EÖTVÖS LORÁND UNIVERSITY

FACULTY OF SCIENCE

MUTUAL INFORMATION ESTIMATION AND ITS UTILIZATION IN DEEP LEARNING

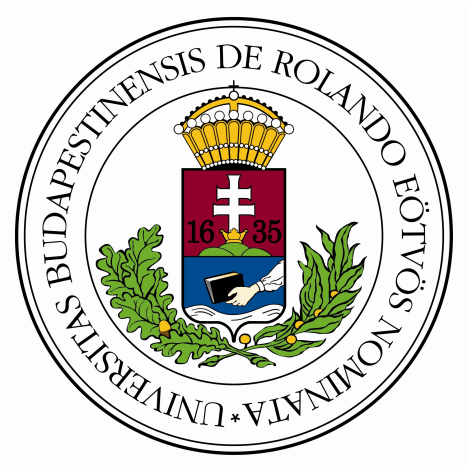
László Sándor Sisák

Applied Mathematics MSc

Supervisors:

Villő Csiszár

Beatrix Benkő



Budapest 2024

Contents

1 Introduction **5**

2 Preliminaries **6**

2.1 Information theory 6

2.1.1 Entropy and differential entropy 6

2.1.2 Mutual information 8

2.1.3 Kullback-Leibler divergence 9

2.2 Artificial neural networks 10

2.2.1 Elements of neural networks 10

2.2.2 Supervised learning via parameter optimization 12

2.2.3 Self-supervised learning 17

2.3 Mutual information in deep learning 20

2.3.1 Mutual information as learning principle 20

2.3.2 Understanding learning dynamics and generalization 22

2.3.3 Limitations and other measures of dependency 23

3 Mutual information estimation **25**

3.1 k-nearest neighbors estimator 25

3.2 Kraskov-Stögbauer-Grassberger estimator 27

3.3 Information noise contrastive estimation 30

3.4 Mutual information neural estimator 32

4 Benchmarking on synthetic data **34**

4.1 Analytical computation 34

4.1.1 Linear synthetic data 34

4.1.2 Gaussian synthetic data 35

4.2 Comparing analytic and estimated mutual information 37

4.2.1 Nearest neighbors-based estimates 37

4.2.2 Neural estimates 39

4.2.3 Comparison of neural and nearest neighbors estimators 41

5 Dataset inference with mutual information score **44**

5.1 Victim encoder training and evaluation 45

5.2 Stealing the victim encoder 46

5.3 Estimating encoder similarity 48

6 Conclusions **50**

I would like to express my gratitude to my thesis supervisors, who have supported me throughout the course of this research and thesis. I would like to thank Beatrix Benkő, who suggested this topic in the first place, helped me find the majority of the cited literature, assisted in understanding and fixing code, and provided guidance on all deep learning-related questions. I am also grateful to Villő Csiszár, who offered her expertise in statistics and patiently answered all of my theoretical questions.

1 Introduction

In information theory, mutual information is a measure that quantifies the amount of information one can learn about a random variable by observing another one. Unlike correlation, it is not limited to measuring linear dependence and it is invariant under homeomorphisms. In deep learning, a measure with such properties finds several applications, from representation learning [2, 33] to studying how neural networks learn and generalize [16, 44]. *In this field of study, measuring the dependence between high-dimensional, continuous, real-valued variables is the primary focus.* Unfortunately, mutual information is notoriously hard to measure for such variables. While in theory it is calculated from the underlying distributions of the variables, in practice we usually have to rely on approximations using finite sets of i.i.d. samples from the joint distribution. We discuss two approaches to estimating mutual information by presenting four mutual information estimators, and evaluating these estimators on synthetic datasets with known distributions. *One aim of this thesis is to provide a comprehensive overview of these estimation methods with the necessary mathematical background, while another is to highlight their connections and applicability in deep learning.*

In Section 2 we lay the theoretical foundations for the rest of the thesis. We first present elementary information theoretic measures in Subsection 2.1: entropy, mutual information and Kullback-Leibler divergence, the properties of these measures and how they are related to one another. Then we discuss the elements of neural networks and deep learning in Subsection 2.2. with a short introduction to self-supervised learning also included. In Subsection 2.3 we provide a brief overview of the role mutual information and information theory plays in the field of deep learning: we reflect on the information bottleneck, InfoMax principle, the information-theoretic generalization bounds and understanding of learning dynamics, finally mention limitations of mutual information and some other measures of dependency that may overcome those. In Section 3 we present in detail four mutual information estimators: two *nearest neighbors*-based methods, the k-NN and KSG estimators, and two lower bounds, MINE and InfoNCE that rely on *neural estimates*. Section 4 contains our experimental setups and results for the benchmarking of them conducted on synthetic data. The distribution of our synthetic data is presented, along with the analytical computation of mutual information in these cases, then the performance of the estimators is evaluated. Finally in Section 5 we introduce the *dataset inference problem* and encoder ownership attribution as applications of mutual information estimation, in an attempt to reproduce the results of [7].

2 Preliminaries

2.1 Information theory

In this section we give a brief information theoretical introduction of the concepts used later. We define entropy, Kullback-Leibler divergence and mutual information, some of the most widely used information theoretic measures. Simple motivating examples and some elementary properties of these measures are also presented.

2.1.1 Entropy and differential entropy

Suppose we have two coins, the first is an ordinary coin that has exactly $\frac{1}{2}$ probability to show heads and $\frac{1}{2}$ probability to show tails when flipped. The other coin is weighted, and has a $\frac{9}{10}$ chance to show heads. One could say that the second coin's outcome is less random than that of the first. Generally it is not so straightforward to grasp the uncertainty of random variables. Entropy is a measure used to quantify the "randomness" or "uncertainty" of a random variable X, introduced by Claude E. Shannon [43]. In the original setting X is a data source, and entropy is a limit on how well a message composed of n i.i.d. samples from X can be compressed. (See Shannon's source coding theorem.) Since then, entropy has seen many other applications.

For continuous random variables, the meaning of entropy isn't as intuitive. The formula for differential entropy presented here was meant to be a continuous analogue of entropy by Shannon, but rather than deriving it, he defined it as is. Later the *limiting density of discrete points* [20] was introduced as a proper continuous analogue of discrete entropy.

Definition 2.1 (Entropy)**.** The entropy of a discrete random variable X over space $\mathcal{X}$ with distribution P_X is

$$H(X) = -\sum_{x\in\mathcal{X}} P_X(x)\log P_X(x) = -\mathbb{E}(\log P_X(X)).$$

The differential entropy of a continuous random variable X over space $\mathcal{X}$ with density function f_X is defined similarly

$$H(X) = -\int_{\mathcal{X}} f_X(x)\log f_X(x)dx = -\mathbb{E}(\log f_X(X)).$$

The base of the logarithm is either 2 (defines the entropy in bits) or e (defines the entropy in *nats*).

Remark 2.1. For discrete entropy, the base of the logarithm is usually taken to be 2 and for differential entropy, base e is more common. This has little significance as long as we remain consistent. For the rest of this work and in all our experiments, the base of the logarithm is always e.

Remark 2.2. For any given discrete random variable X over $\mathcal{X}$, $H(X) \geq 0$ and $H(X) = 0$ iff there exists $c \in \mathcal{X}$ so that $P(X = c) = 1$. For continuous variables, differential entropy $H(X)$ may be negative. For example, if $X \sim \text{Unif}(a, b)$, then $H(X) = \log(b - a)$, which is negative if $b - a < 1$.

Joint entropy of two random variables X and Y can be understood as the entropy of the joint variable (X, Y).

Definition 2.2 (Joint entropy)**.** The joint entropy of n discrete random variables X_1, X_2, ...,X_n over spaces $\mathcal{X}_1$, $\mathcal{X}_2$, ...,$\mathcal{X}_n$ respectively, with joint distribution P is the entropy of the joint distribution

$$H(X_1, \ldots, X_n) = -\sum_{x_1 \in \mathcal{X}_1} \cdots \sum_{x_n \in \mathcal{X}_n} P(x_1, \ldots, x_n) \log P(x_1, \ldots, x_n) = \\ = -\mathbb{E}(\log P(X_1, \ldots, X_n)).$$

The joint entropy of n continuous random variables X_1, X_2, ... X_n over spaces $\mathcal{X}_1$, $\mathcal{X}_2$, ... $\mathcal{X}_n$ respectively, with joint density function f is defined similarly

$$H(X_1, \ldots, X_n) = -\int_{\mathcal{X}_1 \times \cdots \times \mathcal{X}_n} f(x_1, \ldots, x_n) \log f(x_1, \ldots, x_n) dx_1 \ldots dx_n = \\ = -\mathbb{E}(\log f(X_1, \ldots, X_n)).$$

Conditional entropy is also motivated by the mathematics of communication. Suppose that we send a signal over a noisy communication channel. The signal is generated by data source X, but the recipient observes the signal as if it followed the distribution of Y. Conditional entropy quantifies the (expected) remaining uncertainty of the original message after observing Y.

Definition 2.3 (Conditional entropy)**.** The entropy of a discrete random variable Y over space $\mathcal{Y}$ conditioned on discrete random variables X_1, X_2, ... X_n over spaces $\mathcal{X}_1$, $\mathcal{X}_2$, ... $\mathcal{X}_n$ respectively, with joint distribution P is

$$H(Y|X_1, \ldots, X_n) = -\sum_{y \in \mathcal{Y}} \sum_{x_1 \in \mathcal{X}_1} \cdots \sum_{x_n \in \mathcal{X}_n} P(y, x_1, \ldots, x_n) \log P(y|x_1, \ldots, x_n) = \\ = -\mathbb{E}(\log P(Y|X_1, \ldots, X_n)).$$

The entropy of a continuous random variable Y over space $\mathcal{Y}$ conditioned on continuous random variables $X_1, X_2, \ldots, X_n$ over spaces $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n$ respectively, with joint density function f is

$$H(Y|X_1, \ldots, X_n) = -\int_{\mathcal{Y}\times\mathcal{X}_1\times\cdots\times\mathcal{X}_n} f(y, x_1, \ldots, x_n) \log f(y|x_1, \ldots, x_n) dy dx_1 \ldots dx_n$$
$$= -\mathbb{E}(\log f(Y|X_1, \ldots, X_n)).$$

Remark 2.3 (Chain rule for entropy)**.** $H(X, Y) = H(Y) + H(X|Y)$.

2.1.2 Mutual information

Consider our previous example of communication over a noisy channel: the data source follows the distribution of X, but the recipient observes the message as if it followed the distribution of Y. While conditional entropy measured the uncertainty of X after observing Y, mutual information quantifies the information learned of X after reading the message generated by Y.

Definition 2.4 (Mutual information)**.** Given discrete random variables X and Y over spaces $\mathcal{X}$ and $\mathcal{Y}$, with distributions P_X and P_Y, joint distribution $P_{X,Y}$, the mutual information of X and Y is

$$I(X;Y) = \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} P_{X,Y}(x, y) \log \frac{P_{X,Y}(x, y)}{P_X(x)P_Y(y)} = \mathbb{E}\left(\log \frac{P_{X,Y}(X, Y)}{P_X(X)P_Y(Y)}\right).$$

Given continuous random variables X and Y over spaces $\mathcal{X}$ and $\mathcal{Y}$, with density functions f_X and f_Y, joint density function $f_{X,Y}$, the mutual information of X and Y is

$$I(X;Y) = \int_{\mathcal{X}\times\mathcal{Y}} f_{X,Y}(x, y) \log \frac{f_{X,Y}(x, y)}{f_X(x)f_Y(y)} dx dy = \mathbb{E}\left(\log \frac{f_{X,Y}(X, Y)}{f_X(X)f_Y(Y)}\right).$$

Remark 2.4. For any two random variables X, Y $I(X;Y) \geq 0$ and $I(X;Y) = 0$ iff X and Y are independent.

Remark 2.5. $I(X;Y) = H(X) - H(X|Y) = H(X) + H(Y) - H(X, Y)$.

Definition 2.5 (Conditional mutual information)**.** Given discrete random variables X, Y and Z over spaces $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$, with joint distribution $P_{X,Y,Z}$, the mutual information of X and Y conditioned on Z is

$$I(X;Y|Z) = \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} \sum_{z\in\mathcal{Z}} P_{X,Y,Z}(x, y, z) \log \frac{P_Z(z)P_{X,Y,Z}(x, y, z)}{P_{X,Z}(x, z)P_{Y,Z}(y, z)}.$$

Given continuous random variables X, Y and Z over spaces $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$, with joint density function $f_{X,Y,Z}$, the mutual information of X and Y conditioned on Z is

$$I(X;Y|Z) = \int_{\mathcal{X}\times\mathcal{Y}\times\mathcal{Z}} f_{X,Y,Z}(x,y,z) \log \frac{f_Z(z) f_{X,Y,Z}(x,y,z)}{f_{X,Z}(x,z) f_{Y,Z}(y,z)} dxdydz.$$

Remark 2.6 (Chain rule for mutual information)**.**

$$I(X;Y,Z) = I(X;Z) + I(X;Y|Z) = I(X;Y) + I(X;Z|Y).$$

Theorem 2.1 (Data processing inequality)**.** *For a Markov-chain* $X \longleftrightarrow Y \longleftrightarrow Z$*,* $I(X;Y) \geq I(X;Z)$*.*

A result of the latter theorem is that $I(X;Y) \geq I(g_x(X); g_y(Y))$ for all functions g_x, g_y.

2.1.3 Kullback-Leibler divergence

Suppose that we try to guess the probability of heads when flipping a weighted coin. The true chance of showing heads is $\frac{9}{10}$. Intuitively, it is clear that guessing the chance of heads as $\frac{8}{10}$ is closer to the truth than a guess of $\frac{7}{10}$. When modeling more complex problems, the difference between models and reality is much more elusive to measure. Kullback-Leibler divergence is a way of quantifying the distance between the distributions of two random variables X and Y over the same space, introduced by Solomon Kullback and Richard Leibler [26]. While it is often referred to as a distance, it is not a metric over probability distributions in the mathematical sense, as it is not symmetric and does not satisfy the triangle inequality.

Definition 2.6 (Kullback-Leibler divergence)**.** Given discrete probability distributions P_X and P_Y over space $\mathcal{X}$, the KL divergence of P_X and P_Y is

$$D_{KL}(P_X||P_Y) = \sum_{x\in\mathcal{X}} P_X(x) \log \frac{P_X(x)}{P_Y(x)}.$$

Given continuous random variables with density functions f_X and f_Y over space $\mathcal{X}$, the KL divergence of f_X and f_Y is

$$D_{KL}(f_X||f_Y) = \int_{\mathcal{X}} f_X(x) \log \frac{f_X(x)}{f_Y(x)} dx.$$

Remark 2.7. For any two distributions (or density functions) P_X, P_Y over the same space, $D_{KL}(P_X||P_Y) \geq 0$, and $D_{KL}(P_X||P_Y) = 0$ iff $P_X = P_Y$ almost everywhere.

Remark 2.8.

$$I(X;Y) = D_{KL}(P_{X,Y}||P_X P_Y),$$

where $P_{X,Y}$ is the joint distribution (or joint density function) of (X, Y) and P_X, P_Y are the marginals.

2.2 Artificial neural networks

Artificial neural networks are computational models inspired by the structure and functioning of biological neural networks in the human and mammalian brain. They consist of interconnected layers of artificial neurons that process and transform input data to make predictions or classifications. As these models are capable of pattern recognition in high-dimensional inputs, with the computational resource development their usage lead to unprecedented breakthrough results in various fields, including image and speech recognition, natural language processing, and modeling complex systems like protein folding in 3D. Some of their applications gathered significant attention in recent years, such as ChatGPT and various image generation models, leading to a widespread increase in knowledge about them. In this section we provide only a brief introduction to the basic structure and training method of such networks, and for more detailed sources we refer the reader to chapters of [32] and [42].

2.2.1 Elements of neural networks

The basic building block of neural networks is the artificial neuron. As the name suggests, its definition is motivated by biological neurons, even though it is often remarked that they don't model nerve cells accurately. The artificial neuron takes an input $x \in \mathbb{R}^n$, interpreted as the stimuli of other neurons, and if these stimuli reach a certain threshold (usually 0) the neuron *fires* or *activates*, sending stimulus to further neurons.

Definition 2.7 (Artificial neuron)**.** An artificial neuron is a function $h : \mathbb{R}^n \to \mathbb{R}$ in the form

$$h(x) = \Phi(w^T x + b),$$

where $w \in \mathbb{R}^n$ are *weights*, $b \in \mathbb{R}$ is the *bias* and $\Phi : \mathbb{R} \to \mathbb{R}$ is an *activation function*.

Remark 2.9. While there is no universal definition of activation functions, we generally expect it to introduce non-linearity to h and be differentiable almost everywhere. If multiple linear layers are stacked, they can be collapsed to a single linear

layer. Differentiability is necessary for Definition 2.15. The activation function often holds some similarity to the Heaviside step function, which was the activation function used in the original Mark 1 Perceptron machine, built in 1957 [41]. Some popular choices for Φ include:

$$\tanh(x), \quad \text{Sigmoid}(x) = \frac{1}{1+e^{-x}}, \quad \text{ReLU}(x) = \max(0, x), \quad \text{SoftMax}(x)_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}.$$

Definition 2.8 (Artificial neural layer)**.** A neural layer is a function $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^m$ in the form

$$\mathbf{h}(x) = \Phi(W^T x + \mathbf{b}),$$

where $W \in \mathbb{R}^{n \times m}$ are *weights*, $\mathbf{b} \in \mathbb{R}^m$ are *biases* and $\Phi : \mathbb{R}^m \to \mathbb{R}^m$ is an *activation function*, e.g. $\mathbf{h}(x)_i$ is a neuron for all $i \in \{1, \dots m\}$. m is the *width* of the neural layer.

Definition 2.9 (Feedforward neural network)**.** A feedforward neural network is a function $g : \mathbb{R}^n \to \mathbb{R}^m$ in the form

$$g = \mathbf{h}_{m_d} \circ \mathbf{h}_{m_d - 1} \circ \cdots \circ \mathbf{h}_1$$

for some *depth* $m_d \in \mathbb{Z}_{>0}$, so that $\{\mathbf{h}_i : i \in \{1, \dots m_d\}\}$ are all artificial neural layers. In this case we say $\{\mathbf{h}_i : i \in \{1, \dots m_d\}\}$ are the layers of g. The *width* of a feedforward neural network is the maximum width of its layers.

In Definition 2.8, each neuron's output is a function of every element of the input. For this reason, it is also called a fully connected layer. Fully connected layers remain popular for simple tasks, but to apply deep learning in diverse applications, a similarly diverse variety of specialized networks and layers had to be developed. When processing image data, *convolutional layers* are more practical to leverage the spatiality inherent to images.

Definition 2.10 (Convolutional layer [8])**.** A convolutional layer with kernel size $k+1$ is a function $\mathbf{h} : \mathbb{R}^{n_1 \times n_2} \to \mathbb{R}^{n_1 - k \times n_2 - k}$ that satisfies

$$1 \le i \le n_1, 1 \le j \le n_2 : \mathbf{h}(x)_{i,j} = \Phi(w_{i,j}^T \bar{x}_{[i \dots i+k] \times [j \dots j+k]}) + b_{i,j}),$$

where $\bar{x}_{[i \dots i+k] \times [j \dots j+k]}$ denotes a reindexing or *flattening* of $x_{[i \dots i+k] \times [j \dots j+k]}$ so that we can view it as a vector, rather than a matrix. (The exact method of reindexing is not important as long as we remain consistent.)

Inspired by research in neurobiology [19], convolutional neural networks often use alternating convolutional and *max-pooling layers*.

Definition 2.11 (Max-pooling layer)**.** A max-pooling layer with kernel size $k+1$ is a function $\mathbf{h} : \mathbb{R}^{n_1 \times n_2} \to \mathbb{R}^{n_1-k \times n_2-k}$ of the form

$$1 \leq i \leq n_1, 1 \leq j \leq n_2 : \mathbf{h}(x)_{i,j} = \max_{i \leq i' \leq i+k, 1 \leq j' \leq j+k} \{x_{i',j'}\}.$$

A shortcoming of feedforward neural networks with large depth m_d is that the stochastic gradient descent algorithm used to train the network (see Section 2.2.2) converges very slowly. To circumvent this problem, *residual networks* were introduced, where there are connections between layers not directly neighboring each other.

Definition 2.12 (Residual block [17])**.** A residual block is a function $g : \mathbb{R}^n \to \mathbb{R}^m$ in the form

$$g(x) = \hat{g}(x) + x,$$

where $\hat{g}$ is an artificial neural network itself.

Definition 2.13 (Residual neural network)**.** A residual neural network is a function $g : \mathbb{R}^n \to \mathbb{R}^m$ in the form

$$g = g_{m'_d} \circ g_{m'_d-1} \circ \cdots \circ g_1$$

so that $\{g_i : i \in \{1, \dots m'_d\}\}$ are all residual blocks or neural layers.

More intricate connections between layers and neurons are also possible, e.g. recurrent neural networks allow for backward loop of information flow in the network, well-suited for modeling sequential data similar to memory utilization, but we do not detail these further here, as they are not utilized in this work.

2.2.2 Supervised learning via parameter optimization

The number and type of layers, number of neurons in each layer and hyperparameters such as the kernel size of convolutional layers in a network are collectively referred to as the *architecture* of a neural network. The weights and biases of each layer are collectively referred to as the *parameters* θ. In deep learning we fix a neural architecture (and maybe some of the parameters, these are called fixed parameters) and view the neural network as a parametric function. We then attempt to find

a parametrization θ in the set of possible parameters Θ so that we sufficiently approximate an input-target distribution (X, Y) with $(X, g_\theta(X))$. Naturally, we only have access to a finite dataset D of i.i.d. samples from (X, Y). Iterative algorithms attempting to find such θ are called *training algorithms*.

Currently we only have (often very developed) heuristic ideas on how to fix a suitable architecture for any given problem, more so considering the ever-growing literature of possible models. Finding θ, however, is universally achieved through empirical risk minimization.

Empirical risk minimization is not specific to neural networks, it is a setting in statiscical learning, where we seek to estimate and optimize the performance of a model based on a finite set of training data. We assume there is a loss function measuring how different our g_θ hypothesis function's output is from the true outcome.

Definition 2.14 (Loss function)**.** Suppose that the output variable Y takes values from $\mathcal{Y}$. Then the loss function is a function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$, with $L(y, y) = 0$ for any $y \in \mathcal{Y}$.

Example. For commonly considered *classification problems* – where $\mathcal{Y}$ is a finite set – the 0-1 loss is a straightforward choice:

$$L(g_\theta(x), y) = \begin{cases} 0 & \text{if } g_\theta(x) = y, \\ 1 & \text{otherwise.} \end{cases}$$

For such classification problems, smoother loss functions are more ideal. For this reason, some neural networks g_θ, e.g. using SoftMax activation function output class probabilities, i.e. positive real numbers $g_\theta(x)_1, g_\theta(x)_2, \dots g_\theta(x)_m$ so that $\sum_{i=1}^m g_\theta(x)_i = 1$ and predict the class with the highest probability. For networks such as these, the cross-entropy loss can be used:

$$L(g_\theta(x), y) = -\sum_{i=1}^{m} g_\theta(x)_i \log(y_i),$$

where $y > 0$ also sums to 1.
In general cases, we may simply sum the differences of the prediction and y:

$$L(g_\theta(x), y) = \sum_{i=1}^{m} |g_\theta(x)_i - y_i|$$

for a $|\cdot|$ norm of our choice.

Our goal is to find parameters minimizing the expected loss or risk

$$\arg\min_{\theta\in\Theta} \left\{\mathbb{E}(L(g_\theta(X), Y))\right\}.$$

As mentioned already, we can't minimize this risk directly. Rather, we rely on the sample D and the law of large numbers to approximate the expected loss with the empirical risk

$$\arg\min_{\theta\in\Theta} \left\{\frac{1}{|D|}\sum_{(x,y)\in D} L(g_\theta(x), y))\right\}.$$

Directly minimizing the above functional is possible in theory, but not in practice. For most problems, there are no guarantees that the training algorithm finds this global minimum, and optimizing over the set of all possible inputs – such as all possible images for image classification – is computationally infeasible. The difference

$$\left|\mathbb{E}(L(g_\theta(X), Y)) - \frac{1}{|D|}\sum_{(x,y)\in D} L(g_\theta(x), y))\right|$$

for any given neural network g with parameters θ is the generalization gap. In most cases, the generalization gap can only be measured through heuristic methods, such as separating part of the dataset D for testing the model once training with the rest of the dataset is finished. Fortunately neural networks have been found to generalize well, even on tasks such as image classification, where no dataset can hope to capture the full range of possible inputs.

Training neural networks is universally achieved with gradient descent and the backpropagation algorithm. The backpropagation algorithm is a way of computing gradients of the loss function efficiently for feedforward neural networks and other neural architectures used in the field. To perform gradient descent, the loss function L has to be (partially) differentiable. We compute the partial derivatives of $\sum_{(x,y)\in D} L(g_\theta(x), y))$ with respect to θ. This derivative represents the direction in which the loss can be increased from our current set of parameters θ. Intuitively, loss should decrease if we shift the parameters in the opposite direction.

Definition 2.15 (Gradient descent for neural networks)**.** Let $\theta_0 \in \Theta$ some initial parametrization of neural network g. We fix a *learning rate* $\lambda > 0$. Then gradient descent follows the iteration

$$\theta_{i+1} = \theta_i - \lambda\left(\frac{1}{|D|}\sum_{(x,y)\in D} \nabla_\theta L(g_{\theta_i}(x), y))\right).$$

Definition 2.16 (Backpropagation algorithm)**.** Since we defined feedforward neural networks as a composition of layers, we may calculate the derivative in Definition 2.15 using the chain rule.

$$g' = \prod_{i=1}^{m_d} (\mathbf{h}_i' \circ \mathbf{h}_{i-1} \circ \cdots \circ \mathbf{h}_1).$$

We can compute the partial derivatives of the loss with regard to each layer's parameters the derivative using dynamic programming, deriving g layer by layer, (or first block by block, for residual networks) starting with $\mathbf{h}_{m_d}$. Finding the derivative of each individual neural layer is simple for popular activation functions such as ReLU.

Remark 2.10. Due to the massive amount of training data D in modern applications, usually it is not feasible to compute the gradient on the entire D at once. Rather, we divide D into *batches* $B_1, \dots B_l$ of fewer samples, and iterate the following parameter-update:

$$\theta_{i+1} = \theta_i - \lambda \left(\frac{1}{|B_i|} \sum_{(x,y) \in B_i} \nabla_\theta L(g_{\theta_i}(x), y)) \right).$$

Whenever we run out of batches, we randomly shuffle D and divide it into new batches to iterate further. Iterating over all batches of the training data once during training is called a *training epoch*.

Classic gradient descent often gets stuck in local minima. To overcome this limitation, several improvements for the update-iteration have been considered. One major branch of these improvements focus on adding *momentum* to the update rule. A momentum is a weighted moving average of past gradients, which is meant to decrease the risk of the iteration converging to local minima and increasing the speed of global convergence.

Definition 2.17 (Momentum-based gradient descent for feedforward networks)**.** Let $\theta_0 \in \Theta$ some initial parametrization of neural network g and $V_0 = \mathbf{0}$ the initial momentum. We fix a learning rate $\lambda > 0$ and momentum parameter $\beta \in [0, 1)$. Then gradient descent with momentum follows the iteration

$$V_{i+1} = \beta V_i + \lambda \left(\frac{1}{|D|} \sum_{(x,y) \in D} \nabla_\theta L(g_\theta(x), y)) \right)$$

$$\theta_{i+1} = \theta_i - V_{i+1}.$$

While there is no objective best way to add a momentum term to the update rule, by far the most popular improvement of gradient descent is Adaptive Moment Estimation (Adam) [22].

Definition 2.18 (Adam stochastic optimization algorithm)**.** Let $\theta_0 \in \Theta$ some initial parametrization of neural network g and $V_0 = v_0 = \mathbf{0}$ initial momentum. We fix a learning rate $\lambda > 0$, momentum parameters $\beta_1, \beta_2 \in (0, 1]$ and a small $\epsilon > 0$. Adam follows the iteration

$$\hat{\nabla}_{i+1} = \left(\frac{1}{|D|} \sum_{(x,y)\in D} \nabla_\theta L(g_\theta(x), y)) \right),$$

$$V_{i+1} = \beta_1 V_i + (1-\beta_1)\hat{\nabla}_{i+1}, \quad v_{i+1} = \beta_2 v_i + (1-\beta_2)\hat{\nabla}^2_{i+1},$$

$$\hat{V}_{i+1} = V_{i+1}/(1-\beta_1^{i+1}), \quad \hat{v}_{i+1} = v_{i+1}/(1-\beta_2^{i+1}),$$

$$\theta_{i+1} = \theta_i - \lambda \hat{V}_{i+1}/\sqrt{\hat{v}_{i+1} + \epsilon}.$$

Can we hope that there is a parametrization that minimizes the empirical risk $\frac{1}{|D|} \sum_{(x,y)\in D} L(g_\theta(x), y))$ efficiently? Several universal approximation theorems exist, proving that neural networks can approximate a wide variety of functions with arbitrary accuracy. Many of these theorems formulate statements about feedforward neural networks of bounded depth, and more recently, of bounded width. Here we only cite two strong results, the first of which is presented as a slightly weaker statement for easier readability.

Theorem 2.2 (Universal approximation theorem, bounded depth [37])**.** *Let $\Phi : \mathbb{R} \to \mathbb{R}$ continuous, non-polynomial and $K \subseteq \mathbb{R}^n$ any compact set. Let $\mathcal{G}$ be the set of neural networks $g : \mathbb{R}^n \to \mathbb{R}^m$ with depth 2, where the activation function of each layer is Φ (coordinate-wise) or the identity function. Then $\mathcal{G}$ is dense in $C(K, \mathbb{R}^m)$ w.r.t. the supremum norm. (That is, for any continuous function $F : K \to \mathbb{R}^m$ and $\epsilon > 0$, there is a $g \in \mathcal{G}$ function so that $\sup_{x\in K} |F(x) - g(x)| < \epsilon$.)*

Theorem 2.3 (Universal approximation theorem, bounded width [21])**.** *Let $\Phi : \mathbb{R} \to \mathbb{R}$ continuous, non-affine, continuously differentiable in some $x \in \mathbb{R}$ so that $\Phi'(x) \neq 0$ and $K \subseteq \mathbb{R}^n$ any compact set. Let $\mathcal{G}$ be the set of neural networks $g : \mathbb{R}^n \to \mathbb{R}^m$ with width $n + m + 2$, where the activation function of each layer is Φ (coordinate-wise) or the identity function. Then $\mathcal{G}$ is dense in $C(K, \mathbb{R}^m)$ w.r.t. the supremum norm.*

This shows that neural networks can model most functions of practical interest.

2.2.3 Self-supervised learning

Finally, we provide a brief introduction to *representation learning* and *self-supervised learning* (SSL), as one of the mutual information estimators, discussed in Section 3.3, was designed to be a loss function for self-supervised representation learning.

In Section 2.2.2, we assumed that we have access to a dataset D composed of input-output pairs (x_i, y_i). This is the classic theoretical model for *supervised learning*. The drawback of supervised learning is that creating large datasets with task-specific output labels y_i is a time consuming and expensive task in and of itself.

Representation learning seeks to mitigate this problem by preprocessing the input X so that the resulting *embeddings*, *representations* or *abstract features* $Z \subset \mathbb{R}^l$ are easier to process for a wide variety of tasks (X, Y') with a limited dataset. In the context of representation learning, these tasks are called *downstream tasks*. In a supervised setup, learning Z can be done by training a neural network g with depth m_d for a specific task (X, Y) (with a sufficiently large available dataset D), and later reusing the first $m'_d < m_d$ layers. We view this truncated neural network g' with depth m'_d as an *encoder* that maps X to embeddings Z. We can then train other networks with input X and target Y' by using g' to transform the task (X, Y') to $(g'(X), Y') = (Z, Y')$, i.e. an easier task.

This helps learning downstream tasks because it is theorized that a neural network's layers extract increasingly abstract features from the data, and only a few layers preceding the output layer perform task-specific transformations. While the exact inner workings of artificial neural networks are not fully understood, this theory is supported by findings such as the hierarchical organization of features across layers – earlier layers capture low-level (small context) features while deeper layers represent increasingly abstract (large context) concepts – and the identification of *multimodal neurons* in neural networks. A multimodal neuron fires when the network receives stimulus related to an abstract concept. One such neuron is the Donald Trump neuron [10], which fired whenever the network's input was a photo or drawing of Donald Trump, or an image of text that reads "Donald Trump".

In SSL we try to learn useful representations for task-specific problems without ever relying on a labeled dataset $D = \{(x_i, y_i) | 1 \leq i \leq n\}$ of a specific task (X, Y). SSL's input is a dataset D of samples from a random variable X for which we try to learn representations.

Example. Large language models are often trained in a self-supervised manner

using the masking problem. In this problem, we hide a few words of the input text from the model (masking), and task the model with correctly guessing the masked word from the context. A similar method can be employed to train image generation models by masking parts of the input image.

Here we further detail the foundation of multiple SSL frameworks, particularly *contrastive learning* techniques, a probabilistic estimation method called *noise contrastive estimation* (NCE) [15], which can be used to approximate intractable probability distributions. In contrastive learning, we assume that each data point x from D is associated with a *context* y, which provides additional information that helps define the conditional distribution $p(x|y)$. The context y could represent, for instance, a temporal or spatial neighbor of x, or a related data modality. We aim to estimate $p(x|y)$, to achieve this we introduce a parametric (neural) *scoring function* $t_\theta(x, y) > 0$, that assigns a score proportional to the likelihood of x given y:

$$p(x|y) \approx p_\theta(x|y) = \frac{t_\theta(x, y)}{\sum\limits_{x' \in D} t_\theta(x', y)},$$

where the denominator is the normalization term. Computing this sum directly is often infeasible for large datasets, motivating simplification with NCE.

In NCE, instead of modeling $p(x|y)$ explicitly, we frame the problem as a binary classification task. Specifically, we assume that only $x \in D$ is a *positive sample* drawn from the true conditional distribution $p(x|y)$, while all other samples $x' \in D \setminus \{x\}$ are treated as *negative samples* following a predefined noise distribution $q(x')$. The key assumption is that $p(x|y) \geq q(x')$ for all $x' \in D \setminus \{x\}$. In place of directly approximating $p(x|y)$, NCE estimates the probability that a given sample x originates from $p(x|y)$ rather than the noise distribution $q(x)$. Let $\mathbb{I}$ be an indicator variable denoting whether x follows $p(x|y)$ ($\mathbb{I} = 1$) or $q(x)$ ($\mathbb{I} = 0$). The conditional probabilities can then be expressed as:

$$p(\mathbb{I} = 0|x, y) = \frac{(|D| - 1)q(x)}{p(x|y) + (|D| - 1)q(x)},$$

$$p(\mathbb{I} = 1|x, y) = \frac{p(x|y)}{p(x|y) + (|D| - 1)q(x)}.$$

The NCE process requires optimizing the parameters of the scoring function $t_\theta(x, y)$ to maximize the likelihood of the observed data, in practice typically done by minimizing a suitable loss function between the predicted probabilities for the indicator variable $\mathbb{I}$ and the implicitly defined labels (1 for true distribution, 0 for

noise distribution). This optimization encourages the probabilistic model to assign higher scores to samples that follow the true distribution $p(x|y)$, thereby learning an efficient approximation of the conditional distribution without needing to compute the normalization term explicitly.

To understand how this could be used for self-supervised representation learning, let us present as example the SimCLR method [3]. In every training step, we sample a batch $B = \{x_1, x_2, \ldots, x_b\} \subseteq D$ of b data points from the dataset. We then *augment* each data point $x_i \in B$ with transformations that retain all relevant features, two different stochastic data augmentation functions are applied, resulting in augmented *views* of $x_i \in D$. In the case of image data, such augmentation can include rotating, cropping or adding noise to the image x_i, as long as the augmented version remains identifiable as a transformed version of the same item. The two augmented views of the same sample constitute a *positive pair*, while the augmented views of all other data points in the batch act as *negative samples* for them. We denote one with the same x_i as before, and its pair with y_i as context for it. The parametric model, a neural network g_θ maps the augmented views $\{x_1, x_2, \ldots, x_b, y_1, y_2, \ldots, y_b\}$ to embeddings $\{g_\theta(x_1), g_\theta(x_2), \ldots, g_\theta(x_b), g_\theta(y_1), g_\theta(y_2), \ldots, g_\theta(y_b)\} \subset \mathbb{R}^l$. To complete the training step, we minimize a contrastive loss function that essentially implements the NCE idea of contrasting positive pairs against negative pairs, and rewards packing the embeddings $g_\theta(x_i)$ and $g_\theta(y_i)$ closely, far away from negative samples' embeddings. Some variant of the SGD algorithm is used to update the network's parameters θ, then the step is repeated with new batches, until convergence.

Naturally, we need to define an appropriate loss function, as Definition 2.14 can only describe supervised training. Note that we may have more than one pair of elements in $\{x_1, x_2, \ldots x_b, y_1, y_2, \ldots, y_b\}$ that are views of the same data point from D. These all are called *positive pairs*. The loss function of SimCLR defined as follows.

Definition 2.19 (Normalized temperature-scaled cross entropy loss, NT-Xent)**.** For a set of augmented samples $B = \{x_1, x_2, \ldots, x_b\}$, let

$$L_{ij}(x_1, \ldots, x_b) = -\log \frac{\exp(\mathrm{sim}(g_\theta(x_i), g_\theta(x_j))/\tau)}{\sum_{x_k \in B \setminus x_i} \exp(\mathrm{sim}(g_\theta(x_i), g_\theta(x_k)/\tau))},$$

where x_i, x_j is a positive pair, $\mathrm{sim} : \mathbb{R}^l \times \mathbb{R}^l \to \mathbb{R}$ is a similarity metric and $\tau > 0$ is a *temperature parameter*. Then the NT-Xent loss of network g over $\{x_1, x_2, \ldots, x_b\}$ is $L = \sum_{(x_i, x_j) \in B_+} L_{ij}$, where B_+ is the set of ordered positive pairs in B.

In this example, the scoring function is $t_\theta(x, y) = \exp(\text{sim}(g_\theta(x), g_\theta(y))/\tau)$, with the cosine similarity function to the compare embedding vectors, and instead of the whole dataset D, we calculate it over a batch B. The loss function in Definition 2.19 can be written as

$$-L = \sum_{(x,y)\in B_+} \left(\log \frac{t_\theta(x, y)}{\sum\limits_{x'\in B\setminus\{y\}} t_\theta(x', y)} \right) \approx \sum_{(x,y)\in B_+} \log p(\mathbb{I} = 1|x, y)$$

with the notation of the NCE framework.

A different approach to simplify estimating $p(x|y)$ is to define a random variable I over the batch $B = \{x_1, x_2, \dots x_b\}$ given some context y. $I = i$ if and only if (x_i, y) is the positive pair and any other $x' \in B \setminus \{x_i\}$ comes from the noise distribution. If we approximate

$$p(I = i|B, y) = \frac{p(x_i|y)\Pi_{k\neq i}p(x_k)}{\sum_{j=1}^{b}(p(x_j|y)\Pi_{k\neq j}p(x_k))} = \frac{\frac{p(x_i|y)}{p(x_i)}}{\sum_{j=1}^{b}\frac{p(x_j|y)}{p(x_j)}} \approx \frac{t_\theta(x_i, y)}{\sum_{j=1}^{b} t_\theta(x_j, y)},$$

then the scoring function $t_\theta(x, y)$ approximates $\frac{p(x|y)}{p(x)}$ up to a multiplicative constant. See Lemma 3.3 for proof.

2.3 Mutual information in deep learning

Mutual information plays an important role in deep learning, especially in the context of representation learning. This section discusses such forms of utilization. The InfoMax principle is a theoretical framework that builds on the intuition that a good representation should retain as much information about the input as possible. On the other hand, the information bottleneck principle aims to find a balance between retaining information about a target (relevant information) and discarding superfluous information. In contrastive learning, the InfoNCE loss is used to train networks to maximize the mutual information between representations of similar inputs. (See Section 3.3 for details.) Mutual information is also used to understand the inner workings of neural networks.

2.3.1 Mutual information as learning principle

The information bottleneck method is an information theoretic framework for finding the best trade-off between accuracy and complexity (compression) when

summarizing (e.g. clustering) a random variable X, given a joint probability distribution $p(x, y)$ between X and an observed relevant variable Y over $\mathcal{X} \times \mathcal{Y}$ [47]. In this method, we are looking for a compression of X represented by random variable Z to find

$$\inf_{p(z|x)} \{I(X;Z) - \beta I(Z;Y)\},$$

where β is a parameter representing the amount of information Z should retain about Y. One example the authors give is speech compression: for a vast number of downstream tasks a transcript of the speech is sufficient information. The resulting text Z occupies memory space several orders of magnitude smaller than the memory needed to store lossless compression of the audio X. In the context of deep learning, this framework is easily applicable to supervised representation learning. X is the input data, Y is the target label and Z is the representation provided by some neural network. Minimizing the above functional would mean minimizing $I(X;Z)$ (discarding irrelevant information from X) and maximizing $I(Z;Y)$ (retaining relevant information about Y).

We may call a representation *sufficient* if $I(X;Y|Z) = 0$. $Z = g_\theta(X)$ is a representation, this means Z, X and Y form a Markov-chain $Y \longleftrightarrow X \longleftrightarrow Z$ and $I(Y;Z|X) = 0$. Thus

$$I(X;Y) - I(X;Y|Z) = I(Y;Z) - I(Y;Z|X) \Longrightarrow I(X;Y) = I(Y;Z) \text{ iff } Z \text{ sufficient.}$$

This means that access to a sufficient representation Z enables us to predict Y at least as accurately as if we had access to X. In this case, $I(X;Z) = I(X,Y;Z) = I(Y;Z) + I(X;Z|Y)$. Here $I(Y;Z)$ represents information predictive of target Y, and $I(X;Z|Y)$ represents superfluous information.

In the original work [47], the authors constrained the amount of information Z should retain and aimed to compress Z. They remark that alternatively we may constrain the size of Z and maximize the retained information instead. In deep learning it is more intuitive to do the latter. We fix a neural architecture that maps from $\mathcal{X}$ to $\mathcal{Y}$, with a narrow neural layer in between. The output of this narrow layer is the representation Z. The neuron count of the layer applies a natural constraint on the information that can be transferred between the input and output layers, thus we seek to maximize $I(Z;Y)$.

To apply the information bottleneck, we have to measure mutual information between X, Z and Y. As the underlying distribution of these variables is not known in the majority of applications, we have to rely on approximations. This task is more

challenging in machine learning, where the dimension of the representation Z is often in the hundreds, and the dimension of the input X can easily be in the thousands in the case of image data. In most cases both the images and the representations come from continuous distributions, further complicating accurate approximation.

In unsupervised training, the *InfoMax principle* [28] inspired several training objectives. The InfoMax principle is the reasonable intuition that a good representation retains as much information from the input data as possible. The training objectives based on this philosophy are lower bounds on the mutual information between the representations and the input, and parameters are adjusted to maximize this bound. Two such lower bounds are discussed in Sections 3.4 and 3.3.

2.3.2 Understanding learning dynamics and generalization

While deep learning gained popularity for its empirical effectiveness on a variety of hard problems, there is no comprehensive theoretical foundation for understanding how neural networks learn and converge. Mutual information may be the measure necessary to develop such a foundation, or at least to provide a deeper mathematical analysis of neural networks than test accuracy.

Overfitting is one of the oldest problems in the practical use of neural networks. During training, the longer we train a neural network, the lower the empirical error gets on the training dataset. However, it has been observed that after enough iterations, the error starts to increase on the separate test set. (The test set is not used for training, and is drawn from the same distribution as the training set.) When this happens, the network or the parameters are said to be overfitted on the training data, and the network *memorized* the training data. Neural networks were even shown to achieve 0 (supervised) training error when the labels of the training data are randomized [52], i.e. they memorize noise. One proposed measure to understand generalization and memorization is to provide mutual information-based generalization bounds. Several bounds use the mutual information between the input data and the weights [16] (more specifically, the input data and the corresponding outputs of the training algorithm).

Some researchers viewed the data X, target label Y and the outputs Z_1, Z_2, $\ldots, Z_{m_d}$ of each neural layer as a Markov-chain.

$$Y \longleftrightarrow X \longleftrightarrow Z_1 \longleftrightarrow \cdots \longleftrightarrow Z_{m_d}.$$

Here, Z_{m_d} would be the final estimate of Y based on input X, while Z_i, $i < m_d$

are hidden representations. Schwarz-Ziv and Thisby [44] estimated $I(X; Z_i)$ and $I(Y; Z_i)$ repeatedly during training, and found that training with stochastic gradient descent can be separated into two phases. In the first phase the layers increase information on label Y. Then in the second, much longer phase, $I(X; Z_i)$ decreases, indicating that the neural network starts discarding superfluous information from the input. They also found that once the layers converge (when their parameters are not significantly altered by further training) the representations Z_i are very close to the theoretical information bottleneck bound, for different parameters β.

2.3.3 Limitations and other measures of dependency

While Shannon's mutual information is theoretically well motivated, in practice proves to be hard to estimate in high dimensions [5]. The training objectives motivated by the InfoMax-principle are popular in practice, but mutual information alone does not guarantee useful representations. It has been shown that one can engineer representations that retain a lot of information, yet lead to poor performance on downstream tasks [48]. Furthermore, any lower bounds on mutual information were proven to be at most $O(\log N)$ on N samples, see Theorem 1 from [34] and Theorem 1.1 from [31]. Another shortcoming of mutual information is that it does not account for the topology of the data. This is a natural consequence of mutual information's invariance to invertible transformations. Notably, mutual information is blind to the Vapnik-Chervonenkis dimension [50] of supervised classification tasks.

Due to the limitations of mutual information, several other dependency measures are utilized in research. Canonical Correlation Analysis [18], Pearson and Spearman rank correlations are popular, but quantify only linear and monotonic relationships, therefore these are not suitable for capturing potentially non-linear and higher-order interactions between multidimensional variables of complex data e.g. image inputs.

In the era of large quantities of such complex high-dimensional data the focus is shifting towards developing easily computable measures of non-linear dependencies. One way is to process the input data via kernels or project to lower dimensions via learned parametric models e.g. neural networks – for instance, autoencoders are suitable for such task.

Sliced mutual information [11], and especially its k-sliced [12] and max-sliced versions [49] aim for such estimation via considering random projections of X and Y onto k-dimensional subspaces or maximal mutual information preserving projections, and calculating mutual information between the projections. These are

easy to approximate and retain many favorable properties akin to mutual information, such as nonnegativity, a chain rule similar to Remark 2.6 and a relation to Kullback-Leibler divergence similar to Remark 2.8. Unfortunately, the data processing inequality does not hold for sliced mutual information variants, and the gap between sliced and classical mutual information may not be bounded.

The Hilbert-Schmidt independence criterion [14] is a kernel-based measure of dependence for comparing probability distributions by first taking a non-linear feature transformation of each, then evaluating the norm of the cross-covariance between those features in a reproducing kernel Hilbert space. For many kernels, $\mathrm{HSIC}(X, Y) = 0$ iff X and Y are independent, but unlike mutual information, the Hilbert-Schmidt independence criterion incorporates geometry via the choice of kernel. It is both statistically and computationally easy to estimate [46] and it has been successfully utilized for self-supervised learning [27].

The Wasserstein dependency measure [34] replaces the KL-divergence in the classical definition of mutual information with the Wasserstein distance [51] between the joint and the product of marginal distributions, therefore measures the cost of transforming samples from the marginals to samples from the joint, connecting to optimal transport methods.

Since the main focus of this thesis is the estimation of mutual information, such novel masures of dependecy were not considered for comparison.

3 Mutual information estimation

This section gives a theoretical introduction of the four mutual information estimators we test in Section 4. We aim to estimate the mutual information of continuous random variables X and Y over $\mathbb{R}^{d_1}$ and $\mathbb{R}^{d_2}$ respectively, given i.i.d. samples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \sim (X, Y)$. Note that any entropy estimator $\hat{H}(\cdot)$ can be used to estimate mutual information using Remark 2.5, if we let $\hat{I}(X;Y) = \hat{H}(X) + \hat{H}(Y) - \hat{H}(X,Y)$.

We did not consider estimators that rely on further assumptions about the distribution, such as the assumption that X or Y belongs to a specific family of distributions. This excludes estimates such as GMM-MI, which fits normal distributions to the data available [38] or geodesic k-NN, which assumes that the (high-dimensional) datapoints lie on a manifold of significantly lower dimension [30]. While there is a variety of estimation approaches for low-dimensional variables [1], due to our focus on applicability in deep learning, we excluded estimators that are not scalable for any $d_1, d_2 \in \mathbb{N}$. Histogram-based entropy and mutual information estimators designed for discrete distributions could be applicable to a continuous distribution after discretization, but we don't discuss these methods either, as their results are heavily dependent on the granularity of discretization.

Lower bounds on mutual information and unbiased estimates are our primary focus as they have many applications in common, but upper bounds on mutual information have also been developed, such as the upper contrastive logarithmic ratio bound [4] and the leave-one-out upper bound [39]. They are applied in contexts, where one can reasonably assume that X and Y are highly dependent, such as preventing overfitting or ensuring differential privacy. A lower bound we did not include is the Nguyen-Wainwright-Jordan lower bound, which – similarly to the MINE and InfoNCE estimates – trains a neural network to maximize a parametric lower bound.

3.1 k-nearest neighbors estimator

In this section, we first show the k-nearest neighbors (k-NN) entropy estimate of Singh et al. [45] which generalizes the Kozachenko-Leonenko entropy estimate [23], the k-NN estimate for $k = 1$. We estimate $H(X)$ by approximating the probability

density function f_X of X and applying the Monte Carlo method to the estimated $\hat{f}$:

$$\hat{H}(X) = -\frac{1}{n}\sum_{i=1}^{n} \log \hat{f}(x_i).$$

Fix $k \in \{1, 2, \ldots, n\}$ and let

$$r_i = \{|x_i - x_j| : x_j \text{ is the } k\text{-th nearest neighbor of } x_i \text{ in } \{x_1, x_2, \ldots, x_n\} \setminus \{x_i\}\}$$

for the euclidean norm $|\cdot|$. Then the closed d-dimensional ball $B(x_i, r_i)$ contains exactly k of the n sample points, (other than x_i, assuming $|x_j - x_i| = |x_{j'} - x_i|$ iff $j = j'$) thus a reasonable estimate of $\hat{f}(x_i)$ is given by

$$\hat{f}(x_i)\text{Vol}(B(x_i, r_i)) = \hat{f}(x_i)\frac{\pi^{d/2} r_i^d}{\Gamma(d/2+1)} = \frac{k}{n}$$

assuming the probability density is constant in $B(x_i, r_i)$, and we can define the entropy estimator

$$\hat{H}(X) = -\frac{1}{n}\sum_{i=1}^{n} \log\left(\frac{k}{n}\frac{\Gamma(d/2+1)}{\pi^{d/2} r_i^d}\right) = \log\left(\frac{n\pi^{d/2}}{k\Gamma(d/2+1)}\right) + \frac{d}{n}\sum_{i=1}^{n}\log(r_i).$$

Remark 3.1. The estimator can be generalized for any p-norm, in which case we measure $|\cdot|_p$-distances and $\text{Vol}(B(x_i, r_i, p)) = \frac{\Gamma(1+1/p)^d}{\Gamma(1+d/p)} r_i^d$. In its most general form, this estimate is written as

$$\hat{H}(X) = \log\left(\frac{n}{k} c_d\right) + \frac{d}{n}\sum_{i=1}^{n}\log(r_i),$$

where c_d is the volume of the d-dimensional unit ball in the norm of our choice.

Theorem 3.1 ([45])**.** *The asymptotic mean of the above $\hat{H}(X)$ for sample size $n \to \infty$ is given by*

$$\mathbb{E}(\lim_{n\to\infty} \hat{H}(X)) = \Psi(k) - \log k + H(X),$$

where $\Psi(k) = \frac{\Gamma'(k)}{\Gamma(k)}$ is the digamma function.

This means that the estimate above is biased. By correcting this bias, we obtain the asymptotically unbiased k-nearest neighbors entropy estimate.

Definition 3.1 (k-nearest neighbors entropy estimator)**.** Given an i.i.d. sample $x_1, x_2 \ldots, x_n$ of a continuous random variable X over $\mathbb{R}^d$, fix any $k \in \{1, 2, \ldots, n\}$.

Let

$$H_{kNN}(X) = \log\left(\frac{\pi^{d/2}}{\Gamma(d/2+1)}\right) + \frac{d}{n}\sum_{i=1}^{n}\log(r_i) - \Psi(k) + \log n,$$

where r_i is the distance from x_i to its k-th nearest neighbor and Ψ is the digamma function.

Theorem 3.2 ([45])**.** *$lim_{n\to\infty} Var(H_{kNN}(X)) = 0$, i.e. H_{kNN} is consistent.*

Definition 3.2 (k-nearest neighbors mutual information estimator)**.** Given an i.i.d. sample $(x_1, y_1), \ldots (x_n, y_n)$ of absolute continuous random variables (X, Y) over $\mathbb{R}^{d_1} \times \mathbb{R}^{d_2}$, fix any $k \in \{1, 2, \ldots, n\}$. Let

$$I_{kNN}(X;Y) = H_{kNN}(X) + H_{kNN}(Y) - H_{kNN}(X,Y).$$

Remark 3.2. While $I(X;Y) \geq 0$, the same does not hold for $I_{kNN}(X;Y)$, as the negative term $H_{kNN}(X,Y)$ may be overestimated, or the positive terms $H_{kNN}(X)$, $H_{kNN}(Y)$ underestimated.

3.2 Kraskov-Stögbauer-Grassberger estimator

In the nearest neighbors method we limit ourselves to estimating $H(X)$ and $H(Y)$ using information exclusively from the marginal distributions X and Y. Kraskov, Stögbauer and Grassberger present two algorithms that seek to improve the kNN estimator [24] by estimating marginal probability densities with the help of the joint distribution. The algorithms are similar, so both of them are referred to as Kraskov-Stögbauer-Grassberger estimator or KSG in literature.

We use the metric $||(x,y) - (x',y')|| = \max\{|x - x'|, |y - y'|\}$ to find r_i – the distance between (x_i, y_i) and its k-th nearest neighbor – on the joint space $\mathbb{R}^{d_1} \times \mathbb{R}^{d_2}$. In the first algorithm, we count the number of points in the *open* balls $B'_x(x_i, r_i) \subseteq \mathbb{R}^{d_1}$ and $B'_y(y_i, r_i) \subseteq \mathbb{R}^{d_2}$ and denote these numbers $n_{x,i}$ and $n_{y,i}$ respectively. While usually only one of these balls contains the marginal projection of (x_i, y_i)'s k-th nearest neighbor (see Figure 1/a), the authors chose to define the relative density of these balls as $\frac{n_{x,i}+1}{n}$ and $\frac{n_{y,i}+1}{n}$, counting the k-th nearest neighbor in both marginals. In case one of the marginal projections contain no neighbors, this also ensures that we do not calculate $\log 0$, see Definition 3.3.

We use the same heuristic for estimating probability density in (x_i, y_i) as for the k-NN-estimator:

$$\hat{f}_X(x_i)\mathrm{Vol}(B_x(x_i, r_i)) = \frac{n_{x,i}+1}{n}, \quad \hat{f}_Y(y_i)\mathrm{Vol}(B_y(y_i, r_i)) = \frac{n_{y,i}+1}{n}.$$

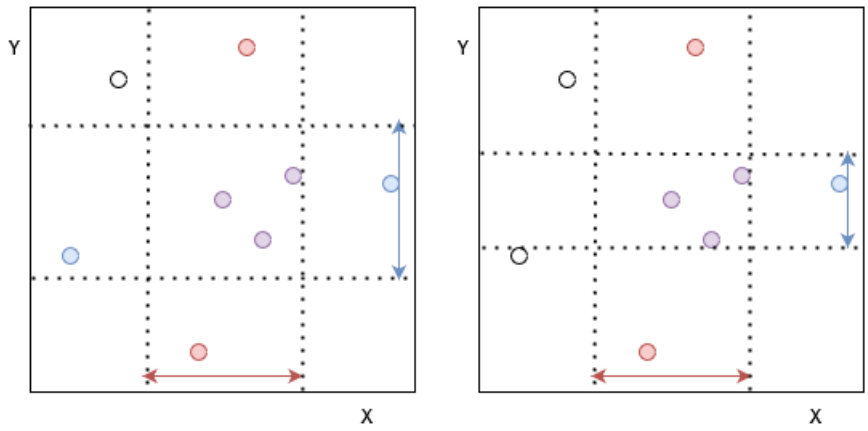


Figure 1: Example of KSG1 (left) and KSG2 (right) algorithms calculating $n_{x,i}$, $n_{y,i}$ and $m_{x,i}$, $m_{y,i}$ respectively, for $k = 2$.

For $B((x_i, y_i), r_i) \subseteq \mathbb{R}^{d_1} \times \mathbb{R}^{d_2}$, $\mathrm{Vol}(B((x_i, y_i), r_i)) = \mathrm{Vol}(B_x(x_i, r_i))\mathrm{Vol}(B_y(y_i, r_i))$ because of the max-norm we defined on $\mathbb{R}^{d_1} \times \mathbb{R}^{d_2}$. Thus

$$\hat{f}(x_i, y_i)\mathrm{Vol}(B_x(x_i, r_i))\mathrm{Vol}(B_y(y_i, r_i)) = \frac{k}{n}.$$

From these probability density estimates, we obtain

$$\hat{H}(X) = \log(n\mathrm{Vol}(B_x(0,1))) + \frac{d_1}{n}\sum_{i=1}^{n}\log(r_i) - \frac{1}{n}\sum_{i=1}^{n}\log(n_{x,i}+1)$$

$$\hat{H}(Y) = \log(n\mathrm{Vol}(B_y(0,1))) + \frac{d_2}{n}\sum_{i=1}^{n}\log(r_i) - \frac{1}{n}\sum_{i=1}^{n}\log(n_{y,i}+1)$$

$$\hat{H}(X,Y) = \log\left(\frac{n}{k}\mathrm{Vol}(B_x(0,1))\mathrm{Vol}(B_y(0,1))\right) + \frac{d_1+d_2}{n}\sum_{i=1}^{n}\log(r_i).$$

Now $\hat{H}(X) + \hat{H}(Y) - \hat{H}(X,Y)$ gives us the following estimate:

$$\hat{I}(X;Y) = -\frac{1}{n}\sum_{i=1}^{n}(\log(n_{x,i}+1) + \log(n_{y,i}+1)) + \log(k) + \log(n).$$

Remark 3.3. In the original article, the estimator is defined using the digamma function. For positive integers, $\Psi(n) = \sum_{i=1}^{n-1}\frac{1}{i} - \gamma \approx (\log(n-1)+\gamma) - \gamma \approx \log(n)$ for large n. The authors found that this formula is unbiased based on their numerical tests.

Conjecture 3.1. *The asymptotic mean of the above $\hat{I}(X;Y)$ for sample size $n \to \infty$ is given by*

$$\lim_{n\to\infty}\mathbb{E}(\hat{H}(X)) = -\Psi(k) + \log k + I(X;Y).$$

We see no proof of this, but the KSG estimator became widely popular in the forms seen in Definition 3.3 and Definition 3.4. This means that the estimate above is biased. By correcting this bias, we obtain the asymptotically unbiased KSG estimate.

Definition 3.3 (Kraskov-Stögbauer-Grassberger mutual information estimator I)**.**

$$I_{KSG1}(X;Y) = -\frac{1}{n}\sum_{i=1}^{n}(\log(n_{x,i}+1)+\log(n_{y,i}+1)) + \Psi(k) + \log(n).$$

For the second KSG algorithm, we denote by $\{(x_{i1}, y_{i1}), (x_{i2}, y_{i2}), \dots (x_{ik}, y_{ik})\}$ the k nearest neighbors of (x_i, y_i) on the joint space and define

$$r_{x,i} = \max\{|x_i - x_{ij}| : j \in \{1,\dots,k\}\} \text{ and } r_{y,i} = \max\{|y_i - y_{ij}| : j \in \{1,\dots,k\}\}$$

so that $r_i = \max\{r_{x,i}, r_{y,i}\}$. We count the number of points $m_{x,i}$ and $m_{y,i}$ in the closed balls $B_x(x_i, r_{x,i}) \subseteq \mathbb{R}^{d_1}$ and $B_y(y_i, r_{y,i}) \subseteq \mathbb{R}^{d_2}$. Following the same line of thought as for previous estimators, we obtain entropy estimates

$$\hat{H}(X) = \log(n\text{Vol}(B_x(0,1))) + \frac{d_1}{n}\sum_{i=1}^{n}\log(r_{x,i}) - \frac{1}{n}\sum_{i=1}^{n}\log(m_{x,i})$$

$$\hat{H}(Y) = \log(n\text{Vol}(B_y(0,1))) + \frac{d_2}{n}\sum_{i=1}^{n}\log(r_{y,i}) - \frac{1}{n}\sum_{i=1}^{n}\log(m_{y,i}).$$

On the joint space, all k nearest neighbors of (x_i, y_i) fall in $B_x(x_i, r_{x,i}) \times B_y(y_i, r_{y,i}) \subset B((x_i, y_i), r_i)$ by the definition of $r_{x,i}$ and $r_{y,i}$. Thus we may improve our heuristic: instead of using $\text{Vol}(B((x_i, y_i), r_i))$ in the equation, we write

$$\hat{f}(x_i, y_i)\text{Vol}(B_x(x_i, r_{x,i}) \times B_y(y_i, r_{y,i})) = \hat{f}(x_i, y_i)\text{Vol}(B_x(x_i, r_{x,i}))\text{Vol}(B_y(y_i, r_{y,i})) = \frac{k}{n}$$

$$\hat{H}(X,Y) = \log(\frac{n}{k}\text{Vol}(B_x(0,1))\text{Vol}(B_y(0,1)) + \frac{1}{n}\sum_{i=1}^{n}(d_1\log(r_{x,i}) + d_2\log(r_{y,i})).$$

Again, $\hat{H}(X) + \hat{H}(Y) - \hat{H}(X,Y)$ gives us an estimate.

$$\hat{I}(X;Y) = -\frac{1}{n}\sum_{i=1}^{n}(\log(m_{x,i}) + \log(m_{y,i})) + \log(k) + \log(n).$$

Conjecture 3.2. *The asymptotic mean of the above $\hat{I}(X;Y)$ for sample size $n \to \infty$ is given by*

$$\lim_{n\to\infty}\mathbb{E}(\hat{H}(X)) = -\Psi(k) + \log k + \frac{1}{k} + I(X;Y).$$

Correcting this bias gives us the second KSG estimate.

Definition 3.4 (Kraskov-Stögbauer-Grassberger mutual information estimator II)**.**

$$I_{KSG2}(X;Y) = -\frac{1}{n}\sum_{i=1}^{n}(\log(m_{x,i}) + \log(m_{y,i})) - \frac{1}{k} + \Psi(k) + \log(n).$$

Now let us assume the following about the distribution $f(x,y)$ of (X,Y) and marginals $f(x)$, $f(y)$: there exist finite constants C_a, C_b, C_c, C'_c, C_d, C'_d and C_e, such that

(a) $f(x,y) \leq C_a$ almost everywhere,

(b) $f(x) \leq C_b$ and $f(y) \leq C_b$,

(c) for all $b > 0$

$$\int f(x,y)\exp(-bf(x,y))dxdy \leq C_c/b,$$

$$\int f(x)\exp(-bf(x))dx \leq C'_c/b,$$

$$\int f(y)\exp(-bf(y))dy \leq C'_c/b,$$

(d) $||\nabla^2 f(x,y)||_{op} \leq C_d$, $||\nabla^2 f(x)||_{op} \leq C'_d$, $||\nabla^2 f(y)||_{op} \leq C'_d$, where $||\cdot||_{op}$ is the operator norm,

(e) $f(x|y) \leq C_e$ and $f(y|x) \leq C_e$.

Theorem 3.3 ([53])**.** *Under assumptions (a), (b), (c), (d), (e), with fixed $k > 1$ and sufficiently large sample size n, the KSG estimator's bias is bounded by*

$$\mathbb{E}(I_{KSG}(X;Y) - I(X;Y)) = O\left(\frac{\log n}{n^{d_x+d_y}}\right).$$

Theorem 3.4 ([9])**.** *Under assumptions (b), (d) and (e) the KSG estimator is consistent.*

3.3 Information noise contrastive estimation

InfoNCE [33] is a noise contrastive representation learning approach (see Section 2.2.3) introduced by van den Oord, Li and Vinyals [33]. They remark that the mutual information-based loss function they defined for noise contrastive estimation (NCE) can also be used for estimating the mutual information between two random variables. We briefly explain the loss function, then focus on the latter application.

The InfoNCE loss over a batch of samples $B = \{x_1, x_2, \dots x_b\}$ given context y_i is

$$L_i = -\mathbb{E}\left(\log \frac{g_\theta(x_i, y_i)}{\sum_{j=1}^{b} g_\theta(x_j, y_i)}\right).$$

Here (x_i, y_i) is the only positive pair, and the neural network g_θ outputs a positive value. We view (x_i, y_i) as random variables following the joint distribution $p(x, y)$, and x_j $(j \neq i)$ as random variables following the noise distribution $p(x_i)p(y_i)$. The expectation is taken over these variables. L_i is a categorical cross-entropy loss for identifying the pair correctly. The prediction of the model – the probability that (x_i, y_i) forms a positive pair – is $\frac{g_\theta(x_i, y_i)}{\sum_{(x,y)\in B} g_\theta(x, y)}$. The authors note that any positive real score can be used in place of $g_\theta(x, y)$.

Remark 3.4. In practice, the loss is calculated over a sample $\{(x_1, y_1), (x_2, y_2), \dots, (x_b, y_b)\}$ as the average

$$L = \frac{1}{b}\sum_{i=1}^{b} L_i.$$

Lemma 3.1. *$g_\theta(x, y)$ approximates $\frac{p(x|y)}{p(x)}$ up to a multiplicative constant.*
Proof.

$$p((x_i, y_i) \text{ is the positive pair}|B, y_i) = \frac{p(x_i|y_i)\Pi_{l\neq i}p(x_l)}{\sum_{j=1}^{b} p(x_j|y_i)\Pi_{l\neq i}p(x_l)} = \frac{\frac{p(x_i|y_i)}{p(x_i)}}{\sum_{j=1}^{b}\frac{p(x_j|y_i)}{p(x_j)}}.$$

As we can see, g_θ minimizes the expectation in L_i if $g_\theta(x, y) = c\frac{p(x|y)}{p(x)}$ for some constant $c > 0$. □

Theorem 3.5. $I(X; Y) \geq \log b - L_i$.

Proof. Assuming g_θ is optimal and minimizes L_i,

$$L_i = -\mathbb{E}\left(\log \frac{\frac{p(x_i|y_i)}{p(x_i)}}{\frac{p(x_i|y_i)}{p(x_i)} + \sum_{j\neq i}\frac{p(x_j|y_i)}{p(x_j)}}\right) = \mathbb{E}\log\left(1 + \frac{p(x_i)}{p(x_i|y_i)}\sum_{j\neq i}\frac{p(x_j|y_i)}{p(x_j)}\right).$$

For all $j \neq i$, (x_j, y_i) is not a positive pair. We can view these x_j as i.i.d. data points coming from the noise distribution $p(x)p(y)$, and use an average to approximate the previous expression.

$$\mathbb{E}_{X,Y}\log\left(1 + \frac{p(x_i)}{p(x_i|y_i)}\sum_{j\neq i}\frac{p(x_j|y_i)}{p(x_j)}\right) \approx \mathbb{E}_{X,Y}\log\left(1 + \frac{p(x_i)}{p(x_i|y_i)}(b-1)\mathbb{E}_{X\times Y}\frac{p(x_j|y_i)}{p(x_j)}\right).$$

For the noise distribution, $p(x_j|y_i) = p(x_j)$ stands, so we may write

$$\mathbb{E}_{X,Y} \log \left(1 + \frac{p(x_i)}{p(x_i|y_i)}(b-1)\mathbb{E}_{X\times Y}\frac{p(x_j|y_i)}{p(x_j)}\right) = \mathbb{E}_{X,Y} \log \left(1 + \frac{p(x_i)}{p(x_i|y_i)}(b-1)\right).$$

We assumed y_i is the context that maximizes the probability of x_i, which means $p(x_i|y_i) \geq p(x_i)$.

$$\mathbb{E}_{X,Y} \log \left(1 + \frac{p(x_i)}{p(x_i|y_i)}(b-1)\right) \geq \mathbb{E}_{X,Y} \log \left(\frac{p(x_i)}{p(x_i|y_i)}b\right) = -\hat{I}(X;Y) + \log b$$

□

Remark 3.5. Note that $L_i \leq -\hat{I}(X;Y) + \log b$ means that even if the loss approaches 0, the estimated mutual information can only be as high as $\log b$. Accurately estimating high mutual information requires very large batch size b.

3.4 Mutual information neural estimator

The mutual information neural estimator (MINE) was introduced by Belghazi et al. [2]. It relies on the alternative definition of mutual information presented in Remark 2.8, and the variational representation of the Kullback-Leibler divergence known as the Donsker-Varadhan formula, introduced with the following theorem.

Theorem 3.6 ([6])**.** *For probability distributions P, Q over Ω*

$$D(P||Q) = \sup_{T:\Omega\to\mathbb{R}} (\mathbb{E}_P(T) - \log(\mathbb{E}_Q(e^T))),$$

where the supremum is taken over all functions T such that the two expectations are finite.

This means that given random variables X and Y with probability density functions $f_X : \mathbb{R}^{d_1} \to \mathbb{R}$ and $f_Y : \mathbb{R}^{d_2} \to \mathbb{R}$ respectively, for any set of functions $\mathcal{F}$ with $\mathcal{F} \subset \{T : \mathbb{R}^{d_1+d_2} \to \mathbb{R} : \mathbb{E}_{X,Y}(T) < \infty, \log(\mathbb{E}_{X\times Y}(e^T)) < \infty\}$,$the following holds : I(X;Y) = D(f_{X,Y}(x,y)||f_X(x)f_Y(y)) \geq \sup_{T\in\mathcal{F}}(\mathbb{E}_{X,Y}(T) - \log(\mathbb{E}_{X\times Y}(e^T)))$, where $f_{X,Y}(x,y)$ is the probability density function of the joint variable (X,Y). Let $\mathcal{F}'$ be the set of functions parameterized by neural networks that map from $\mathbb{R}^{d_1+d_2}$ to $\mathbb{R}$, and for any $g_\theta \in \mathcal{F}'$ neural network let $L(g_\theta) = \mathbb{E}_{X,Y}(g_\theta) - \log(\mathbb{E}_{X\times Y}(e^{g_\theta}))$. Theorems 2.2 and 2.3 guarantee that the functions in $\mathcal{F}'$ can (in theory) arbitrarily approximate the true supremum $I(X;Y)$ with $L(g_\theta)$.

Given a dataset D of samples from (X, Y), MINE approximates $I(X;Y)$ by optimizing the parameters θ of a fixed neural network architecture g. The network is trained to maximize $L(g_\theta)$, i.e. minimize $-L(g_\theta)$ as a loss function. Evidently, this is achieved through minimizing the empirical expectations $-\hat{L}(g_\theta)$ over the set of samples. The samples from (X, Y) are given, and the samples from $X \times Y$ are obtained by randomly shuffling Y to obtain a dataset $D' = \{(x_i, y_{\pi(i)}) | 1 \leq i \leq n\}$. The expectations in $\hat{L}(g_\theta)$ are calculated using the law of large numbers, by averaging over a number of samples before updating the network parameters.

Theorem 3.7 ([2])**.** *MINE is strongly consistent.*

Now let us assume there exists constants C_a, C_b and C_c such that

(a) $|g_\theta| < C_a$,

(b) g_θ is C_b-Lipschitz w.r.t. θ,

(c) $\Theta \subset \mathbb{R}^{C_p}$ is bounded, $||\theta|| < C_c$ for some $C_c \in \mathbb{R}$.

Theorem 3.8 ([2])**.** *Under assumptions (a), (b) and (c), for all $\epsilon > 0$ and $\delta > 0$*

$$P(|L(g_\theta) - \hat{L}(g_\theta)| < \epsilon) > 1 - \delta,$$

whenever the number of samples n satisfies

$$n > \frac{2C_a^2 C_p \log(16 C_c C_b \sqrt{C_p}/\epsilon) + 2C_a C_p + \log(2/\delta)}{\epsilon^2}.$$

InfoNCE is closely related to MINE. Let $\exp T(x, y) = g_\theta(x, y)$.

$$\begin{aligned} L_i = \mathbb{E}\left(\log \frac{\exp T(x_i, y_i)}{\sum_{j=1}^{b} \exp T(x_j, y_i)} \right) &= \mathbb{E}(T(x_i, y_i)) - \mathbb{E}\left(\log \sum_{j=1}^{b} \exp T(x_j, y_i) \right) \\ &\leq \mathbb{E}(T(x_i, y_i)) - \mathbb{E}\left(\log \sum_{j \neq i} \exp T(x_j, y_i) \right) \\ &= \mathbb{E}(T(x_i, y_i)) - \mathbb{E}\left(\log \left(\frac{1}{b-1} \sum_{j \neq i} \exp T(x_j, y_i) \right) + \log(b-1) \right). \end{aligned}$$

This is equivalent to MINE up to a constant, which means InfoNCE maximizes the same lower bound as MINE (Theorem 3.6). Theorems 3.7 and 3.8 also carry over. Regardless, the authors claim that InfoNCE performed better than MINE on trivial tasks and gave identical performance on non-trivial tasks.

4 Benchmarking on synthetic data

In this section we focus on comparing the mutual information estimators on synthetic data. In our experiments, we implemented the formulas for the k-nearest neighbors and Kraskov-Stögbauer-Grassberger estimators as written in this thesis, and used the implementations of [4] for the remaining two estimators, available at `https://github.com/Linear95/CLUB`. For most of the experiments, we used a random seed of 2023. Our code is available at `http://github.com/sisakls/mi`.

We tested on synthetic data derived from uniform or normal distributions. Normal distributions are popular for benchmarking [13, 36], but recent work [5] introduces more challenging benchmarks for testing, they show that testing on multivariate normal distributions gives a biased and overly optimistic view of estimator performance, and KSG does not perform well on problems involving high-dimensions or sparse interactions.

4.1 Analytical computation

Evaluating the accuracy of mutual information estimators on real data is usually not possible, as the underlying distributions are not known. For this reason, we generate synthetic data from known distributions, where accurately calculating mutual information is tractable.

4.1.1 Linear synthetic data

For linear synthetic data, $X \sim \text{Unif}(0, 1)$ and $Y = X+Z$, where $Z \sim \text{Unif}(-\alpha/2, \alpha/2)$ for $\alpha < 1$. When generating an d dimensional linear dataset, we draw d i.i.d. samples from distributions X and Z, one for each dimension. Essentially, each d dimensional data point is an d-sized sample of the 1 dimensional linear data, and (aside from dimensionality) the data have a single parameter, α. The mutual information for $d = 1$ dimension is

$$I(X;Y) = H(Y) - H(Y|X) = H(Y) - H(Z) = H(Y) - \log \alpha.$$

It remains to calculate $H(Y)$. We can find the pdf of Y using convolution:

$$f_Y(y) = \int_{-\infty}^{\infty} f_X(x) f_Z(y - x) dx.$$

As $f_Z(y-x) = \frac{1}{\alpha}$ iff $-\alpha/2 \leq y-x \leq \alpha/2$ iff $y-\alpha/2 \leq x \leq y+\alpha/2$, and 0 elsewhere, we may write

$$f_Y(y) = \int_{y-\alpha/2}^{y+\alpha/2} f_X(x)\frac{1}{\alpha}dx = \frac{1}{\alpha}\lambda^*([0,1] \cap [y-\alpha/2, y+\alpha/2]),$$

where λ^* is the Lebesgue measure, or simply the length of these intervals. Which means

$$f_Y(y) = \begin{cases} 0 & \text{if } y \in (-\infty, -\alpha/2) \cup (1+\alpha/2, \infty) \\ \frac{1}{\alpha}(y+\alpha/2) & \text{if } y \in [-\alpha/2, \alpha/2] \\ 1 & \text{if } y \in (\alpha/2, 1-\alpha/2) \\ \frac{1}{\alpha}(-y+1+\alpha/2) & \text{if } y \in [1-\alpha/2, 1+\alpha/2], \end{cases}$$

thus

$$H(Y) = -\int_{-\infty}^{\infty} f_Y(y)\log f_Y(y)dy = \\ -\int_{-\alpha/2}^{\alpha/2}\left(\frac{y}{\alpha}+\frac{1}{2}\right)\log\left(\frac{y}{\alpha}+\frac{1}{2}\right) - \int_{1-\alpha/2}^{1+\alpha/2}\left(\frac{1-y}{\alpha}+\frac{1}{2}\right)\log\left(\frac{y}{\alpha}+\frac{1}{2}\right) = \\ \alpha/4+\alpha/4 = \alpha/2. \quad (1)$$

Substituting (1) in $I(X;Y)$ we obtain

$$I(X;Y) = H(Y) - H(Z) = \alpha/2 - \log\alpha$$

for $d = 1$ dimension. The mutual information for $d = 2$ dimensions is

$$I(X;Y) = I(X_1, X_2; Y_1, Y_2) = I(X_1; Y_1) + I(X_2; Y_2) = 2(\alpha/2 - \log\alpha)$$

because of the independence of (X_1, Y_1) and (X_2, Y_2). For general d, by induction

$$I(X;Y) = d(\alpha/2 - \log\alpha).$$

4.1.2 Gaussian synthetic data

For d-dimensional (multivariate) Gaussian synthetic data, a random $2d \times 2d$ covariance matrix was generated using Cholesky decomposition. That is, for every $M \in \mathbb{R}^{n\times n}$ symmetric positive definite matrix, there exists $L \in \mathbb{R}^{n\times n}$ lower-triangular matrix such that $M = LL^T$. Thus we generate L by defining all diagonal

entries as 1 and generating all entries under the diagonal as i.i.d. samples from Unif$(-1,1)$. The rest of the entries are 0. The covariance matrix is defined as $\Sigma = LL^T \in \mathbb{R}^{2d\times 2d}$. $(X;Y) \sim N(\mathbf{0},\Sigma)$, where X is the first d entries and Y is the last d entries. Note that for a normal distribution with $\mathbf{0}$ expectation and covariance matrix $\hat{\Sigma}$,

$$H(Z) = -\mathbb{E}(\log N(0,\hat{\Sigma})) = -\mathbb{E}(\log(2\pi)^{-d/2}\det(\hat{\Sigma})^{-1/2}\exp(-\frac{1}{2}Z^T\hat{\Sigma}^{-1}Z)) = \\ = \frac{d}{2}\log(2\pi) + \frac{1}{2}\log\det(\hat{\Sigma}) + \frac{1}{2}\mathbb{E}(Z^T\hat{\Sigma}^{-1}Z), \quad (2)$$

where $\mathbb{E}(Z^T\hat{\Sigma}^{-1}Z) = \mathbb{E}(\mathrm{tr}(Z^T\hat{\Sigma}^{-1}Z)) = \mathrm{tr}(\hat{\Sigma}^{-1}\mathbb{E}(ZZ^T)) = \mathrm{tr}(\hat{\Sigma}^{-1}\hat{\Sigma}) = \mathrm{tr}(I_d) = d$. Substituting the latter in (2) yields

$$H(X) = \frac{d}{2}\log(2\pi) + \frac{1}{2}\log\det(\Sigma) + \frac{1}{2}d = \frac{d}{2}(1+\log(2\pi)) + \frac{1}{2}\log\det(\Sigma).$$

We can conclude that the mutual information of this multivariate Gaussian distribution is

$$I(X;Y) = H(X) + H(Y) - H(X;Y) = \frac{1}{2}\log\left(\frac{\det(\Sigma_X)\det(\Sigma_Y)}{\det(\Sigma)}\right),$$

where Σ_X is the $d\times d$ (upper left) submatrix belonging to X and Σ_Y is the $d\times d$ (lower right) submatrix belonging to Y.

For some experiments, we want the mutual information of the distribution to scale linearly with dimension d, as it does for linear synthetic data. We achieve this by ensuring that the distribution generating the i-th coordinate of X is independent from all other distributions generating coordinates of (X,Y), save for the i-th coordinate of Y. Doing so will render the distribution a lot less complex, but we will have some basis to draw conclusions from comparing the estimates between different dimensions. In these cases, the matrix L used for generating the covariance matrix of the distribution was fixed as

$$L = \begin{pmatrix} I & \mathbf{0} \\ \beta I & I \end{pmatrix}, \text{ thus } LL^T = \begin{pmatrix} I & \beta I \\ \beta I & (1+\beta^2)I \end{pmatrix},$$

where I is the d-dimensional identity matrix and $\beta \in \mathbb{R}$ scalar.
When we fix L this way, one easily checks that based on previous statements

$$I(X;Y) = \frac{d}{2}\log\left(\frac{1\cdot(1+\beta^2)}{1}\right).$$

4.2 Comparing analytic and estimated mutual information

4.2.1 Nearest neighbors-based estimates

The accuracy of each estimator was tested on both linear and Gaussian synthetic data. The parameters for nearest neighbors-based estimators (kNN, KSG1, KSG2) were k and the norm used on the (marginal) space.

There is an additional parameter $\epsilon > 0$ for such estimators, which is added to the distance of a point x_i and its k-th nearest neighbor. This is because in practice the k-th nearest neighbor of x_i may not be contained in the closed ball $B(x_i, r_i)$ due to floating point error, but is surely contained in $B(x_i, r_i')$, where $r_i' = r_i + \epsilon$. We experimentally confirmed that a small ϵ does not significantly alter the estimates (ϵ between orders of magnitude 10^{-12} and 10^{-8}), then used $\epsilon = 10^{-10}$ for all other experiments.

In the first experimental setup, the performance of estimates was evaluated on both datasets for varying parameters and sample sizes. The estimators approximated $I(X;Y)$ for a 20-dimensional (X,Y) joint distribution ($d_X = d_Y = 10$). For the linear dataset, $\alpha = 0.6$. The nearest neighbors estimates were evaluated on 100, 300, 1000, 3000 and 10000 samples. These were not independently generated datasets, rather a single dataset of 10000 samples was created, and we estimated $I(X;Y)$ on the first 100 samples, then the first 300 samples, etc. until at last all the samples were used. This means that the x axis of Figures 2-5 is roughly logarithmic.

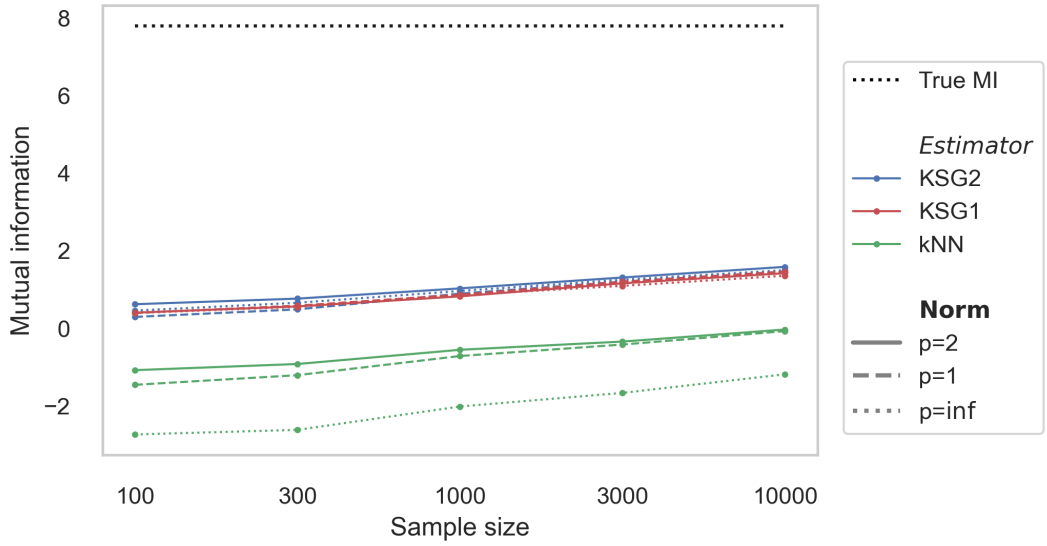


Figure 2: Accuracy of first nearest neighbors-based estimators on increasingly bigger samples of 20-dimensional Gaussian joint distribution (X,Y). Different lines of the same color mark different norms.

On Figures 2 and 4 the accuracy of the estimators with various norms are plotted. On the Gaussian dataset, the kNN estimator performed very poorly regardless of the norm, but the L_1 and L_2 norm seemed to be the least inaccurate. The KSG1 and KSG2 estimators were very insensitive to the norm used. On the linear dataset, L_2 norm was the most accurate for the kNN and KSG1 estimators. The KSG2 estimator was still rather insensitive to the norm, but the L_1 norm performed a little better than the L_2 and L_∞ norms. Based on the above observations we fixed the L_2 norm for kNN and KSG1 and L_1 norm for KSG2 for the remaining experiments.

Next, the estimators were tried with various k parameters. We found that the greater k is, the worse each approximation gets. The results can be seen on Figures 3 and 5 for $k \in \{1, 2, 3\}$. We tried k up to $k = 5$ and even repeated the experiment fixing different norms, each time finding that the performance of all three estimators gets strictly worse whenever k increases. This coincides with the findings of Kraskov, Stögbauer and Grassberger. (See figure 16 in [24].)

Over all, the kNN estimator was clearly the least accurate of nearest neighbors-based estimators, but all three estimators were very inaccurate over as little as 10 dimensions.

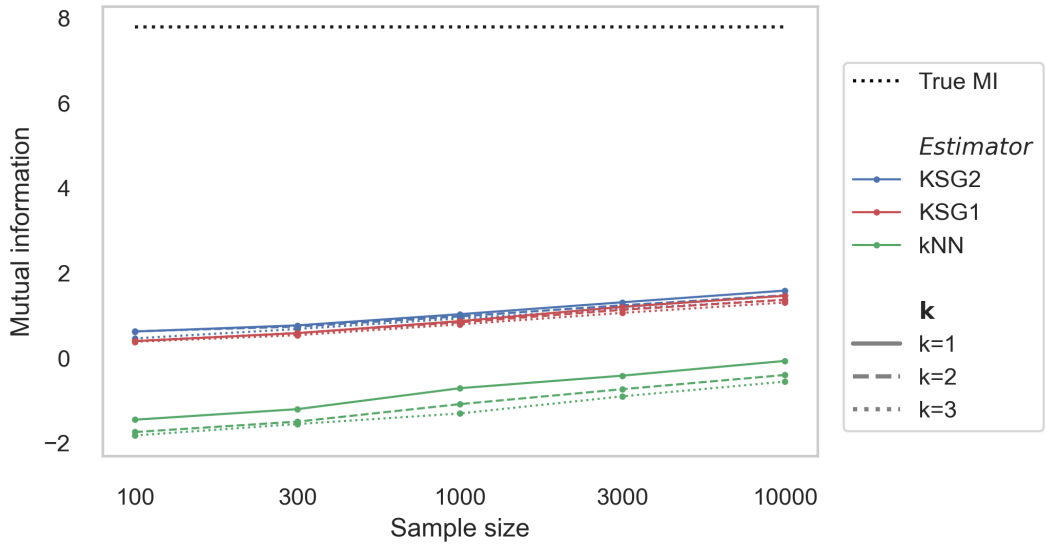


Figure 3: Accuracy of k nearest neighbors-based estimators on increasingly bigger samples of 20-dimensional Gaussian joint distribution (X, Y). Different lines of the same color mark different k parameters. kNN and KSG1 estimates used the euclidean norm, KSG2 used the L_1 norm.

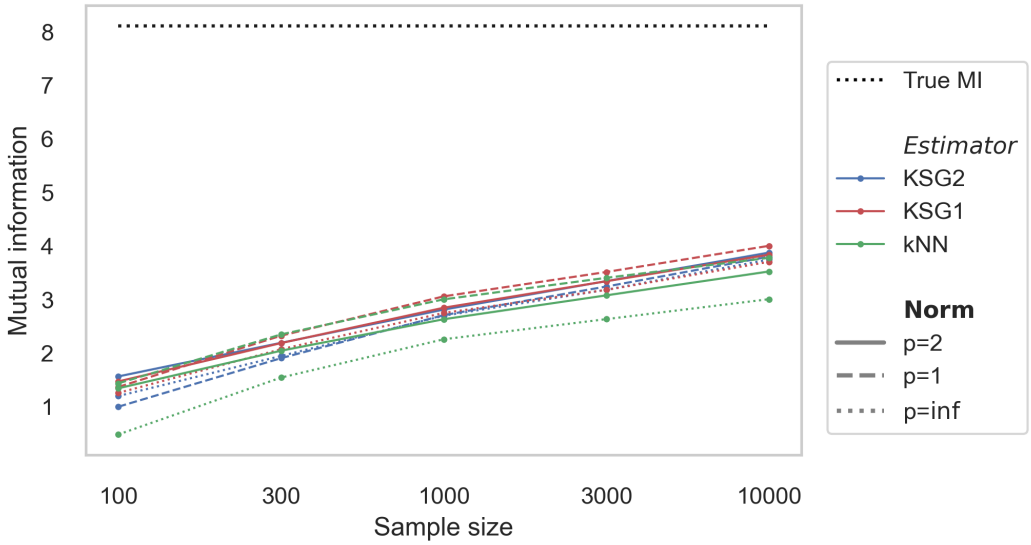


Figure 4: Accuracy of first nearest neighbor-based estimators on increasingly bigger samples of 20-dimensional linear joint distribution (X, Y). Different lines of the same color mark different norms.

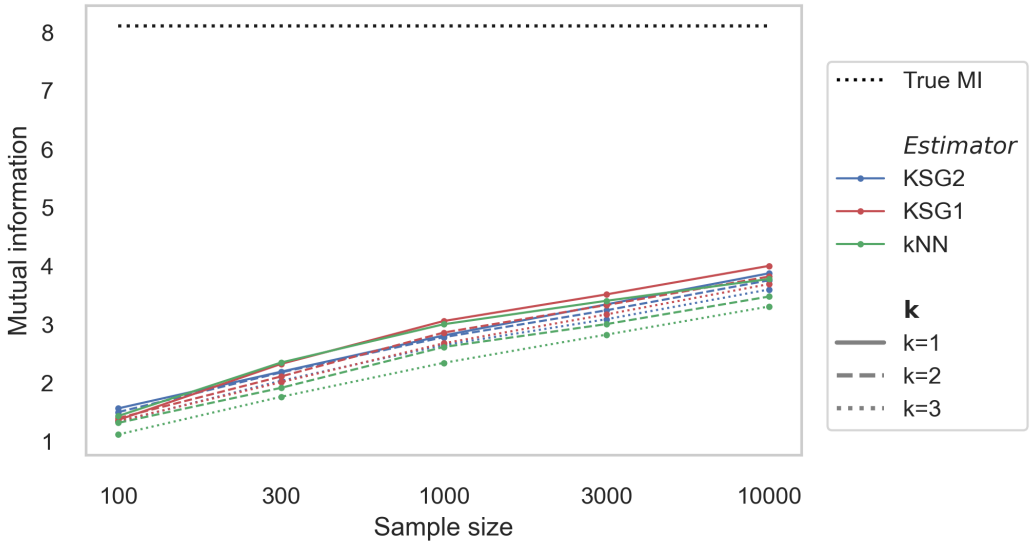


Figure 5: Accuracy of k nearest neighbors-based estimators on increasingly bigger samples of 20-dimensional linear joint distribution (X, Y). Different lines of the same color mark different k parameters. kNN and KSG1 estimates used the euclidean norm, KSG2 used the L_1 norm.

4.2.2 Neural estimates

Again, we tested the accuracy of MINE and InfoNCE on both the Gaussian and linear datasets. The parameters of the datasets were identical to the ones used in the previous subsection. The parameters of these neural estimators are the same as the ones used in any neural network training setup: the choice of neural architecture, number of training steps, the gradient descent variant and its hyperparameters.

The architecture was a simple, shallow neural network that first mapped to a layer of p neurons and then to the single output neuron. $p \in \mathbb{N}$ was a parameter left to be experimented with. This is a simple architecture, but the authors evaluated InfoNCE similarly, proving its effectiveness. The variant of gradient descent used was the Adam optimization algorithm (see Definition 2.18). While Adam operates with many parameters, it is a very robust optimization algorithm that was proven not to require much fine-tuning. Aside from learning rate λ, we fixed all of Adam's parameters at their implemented default values: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.

The models were trained with a batch size of 3334. We intended to train using a batch size of 10000 to enable the models the best possible performance given the setup (as the largest training set in our experiments consisted of 10000 samples), but this was not possible due to hardware limitations.

The parameter p was determined in an experimental setup similar to that used in the previous section: Gaussian and linear datasets of 100, 300, 1000, 3000 and 10000 samples, not independently generated, but each of them subsets of the largest dataset. Both estimators were trained on each dataset with a learning rate of $\lambda = 0.005$ and each dataset was processed 300 times before the model was evaluated. The final mutual information score given by the models was the average of the last 20 estimations of the training. This was needed because the values estimated by MINE showed a large variance, even after convergence. The results for $p \in \{10, 20, 30\}$ can be seen on Figures 6 and 7.

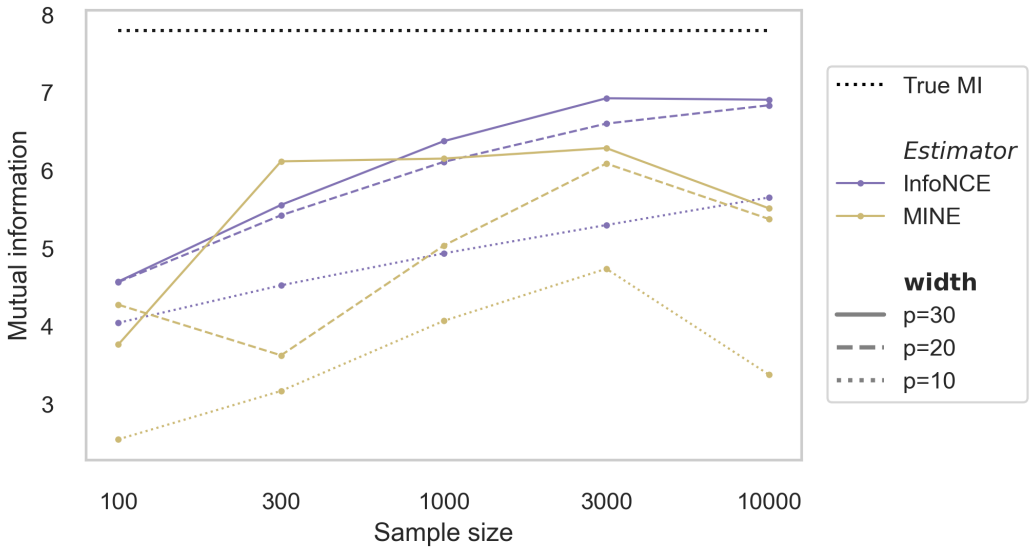


Figure 6: Accuracy of neural estimators on increasingly bigger samples of 20-dimensional Gaussian joint distribution (X, Y). Different lines of the same color mark different widths p of the hidden layer.

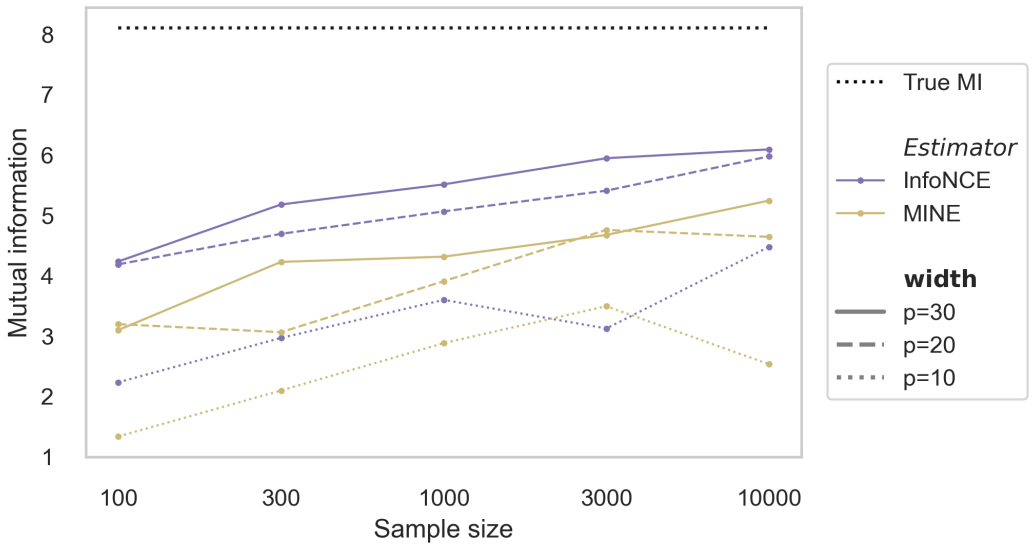


Figure 7: Accuracy of neural estimators on increasingly bigger samples of 20-dimensional linear joint distribution (X, Y). Different lines of the same color mark different widths p of the hidden layer.

As one would expect, higher model complexity led to more accurate lower bound estimates. The models were tried with p values up to 50, but increasing the model complexity to 40 and 50 showed diminishing returns in the accuracy of estimation. As unnecessarily complex neural architectures cause overfitting on training data (i.e. allowing models to learn features that arise from noise in the data), a width of $p = 30$ was fixed on both models for future experiments.
It remained to determine an appropriate learning rate and the number of training epochs. These parameters are much less sensitive than model complexity, and they do not impact model performance as long as we don't end the training before convergence. These training parameters were tested after p to avoid misinterpreting overfitting with an increase in accuracy. After some experimentation, we fixed $\lambda = 0.02$ for InfoNCE, with a training duration of 100 epochs and $\lambda = 0.004$ for MINE, with a training duration of 800 epochs.

4.2.3 Comparison of neural and nearest neighbors estimators

In the next experimental setup, we tested the same estimators on datasets of fixed sample size (10000 samples) and increasing dimension. As mentioned in Section 4.1.2, we used a fixed LL^T covariance matrix for the Gaussian distribution, with $\beta = 2$. For the linear dataset, $\alpha = 0.6$, this ensures that mutual information is roughly the same in both datasets.

Similarly to previous setups, we did not generate new samples for each dimension,

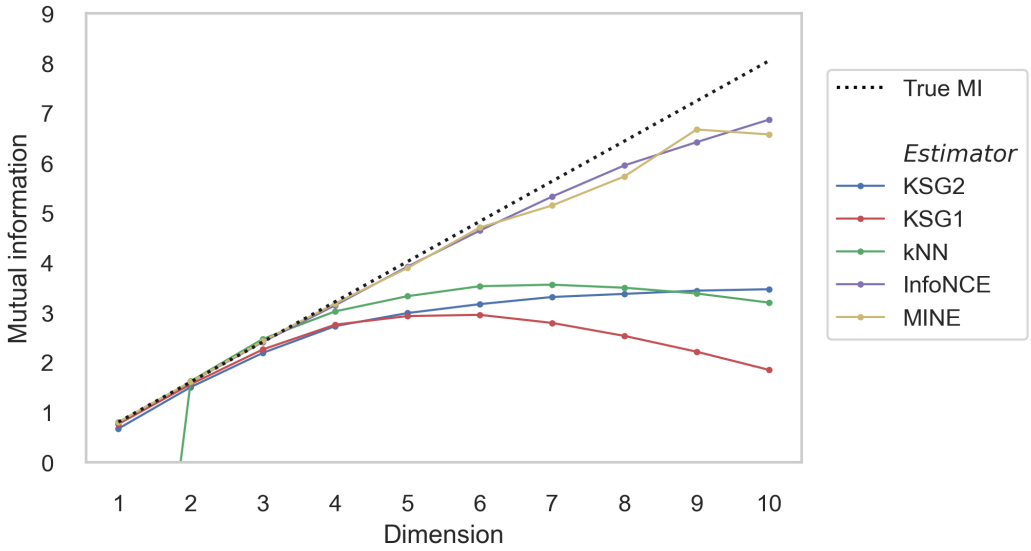


Figure 8: Accuracy of all estimators on Gaussian synthetic data of increasing (marginal) dimension. The y axis is cut off at 0 for readability, despite kNN predicting a value far below 0 for 1-dimensional marginals.

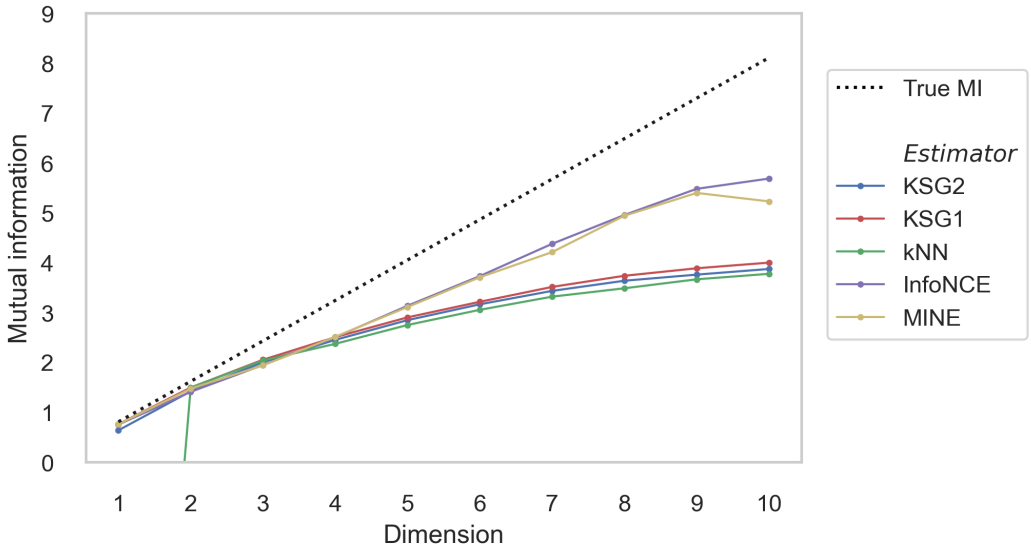


Figure 9: Accuracy of all estimators on linear synthetic data of increasing (marginal) dimension. The y axis is cut off at 0 for readability, despite kNN predicting a value far below 0 for 1-dimensional marginals.

rather we measured the mutual information between the first dimensions, first 2 dimensions, first 3 dimensions ect. of X and Y on a fixed 10-dimensional dataset.

The results can be seen on Figures 8 and 9. Unsurprisingly, the accuracy of all estimators deteriorated as the dimensions increased. The nearest neighbors-based estimators could not keep up with neural estimators. On Gaussian data, the kNN and KSG1 estimated a lower value for 8, 9 and 10 dimensions than for 6 or 7 di-

mensions, despite the true MI increasing linearly with dimensions. Interestingly, the kNN estimator was extremely inaccurate on 1-dimensional marginals. The neural estimators performed better on high-dimensional data than any of the other estimates, and at high dimensions, achieved noticeably better performance on the Gaussian dataset than on the linear.

In the next experimental setup, we repeated the experiment with a different 10-dimensional dataset. We generated a 10000 sample dataset with marginal dimension 2, then padded it to 10 dimensions with i.i.d. samples from a standard normal distribution. These 8 noise dimensions do not increase the MI of the 2-dimensional data. Again, we estimated the true MI with all methods on 2, 3, ... 10 dimensional marginals. $\beta = 4$ for the Gaussian dataset and $\alpha = 0.3$ for the linear dataset.

The results can be seen on Figures 10 and 11. Again, the neural estimators outperformed the nearest neighbors-based estimates, and achieved greater accuracy on the Gaussian dataset. The kNN estimator was the least robust against the noise added in this experiment, the KSG1 and KSG2 estimates noticeably deteriorated as noise was added. The neural estimators consistently reached the same estimate despite the increase in dimensions.

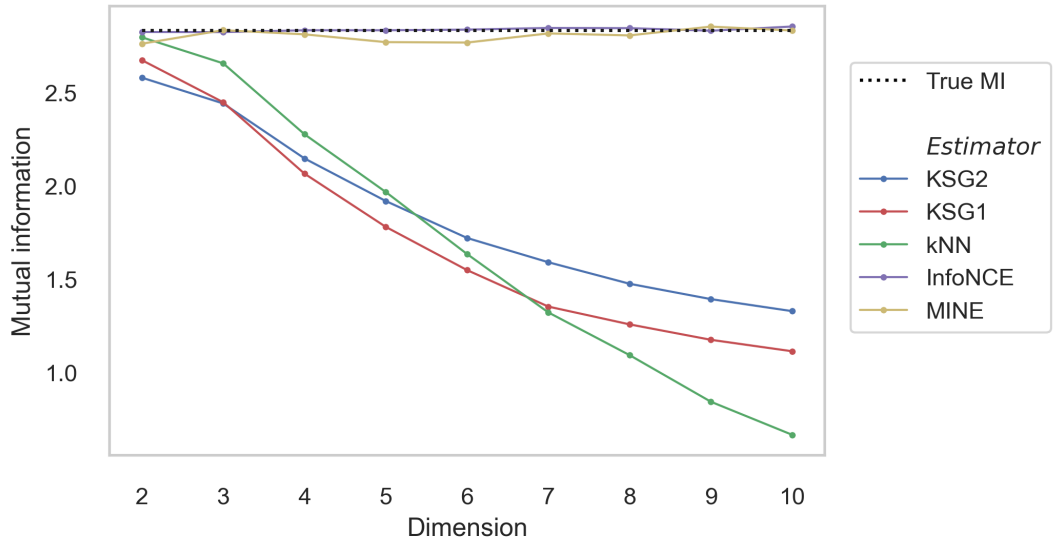


Figure 10: Accuracy of all estimators on Gaussian synthetic data with an increasing number of noise dimensions.

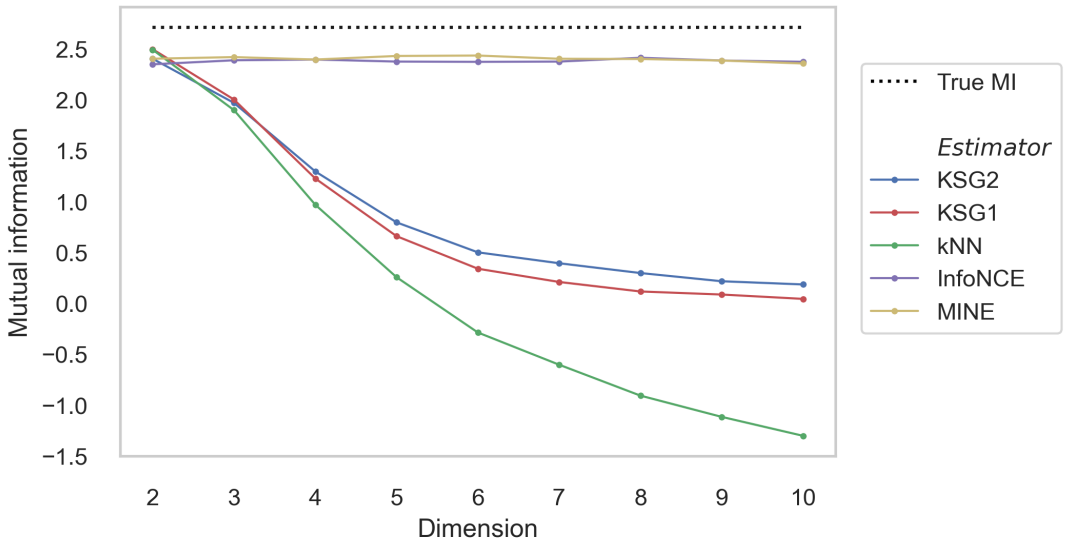


Figure 11: Accuracy of all estimators on linear synthetic data with an increasing number of noise dimensions.

5 Dataset inference with mutual information score

Some commercial service providers grant users paid query access to their model through public APIs, making these encoders vulnerable to "model stealing" attacks. The attacker can query inputs on the hosted neural network and use the observed outputs to train a network that achieves similar performance. Dataset inference [29] offers the possibility to resolve ownership by verifying whether a network is a stolen derivative of another one. The high costs associated with training large models, such as CLIP [40] or large language models, make them valuable targets for theft. Unfortunately, dataset inference normally relies on decision boundaries to verify ownership, which do not exist for encoders trained on a large volume of unlabeled data. In this section, we aim to utilize mutual information estimation for dataset inference following [7]. Here we don't discuss the main result of this work, only focus on section 4.1, which proposes a mutual information-based dataset inference method for self-supervised models.

In the attack setup outlined by the authors, the attacker can't access the weights of the *victim encoder* g_v or the private dataset D_v used for training g_v, but he can construct a model g_s with a similar architecture. The victim model is hosted as a black box for the attacker. The attacker steals g_v by querying his dataset D_s on g_v, obtaining a dataset $\{x, g_v(x)|x \in D_s\}$ which he can use to train the *stolen model* g_s. The attacker trains g_s in a contrastive manner. At every training step, a positive pair for input x is the pair $g_v(x), g_s(x)$. As $g_v(x)$ is independent of g_s, such definition of positive pairs will train g_s to copy g_v's outputs.

The defense method relies on a trusted third party to query part of the private training data $D \subseteq D_v$ on g_v, g_s and a randomly initialized model g_r. Assuming D_v contains i.i.d. samples from some random variable X, this yields samples $g_v(D)$, $g_s(D)$ and $g_r(D)$ from random variables (representations) $g_v(X)$, $g_s(X)$ and $g_r(X)$ respectively. It is assumed that if the encoder g_s is stolen, $I(g_v(X); g_s(X)) \gg I(g_v(X); g_r(X))$. This mutual information can't be calculated directly, because for datasets of practical interest, the distribution of X is unknown. Thus the samples $g_v(D)$, $g_s(D)$, $g_r(D)$ are used to estimate $I(g_v(X); g_s(X))$ and $I(g_v(X); g_r(X))$. Furthermore, the authors propose a novel *mutual information score*, which is mutual information normalized to be a score between 0 and 1.

Definition 5.1. For the mutual information $I(g_v(X); g_s(X))$, the mutual information score is

$$S(g_s, g_v) = \frac{I(g_v(X); g_s(X)) - I_{min}}{I_{max} - I_{min}},$$

where $I_{max} = I(g_v(X); g_v(X))$ and $I_{min} = I(g_v(X); g_r(X))$.

We rely on and extend the publicly available codebase accompanying the publication, our modifications are also available at this link.

5.1 Victim encoder training and evaluation

In the cited and followed article, victim encoders with ResNet34 architecture [17] were trained on images of the CIFAR10 [25] dataset for 200 epochs in contrastive manner using the InfoNCE loss receiving temperature-scaled logits – we refer the reader to Subsection 3.3 for detail on it, as particularly its temperature parameter plays a crucial role in the successful optimization process. According to the authors, the temperature changed between 0.1, 0.15, 0.2 and 0.25, but unfortunately it was not further specified for training victim encoders. The ResNet34 architecture was used with stride 1 and a 3×3 kernel size in its first convolution layer instead of the default 7×7, as usually for CIFAR10 image size, and did not use max-pooling layer. During training, the ResNet34's 512-dimensional output was first mapped to 512 neurons with a linear layer using ReLU activation function, then a 128-dimensional output using a linear layer with no activation function. The *head* – these two linear layers – was discarded after training (a common method in self-supervised learning). The Adam optimization algorithm was used to update the weights of models. Adam's initial learning rate was kept constant in all cases and changed during training via the cosine annealing scheduler, but the weight decay or the exact initial learning rate was not specified in the paper. The batch size was specified as

Setting	Training loss after 200 epochs	Linear eval. training acc.	Linear eval. test acc.
reported	-	-	87.4
temperature=0.1	1.08	83.7	81.6
temperature=0.15	1.94	85.9	84.3
temperature=0.2	2.79	86.7	84.7
temperature=0.25	3.37	86.4	84.6

Table 1: Our linear evaluation results of victim encoders on CIFAR10, trained with different temperature parameters. The first row shows the reported result from Table 7 of [7].

256 or 512. We tested both batch sizes and found that they perform similarly, with a batch size of 256 yielding slightly better results.

When attempting to reproduce the model trainings, we heavily relied on the reported linear evaluation results, see Table 8 of [7]. During linear evaluation, the encoder's parameters were fixed and only a linear layer was trained to perform the default classification task of the CIFAR10 dataset. Unfortunately, we could only reproduce the linear evaluation accuracy of victim encoders with the available code (`github.com/cleverhans-lab/DatasetInferenceForSelfSupervisedModels`), we got similar results only after some experimentation.

In the provided code, the cosine annealing learning rate scheduler is called with a maximum number of iterations set as the length of the data loader (i.e. $\left\lfloor \frac{\#\text{samples}}{\text{batch size}} \right\rfloor$), while the implementation shows that they step the scheduler only once per epoch. While this was not major error, editing the code to call the scheduler with maximum number of iterations set as 200 (the number of epochs) slightly improved performance.

It remained to correctly set the learning rate and weight decay used by the optimization algorithm, and set the appropriate temperature for scaling the InfoNCE logits. We found that the results were most sensitive to the temperature parameter, and left the learning rate and weight decay at their default values provided in the code, $3 \cdot 10^{-4}$ and 10^{-4} respectively.

5.2 Stealing the victim encoder

As we mentioned earlier, we failed to recreate the stolen model as described in the paper. We focused on reproducing the results of Table 8, thus the stolen encoder was trained by querying 10000 samples from the SVHN dataset on our best performing victim encoder. The authors remark that the SVHN dataset's test and train data

distribution is not the same, therefore they reshuffled the whole dataset and used it after an $80 - 20\%$ train-test split. The authors test stealing for a variety of sample numbers from 1000 to 50000, we chose 10000 because it was close to the default value provided in the original code. For the stolen encoder, a ResNet34 architecture has been used (with the same deviations from default in kernel size, stride and layers as the victim), and it was trained contrastively for 100 epochs with InfoNCE loss using a temperature of 0.1. During contrastive training, the the positive pairs consisted of the representation returned by the victim encoder, and the representation of the same data point returned by the stolen encoder. Naturally, the victim encoder's parameters were fixed. The weights of the stolen encoder were optimized with the Adam optimization algorithm. For the InfoNCE loss, the authors hard-coded the learning rate as $3 \cdot 10^{-4}$, the weight decay as 10^{-4} and the batch size as 256, but these parameters were not exactly specified in the paper. The learning rate was adjusted with the cosine annealing scheduler.

Again, we had to modify the code to invoke the cosine annealing scheduler with the appropriate maximum number of iterations. The SVHN dataset was initialized using PyTorch's built-in function, which verifies the integrity of the dataset and prevents the loading of modified or remixed data. To address this, we created a custom function identical to PyTorch's implementation but without the integrity check, enabling us to use a remixed SVHN dataset as reported in the paper. We also observed that the samples are always queried in the same order across all epochs. Initially suspected to harm the SGD-based optimization, this behavior was later understood to serve the purpose of avoiding overlapping data when both stealing and testing the model on the test dataset only. The stealing experiment targeted our best-performing victim model, the one trained with a temperature of 0.2.

Model / Setting	Training loss	CIFAR10 training acc.	CIFAR10 test acc.	SVHN training acc.	SVHN test acc.
our victim	-	86.7	84.7	56.5	52.1
reported stolen	-	-	55.6	-	58.9
no heads	6.08	41.3	39.2	38.2	36.4
victim head	8.55	36.0	35.2	26.8	24.8
stolen head	6.07	42.8	41.7	37.7	36.4
two heads	6.15	41.1	39.5	30.5	27.4
old scheduler	6.08	40.8	39.7	37.4	35.1
$lr = 1.5 \cdot 10^{-3}$	6.07	38.8	37.8	30.7	28.6

Table 2: Our linear evaluation results on stolen encoders. *victim* is the performance of the targeted victim model, all others are stolen encoders: *old scheduler* refers to cosine annealing scheduler called with the original parameters, *lr* is short for learning rate. The second row shows the reported result from Table 8 of [7].

The final parameter to be determined was wether we should add a head to the ResNet34 architecture, similarly to the victim encoder's training. The code offered separate parameters to determine the use of a linear layer for the stolen and victim models. When only one of the models has a head, the final linear layer has to map to an 512-dimensional output to ensure matching output dimensions. We tried all four possible combinations, despite one of them (head on victim only) being seemingly unsupported. Despite our efforts, we were unable to train a stolen encoder that reproduced the linear evaluation accuracies reported in the paper, a summary of the results is provided in Table 2. We also tried training on the original SVHN dataset, training with the original cosine annealing scheduler parameters and with higher learning rate, but the stolen encoder never reached 50% accuracy on CIFAR10 during linear evaluation.

5.3 Estimating encoder similarity

We were unable to reproduce the stolen model but proceeded to analyze the results of mutual information estimation in this setting. The authors implemented the k-NN estimator to measure mutual information between the representations generated by the models. Following the Definition 5.1, an untrained model with randomly initialized weights is used to normalize the score, but here we mainly report the raw mutual information estimation values instead. We estimated the mutual information between the victim model and a copy of the victim model, randomly initialized models, an independent model, and one of our other victim models, trained with 0.15 temperature on the CIFAR10 dataset (see Section 5.1 for the latter). We trained the independent model on the STL10 dataset, with temperature 0.2 and all other parameters identical to the victim. We also include the estimate between the victim and the stolen encoder trained with no heads. As we could not produce a stolen model that matched the accuracy reported in the paper, this is more of a curiosity than a conclusive result.

The provided script used to calculate mutual information estimation utilizing the kNN-estimation method could only query samples from the ImageNet dataset (it was hard-coded), but after minimal editing, we were able to query samples of the CIFAR10 dataset. We did not shuffle CIFAR10 to ensure reproducible, consistent results. This means that the values in Table 3 are obtained from the first 2000, 4000 and 8000 datapoints of CIFAR10 respectively. Notably, the available implementation contained a syntax error.

The results of these experiments can be seen in Table 3. The estimated mutual

information is almost always negative, but the results indicate that there is a relation between the data used to train a model and the estimate. The random model showed inconsistent results varying between -285.64 and -377.3, thus we consider random models to be an unfitting baseline for calculating I_{min} as seen in Definition 5.1. On the other hand, the independently trained encoder consistently produced similar results.

Model	2000 samples	4000 samples	8000 samples
victim	4.8	5.5	6.2
stl10	-73.2	-76.3	-80.7
cifar10	-21.7	-22.6	-25.3
random	-337.1	-359.1	-318.8
stolen	-1450.6	-1394.8	-1329.4

Table 3: Mutual information estimated between the victim encoder and various other encoders. The *victim* is a copy of our best encoder, *cifar10* is the victim encoder trained with 0.15 temperature and *stl10* is the independently trained model. Results from the random encoder are the average of three trials.

6 Conclusions

Mutual information has promising theoretical properties, such as a natural connection to entropy and Kullback-Leibler divergence (Remarks 2.5 and 2.8) or the data processing inequality (Theorem 2.1). In Section 2.3 we reviewed many of the possible applications of mutual information in the field of deep learning, along with their limitations. As we expected, our experiments on synthetic data confirmed that the theoretical limitations ([34], Theorem 1 and [31], Theorem 1.1) carry over to practice. Current mutual information estimation methods seem inadequate for accurately estimating high mutual information in high-dimensional settings, which would be crucial for applicability in deep learning. Deep learning research appears to be shifting its focus towards projecting random variables onto lower-dimensional spaces, measuring dependency between these projections, and, more broadly, towards techniques from algorithmic information theory.

Unfortunately, during the writing of this thesis, we encountered issues inherent to deep learning research itself. Deep learning and artificial intelligence have been popular research topics over the past decade, attracting significant public and professional attention. The volume of papers published in the field has increased astoundingly in recent years, making it difficult to follow the literature in the first place. Despite ongoing efforts [35], the deep learning community seemingly still lacks appropriate infrastructure to review publications with sufficient academic rigor.

References

[1] Jan Beirlant, Edward J Dudewicz, László Györfi, Edward C Van der Meulen, et al. Nonparametric entropy estimation: An overview. *International Journal of Mathematical and Statistical Sciences*, 6(1):17–39, 1997.

[2] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeswar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and R Devon Hjelm. Mine: Mutual information neural estimation, 2021.

[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[4] Pengyu Cheng, Weituo Hao, Shuyang Dai, Jiachang Liu, Zhe Gan, and Lawrence Carin. Club: A contrastive log-ratio upper bound of mutual information. In *International conference on machine learning*, pages 1779–1788. PMLR, 2020.

[5] Paweł Czyż, Frederic Grabowski, Julia Vogt, Niko Beerenwinkel, and Alexander Marx. Beyond normal: On the evaluation of mutual information estimators. *Advances in Neural Information Processing Systems*, 36, 2024.

[6] M. D. Donsker and S. R.S. Varadhan. Asymptotic evaluation of certain markov process expectations for large time. iv. *Communications on Pure and Applied Mathematics*, 36(2):183–212, 1983.

[7] Adam Dziedzic, Haonan Duan, Muhammad Ahmad Kaleem, Nikita Dhawan, Jonas Guan, Yannis Cattan, Franziska Boenisch, and Nicolas Papernot. Dataset inference for self-supervised models. *Advances in Neural Information Processing Systems*, 35:12058–12070, 2022.

[8] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[9] Weihao Gao, Sewoong Oh, and Pramod Viswanath. Demystifying fixed k-nearest neighbor information estimators. *IEEE Transactions on Information Theory*, 64(8):5629–5661, 2018.

[10] Gabriel Goh, Nick Cammarata, Chelsea Voss, Shan Carter, Michael Petrov, Ludwig Schubert, Alec Radford, and Chris Olah. Multimodal neurons in artificial neural networks. *Distill*, 6(3):e30, 2021.

[11] Ziv Goldfeld and Kristjan Greenewald. Sliced mutual information: A scalable measure of statistical dependence. *Advances in Neural Information Processing Systems*, 34:17567–17578, 2021.

[12] Ziv Goldfeld, Kristjan Greenewald, Theshani Nuradha, and Galen Reeves. k-sliced mutual information: A quantitative study of scalability with dimension. *Advances in neural information processing systems*, 35:15982–15995, 2022.

[13] Gokul Gowri, Xiao-Kang Lun, Allon M Klein, and Peng Yin. Approximating mutual information of high-dimensional variables using learned representations. *arXiv preprint arXiv:2409.02732*, 2024.

[14] Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *International conference on algorithmic learning theory*, pages 63–77. Springer, 2005.

[15] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304. JMLR Workshop and Conference Proceedings, 2010.

[16] Hrayr Harutyunyan, Maxim Raginsky, Greg Ver Steeg, and Aram Galstyan. Information-theoretic generalization bounds for black-box learning algorithms. *Advances in Neural Information Processing Systems*, 34:24670–24682, 2021.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[18] Harold Hotelling. Relations between two sets of variates. In *Breakthroughs in statistics: methodology and distribution*, pages 162–190. Springer, 1992.

[19] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

[20] Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

[21] Patrick Kidger and Terry Lyons. Universal approximation with deep narrow networks. In *Conference on learning theory*, pages 2306–2327. PMLR, 2020.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[23] Leonenko Kozachenko. Sample estimate of the entropy of a random vector. *Probl. Peredachi Inf.*, 1987.

[24] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.

[25] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[26] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[27] Yazhe Li, Roman Pogodin, Danica J Sutherland, and Arthur Gretton. Self-supervised learning with kernel dependence maximization. *Advances in Neural Information Processing Systems*, 34:15543–15556, 2021.

[28] Ralph Linsker. Self-organization in a perceptual network. *Computer*, 21(3):105–117, 1988.

[29] Pratyush Maini, Mohammad Yaghini, and Nicolas Papernot. Dataset inference: Ownership resolution in machine learning. *arXiv preprint arXiv:2104.10706*, 2021.

[30] Alexander Marx and Jonas Fischer. Estimating mutual information via geodesic k nn. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pages 415–423. SIAM, 2022.

[31] David McAllester and Karl Stratos. Formal limitations on the measurement of mutual information. In *International Conference on Artificial Intelligence and Statistics*, pages 875–884. PMLR, 2020.

[32] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[33] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[34] Sherjil Ozair, Corey Lynch, Yoshua Bengio, Aaron Van den Oord, Sergey Levine, and Pierre Sermanet. Wasserstein dependency measure for representation learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[35] Andi Peng, Jessica Zosa Forde, Yonadav Shavit, and Jonathan Frankle. Strengthening subcommunities: Towards sustainable growth in ai research. In *ICLR*, 2022.

[36] Georg Pichler, Pierre Jean A Colombo, Malik Boudiaf, Günther Koliander, and Pablo Piantanida. A differential entropy estimator for training neural networks. In *International Conference on Machine Learning*, pages 17691–17715. PMLR, 2022.

[37] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195, 1999.

[38] Davide Piras, Hiranya V Peiris, Andrew Pontzen, Luisa Lucie-Smith, Ningyuan Guo, and Brian Nord. A robust estimator of mutual information for deep learning interpretability. *Machine Learning: Science and Technology*, 4(2):025006, 2023.

[39] Ben Poole, Sherjil Ozair, Aaron Van Den Oord, Alex Alemi, and George Tucker. On variational bounds of mutual information. In *International Conference on Machine Learning*, pages 5171–5180. PMLR, 2019.

[40] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[41] F. Rosenblatt. The perceptron - a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January 1957.

[42] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach (4th ed.)*. Pearson, 2021.

[43] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[44] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[45] Harshinder Singh, Neeraj Misra, Vladimir Hnizdo, Adam Fedorowicz, and Eugene Demchuk. Nearest neighbor estimates of entropy. *American journal of mathematical and management sciences*, 23(3-4):301–321, 2003.

[46] Le Song, Alex Smola, Arthur Gretton, Justin Bedo, and Karsten Borgwardt. Feature selection via dependence maximization. *Journal of Machine Learning Research*, 13(5), 2012.

[47] Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method, 2000.

[48] Michael Tschannen, Josip Djolonga, Paul K Rubenstein, Sylvain Gelly, and Mario Lucic. On mutual information maximization for representation learning. *arXiv preprint arXiv:1907.13625*, 2019.

[49] Dor Tsur, Ziv Goldfeld, and Kristjan Greenewald. Max-sliced mutual information. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[50] Vladimir Vapnik and Alexey Chervonenkis. Theory of pattern recognition, 1974.

[51] Leonid Nisonovich Vaserstein. Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredachi Informatsii*, 5(3):64–72, 1969.

[52] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

[53] Puning Zhao and Lifeng Lai. Analysis of knn information estimators for smooth distributions. *IEEE Transactions on Information Theory*, 66(6):3798–3826, 2019.