Intrinsic Dimensionality and Neural Networks

Dominique Barbe

The University of Melbourne, Victoria 3010, Australia

Abstract. Neural networks exhibit impressive performances, yet their nature is still poorly understood. We explored two directions in an attempt to increase the knowledge about them. We ran experiments, looking at how the dimensionality of the data changes through the network. And we defined an infinite model, more suited to theoretical analysis.

Keywords: neural networks, intrinsic dimensionality

1 Introduction

This document is an attempt to summarize the work realized at the University of Melbourne, under the supervision of Prof. James Bailey, from the 30/01/2017 to the 30/06/2017. I will go through the experiments I did, the theoretical model for neural networks developed (with the help of Michael E. Houle), and the intuition that guided me here and there.

Should you have any question regarding this document (or anything I worked on, like the code), feel free to contact me at `dominique.barbe@ens-rennes.fr`.

2 Preliminaries

This document assumes some familiarity with neural networks and the Local Intrinsic Dimensionality. We give a quick recap of these two notions here.

2.1 An overview of feedforward neural networks

Consider the following quantities:

- Let $(n, m) \in \mathbb{N}^2$.
- Let $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ be the *weight matrix* (or *weights*).
- Let $\boldsymbol{b} \in \mathbb{R}^m$ be the *bias vector*.
- Let $a : \mathbb{R} \to \mathbb{R}$ be the *activation function*; we also note a the component-wise application of a.

Consider the following function:

$$\begin{aligned} \mathbb{R}^n &\to \mathbb{R}^m \\ f : \boldsymbol{x} &\mapsto a\left(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b}\right) \end{aligned} \tag{1}$$

We call the function f a layer; it is parameterized by its weight matrix $\boldsymbol{W}$ and its bias vector $\boldsymbol{b}$. A *k-layers feedforward neural network* is a function defined by the composition of k layers.

For a more detailed introduction to neural networks, one can refer to [14].

2.2 The Intrinsic Dimensionality

Definition 1 ([8]). *Let* $\mathbf{X}$ *be a random distance variable. For any* r *such that* $F_{\mathbf{X}}(r) > 0$*, the* intrinsic dimensionality (ID) *of* $\mathbf{X}$ *at* r *is given by*

$$\mathrm{ID}_{F_{\mathbf{X}}}(r) \triangleq \lim_{\epsilon \to 0^+} \frac{\ln F_{\mathbf{X}}((1+\epsilon)r) - \ln F_{\mathbf{X}}(r)}{\ln((1+\epsilon)r) - \ln r} = \frac{r \cdot F'_{\mathbf{X}}(r)}{F_{\mathbf{X}}(r)}, \tag{2}$$

wherever the limit exists. The second equality follows by applying l'Hôpital's rule to the limits, provided that $F_{\mathbf{X}}$ *is positive and differentiable over an open interval containing* r*.*

Under this distributional interpretation, the original data set determines a sample of distances from a given point. The intrinsic dimensionality of this distance distribution $F_{\mathbf{X}}$ is estimated. The definition of $\mathrm{ID}_{F_{\mathbf{X}}}$ can be extended to the case where $r = 0$, by taking the limit of $\mathrm{ID}_{F_{\mathbf{X}}}(r)$ as $r \to 0^+$, whenever this limit exists:

$$\mathrm{ID}_{F_{\mathbf{X}}}(0) \triangleq \lim_{r \to 0^+} \mathrm{ID}_{F_{\mathbf{X}}}(r)\,. \tag{3}$$

The intuition and the formal definition we gave involved distances distributions. However, this definition can be extended to any multivariate real-valued function that is non-zero over some open ball around x (the point where the ID is computed).

An important result is the Local ID Representation theorem, which states conditions under which a function F is fully characterized by ID_F.

Local ID Representation Theorem

Theorem 1 ([7]). *Let* $F : \mathbb{R} \to \mathbb{R}$ *be a real-valued function. Let* $v \in \mathbb{R}$ *be a value for which* $\mathrm{ID}_F(v)$ *exists. Let* $x, w \in \mathbb{R}$ *be values for which the quantities* x/w *and* $F(x)/F(w)$ *exist and are positive. Then if* F *is non-zero and continuously differentiable everywhere over* $[\min\{x,w\}, \max\{x,w\}]$*, except perhaps at* v *itself,*

$$\frac{F(x)}{F(w)} = \left(\frac{x}{w}\right)^{\mathrm{ID}_F(v)} \cdot G_{F,v,w}(x), \text{ where} \tag{4}$$

$$G_{F,v,w}(x) \triangleq \exp\left(\int_x^w \frac{\mathrm{ID}_F(v) - \mathrm{ID}_F(t)}{t}\,\mathrm{d}t\right), \tag{5}$$

whenever the integral exists.

Moreover, let $c > 1$ *be a constant. We have:*

- *if* $v \neq 0$*,*

$$\lim_{\substack{w \to 0 \\ |x-v| \leq c|w-v|}} G_{F,0,w}(r) = 1 \tag{6}$$

- *if* $v = 0$*,*

$$\lim_{\substack{w \to 0^+ \\ 0 < w/c \leq r \leq cw}} G_{F,0,w}(r) = \lim_{\substack{w \to 0^- \\ 0 < w/c \leq r \leq cw}} G_{F,0,w}(r) = 1 \tag{7}$$

Informally, given a function F and three values, a b and c relatively close to each others. We have:

$$\frac{F(a)}{F(b)} \simeq \left(\frac{a}{b}\right)^{\mathrm{ID}_F(c)} \tag{8}$$

In other words, we can switch from a reasoning on the value of F to simply ratios of inputs a/b, the ID doing the link between the two. The notion of intrinsic dimensionality is analogous to the derivative, but allows working with quotients instead of differences:

$$F(a) - F(b) \simeq (a - b) \cdot F'(c) \tag{9}$$

3 Experiments with networks

3.1 Introduction

Most of the internship consisted in a serie of experiments, trying to measure the complexity of the data (using the ID) at different steps through the network and at different moments during training. None of the experiments gave us new, interesting insights. We will go through most of them.

3.2 Experimental setup

An auto-encoder is a neural network that tries to reproduce its input (in other words, it tries to approximate the identity function). It consists of two part: the encoder — a first serie of layers of decreasing size — and the decoder — a serie of layers of increasing size.

Most of the experiments use auto-encoders, as these networks do not require labels for the data. They try to find a generic model for their training set (generic because their is other goal than reconstructing the dataset). How they model the data is what we want to analyze. We wanted to see how much a network would simplify the data it receives: does the data reconstructed by the network has a low or a high dimensionality?

There are two kind of plots: LID distributions, and mean LID evolution. Most of the experiments involve the following process: reconstructing the dataset (i.e taking its image by the network), and estimating the LID for all/sampled points. The LIDs collected are either plotted as a distribution, or averaged and plotted (against the training epoch most of the time).

We use the MNIST [1] and CIFAR-10 [2] datasets for all experiments.

3.3 Output complexity

We compare different autoencoders. For each of them, we take the image of a dataset by a network, and compute the LID distribution of the transformed dataset. The LID is estimated using the MLE ($k = 100$ neighours are used). It is computed for every point in the training set, and the probability distribution

function of the distribution of the LID is plotted. The results are shown in Figure 1 and 2.

The following networks are used:

- The networks named "Dx" are networks using one dense hidden layer, with x hidden neurons. Rectified linear units are used for the activation in the hidden layer, and sigmoid units for the output.
- The networks named "Cx" are convolutional neural network, using x features maps of 5 by 5 pixels. Rectified linear units are used for the activation, followed by a max polling layer of size 2 by 2. The output layer also uses sigmoid functions for the activation.

Remark: I didn't spend that much time tweaking the training of the networks. Their performances may be good, but still not exploit to the fullest their individual architectures. This can explain why, for example, the networks D8 and D16 perform similarly on the MNIST dataset, despite one of them being more powerful.

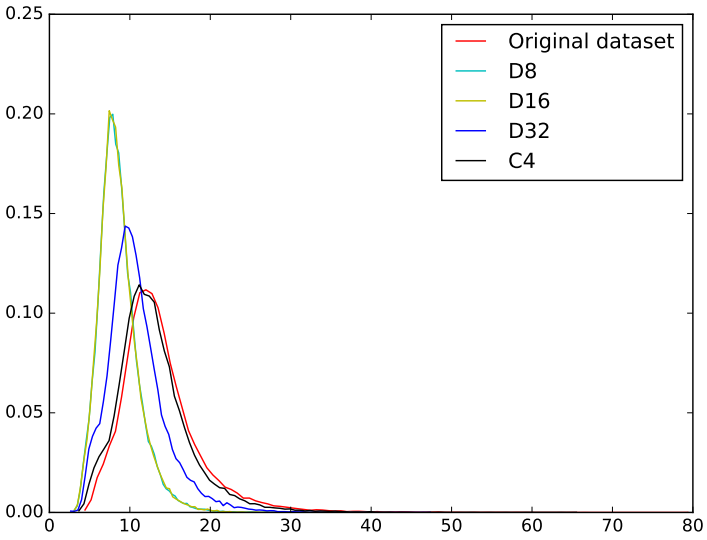


Fig. 1: LID distribution for the MNIST dataset

3.4 Output complexity during training

We examined the evolution of the output complexity during training. For this serie of experiments we plotted the training accuracy and the mean LID against the training epoch.

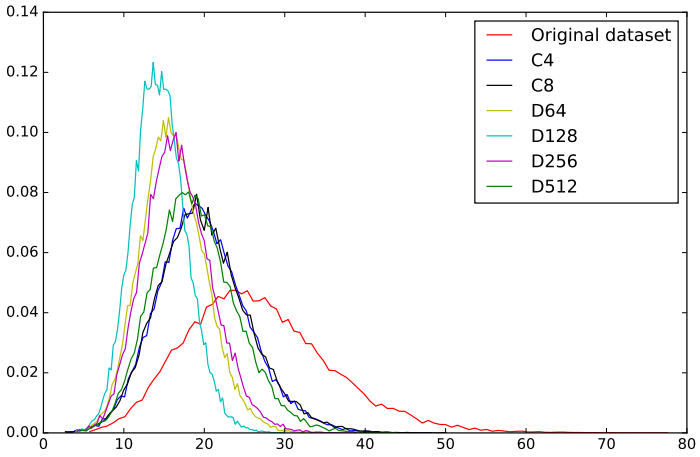
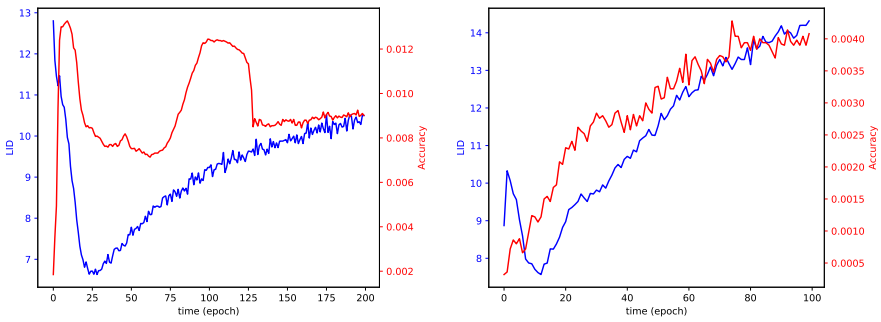


Fig. 2: LID distribution for the CIFAR-10 dataset

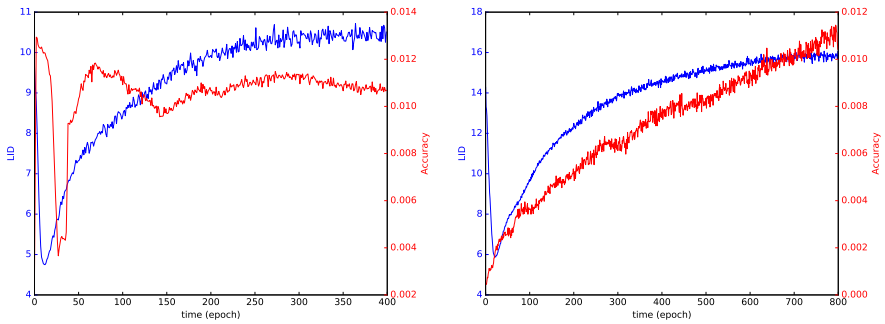
A first experiment study the short-term dynamic. Here, a 2-layers auto-encoder with 32 hidden units is used. The activation functions are ReLU for the hidden layer and sigmoid for the output layer (for re-scaling purpose). The ID is computed on the test set, estimated using 1000 samples with the Maximum Likelihood Estimation [9]. The learning rate is set low (.002), as convergence is not more important here than the dynamic. Results are shown on Figure 3.

The second experiment studies the long-term dynamic. Again, a 2-layer auto-encoder is used, with 32 hidden units for MNIST and 64 for CIFAR. The learning rates are respectively $lr = 0.005$ and $lr = 0.001$. Results are shown on Figure 4.



(a) MNIST dataset (b) CIFAR-10 dataset

Fig. 3: Accuracy and output ID plotted against the training epoch



(a) MNIST dataset (b) CIFAR-10 dataset

Fig. 4: Accuracy and output ID plotted against the training epoch

These plots show a decrease followed by an increase in output ID. Interpreting the ID as a complexity measure, these plots can be interpreted as such: the parameters being random at first, the data undergo random transformations, and thus the output complexity starts high. As patterns are learned, the network fits a simple model to the data, decreasing the output complexity. Finally, finer details are learned; the approximation getting better, the complexity increases again, getting closer to the one of the original dataset.

4 Experiments with neurons

All the networks studied here have the same architecture unless told otherwise:

- feedforward autoencoders with one hidden layer.
- the hidden layer uses rectified linear units.
- the output layer uses sigmoid units.

Three datasets are used: MNIST [1], Cifar-10 [2] and Hasy [3]. All three are images sets. For each dataset, all but 10000 images are used for the training set, and the remaining 10000 for the validation set.

4.1 Introduction

Rectified linear units allow for an easy partition of a dataset in two: either an input has a null activation (**low-activation set**) or a strictly positive activation (**high activation set**). We want to investigate the properties of these two sets, for a given neuron in a given network.

First, we ran the following experiment: we took an autoencoder trained on the MNIST dataset. For each neuron, we partitioned the validation set in two (using the previously described method, that is computing the high and low-activation sets). We then computed the following metrics:

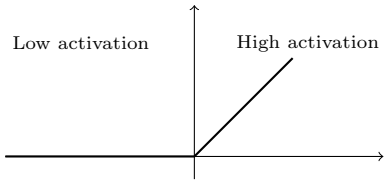


Fig. 5: Rectified Linear Unit

- size of the two sets.
- dominant class of the two sets.
- proportion of the dominant class.
- mean LID of the two sets (using ALIDv4, k=12, sampled on 500 points).

Note: the LID of the original images is computed. The LID is computed for each set independently (that is: neighbours are drawn from the same set, not the whole dataset).

Below are some examples of such metrics for a few neurons (taken from the same network):

- Neuron (a):
  - low-activation set: 159 images. Class 6 is dominant (49% of the images). LID = 11.87.
  - high-activation set: 9841 images Class 1 is dominant (11% of the images). LID = 14.91.
- Neuron (b):
  - low-activation set: 4796 images. Class 3 is dominant (17% of the images). LID = 15.55.
  - high-activation set: 5204 images Class 6 is dominant (18% of the images). LID = 14.14.
- Neuron (c):
  - low-activation set: 9612 images. Class 1 is dominant (11% of the images). LID = 15.37.
  - high-activation set: 388 images Class 7 is dominant (36% of the images). LID = 12.44.

We observed the following correlations:

- The two sets can be unbalanced. One is huge and the other is small. The huge set has high LID (close to the one of the original dataset); the small set has low ID. The huge set doesn't have a clear dominant class (often 11%-12% of its images for 10 classes); the small set has a clear dominant class.
- The two sets can be balanced. They then tend to have the same LID, and a dominant class of the same proportion.

From here, we concluded that:

- Neurons can be classified into three categories:

  - *normal neurons* are unbalanced, having a small low-activation set and a huge high-activation one (They are the most common, hence the name).
  - *balanced neurons* are balanced. Their two sets are of the same size.
  - *inverted neurons* are unbalanced, having a huge low-activation set and a small high-activation one.
- Some neurons clearly belong to one of those three categories; some are harder to classify (neither really unbalanced nor balanced).
- From now on, we will only compute the size of the high-activation set of the different neurons we analyze. The empiric correlations described earlier show that it is enough to determine the type of a given neuron.

We then tried to answer the following question: what has an influence on the distribution of the type of neurons? We explored three parameters: the initialization of the weights, the size of the hidden layer and the training database.

4.2 Influence of the initialization

We trained 7 different networks using the same architecture (256 hidden neurons), the same training set (MNIST here) and the same training process (i.e. number of epochs and learning rate). The only difference between them are the random weight initialization.

For each network, we compute the size of the high-activation set for each of its neuron. Then, the distribution of these sizes is plotted. The results are displayed on Figure 6.

The initialization doesn't seem to have a significant effect on the distribution of the type of neurons. Further experiments with the other 2 datasets confirmed this trend (the plots are not shown). Here, for every network trained, most neurons are *normal*, a small number are *inverted*, and almost none are *balanced*.

4.3 Influence of the size of the hidden layer

We trained 5 different networks, using between 64 and 1024 hidden neurons (still with the MNIST dataset). The results are displayed on Figure 7.

Note: the input dimension is 784 here; 1024 is already beyond that.

Again, the number of hidden units doesn't seem to have a significant effect on the distribution of the type of neurons.

Still, networks using a high number of neurons may have a smoother distribution. Maybe this is an artifact of the discretization required to trace the plot. Maybe there is a limit distribution when the number of hidden neurons tends toward infinity. We may investigate this question later, but we postponed it, as auto-encoder with too many hidden units do not make sense.

4.4 Influence of the dataset

We trained networks using 256 hidden neurons on the three datasets (MNIST, Cifar-10 and Hasy). Again, the distribution of the sizes of the high-activation sets are plotted together. The results are displayed on Figure 8.

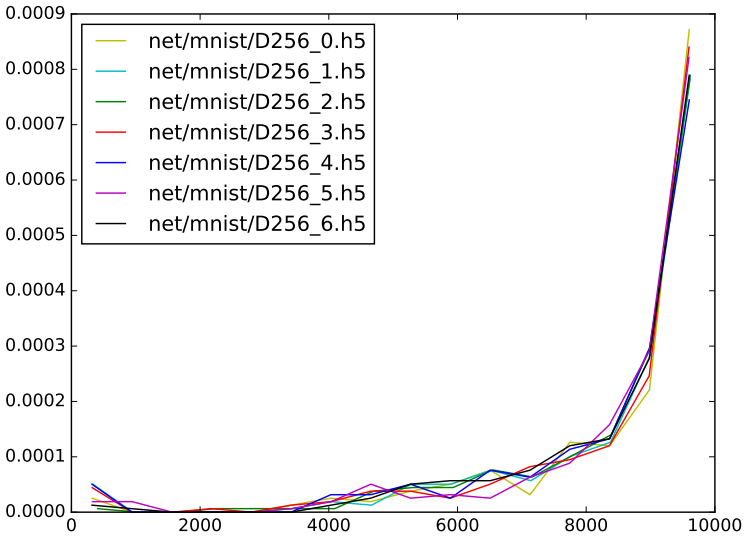


Fig. 6: Comparison with 7 different initialization on the repartition of the neuron types

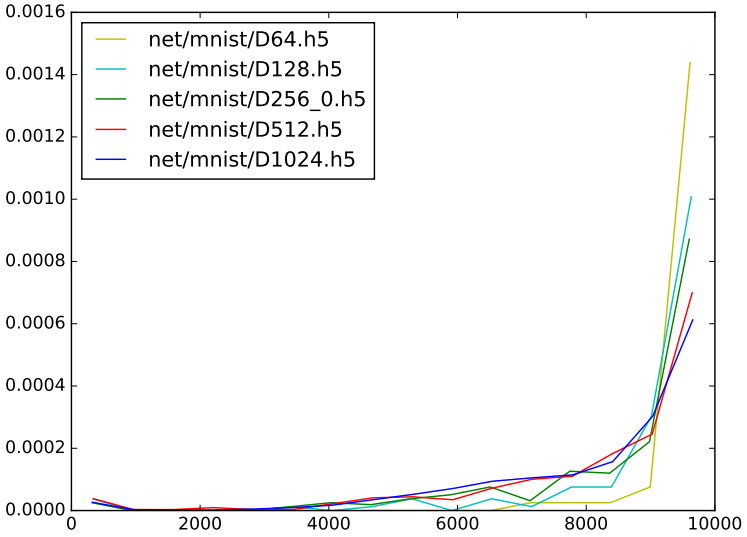


Fig. 7: Comparison with 5 different number of hidden neurons on the repartition of the neuron types

Remark: The distributions are averaged over 7 different initialization. We have seen that the initialization doesn't have a significant impact on the type of the neurons; we can take advantage of this fact to obtain more precise plots by using many networks to analyse more neurons for a given architecture/database.

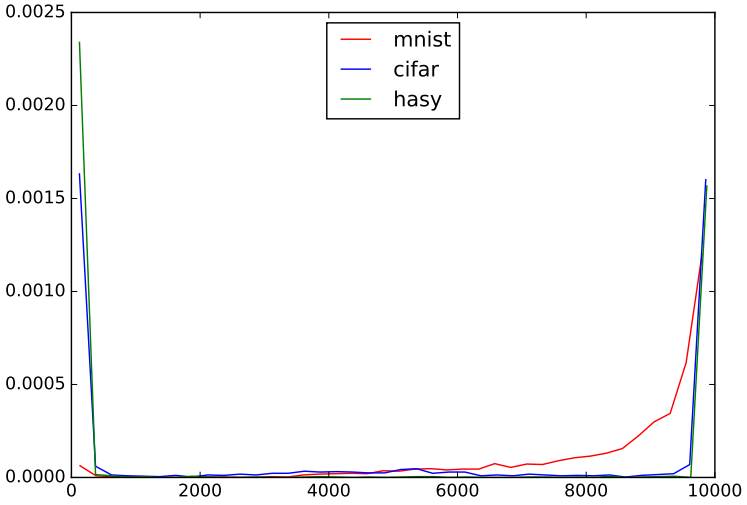


Fig. 8: Comparison with 3 different datasets on the repartition of the neuron types

The dataset has a large impact on the distribution of the type of neurons. MNIST produces mostly *normal* neurons, where Hasy creates mostly *inverted* neurons. Cifar-10 lying in between the two.

5 Some results on ID

Before moving on with the infinite model for neural network, we give some mildly interesting results on ID

5.1 Composition of functions

We give a set of theorems on the behaviour of ID when composing (multivariate) functions. Let $g : \mathbb{R}^n \mapsto \mathbb{R}^m$ and $\boldsymbol{x} \in \mathbb{R}^n$. We note:

- x_i the ith component of $\boldsymbol{x}$.
- $g_j : \mathbb{R}^n \mapsto \mathbb{R}$ the function obtained from g by taking only its jth output.

- ${}_ig : \mathbb{R} \mapsto \mathbb{R}^m$ the function obtained from g obtained by treating all its inputs as constants (equal to the coordinates of $\boldsymbol{x}$), except the ith one. We note that $g(\boldsymbol{x}) = {}_ig(x_i)$.
- ${}_ig_j : \mathbb{R} \mapsto \mathbb{R}$ the function obtained from ${}_ig$ by taking only its jth output. We note that $g_j(\boldsymbol{x}) = {}_ig_j(x_i)$.

When composing functions, we have the following relations on ID:

Theorem 2. *Let* $\mathbb{R} \xrightarrow{g} \mathbb{R} \xrightarrow{f} \mathbb{R}$. *We have:*

$$\mathrm{ID}_{f\circ g}(x) = \mathrm{ID}_g(x).\mathrm{ID}_f(g(x)) \tag{10}$$

Theorem 3. *Let* $\mathbb{R}^n \xrightarrow{g} \mathbb{R} \xrightarrow{f} \mathbb{R}$. *We have:*

$$\mathrm{ID}_{f\circ g}(\boldsymbol{x}) = \mathrm{ID}_g(\boldsymbol{x}).\mathrm{ID}_f(g(\boldsymbol{x})) \tag{11}$$

Theorem 4. *Let* $\mathbb{R} \xrightarrow{g} \mathbb{R}^m \xrightarrow{f} \mathbb{R}$. *We have:*

$$\mathrm{ID}_{f\circ g}(x) = \sum_{j=1}^{n} \left[\mathrm{ID}_{g_j}(x).\mathrm{ID}_{{}_jf}(g_j(x))\right] \tag{12}$$

Theorem 5. *Let* $\mathbb{R} \xrightarrow{g} \mathbb{R}^m \xrightarrow{f} \mathbb{R}$. *We have:*

$$\mathrm{ID}_{f\circ g}(\boldsymbol{x}) = \sum_{j=1}^{n} \mathrm{ID}_{{}_jf\circ g_j}(\boldsymbol{x}) \tag{13}$$

Proof. (Theorem 2) For $\mathbb{R} \xrightarrow{g} \mathbb{R} \xrightarrow{f} \mathbb{R}$:

$$\begin{aligned}
\mathrm{ID}_{f\circ g}(x) &= \frac{x.(f\circ g)'(x)}{(f\circ g)(x)} \\
&= \frac{x.g'(x)}{g(x)}.\frac{g(x).f'(g(x))}{f(g(x))} \\
&= \mathrm{ID}_g(x)\mathrm{ID}_f(g(x))
\end{aligned}$$

(Theorem 3) For $\mathbb{R}^n \xrightarrow{g} \mathbb{R} \xrightarrow{f} \mathbb{R}$:

$$\begin{aligned}
\mathrm{ID}_{f\circ g}(\boldsymbol{x}) &= \frac{\boldsymbol{x}\cdot\nabla(f\circ g)(\boldsymbol{x})}{(f\circ g(\boldsymbol{x}))} \\
&= \sum_{i=1}^{n}\left[\frac{x_i.\,{}_ig'(x_i).f'({}_ig(x_i))}{f(g(x))}\right] \\
&= \sum_{i=1}^{n}\left[\frac{x_i.\,{}_ig'(x_i)}{{}_ig(x_i)}.\frac{{}_ig(x_i).f'({}_ig(x_i))}{f(g(x))}\right] \\
&= \sum_{i=1}^{n}\left[\mathrm{ID}_{{}_ig}(x_i).\frac{g(\boldsymbol{x}).f'(g(\boldsymbol{x}))}{f(g(x))}\right] \\
\mathrm{ID}_{f\circ g}(\boldsymbol{x}) &= \mathrm{ID}_g(x)\mathrm{ID}_f(g(x))
\end{aligned}$$

(Theorem 4) For $\mathbb{R} \xrightarrow{g} \mathbb{R}^m \xrightarrow{f} \mathbb{R}$:

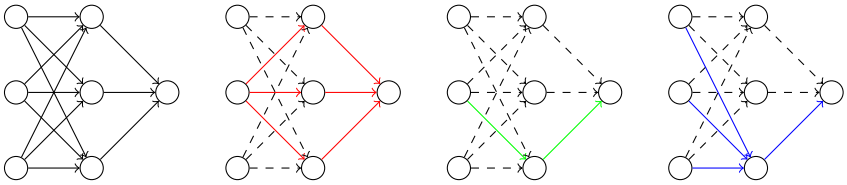
$$\begin{aligned}
\mathrm{ID}_{f\circ g}(x) &= \frac{x.(f\circ g)'(x)}{(f\circ g)(x)} \\
&= \sum_{j=1}^{n} \frac{x.g_j'(x).\,{}_jf(x)}{f(g(x))} \\
&= \sum_{j=1}^{n} \left[\frac{x.g_j'(x)}{g_j(x)}.\frac{g_j(x).\,{}_jf'(g_j(x))}{f(g(x))}\right] \\
&= \sum_{j=1}^{n} \left[\mathrm{ID}_{g_j}(x).\frac{g_j(x).\,{}_jf'(g_j(x))}{{}_jf(g_j(x))}\right] \\
\mathrm{ID}_{f\circ g}(x) &= \sum_{j=1}^{n} \Big[\mathrm{ID}_{g_j}(x).\mathrm{ID}_{{}_jf}(g_j(x))\Big]
\end{aligned}$$

(Theorem 5) For $\mathbb{R}^n \xrightarrow{g} \mathbb{R}^m \xrightarrow{f} \mathbb{R}$:

$$\begin{aligned}
\mathrm{ID}_{f\circ g}(\boldsymbol{x}) &= \sum_{i=1}^{n} \mathrm{ID}_{f\circ_i g}(x_i) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} \Big[\mathrm{ID}_{{}_ig_j}(x_i).\mathrm{ID}_{{}_jf}({}_ig_j(x_i))\Big] \\
&= \sum_{j=1}^{n}\sum_{i=1}^{n} \Big[\mathrm{ID}_{{}_ig_j}(x_i).\mathrm{ID}_{{}_jf}({}_ig_j(x_i))\Big] \\
&= \sum_{j=1}^{n}\sum_{i=1}^{n} \Big[\mathrm{ID}_{{}_ig_j}(x_i).\mathrm{ID}_{{}_jf}(g_j(\boldsymbol{x}))\Big] \\
&= \sum_{j=1}^{n} \Big[\mathrm{ID}_{g_j}(\boldsymbol{x}).\mathrm{ID}_{{}_jf}(g_j(\boldsymbol{x}))\Big] \\
\mathrm{ID}_{f\circ g}(\boldsymbol{x}) &= \sum_{j=1}^{n} \mathrm{ID}_{{}_jf\circ g_j}(\boldsymbol{x})
\end{aligned}$$

Notes on Theorem 5: Theorem 5 give a non-trivial (yet simple to prove) decomposition of the ID of the composition of two functions. The proof involves breaking the overall path taken by the inputs into every individual possible path, and recomposing them.

We can plot the interplay between the different coordinates of $\mathbb{R} \xrightarrow{g} \mathbb{R}^m \xrightarrow{f} \mathbb{R}$ (see Figure 9). The first decomposition is done along each input coordinate. Then the sum is decomposed again using Theorem 4. Finally, it is recomposed using Theorem 3.



(a) Connections (b) Decomposition 1 (c) Decomposition 2 (d) Recomposition

Fig. 9: Paths involved in the proof of Theorem 5.

5.2 The weighted ID of common functions

We try to find explicit formulas for the wID of common transformations involved in neural networks (linear combinations and activation functions).

Linear combinations

Theorem 6. *Let* $(a, b) \in \mathbb{R}^2$. *Let* $\begin{array}{rccc} f : & \mathbb{R} & \to & \mathbb{R} \\ & x & \mapsto & a.x + b \end{array}$. *Then:*

$$\mathrm{wID}_f(x) = a.x \tag{14}$$

Theorem 7. *Let* $\boldsymbol{a} \in \mathbb{R}^n$ *and* $b \in \mathbb{R}$. *Let* $\begin{array}{rccc} f : & \mathbb{R}^n & \to & \mathbb{R} \\ & \boldsymbol{x} & \mapsto & \boldsymbol{a} \cdot \boldsymbol{x} + b \end{array}$. *Then:*

$$\mathrm{wID}_f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{a} \tag{15}$$

$$\mathrm{wID}_{_j f}(x) = x.a_j \tag{16}$$

Sigmoid

Theorem 8. *Let* σ *be the sigmoid function:* $\begin{array}{rccc} \sigma : & \mathbb{R} & \to & \mathbb{R} \\ & x & \mapsto & \frac{e^x}{1+e^x} \end{array}$. *The sigmoid function has the two following properties:*

$$\sigma(x) = 1 - \sigma(-x) \tag{17}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{18}$$

Its weighted ID is:

$$\mathrm{wID}_\sigma(x) = x.\sigma(x).\sigma(-x) \tag{19}$$

Proof. (First property)

$$1 - \sigma(-x) = \frac{1 + e^{-x}}{1 + e^{-x}} - \frac{e^{-x}}{1 + e^{-x}} = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x} = \sigma(x)$$

(Second property)

$$\sigma'(x) = \frac{e^x.(1+e^x) - e^x.e^x}{(1+e^x)^2} = \frac{e^x}{1+e^x} - \left(\frac{e^x}{1+e^x}\right)^2$$
$$= \sigma(x) - \sigma(x)^2 = \sigma(x).(1-\sigma(x))$$

(Weighted ID)

$$\mathrm{wID}_\sigma(x) = x.\sigma'(x) = x.\sigma(x).(1-\sigma(x)) = x.\sigma(x).\sigma(-x)$$

Remark: The weighted ID of the sigmoid function has the following shape:

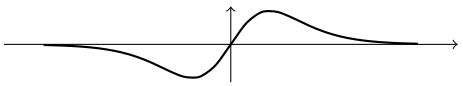


Fig. 10: The weighted ID of the sigmoid function

ReLU

Theorem 9. *Let* $\mathrm{ReLU} : x \mapsto \max(0, x)$ *be the Rectified Linear Unit function. Its weighted ID is:*

$$\mathrm{wID}_{\mathrm{ReLU}}(x) = \mathrm{ReLU}(x) \tag{20}$$

Proof. Let $x \in \mathbb{R}^*$.

$$\mathrm{wID}_{\mathrm{ReLU}}(x) = x.\mathrm{ReLU}'(x) = \begin{cases} x.1 \text{ if } x > 0 \\ x.0 \text{ if } x < 0 \end{cases} = \mathrm{ReLU}(x)$$

6 A model for infinite neural networks

6.1 Introduction

The universal approximation theorem states that a feedforward neural network with one hidden layer can approximate any continuous function on compacts of $\mathbb{R}^n$, as the number of hidden neurons tends to infinity [15]. This theorem is our main reason to consider infinite networks, but other results push us in the same directions.

Chiyuan Zhang et al. [13] showed through experiments that deep neural networks easily fit random noise. Yet they are also able to generalize properly from data that make sense. The networks they manipulate are over-parameterized. Despite this fact, they avoid overfitting. This suggest that the capacity (i.e the number of parameters) is not the proper metric to measure the expressive power of a network. This quantity could then be abstracted by considering infinite networks. Moreover, even when neural networks achieve a perfect accuracy on their

training sets, their accuracy at test time still increases as hidden neurons are added. In other words, even when the training examples are perfectly recognized there is still something to learn.

Our intuition is that real (i.e finite) networks are an approximation of infinite objects: neural networks with an infinite number of neurons at every hidden layer. Real networks are the result of a sampling process from the space of the possible weights. Increasing the size of a network then allows for a better approximation. The quality of the approximation (and thus the performances of the network) is determined by the quality of the sampling process. Overfitting would then not be linked to the number of parameters, but rather the capacity of the sampling process to select the right neurons for difficult regions of the input space.

Generally speaking, a continuous point of view seem to be a better support for intuition, design and theoretical analysis in machine learning. This is the point of view adopted with the ID, where one manipulates distributions instead of fixed set of points. When practical work has to be done the ID is estimated, the data being interpreted as samples from an underlying distribution. We wish to achieve a similar result with neural networks: to get rid of the discrete complexity (the connections, the number of neurons, etc), to abstract it into series of continuous transformations.

6.2 Proposed model

The model we propose is inspired by Hazam et al. [16]. They model infinite neural networks with an arbitrary depth to create new kernels mimicking deep learning. In order to compute analytically their kernels, they restrict themselves to Gaussian distributions of the weights. We lift this restriction to get a more generic model.

In this section, we use the following notations.

- let $\langle \cdot \rangle$ be the scalar product in $\mathbb{R}^n$
- let $a : \mathbb{R} \to \mathbb{R}$ be an activation function.

We consider a neural network with one hidden layer and one output neuron. We isolate a hidden neuron, and analyze its contribution to the output. This hidden neuron is associated with a weighting of the input $\boldsymbol{w} \in \mathbb{R}^n$, and a weighting of its output $u \in \mathbb{R}$. Its contribution to the global output is:

$$a(\langle \boldsymbol{x}, \boldsymbol{w} \rangle) \cdot u \tag{21}$$

In practice with neural networks, the hidden layers contains a set number of hidden neurons. The ith neuron is associated with a weight $\boldsymbol{w}_i \in \mathbb{R}^n$. The output of each neuron is weighted by a number $u(\boldsymbol{w}_i) \in \mathbb{R}$. The final output is a weighted sum of the activation of each neuron:

$$\sum_i a(\langle \boldsymbol{x}, \boldsymbol{w}_i \rangle) \cdot u(\boldsymbol{w}_i) \tag{22}$$

We give a theoretical model for infinite neural networks. Instead of having a fixed, finite number of hidden neurons (and their respective weights), we consider all possible weightings of the input. The final output is again a weighted sum of the activation of each neuron. Consider the following function parameterized by $u : \mathbb{R}^n \to \mathbb{R}$:

$$\begin{array}{rl} & \mathbb{R}^n \to \mathbb{R} \\ F_u : \boldsymbol{x} & \mapsto \int_{\mathbb{R}^n} a(\langle \boldsymbol{x} \cdot \boldsymbol{w} \rangle) \cdot u(\boldsymbol{w}) d\boldsymbol{w} \end{array} \tag{23}$$

This infinite network is parameterized by $u : \mathbb{R}^n \to \mathbb{R}$, a continous and integrable fonction weighting each activation, and μ, a probability distribution of the different weights.

Such a function computes every possible weighted input $\langle \boldsymbol{x} \cdot \boldsymbol{w} \rangle$, then computes the corresponding activations, and finally weights these activations according to u. Neural networks with one hidden layer can then be seen as an approximation of such functions. Instead of considering all possible initial weights $\boldsymbol{w}$, only a finite number are chosen; the integral then becomes a finite sum. In other words, the hidden neurons of a neural networks (and their associated weights) can be thought as the result of a sampling process of an underlying distribution.

This model has other differences with the finite one (presented earlier) than adopting a continuous point of view. The output layer is not equipped with an activation function, for two reasons. First, the universal approximation theorem (for finite networks) do not require activation functions on the output layer. Second, the last activation function is commonly used for convenience or interpretation reasons: a sigmoid function would fit the output in $(0, 1)$, a softmax layer fit the output to a probability distribution (the output values are in $(0, 1)$ and sum to 1), etc. To keep the analysis generic, we remove the last activation function.

6.3 An interpretation for binary classification

Equation 23 can be rewritten to have a simple interpretation in the case of binary classification. u being a weighting of all the possible activations, it resembles a probability density function. However, it is not always positive, and its integral is not 1. We thus define the following quantities:

$$u_+ = max(0, u) \qquad U_+ = \int u_+ \qquad \mu_+ = \frac{u_+}{U_+} \tag{24}$$

$$u_- = max(0, -u) \qquad U_- = \int u_- \qquad \mu_- = \frac{u_-}{U_-} \tag{25}$$

u_+ and u_- are the positive and negative parts of u; they are the two positive functions verifying $u = u_+ - u_-$. Then, μ_+ and μ_- are two probability density functions, obtained from u_+ and u_- by re-scaling them. We note that these distributions describe disjoint regions of $\mathbb{R}^n$; in other words, when one is (strictly)

positive, the other has to be null. We now rewrite Equation 23:

$$F_u(\boldsymbol{x}) = \int_{\mathbb{R}^n} a(\langle \boldsymbol{x} \cdot \boldsymbol{w} \rangle) \cdot u(\boldsymbol{w}) d\boldsymbol{w} \tag{26}$$

$$= \int_{\mathbb{R}^n} a(\langle \boldsymbol{x} \cdot \boldsymbol{w} \rangle) \cdot u_+(\boldsymbol{w}) d\boldsymbol{w} - \int_{\mathbb{R}^n} a(\langle \boldsymbol{x} \cdot \boldsymbol{w} \rangle) \cdot u_-(\boldsymbol{w}) d\boldsymbol{w} \tag{27}$$

$$= U_+ \int_{\mathbb{R}^n} a(\langle \boldsymbol{x} \cdot \boldsymbol{w} \rangle) d\mu_+(\boldsymbol{w}) - U_- \int_{\mathbb{R}^n} a(\langle \boldsymbol{x} \cdot \boldsymbol{w} \rangle) d\mu_-(\boldsymbol{w}) \tag{28}$$

$$= U_+ \mathbb{E}_{\boldsymbol{w} \sim \mu_+} \left[a(\langle \boldsymbol{x} \cdot \boldsymbol{w} \rangle) \right] - U_- \mathbb{E}_{\boldsymbol{w} \sim \mu_-} \left[a(\langle \boldsymbol{x} \cdot \boldsymbol{w} \rangle) \right] \tag{29}$$

Thus, the output of the network is the (weighted) averaged activation over one region of the weights space, minus the (weighted) averaged activation over a disjoint region of the weight space.

We assume a to be both positive and increasing (like ReLU or sigmoid). Intuitively, when the input $\boldsymbol{x}$ is in a dense regions of μ_+, many scalar products $\langle \boldsymbol{x} \cdot \boldsymbol{w} \rangle$ will be high. As a result, many activations will be high (a being positive and increasing). Thus, the expectation $U_+ \mathbb{E}_{\boldsymbol{w} \backsim \mu_+} \left[a(\langle \boldsymbol{x} \cdot \boldsymbol{w} \rangle) \right]$ will tend to be high. Conversely $\boldsymbol{x}$ will be in a void region of μ_-; by a similar argument $U_- \mathbb{E}_{\boldsymbol{w} \backsim \mu_-} \left[a(\langle \boldsymbol{x} \cdot \boldsymbol{w} \rangle) \right]$ will tend to be low (close to 0). Thus, when the input $\boldsymbol{x}$ is in a dense regions of μ_+, the output of the network $F_u(\boldsymbol{x})$ will tend to be high.

A symmetric argument can be given for inputs in dense regions of μ_- giving a low (negative) output. The function F_u can be seen as binary classifier, where μ_+ describes the density of the first class, associated with positive outputs, and μ_- describes the density of the second class, associated with negative outputs.

6.4 Possible extensions

Adding more layers We gave a model suited for neural networks with one hidden layer. It is possible to extend this construction for any number of layer.

We give an example for the modeling of a 3-layers feedforward neural network: we start from the function F_u, a weighting by a given u of all the possible activations. We consider the activation of this quantity: $a(F_u)$. This is one possible activation for the second (hidden) layer. A weighting of all possible activations gives an infinite model for a 3-layers feedforward neural network. Let $v : \mathbb{R}^{\mathbb{R}} \to \mathbb{R}$. We define:

$$G_v = \int a\left(F_u(\boldsymbol{x})\right) \cdot v(u) du \tag{30}$$

(We assume that v is chosen such that the above integral is well defined.)

Infinite networks with one layer were parameterized with a real-valued function over $\mathbb{R}$, similar to a probability density. Networks with two layers are now parameterized with real-valued function of functions, similar to probability measures. This construction can be repeated an arbitrary number of times. The interpretation of such models may be harder though. This direction is left open for future studies.

The effect of perturbations Let $\boldsymbol{x} \in \mathbb{R}^n$ be a reference location in space, the unit vector $\hat{\mathbf{y}} \in \mathbb{R}^n$ be the direction of perturbation, and $u : \mathbb{R} \to \mathbb{R}$ a suitable parameter for an infinite network. Consider the following function:

$$\begin{array}{rl} & \mathbb{R}_+ \to \mathbb{R} \\ \bar{F_u} : \epsilon & \mapsto F_u(\boldsymbol{x} + \epsilon\hat{\mathbf{y}}) \end{array} \tag{31}$$

$\bar{F_u}(\epsilon)$ is the output of the network for an input $\boldsymbol{x}$ modified by an amount ϵ in a direction $\hat{\mathbf{y}}$. We want to study how much the output of the network can change for small perturbations. We thus compute the ID of $\bar{F_u}$ around 0. Let $A_{\boldsymbol{w}} : \epsilon \mapsto a(\langle \boldsymbol{x} + \epsilon\hat{\mathbf{y}} \cdot \boldsymbol{w} \rangle)$. We have:

$$\mathrm{wID}_{\bar{F_u}}(\epsilon) = \epsilon \cdot \left(\int_{\mathbb{R}^n} A_{\boldsymbol{w}}(\epsilon) \cdot u(\boldsymbol{w}) d\boldsymbol{w} \right)' (\epsilon) \quad \text{(See Equation 2)} \tag{32}$$

$$= \int_{\mathbb{R}^n} \epsilon \cdot A'_{\boldsymbol{w}}(\epsilon) \cdot u(\boldsymbol{w}) d\boldsymbol{w} \tag{33}$$

$$= \int_{\mathbb{R}^n} \mathrm{wID}_{A_{\boldsymbol{w}}}(\epsilon) \cdot u(\boldsymbol{w}) d\boldsymbol{w} \tag{34}$$

The quantities $\bar{F_u}(\epsilon)$ (output of the network), $A_{\boldsymbol{w}}(\epsilon)$ (activation of a neuron) and $\mathrm{wID}_{A_{\boldsymbol{w}}}(\epsilon)$ (weighted ID of a function with an analytic form) can all be computed. Thus, $\mathrm{wID}_{\bar{F_u}}$ could be computed in a theoretical case (if u is known) or approximated in a practical case (by approximating u from a finite network). The effect of perturbations could then be studied with the Local ID Representation Theorem:

$$\frac{\bar{F_u}(\epsilon_1)}{\bar{F_u}(\epsilon_2)} \simeq \left(\frac{\epsilon_1}{\epsilon_2} \right)^{\mathrm{ID}_{\bar{F_u}}(0)} \tag{35}$$

Further research in this direction could help understanding the efficiency of adversarial perturbations on neural networks [17]. (Now, this is completely speculative, but:) the quotient ϵ_1/ϵ_2 can be rewritten using the LID of the dataset at x, using again the Local ID Representation Theorem. Let ϵ_1 be chosen such that it corresponds to the first nearest neighbour by expectation of x ($F_{\mathbf{X}}(\epsilon_1)$)

Expressive power We already know the following results:

- (finite) neural networks can approximate any continuous function [15]
- (finite) neural networks have a perfect finite sample expressivity [13]: they can learn any finite set of input/output, provided that the number of hidden neurons is large enough.

A natural question to ask is: what is the expressivity of this new model. We adopt the point of view of the ID for this question: the data are not a finite set of input/output pairs; rather, the training set is the result of a sampling process from an underlying distribution. There are a few possibilities from there:

- is the suggested model (both with one layer or more) capable of modeling any function $f : \mathbb{R}^n \to \mathbb{R}^m$? Can it do so with just a restricted class of functions (continuous functions, functions with a compact support, etc)?

– if this is not possible, can it approximate any function, or some class of function? In which sense would it approximate ($\mathbb{L}^1$? $\mathbb{L}^2$? $\mathbb{L}^\infty$?)?

7 Conclusion

Although most of the internship consisted in experimenting with neural networks, the most promising results are on the theoretical side, with the infinite model. Abstracting the discrete details of both the architecture and the database yield simple objects to work with. We believe that further research in this direction could lead to a better understanding of the learning process, to a better understanding of the adversarial setting and to a better intuition regarding design principles.

References

1. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278-2324, November 1998. http://yann.lecun.com/exdb/mnist/
2. Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. 2009. https://www.cs.toronto.edu/~kriz/cifar.html
3. Martin Thoma. The HASYv2 dataset. CoRR, abs/1701.08380, 2017. http://arxiv.org/abs/1701.08380
4. Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. In *International Journal of Computer Vision (IJCV)*, 115(3):211—252, 2015. https://arxiv.org/abs/1409.0575
5. David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. doi:10.1038/nature16961 https://storage.googleapis.com/deepmind-media/alphago/AlphaGoNaturePaper.pdf
6. Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car. Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car. CoRR, arXiv:1704.07911, 2017. https://arxiv.org/abs/1704.07911
7. M. E. Houle. Inlierness, outlierness, hubness and discriminability: an extreme-value-theoretic foundation. Technical Report NII-2015-002E, NII, Mar 2015. http://www.nii.ac.jp/TechReports/public_html/15-002E.pdf
8. Michael E. Houle. Dimensionality, Discriminability, Density & Distance Distributions. In ICDMW, pages 468–473, 2013.
9. L. Amsaleg, O. Chelly, T. Furon, S. Girard, M. E. Houle, K. Kawarabayashi, and M. Nett. Estimating local intrinsic dimensionality. In *KDD*, pages 29–38, 2015. http://mistis.inrialpes.fr/~girard/Fichiers/p29-amsaleg.pdf
10. D. R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *STOC*, pages 741–750, 2002.

11. M. E. Houle, H. Kashima, and M. Nett. Generalized expansion dimension. In *ICDMW*, pages 587–594, 2012.
12. Linnainmaa, Seppo. Taylor expansion of the accumulated rounding error. BIT Numerical Mathematics. 16 (2): 146160. doi:10.1007/bf01931367
13. Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht and Oriol Vinyals. Understanding deep learning requires rethinking generalization ICLR, 2017. http://arxiv.org/abs/1611.03530
14. A., Nielsen, Michael. Neural Networks and Deep Learning (2015-01-01). http://neuralnetworksanddeeplearning.com/chap2.html
15. Gybenko, G. Approximation by superposition of sigmoidal functions. Mathematics of Control, Signals and Systems, 2(4), 303-314, 1989. http://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Fall.2016/pdfs/Cybenko.pdf
16. Tamir Hazan and Tommi S. Jaakkola. Steps Toward Deep Kernel Methods from Infinite Neural Networks. CoRR, abs/1508.05133, 2015. http://arxiv.org/abs/1508.05133
17. Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow and Rob Fergus. Intriguing properties of neural networks ICLR, abs/1312.6199, 2014. http://arxiv.org/abs/1312.6199
18. Alexey Kurakin, Ian J. Goodfellow and Samy Bengio Adversarial examples in the physical world CoRR, abs/1607.02533 2016. http://arxiv.org/abs/1607.02533