Elementary response of deep neural networks

Yann Traonmilin[1]and A. Newson[2]

[1]CNRS,Univ. Bordeaux, Bordeaux INP, IMB, UMR 5251,F-33400 Talence, France.

[2]Telecom Paris, Palaiseau, France.

The general goal of this work is to study the response of neural networks on elementary signals, deduce limits about theirs possibilities when the training database lack information and highlight the role of regularization.

1 Notations

- Vector of data $x, y, z \in \mathbb{R}^n$
- Dimensions : n dimension of data vectors, N number of examples in the database, d dimensionality of the latent space
- Norms on $\mathbb{R}^n$, ℓ^2-norm $\|\cdot\|_2$
- Training database $X = (x_i)_{i=1,N}$
- Functions defining a neural network f, g, h ...
- Weights w, a neural network defined by its weights w : f_w
- Code $z \in \mathbb{R}^d$ (for autoencoders)
- Loss function $\mathcal{L}_f(X)$, for network f
- Non-linearity function σ

2 Lowest complexity autoencoders

The goal of an autoencoder is to project data to and from the *latent* space $\mathbb{R}^d$, which is supposed to be lower-dimensional than the data space $\mathbb{R}^n$, and thus represent the data much better, and in a more compact fashion. For example, the latent space could represent the *size* of an object. This would clearly be more efficient than representing images of an object with different sizes, which would take up a lot of useless space.

In an autoencoder, however, we must fix the dimension d of the latent space. We would like this to be as small as possible, while representing the data correctly. Representing the data correctly simply means that the autoencoder loss is zero. In this manner, we define an *admissable autoencoder*:

Definition 2.1 (Admissible autoencoder). *We say that $f = f_D \circ f_E$ is an admissible autoencoder for some data X if $L_f(X) = 0$, where $f_E : \mathbb{R}^n \rightarrow \mathbb{R}^d$ and $f_E : \mathbb{R}^d \rightarrow \mathbb{R}^n$ are the encoder and decoder, respectively.*

Note that this is only interesting when $d < n$, otherwise we can simply set $f_D = f_E = Id$. Our goal will be to create admissible autoencoders with smallest latent space dimensionality possible, when applied to elementary signals. The first type of signals we will look at will be *step functions*. This is defined in the following manner.

Definition 2.2 (Step function). *We say that a signal $x \in \mathbb{R}^n$ is a step function, parameterised by θ if:*

$$x(j) = 1 \text{ if } j \leq \theta;\ 0 \text{ otherwise.} \tag{1}$$

In this context, we wish to study the following questions:

- Robustness/generalization of networks trained on such data:
  - What happens when some data is missing ?
  - Do sub-optimal networks exist, which work well on incomplete databases, but not on the complete database ?
  - Can we determine generalisation error bounds, with respect to the amount of missing data ?
- Linearity of the latent space:
  - Do encoders exist which encode the signals in a linear fashion, that is to say that there is a linear relationship between the code z and the true parameter θ ?
  - Do decoders exist which decoder the signals in a linear fashion, that is to say when we move by an amount δ in the latent space, there is a change of δ in the parameter θ of the output image ?
  - Can we design architectures/regularisations which encourage linear behaviour ?

3 A simple example to start with: a two layer MLP autoencoder

We first show that our step functions can be fully encoded by a 2-layer multi-layer perceptron. In the case of the step functions, we set $d = 1$ (only one parameter is needed to describe them).

3.1 Encoder

It is obvious that the following encoder $f_E : x \to \sum_{i=1,n} x(i)$ (the sum of the signal's values) gives the step function's position θ. In the following, we fix this encoder.

3.2 Decoder

Let $z_0 \in \mathbb{R}^n$ defined by $z_0 = (1/i)_{i=1,n}$. Let $\sigma : \mathbb{R} \to \mathbb{R}_+$ defined by $\sigma(u) = 0$ for $u < 1$ and $\sigma(u) = 1$ otherwise. This is a "hard" sigmoid. We consider the decoder $f_D : z \to \sigma(zz_0)$.

We have the following lemma

Lemma 3.1. *Let Consider the* 2*-layers network* $f^* := f_D \circ f_E$ *then for all step functions* x *we have*

$$f(x) = x$$

Proof. Let x be a step function. Let θ be a scalar such that $x(i) = 1$ for $i \in [1, \theta]$, $x(i) = 0$ otherwise. We have $f_E(x) = \theta$ (see previous Subsection 3.1). This gives

$$y = f_D \circ f_E(x) = f_D(\theta) = \sigma(\theta z_0) \tag{2}$$

Let $i \leq \theta$, this implies $1 \leq \theta z_0(i)$ and $y(i) = 1$. Let $i > \theta$, this implies $1 > \theta z_0(i)$ and $y(i) = 0$. Thus, we have that for all i, $y(i) = x(i)$. $\square$

This provides a starting point for this work. The first task will be to implement such a network (two-layer MLP), train it, and study the decoder's weights, to verify that the training has indeed found the correct weights. From there, the other questions (generalisation, linearity) can be studied.