The Backpropagation Algorithm and its Approximations

Samuel Lippl

April 10, 2018

Open Problems: Overview

1 Neural Networks

Neural networks are very general notions that encompass many different varieties. There have been attempts to make certain restrictions on neural networks that are derived from their purpose, see, for instance, [1]. I would argue that these restrictions make the definition more complicated to prove but not more powerful to use. Therefore, my suggestion for a definition largely orients itself by [3] although I have slightly tweaked the notation where I deemed it sensible.

1.1 Definition (Neural Networks). Consider a directed graph $(\mathcal{V},\mathcal{E})$ where $\mathcal{E}\subseteq\mathcal{V}\times\mathcal{V}$. The elements of $\mathcal{V}$ are called *Processing Units* (PU). Every PU has to be connected to some other PU. Among the PUs are the *Input Units* (IU) $\mathcal{I}\subseteq\mathcal{V}$ and the *Output Units* (OU) $\mathcal{O}\subseteq\mathcal{V}, \mathcal{I}\cap\mathcal{O}=\emptyset$ where each IU is connected to some OU by a directed graph[1]. Each PU v is associated with a differentiable function $f_v:\mathbb{R}\to\mathbb{R}$ where $(f_v)_{v\in\mathcal{V}}=:\mathcal{F}$ and every edge $(v,w)\in\mathcal{E}$ is associated with a weight $\xi_w^v\in\mathbb{R}$. We write $\mathcal{E}_w:=\{v\in\mathcal{V}:(v,w)\in\mathcal{E}\}$ and $\mathcal{E}^v:=\{w\in\mathcal{V}:(v,w)\in\mathcal{E}\}$. Now, every PU $v\in\mathcal{V}$ has an input $i(v)\in\mathbb{R}$ and an output $o(v)=f_v(i(v))\in\mathbb{R}$. We call a *network state* $S=(i(v),o(v))_{v\in\mathcal{V}}$ *compatible* if and only if for all $w\in\mathcal{V}$

$$i(w)=\sum_{v\in\mathcal{E}_w}\xi_w^v o(v) \tag{1}$$

A *network function* N_ξ ($\xi\in\mathbb{R}^{\mathcal{E}}$ denotes the weights ξ_w^v) associates to each input $i\in\mathbb{R}^{\mathcal{I}}$ a compatible network state $N(i)$ where we denote the outputs of the network state by $O_\xi(i)$.
Finally, a neural network can be trained where we consider a finite number of examples $(i,o)_1,\dots,(i,o)_k$ and update the weights according to a training function $\mathcal{T}:\mathbb{R}^{\mathcal{E}}\times\left(\mathbb{R}^{\mathcal{I}}\times\mathbb{R}^{\mathcal{O}}\right)^{<\mathbb{N}}\to\mathbb{R}^{\mathcal{E}}$ such that the new weights are $\mathcal{T}(\xi,(i,o))$ where for any set M, $M^{\mathbb{N}}=\bigcup_{n\in\mathbb{N}_0}M^n$ and $M^0=\emptyset$.
We call $(\mathcal{V},\mathcal{E},\mathcal{F},\xi,N,\mathcal{T})$ a *neural network*.

[1] The notion of Input und Output Units is intuitively clear. For now, I will not restrict my definition and I can even imagine situations where $\mathcal{I}\cap\mathcal{O}\neq\emptyset$ would be possible.

It will often be helpful to associate $\mathcal{E}$ with a relation $\rightarrow$ where $v \rightarrow w \equiv (v, w) \in \mathcal{E} \equiv w \leftarrow v$.

1.2 Definition. A neural network is called *feed-forward* if and only if $\rightarrow$ is acyclical.

More generalizations would be possible; for instance, we may consider general aggregation functions $f\left(o(v)\right)_{v \in \mathcal{E}_w}$ instead of the additive structure above. If that becomes necessary, I will consider such a generalization. For now, however, we are only concerned with feed-forward neural networks.

1.1 Important characteristics

Computers mostly process data sequentially where each unit transforms an input into an output that is given to the next unit. Here, we do not rely on parallel processing.[2]

1 Open Problem (Parallelity in Computers and Neuronal Networks). Is parallel processing in computers and neuronal networks comparable?

If, for some reason, one processing unit does not work properly, the network will face the possibility of global failure. For artificial computers, this is acceptable because we can find other workarounds. I can imagine that processing does not directly depend on the physical parts and there is an error correction on another level. Additionally, an appropriate reaction to unreliable units is improving the units, not only improving error compensation. With respect to our brain, we do not have this luxury:

- Cells die eventually, of natural causes, by attack or by external impact.
- Cells are vastly more complex than artificial processing units — arguably even more complex than the most sophisticated computers. While much of this complexity aims to make the cell more reliable, there are also many procedures where something can go wrong.
- Cells may fire spontaneously.

Most importantly, while we can simply modify production of processing units, there is no conscious process behind neurons that could make these global changes to our system. An important characteristic of neural networks, with respect to their model of our brains is therefore a stochastic error rate that is compensated by the parallel pathways. This compensation is illustrated in [3, p. 51]. If we consider artificial neural networks such an error prevention is not as important. However, another feature of parallelity benefits both neural and neuronal networks: by having more than one transformation of the same processing units, a transformation of the input space allows a different perspective on the problem which may substantially simplify a problem. Take, for instance, our visual pathways: the input to our retina is transformed to different characteristics like orientation or relative brightness (see [2, pp. 257-276]) and artifical visual recognition systems have made use of similar strategies [3, pp. 70-73]. This example demonstrates that there are useful analogies between artificial and neuronal processing. It is therefore important to recognize where problems in domain have corresponding problems in the other domain and where solutions to such problems in one domain would be feasible in the other. For instance, parallelity as a feature space transformation is a useful characteristic in an artifical and a biological framework while parallelity as an error compensation is not as important *in silica*. The

[2] Modern computers, to an extent, work with parallel processing. While the extent would be interesting to compare, I will elaborate on the intent below.

fact that parallelity is such a crucial aspect of neural networks has led [1] to claim parallelity as a necessary feature of neural networks. In their definition of neural networks, Guresen and Kayakutlu require that "at least two of the multiple [Processing Elements are] connected in parallel" [1, p. 428] because, they argue, "structures [...] with one or more [Processing Elements] connected serially cannot be referred as [a neural network] because it will lose the power of parallel computing and starts to act more like existing computers than a brain" [1, p. 428]. In principle, I agree with their point. I would argue, however, that it is not helpful to include such a criterion in the definition. After all, if there is one parallel pathway in a massive serial network, would this correspond to a neural network according to their notion. It seems to me that this definition therefore opens an issue that cannot be resolved, analogous to trying to define "many grains of sand" – without doubt, one grain of sand is necessary to have many but it is not sufficient and we can probably not find a hard definition of such a vague statement. We would be better advised to simply define the characteristic "many" refers to, i. e. the *number* of grains of sand. Then, we can use more rigorous notions instead of many grains of sand. Comparably, parallel processing such that it is advantageous cannot be covered by such a broad definition – not only does parallelity exist on a spectrum, it also strongly depends on the context. I would therefore suggest that an attempt to find a way to talk about parallelity may be more fruitful than Guresen and Kayakutlu's hard borders. I have not found the definition of a comparable *degree of parallelity* and Guresen and Kayakutlu themselves also do not elaborate on what they mean by parallel connections.

2 Open Problem (Degree of parallelity). How may we define the *degree of parallelity* of a neural network?

1.2 Backpropagation Algorithm

The backpropagation algorithm is a particularly popular training function $\mathcal{T}$ that is defined on all feed-forward networks.

3 Open Problem (Backpropagation on recurrent networks). How may we define backpropagation for a given network function on a recurrent network?

We define a differentiable loss function L for a given set of training examples

$$L : \mathbb{R}^{\mathcal{E}} \times \left(\mathbb{R}^{\mathcal{I}} \times \mathbb{R}^{\mathcal{O}}\right)^{<\mathbb{N}} \to \mathbb{R}_0^+, (\xi, (i,o)_1, \ldots, (i,o)_k) \mapsto L(\xi|i;o) \tag{2}$$

that compares the network's prediction with the actual output to compute a loss. Log-likelihood functions are popular loss functions, most notably the quadratic loss function

$$L(\xi|i;o) = \frac{1}{2}\sum_{j=1}^{k} \|O_\xi(i_j) - o_j\|_2^2 \tag{3}$$

The computation of the network weights is now an optimization problem:

$$\xi = \arg\min_{\xi \in \mathbb{R}^{\mathcal{E}}} L(\xi|i;o) \tag{4}$$

where i and o are given.

As both L, all PU functions f_v and the aggregation (1) are differentiable, we can compute the loss

gradient

$$\nabla L = \left(\frac{\partial L}{\partial \xi}\right)_{\xi \in \mathbb{R}^{\mathcal{E}}} \tag{5}$$

This allows us to determine the minimum by gradient descent with the iterative update

$$\xi^{(t)} = \xi^{(t-1)} + \alpha \nabla L(\xi | i; o) \tag{6}$$

The backpropagation algorithm is a method of gradient descent that allows us to determine the gradient. Before I go on to explain how the algorithm works, I will note that it does not prevent us from stopping in local minima, even in the case of a convex loss function.

4 Open Problem (Convex processing functions). What if $\mathcal{F}$ is convex? This is normally not the case, as we mostly map the input to $(0, 1)$, for example with the sigmoid function $f_c(x) = \frac{1}{1+e^{-cx}}$, but it would still be interesting to consider. As far as I see, with $\mathcal{F}$ and L convex in $N_\xi(i)$, L should be convex in i.

As $\nabla L(i; o|\xi) = \sum_{j=1}^{k} \nabla L(i_j; o_j|\xi)$ we can consider, without loss of generality, one training example and a univariate output o. The framework I present has been inspired by [3] although I would argue that my approach is a bit more direct.
As $\frac{\partial L}{\partial \xi}(\xi|i; o) = \frac{\partial L}{\partial O_\xi(i)}(\xi|i; o) \cdot \frac{\partial O_\xi(i)}{\partial \xi}$, it suffices to compute the gradient ∇N_ξ. In order to achieve that, our first goal is to compute the gradients $\frac{\partial O_\xi}{\partial i(v)}, \frac{\partial N_\xi}{\partial o(v)}$.

First step: Feed-forward computation In the first step, we extend the neural network by not only computing i and o but also new functions i' and o' where

$$o'(v) := \frac{\partial o(v)}{\partial i(v)} = f'\left(i(v)\right) \quad i'(w) := \left(\frac{\partial i(w)}{\partial o(v)}\right)_{v \in \mathcal{E}_w} = (\xi_w^v)_{v \in \mathcal{E}_w} \tag{7}$$

Note that these functions are only locally stored but only i and o are used as input in the following PUs.

Second step: Backpropagation We now recursively compute $\frac{\partial O_\xi}{\partial i(v)}, \frac{\partial O_\xi}{\partial o(v)}$. Consider a PU v and suppose that for all $w \in \mathcal{E}^v$, we have already computed $\frac{\partial O_\xi(i)}{\partial i(w)}$. Then,

$$\frac{\partial O_\xi(i)}{\partial o(v)} = \frac{\partial O_\xi(i)}{\partial \left(i(w)\right)_{w \in \mathcal{E}^v}} \cdot \frac{\partial \left(i(w)\right)_{w \in \mathcal{E}^v}}{\partial o(v)} = \sum_{w \in \mathcal{E}^v} \xi_w^v \frac{\partial O_\xi(i)}{\partial i(w)}$$

On the other hand,

$$\frac{\partial O_\xi(i)}{\partial i(v)} = \frac{\partial O_\xi(i)}{\partial o(v)} o'(v)$$

which, as we have a feed-forward network, allows us to iteratively compute the gradients in question. Finally, we note that for any $(v, w) \in \mathcal{E}$

$$\frac{\partial i(w)}{\partial \xi_w^v} = o(v) \tag{8}$$

and therefore

$$\frac{\partial O_\xi(i)}{\partial \xi_w^v} = \frac{\partial O_\xi(i)}{\partial i(w)} o(v) \tag{9}$$

which yields the necessary gradient

$$\nabla L(\xi|i; o) = \frac{\partial L}{\partial O_\xi(i)}(\xi|i; o) \frac{\partial O_\xi(i)}{\partial \xi_w^v} \tag{10}$$

and allows us to update ξ according to (6).

2 Biological networks

The biological plausibility of the backpropagation algorithm is generally being doubte

References

[1] Erkam Guresen and Gulgun Kayakutlu. "Definition of artificial neural networks with comparison to other networks". In: *Procedia Computer Science* 3 (2011), pp. 426–433. DOI: 10.1016/j.procs.2010.12.071.

[2] Dale Purves et al. *Neuroscience*. 5th ed. Sinauer Associates, Inc., 2012.

[3] Raúl Rojas. *Neural Networks*. Berlin: Springer-Verlag, 1996.