Coupled Local Supervised Learning Notes

Monica Conte

March 2023

In this document I review the theory on *Coupled Local Supervised Learning* (CLSL) as explained in [1] and applied in [2]. This method constitutes a novel approach in the design of materials that possess the ability of learning.

Contents

1 Intro

As opposed to materials designed with particular sets of functions, CLSL approach allows to create systems that don't possess explicit information about the desired functionality but physically adapts to applied external forces and develop the ability to perform tasks. This method belongs to a class of strategies based on learning, where systems can adjust or be adjusted microscopically in response to training set to develop desired functionalities. Until recently, such idea was primarily applied in the context of non-physical networks, like artificial neural networks (ANN).

There are two types of learning.

- **Global Learning** involves the minimization of a global function and the subsequent *tuning* of the leaning degrees of freedom. **An external intervention at microscopic level is required.**
- In **Local Lerning** the evolution is **autonomous and requires no external designer** for the evaluation of the current state and subsequent modification. This is particularly useful in physical systems, since their microscopical elements do not perform computations and do not encode a priori information on the desired functionality.

CLSL implements local learning on physical netwroks with the aim of training it to achieve desired responses on *target nodes* as a response to external constraints applied to *source nodes*. The method is inspired by *contrastive learning* and by the strategy of *equilibrium propagation*. In the paper, it is applied to two types of physical systems, a flow network and an elastic network. Experimental issues are finally taken into consideration and the theory of the method is modified accordingly.

2 ANN framework

The building blocks of ANN are *neurons*. They are elements that take inputs and produce outputs by performing some mathematical operations on the inputs. Fig. 1 shows a neuron with two inputs x_1 and x_2. The output depends on the inputs, the wieght and the bias. Training a neuron means modifying the wieghts in such a way that it outputs a desired value.

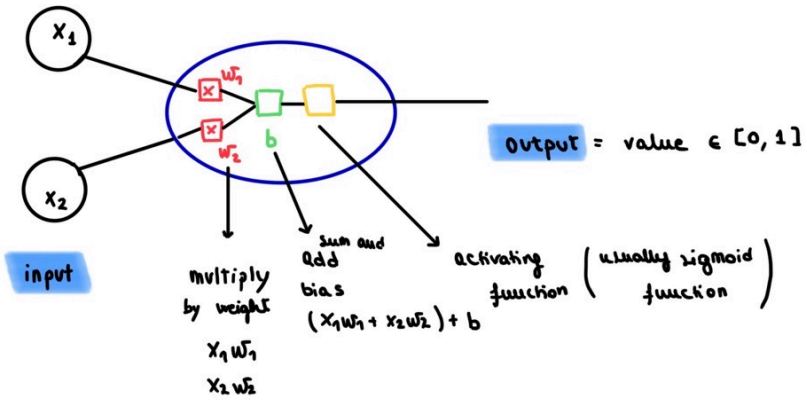

Figure 1: Two inputs (x_1 and x_2) neuron. Blue circle contains the operations perfomred in a neuron: **multiply inputs to wieghts** associated to the links, **sum inputs and add a bias** b, **apply an activating function**. The outcome is a value belonging to $[0, 1]$.

Neural networks are structures made of many neurons. These neurons are organized in layers, an example in Fig. 2. There are three types of layers: input layer, hidden layers and output layers. Input layers read the inputs, operations are perfomed in the hidden layer and output is generated in ouput layer. Hence, a neural network can be seen as a set of *nodes* (the neurons) connected by *edges* (the links with wieghts). Given an input, each neuron in each layer performs coputations and gives output that are inputs for the follwoing neuron. This process of proceeding from the inputs to the outputs is called **feedforward**.

A neural network needs *training* in order to be able to perform a task. The standard way to train a network is by using a *loss function*, which quantifies how good the output of the network is compared to the desired output. Often, the mean-squared error (MSE) is used

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_{\text{desired}} - y_{\text{predicted}})^2$$

where the sum is take over the training samples and y indicates the value of the output in a specific neuron. From this simple formula we can already uderstand that a **smaller error implies a better prediction**.

The trainig method proceeds in changing the weights in the network in the direction of the minimization of the loss funtion. It is therefore useful to denote as $L(\mathbf{w})$ the loss function as a function of the set of wieghts $\mathbf{w}$. In oerder to understand how the loss function changes when the weight are modified, we consdier the derivative of the loss function with respect to the weights. Let us take for example the first wieght w_1, the derivative $\partial L/\partial w_1$ is found by

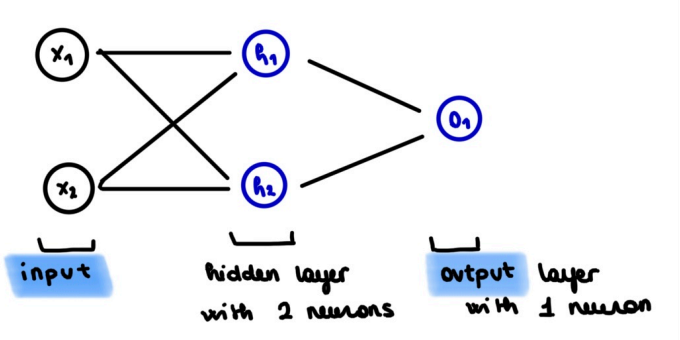

Figure 2: Example of a neural network: a structure made of neurons organized in layers.

applying the chain rule

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{\text{predicted}}} \frac{\partial y_{\text{predicted}}}{\partial w_1}$$

which is applied again to the second therm on the RHS following a path in the neural network which traces back to the location of the weight w_1. This procedure is called **backpropagation** since it acconts in propagating back the error in the network and identify the biggest sources of error.

Once the derivative of the loss function is calculated, one needst o find a way of optimizing the netwrok accordingly. A possible way of minimizing the loss function is by using **gradient descent** method. The weights are updated as follows:

$$w \ \ \rightarrow w - \eta \frac{\partial L}{\partial w}$$

and the logic behind is that it changes the wiehgt towards the direction of descent. In fact, a positive derivative means that a positive increase of the wieght increases the loss function, from which a negative sign in front of the term. Viceversa for the negative sign.

The training proceeds in steps and in each step the weights are optimized to decrease the error. There are many training samples in a dataset and a common training procedure randomly selects, at each step, one of the samples in the set and performs gradient descent. This procedure is called **stochastic gradient descent**.

3 CLSL framework

We now consider a physical neural network (PNN). It is, again, a strucutre made of nodes and edges but, as opposed to ANN, they are not organized in layers and in principle there is no fixed pattern. However, just like an ANN, we divide the nodes in *source nodes*, *hidden nodes* and *target nodes*. A generic PNN is shown in Fig. 3. These elements assume values of physical qunatities and can be realized in real life. For example, in electric netwroks, the nodes assume vlaue of potential and the edges of resistances. The training goal here is to train the netwrok to give

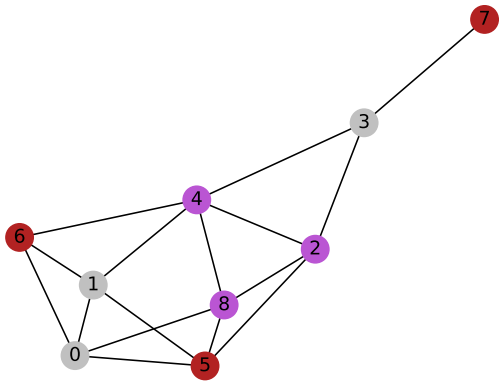

Figure 3: Example of a generic PNN. $5, 6, 7$ red dots denote input nodes, $0, 1, 3$ gray nodes denote hidden nodes and $2, 4, 8$ purple nodes denote target nodes.

desired outputs in the output nodes given cerain inputs at the source nodes, just like in the ANN.

In a PNN, the values on the nodes follow from the minimization of an *energy function*, as opposed to the feedforward procedure of ANN. For example, in electric networks, the function minmized is the power dissipated by the resistances. This, in turn, corresponds to satisfying Kichoff's law on current in the circuit of nodes and resistances (see Appendix (?)), i.e. conservation of charge. In flow networks, minimization of power means conservation of flow (?).

3.1 General formulation of the method

Let us consdier a network with nodes indicized by μ and edges indicized by j. The *physical degrees of freedom* (pof) belong to the set of variables $\{x_\mu\}$. The *learning degrees of freedom* (lof) belong to the set w_j and play the roles of wieghts connecting the nodes. Given an initial condition $x_\mu(t = 0)$ (part of the x_μ in the set given) and a set of weights, the physical degrees of freedom evolve to a steady state tha minimizes a *physical cost function* $E(x_\mu; w_j)$. Note that both $\{x_\mu\}$ and E depend on $\{w_j\}$.

Inspired by *contrastive Hebbian learning* (CHL), one considers two states of the network. The *free state* x_μ^F in which the source nodes x_S are constrained while the target x_T^F and hidden x_H^F

equilibrate to the steady state. The *clamped state* x_μ^C in which the source x_S and the target x_T^C nodes are constrained while only the hidden nodes x_H^C equilibrate to the steady state. In the clampd state, the target nodes are nudged by an *external supervisor* towards the desired X_T values of outputs

$$x_T^C = x_T^F + \eta \left[X_T - x_T^F \right]$$

with $\eta \ll 1$. In contrast with CHL, the nudge parameter is small, which fixes the problem of free and clamped state reaching two different local minima of the energy function.

The authors propose the following updating rule for the lof:

$$\dot{\mathbf{w}} = \alpha \eta^{-1} \partial_{\mathbf{w}} \left\{ E(x_S, x_H^F, x_T^F) - E(x_S, x_H^C, x_T^C) \right\}$$

- α is a scalar **learning rate**
- η **is the nudging aplitude.**

The motivation behind this rule comes from the fact that we consdier a **physical** network, where generally one can partition the total energy E as a sum over edegs $E = \sum_j E_j(x_\mu(j); w_j)$ of energies E_j that only depend on the pof on the nodes that connect the edge j. Therefore, the learning rule is **local**. But does this rule effectively trains the network? The autors show (Appendix A in [1]) that this updating scheme minimizes an *effective cost function* that is not identical to the standard cost function but shares important features with it. They argue that minimizing the effective function mimics the minimization of a standard cost function. From how well the networks are trained, they observe that this affirmation is plausible. The CLSL approach share many similarities with Equilibrium Propagation(EP) [] and the learning rule has the same form. However, in EP the learning rule minimizes an actual cost function.

Summarizing, the learning approach involves the minimization of an energy function and the implementation of the proposed updating rule. The former is motivated by physical plausability and the latter by inspiration from EP. A comparison with minimization of an actual cost function is needed. They say EB is a type of contrastive learning.

3.2 Application to Electrical Netwroks

The framework above can be applied in the context of electrical networks [2], in which nodes are connected by *variable resistors*. When input voltages are applied at the source nodes V_S, the voltages at the output V_O and hidden V_H nodes are determined such that the total power dissipated by the resistances $\mathcal{P}$ is minimized

$$\mathcal{P} = \frac{1}{2} \sum_j \frac{\Delta V_j^2}{R_j}$$

where j runs over the resistances R_j and ΔV_j is the potential difference between the nodes that the resistance links.

We define the two types of states, as above. In the free state the source nodes V_S^F are constrained, while V_H^F and V_T^F are determined by minimizing $\mathcal{P}$. In the clamped state, the source nodes V_C^F are fixed to the source values, the target nodes are clamped

$$V_T^C = V_C^F + \eta \left(V^D - V_C^F \right)$$

towards the desired output voltages V^D, while the hidden nodes are left to minimize $\mathcal{P}$. The parameter $0 < \eta \leq 1$ quanitifies the importance of the nudge.

The final goal is to deliver the desired output V^D to the target nodes. To do so, the network gets optimized by updating the values of the resistances following a certain *learning rule*. The adaptation of the learning rule of CLSL in the context of electrical circuits would lead to the following **updating rule for resistances**

$$\dot{R}_j = \frac{\alpha}{2\eta R_j^2}\left[\left(\Delta V_j^C\right)^2 - \left(\Delta V_j^F\right)^2\right] \tag{1}$$

where R_j is the selected resistance to update, ΔV_j^C is the potential difference between the nodes that it connects in the clamped state and ΔV_j^F the difference in the free state. In the paper [2] they define

$$\gamma = \frac{\alpha}{2\eta}$$

as the learning rate.

In the experiment, they have discrete resistors and the rule above gets approximated with the following

$$\dot{R}_j = \begin{cases} +\delta R & \text{if } |\Delta V_j^C| > |\Delta V_j^F| \\ -\delta R & \text{otherwise} \end{cases} \tag{2}$$

which consists in taking the sign of Eq. (1) and multipy it by the value δR.

We can also reformulate the theory in terms of conductances $g_j = 1/R_j$, the updating rule of wihch can be derived from the power dissipated

$$\mathcal{P} = \frac{1}{2}\sum_j \frac{\Delta V_j^2}{R_j} = \frac{1}{2}\sum_j g_j \Delta V_j^2.$$

In terms of the learning rate γ, the **updating rule for conductances** read

$$\dot{g}_j = \gamma\left[\left(\Delta V_j^F\right)^2 - \left(\Delta V_j^C\right)^2\right] \tag{3}$$

4 Simple Start: Voltage Divider

One of the simplest networks we can consider is a simple votage divider, shown in Fiugre (). It has three nodes, two sources and one ouput, with potential V_{S1}, V_O and V_{S2}. The goal in training this network is to deliver a voltage of $V_O = 4V$ when imposing voltages of $V_{S1} = 5V$ and $V_{S2} = 0V$. At the beginning, the resistances are initiated to a value of $R = 50k\Omega$. In this first section, we are trying to reproduce the results shown in the paper [2], we are thus adopting the same setting and choise of parameters

References

[1] Menachem Stern et al. "Supervised Learning in Physical Networks: From Machine Learning to Learning Machines". In: *Physical Review X* 11.2 (May 2021). ISSN: 2160-3308. DOI: 10.1103/physrevx.11.021045. URL: http://dx.doi.org/10.1103/PhysRevX.11.021045.

[2] Sam Dillavou et al. "Demonstration of Decentralized Physics-Driven Learning". In: *Physical Review Applied* 18.1 (July 2022). ISSN: 2331-7019. DOI: 10.1103/physrevapplied.18.014040. URL: http://dx.doi.org/10.1103/PhysRevApplied.18.014040.