DISMA Dipartimento di Scienze Matematiche G. L. Lagrange

ECCELLENZA 2018 • 2022

A mean field control approach to deep learning

Adriano FESTA
adriano.festa@polito.it
joint work with A. Leone

Two days online workshop on mean field games

POLITECNICO DI TORINO

Outline

Machine Learning

Artificial Intelligence, Machine Learning and Deep Learning

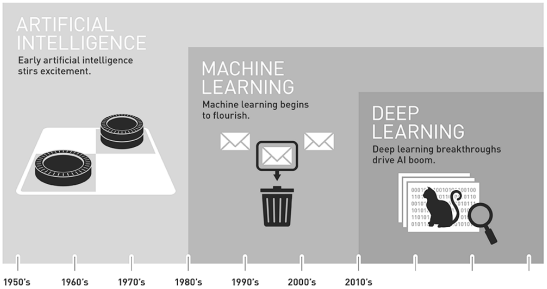


Figure: Adaptation from
https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

We consider supervised learning. We are given a training dataset $\left\{x^i, c^i\right\}_{i=1}^m$, with $x^i \in \mathcal{X} \subset \mathbb{R}^n$ the input variables and $c^i \in \mathcal{C}$ the corresponding outputs, with $\mathcal{C}$ the set of labels. The goal is to learn the mapping function $F : \mathcal{X} \to \mathcal{C}$, with $c^i = F\left(x^i\right)$, well enough to predict the output for any new input data.

Deep Neural Networks

A fully connected, feedforward deep neural network (DNN) is a multilayer stack of simple units, called artificial neurons.
Each neuron computes a weighted sum of the inputs coming from every neuron of the previous layer, computes a non-linear activation function of the result and produces a real value, which is passed to every neuron at the next layer.

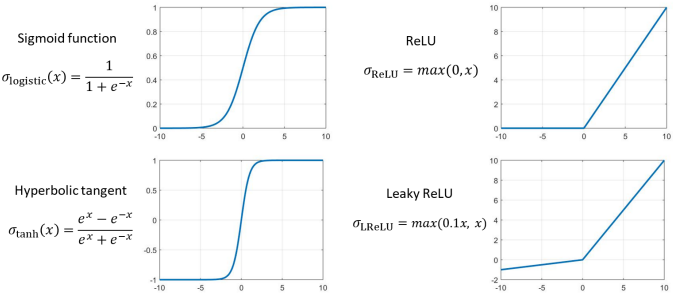
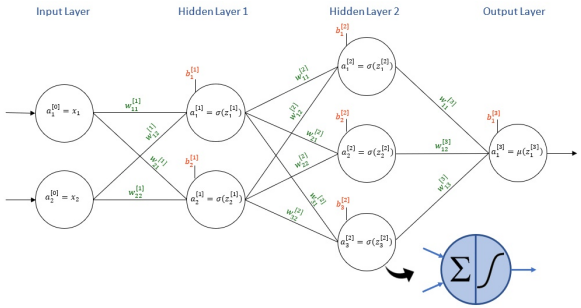


Figure: Commonly used activation functions.Their main purpose is to determine whether a neuron should be activated ("fired") or not, based on whether the neuron's input is large enough, thus mimicking the the behavior of a neuron in the brain.

Feedforward propagation



$$a^{[0]} = x \in \mathbb{R}^{n_0}, \qquad z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}, \qquad a^{[l]} = \sigma\left(z^{[l]}\right) \in \mathbb{R}^{n_l},$$

with $l = 1, 2, \ldots, L$, $a_j^{[l]}$ output from neuron j at layer l, σ activation function acting component-wise, $a_j^{[l]}$ weighted input for neuron j at layer l, $W^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$ weights and $b^{[l]} \in \mathbb{R}^{n_l}$ biases at layer l.

$a^{[L]} = \mu\left(z^{[L]}\right) \in \mathbb{R}^{n_L}$ output from the network, with μ hypothesis function.

A classic approach of Deep Learning

Training the process/ tuning the parameters

The output from the network is given by

$$a^{[L]} = \phi_L \circ \phi_{L-1} \circ \cdots \circ \phi_l \circ \ldots \phi_1(a^{[0]}) \equiv g(x), \tag{1}$$

with $\phi_l(u) = \sigma\left(W^l u + b^l\right)$ for $l = 1 \ldots L-1$ and $\phi_L(u) = \mu\left(W^L u + b^L\right)$.
Therefore $g : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ is given in terms of weights and biases. We denote these parameters by θ.

GOAL: Produce a function that can approximate in the best way the data-label relation and that can produce accurate prediction.

We introduce a cost function, e.g. a mean squared error function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} C_{x^i} = \frac{1}{m}\sum_{i=1}^{m} \frac{1}{2}\|c^i - g(x^i)\|_2^2. \tag{2}$$

Choosing θ in a way that minimizes the cost function is generally called training the network.

Stochastic gradient descent

Gradient descent method: given an initial solution chosen arbitrarily, repeatedly apply the following update rule until a stopping criterion has been met:

$$\theta \to \theta - \alpha \nabla J(\theta), \tag{3}$$

with α the learning rate.

Problem: large number of parameters and training data.

Stochastic gradient descent method: replace the actual gradient by an estimate calculated from a randomly selected subset of the data.

Example: mini-batch gradient descent. Let $s \ll m$, then

1. Choose s integers, $\{k_1, k_2, \dots, k_s\}$, uniformly at random from $\{1, 2, \dots, m\}$.
2. Update

$$\theta = \theta - \alpha \frac{1}{s} \sum_{i=1}^{s} \nabla C_{x^{k_i}}(\theta). \tag{4}$$

In this iteration, the set $\left\{ x^{k_i} \right\}_{i=1}^{s}$ is known as a mini-batch.

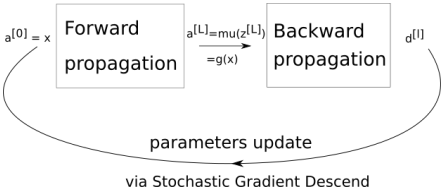
Backpropagation algorithm

Let $\delta^{[l]} = \nabla_{z^{[l]}} C \in \mathbb{R}^{n_l}$, i.e. $\delta_j^{[l]} = \partial C / \partial z_j^{[l]}$, for $1 \leq j \leq n_l$ and $1 \leq l \leq L$.
The quantity $\delta_j^{[l]}$ is often called the error in the j-th neuron at layer l.
The rate of change of the cost w.r.t. any weight and bias in the network is given by the following representation formulas:

$$
\begin{aligned}
&\delta^{[L]} = \mu'\left(z^{[L]}\right) \odot \nabla_{a^{[L]}} C, \quad \text{with } \odot \text{ the component-wise product,} \\
&\delta^{[l]} = \sigma'\left(z^{[l]}\right) \odot \left(W^{[l+1]}\right)^T \delta^{[l+1]} \qquad \text{for} \quad 1 \leq l \leq L-1. \\
&\frac{\partial C}{\partial b_j^{[l]}} = \delta_j^{[l]} \qquad \text{for} \quad 1 \leq l \leq L-1. \\
&\frac{\partial C}{\partial w_{jk}^{[l]}} = \delta_j^{[l]} a_k^{[l-1]} \qquad \text{for} \quad 1 \leq l \leq L-1.
\end{aligned}
\tag{5}
$$

Training of a feedforward deep neural network

- Initialization: Fix *Niter*
- For $it = 1, \ldots,$ *Niter*
  - (Forward propagation)
    - Given $x^{\bar{i}}$, $a^{[0]} = x^{\bar{i}}$
    - For $l = 1 \ldots, L-1$
      $z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}, \quad a^{[l]} = \sigma\left(z^{[l]}\right)$
      $z^{[L]} = W^{[L]} a^{[L-1]} + b^{[L]}, \; a^{[L]} = \mu\left(z^{[L]}\right)$
      Compute cost function C
  - (Backward propagation)
    - $\delta^{[L]} = \mu'\left(z^{[L]}\right) \odot \nabla_{a^{[L]}} C$
    - For $l = L-1, \ldots, 1$
      $\delta^{[l]} = \sigma'\left(z^{[l]}\right) \odot \left(W^{[l+1]}\right)^T \delta^{[l+1]}$
  - (Update the parameters):
    - For $l = L-1, \ldots, 1$
      $W^{[l]} \to W^{[l]} - \alpha \delta^{[l]} a^{[l-1]\,T}$
      $b^{[l]} \to b^{[l]} - \alpha \delta^{[l]}$



Deep learning as a discrete optimal control problem

Forward propagation in deep Residual Networks (ResNet):

$$y_i^{[0]} = x_i, \qquad y_i^{[j+1]} = y_i^{[j]} + f\left(y_i^{[j]}, \theta^{[j]}\right), \qquad j = 0, \dots, N-1, \tag{6}$$

where f is the so called residual function: $f\left(y_i^{[j]}, \theta^{[j]}\right) = \sigma\left(W^{[j]} y_i^{[j]} + b^{[j]}\right)$.
Let us rewrite $y_i^{[j+1]} = y_i^{[j]} + hf\left(y_i^{[j+1]}, \theta^{[j]}\right)$, with $h > 0$.
This is the discretization on $[0, T]$ of the initial value problem

$$\dot{y}_i = f(y_i, \theta), \qquad y_i(0) = x_i, \tag{7}$$

with step-size h and with the forward Euler method.
The training problem can be written as a discrete optimal control problem:

$$\begin{aligned} &\min_{\theta} \sum_{i=1}^{m} \phi_i\left(y_i^{[N]}\right) + \mathcal{R}(\theta), \\ &\text{s.t.} \quad y_i^{[j+1]} = y_i^{[j]} + hf\left(y_i^{[j]}, \theta^{[j]}\right), \qquad j = 0, \dots, N-1. \end{aligned} \tag{8}$$

where $\phi_i := \phi(\mu(\cdot), c_i)$, with $\phi : \mathcal{C} \times \mathcal{C} \to \mathbb{R}$ a function that is minimized when its arguments are equal, and $\mathcal{R}(\theta)$ a regularization function.

DISMA Dipartimento di Scienze Matematiche G. L. Lagrange
ECCELLENZA 2018 · 2022

POLITECNICO DI TORINO

A Mean Field Control formulation for Deep Learning

DISMA Dipartimento di Scienze Matematiche G. L. Lagrange
ECCELLENZA 2018 · 2022

Hamiltonian boundary value problem

Let us consider the following Mayer problem which is a continuous version of the discrete problem (8):

$$\begin{aligned} &\max_u \phi(y(T)) \\ s.t. \qquad &\dot{y} = f(y(t), u(t)), \quad y(0) = x, \quad t \in [0, T]. \end{aligned} \tag{9}$$

Clearly, we can solve it with the classical Optimal Control approach introducing $\mathcal{H}(y, p, u) = p^T \cdot f(y, u)$, and geting the following constrained boundary value problem:

$$\begin{cases} \dot{y} = \nabla_p \mathcal{H}, \\ \dot{p} = -\nabla_y \mathcal{H}, \end{cases} \qquad \begin{cases} y(0) = x, \\ p(T) = \nabla \phi(y(T)), \end{cases} \tag{10}$$

$$\nabla_u \mathcal{H} = 0. \tag{11}$$

and so on...
But we get a very high dimensional optimal control problem... $x \in \mathcal{X}^m \subset (\mathbb{R}^n)^m$

Mean field formulation of DPP and HJB

We borrow some ideas from [Weinan E,Han,Li 2019].

Input-label pair (x, c) considered as a random variable drawn from a probability measure.

The problem becomes searching an optimal transform that propagates the input population to the desired target distribution and can be formulated as a mean-field optimal control problem:

$$\begin{aligned} &\inf_{\theta_t \in L^\infty} \mathbb{E}_{(x,c)\sim\mu_0} \left[\phi(y_T, c) + \int_0^T \mathcal{L}(y_t, \theta_t) dt \right] \\ &s.t. \quad \dot{y} = f(y_t, \theta_t), \end{aligned} \tag{12}$$

where $\mathcal{L}^\infty \equiv \mathcal{L}^\infty([0, T], \mathbb{R}^p)$ denotes the set of essentially-bounded measurable functions and μ_0 is the joint distribution of the initial states x and terminal target c.

Let

$$J(t, \mu, \theta_t) := \mathbb{E}_{(y_t,c)\sim\mu_t} \left[\phi(y_T, c) + \int_t^T \mathcal{L}(y_t, \theta_t) dt \right] \quad \text{s.t. } \dot{y} = f(y_t, \theta_t), \tag{13}$$

$$v^*(t, \mu) = \inf_{\theta_t \in L^\infty} J(t, \mu, \theta_t). \tag{14}$$

Mean field formulation of DPP and HJB

Theorem (Mean-Field DPP & HJB (Weinan E,Han,Li 2019))

Under suitable assumptions for f, $\mathcal{L}$, ϕ, μ_0, both $J(t,\mu,\theta_t)$ and $v^*(t,\mu)$ are Lipschitz continuous on $[0,T]\times\mathcal{P}_2(\mathbb{R}^{n+d})$. For all $0\le t\le\hat{t}\le T$, the principle of dynamic programming suggests

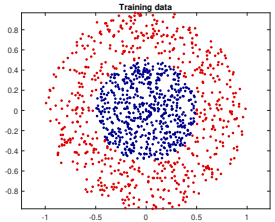
$$v^*(t,\mu)=\inf_{\theta_t\in L^\infty}\mathbb{E}_{(y_t,c)\sim\mu_t}\left[\int_t^{\hat{t}}\mathcal{L}(y_t,\theta_t)+v^*(\hat{t},\hat{\mu})dt\right] \quad \text{subject to } \dot{y}=f(y_t,\theta_t), \tag{15}$$

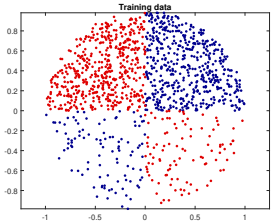
where $\hat{\mu}$ denotes the terminal distribution at $\hat{t}$. Furthermore, through standard computations we get the following equation:

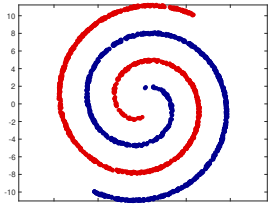
$$\begin{cases}\partial_t v+\inf_{\theta_t\in L^\infty}\langle\partial_\mu v(\mu)(\cdot),f(\cdot,\theta_t)\rangle_\mu+\langle\mathcal{L}(\cdot,\theta_t)\rangle_\mu=0\\ v(T,\mu)=\langle\phi(\cdot)\rangle_\mu,\end{cases} \tag{16}$$

where $\langle f(\cdot),g(\cdot)\rangle_\mu:=\int f(w)^T g(w)d\mu(w)$ and accordingly denote $\langle f(\cdot)\rangle_\mu=\int f(w)d\mu(w)$. Finally, if $\theta^*:(t,\mu)\mapsto\mathbb{R}^p$ is a feedback policy that achieves the infimum in (15), then θ^* is an optimal solution of the problem (12).

A classification problem







Mean field Optimal Control Approach

Our problem becomes then:

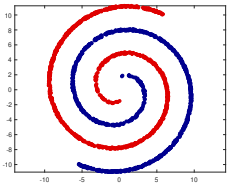
$$\begin{cases} \partial_t v + \inf_{\theta_t \in L^\infty} \langle \partial_\mu v(\mu)(\cdot), f(\cdot, \theta_t) \rangle_\mu + \langle \mathcal{L}(\cdot, \theta_t) \rangle_\mu = 0 \\ v(T, \mu) = \langle \phi(y^{[T]}) \rangle_\mu, \end{cases} \tag{17}$$

where $\theta_t = \arg\inf_{\theta_t \in L^\infty} \langle \partial_\mu v(\mu)(\cdot), f(\cdot, \theta_t) \rangle_\mu + \langle \mathcal{L}(\cdot, \theta_t) \rangle_\mu$ is used in the forward propagation $\dot{y} = f(y_t, \theta_t)$, where $f\left(y_i^{[l]}, \theta^{[l]}\right) = \theta^{[l]} \in \mathbb{R}^2$, $\langle f(\cdot) \rangle_\mu = \int f(w) d\mu(w)$. The running cost is $\mathcal{L}(\cdot, \theta_t) = |\theta_t|^2/2$.

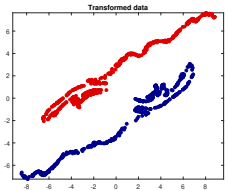
We adopt:

- 3 datasets. 2000 training points (60% training set, 20% validation set, 20% test set).
- Points divided into two groups: blue (label 0) and red (label 1) points.
- Sigmoid function as hypothesis function $\mu(x) = \frac{1}{1-e^{-(x_1-x_2)}}$, within a the cost function of the form $\phi(y^{[N]}) = \sum_{i=1}^m \|\mu(y_i^{[N]}) - c^i\|^2$.
- We discretize the space of mesures and the HJ via standard semiLagrangian schemes to compute the solution in $[0, 1]$ with $\Delta x = 0.1$, $\Delta t = 0.05$,
- Other parameters: $T = 1$, $N_T = 20$ (#layers).
- Equilibrium point found with a fixed point iteration $Niter = 150$.

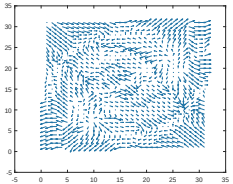
Tests on the Dynamic Programming Principle approach



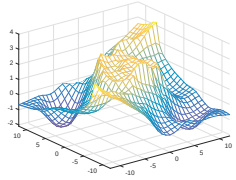
Training data



Final configuration



Control in feedback form



Value function

Figure: Some examples of the results obtained by discretizing the HJB equation with a semi-Lagrangian scheme.

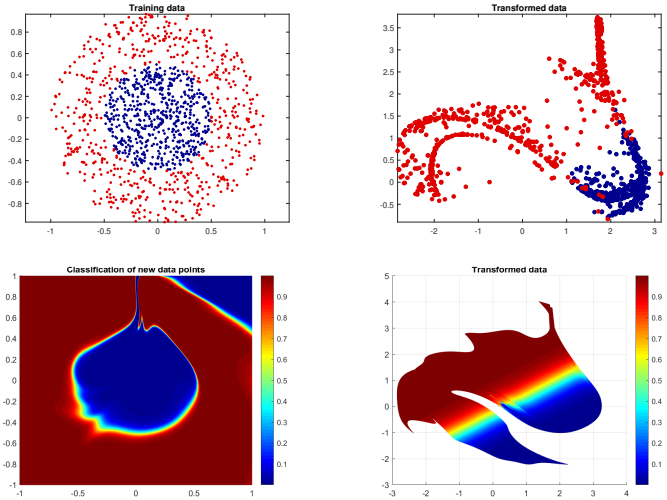


Figure: Classification results concentric circles. The propagation transforms the input features (the coordinates of the points in the plane) so that the two classes of points can be linearly separated.

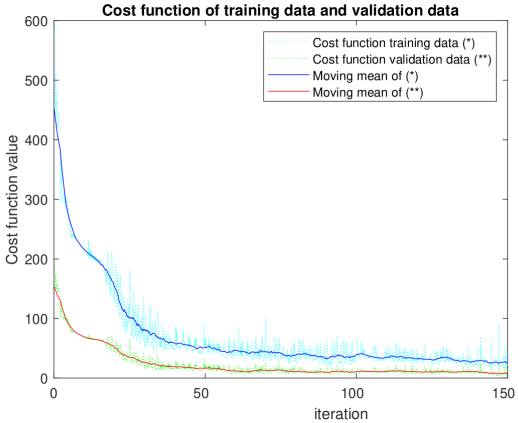


Figure: Cost function in the case of concentric circles. The cost related to the training data decreases with respect to the fixed point iterations, meaning that the network is learning the right parameters to classify the training data. Also the cost function of the validation data decreases, thus there is no overfitting. By adding a threshold equal to 0.5, we got an error for the test set equal to 0.03.

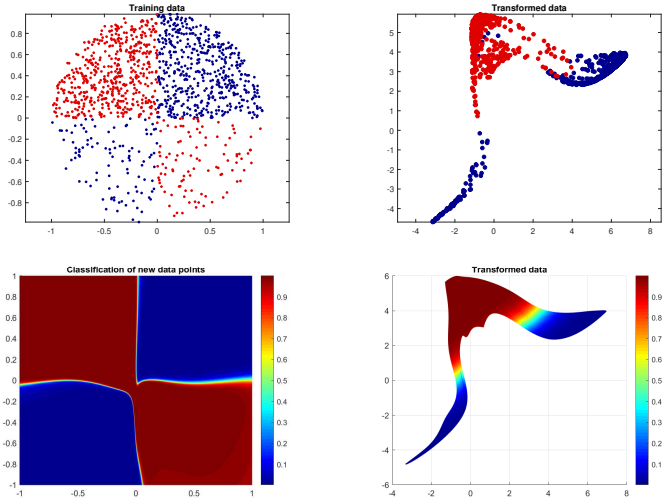


Figure: Classification results circular sectors. The propagation transforms the input features (the coordinates of the points in the plane) so that the two classes of points can be linearly separated.

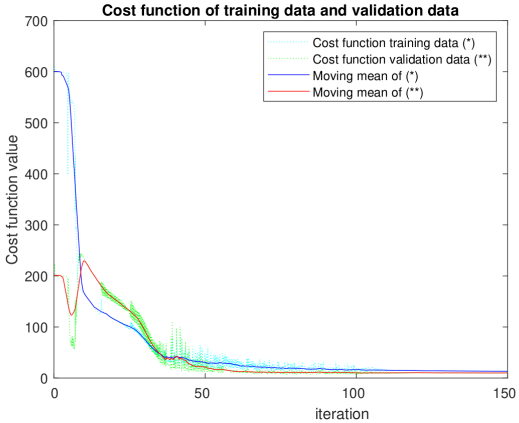


Figure: Cost function in the case of circular sectors. The cost related to the training data decreases with respect to the fixed point iterations, meaning that the network is learning the parameters to accurately classify the training data. Also the cost function of the validation data decreases, thus there is no overfitting. By adding a threshold equal to 0.5, we got an error for the test set equal to 0.04.

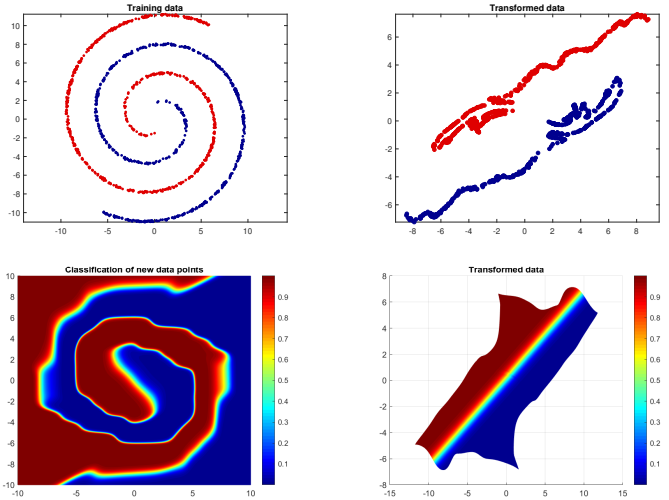


Figure: Classification results swiss roll. The propagation transforms the input features (the coordinates of the points in the plane) so that the two classes of points can be linearly separated.

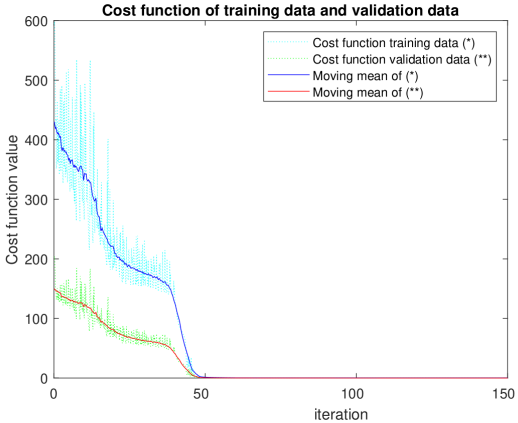


Figure: Cost function in the swiss roll case. The cost related to the training data decreases with respect to the fixed point iterations and reaches the 0 value, meaning that at the end the network can correctly classify the training data with the learned parameters. Also the cost function for the validation data decreases, thus there is no overfitting. By adding a threshold equal to 0.5, we got an error for the test set equal to 0.

Animations on binary classification problems

Animations showing the forward propagation of points through a 20 layers-Network. Three different training datasets are considered, containing points in $\mathbb{R}^2$ dived into two classes, red points and blue points. The network transforms the input data to make the two classes linearly separable (it is a two-class linear classifier).

Conclusions and perspectives

Deep neural networks may be considered as discrete-time nonlinear dynamical systems and their internal parameters can be recast as controls, allowing the formulation of the training process as an optimal control problem.
Due to the large number of data, the use of mean field control can be useful.

Advantages

- It is easier to analyse continuous dynamical systems than discrete ones.
- We can use sophisticated numerical discretization methods from optimal control.
- Development of a mathematical theory for deep learning.

Clear drawbacks

- The problem is more complicated and it requires highly sophisticated mathematical tools.
- The complexity of the computation may be very large.
- We loose some of the points of deep learning.

To-do-list

- Experiment with nonlinear evaluators from machine learning.
- Compare with methods on the market.
- Consider neural networks with an explicit mean-field structure (e.g. batch-normalization).

Partial Bibliography

M. Benning, E. Celledoni, M. J. Ehrhardt, B. Owren,C.-B. Schönlieb Deep learning as optimal control problems: Models and numerical methods, Journal of Computational Dynamics 6.2 (2019), pp. 171-198.

E W., Han, J., Li, Q. A mean-field optimal control formulation of deep learning. Research in the Mathematical Sciences, 6(1), (2019) 10.

A. Leone. The use of Optimal Control Theory in deep learning problems. Master Thesis in Mathematical Engineering, University of L'Aquila, 2020.

E. Haber and L. Ruthotto. Stable architectures for deep neural networks, Inverse Problems 34.1 (Dec. 2017), p. 014004.

C. F. Higham and D. J. Higham. Deep Learning: An Introduction for Applied Mathematicians, SIAM Review 61.3 (Jan. 2019), pp. 860-891.

G. H. Liu and E. A. Theodorou. Deep Learning Theory Review: An Optimal Control and Dynamical Systems Perspective, arXiv: 1908.10920 [cs.LG] (2019).

Thank you for your attention!