Apprentissage automatique, approximation numérique adaptative et méthodes VOF (Volume Of Fluid-Machine Learning)
	hanks{thanks to O. Pironneau (LJLL), S. B. Diallo (ex. ESILV), H. Jourdren (CEA), M. Ancellin (post-doc CEA)}

B. Després (LJLL-SU)
From ML to interface tracking/reconstruction

- Modern ML softwares: TensorFlow (Google 15’), Keras (Chollet 18’), ScikitLEarn (Inria 10’), Pytorch (Facebook), Julia, Matlab, ...

\[ \partial_t \alpha + \nabla \cdot (a\alpha) = 0, \quad \alpha_0(x) = I_\omega(x) \text{ an indicatrix function} \]

- It questions numerical analysis and scientific computing: Yarotsky 17’, Daubechies-DeVore et al 19’, Hesthaven 18’, Zaleski 19’, ...
- Last part of this presentation: ML for Volume of Fluid (VOF)
• Take a large dataset: \( \mathcal{D} = \{(x_i, y_i), \ i = 1, \ldots \} \subset \mathbb{R}^m \times \mathbb{R}^n \)

\[ f^{\text{obj}} \]

\( (x_i) \) \hspace{2cm} \( (y_i) \)

Postulate: the dataset corresponds to an objective function

\[ f^{\text{obj}} : \mathbb{R}^m \longrightarrow \mathbb{R}^n \]

with \( x_i \in \mathbb{R}^m \), \( y_i = f^{\text{obj}}(x_i) + \varepsilon_i \in \mathbb{R}^n \), and noise \( \varepsilon_i \in \mathbb{R}^n \) (hopefully small).

• Examples:
  a) \textbf{MNIST}: \( x_i \in \mathbb{R}^{28 \times 28} \) and \( y_i \in [0, 1]^10 \) with \( \sum_i y_i = 1 \). Here \( m = 784 \) and \( n = 10 \).
  b) The Takagi function: \( f^{\text{obj}}(x) = x^2 \), \( m = n = 1 \).
  c) VOF: up to \( m = 125 \) in 3D, and \( n = 1 \).
Supervised learning of the objective function

• Take a linear function $f$ with weight $W \in \mathcal{M}_{mn}(\mathbb{R})$ and bias $b \in \mathbb{R}^n$

$$f : \mathbb{R}^m \longrightarrow \mathbb{R}^n,$$

$$x \longmapsto f(x) = Wx + b. \quad (1)$$

• Consider the convex cost function $J(W, b) = \frac{1}{\text{card}\mathcal{D}} \sum_{(x,y) \in \mathcal{D}} |f(x) - y|^2$. An optimal value satisfies $J(W_*, b_*) \leq J(W, b) \quad \forall (W, b)$.

Let $y$ and $p(z)$ be discrete probabilities:

$y_i \in [0, 1], \sum y_j = 1 ; p_i = \frac{\exp z_i}{\sum_{j=1}^n \exp z_j} \in (0, 1)$.

For classification, one takes cross-entropy (Kullback-Leibler divergence)

$$J(W, b) = - \sum_{(x,y) \in \mathcal{D}} (\log p(f(x)), y) \geq 0.$$

This cost function is convex.
Two more ideas: recursivity and non linearity

- **Recursivity** = composition of functions.
  - \( a_0 = m \) is the **input layer**
  - \( a_{p+1} = n \) is the **output layer**
  - \((a_1, a_2, \ldots, a_p) \in \mathbb{N}^p\) are the **(dense) hidden layers with neurons**

- Consider
  \[
  f_r : \mathbb{R}^{a_r} \longrightarrow \mathbb{R}^{a_{r+1}},
  \]
  \[
  X_r \mapsto f_r(X_r) = W_r X_r + b_r
  \]
  and the function
  \[
  f = f_p \circ f_{p-1} \circ \ldots \circ f_2 \circ f_1 \circ f_0.
  \]

**Depth** \( p \) is the number of layers.
**Width** \( N = \max_r a_r \) is the maximal number of neurons per layer.
Non linearity

- Non linearity is added with an activation function.

**Sigmoid** $\in C^1(\mathbb{R})$. A sigmoid $\sigma$ is monotone, $0 < \sigma' < 1$, with limit value 0 at $-\infty$ and limit value 1 at $+\infty$.

**ReLU** $\in C^0(\mathbb{R})$. It is defined by $R(x) = \max(0, x)$.

Thresholding at level $a \geq 0$ yields $R^a(x) = \min(R(x), a)$.

Generalization component wise to activation functions $\mathbb{R}^q \rightarrow \mathbb{R}^q$.

A function defined through a generic **neural network** is:

For $g_r = \sigma of_r$ or $g_r = Rof_r$, define $f = f_{p+1}og_{p} \ldots og_{2}og_{1}og_{0}$. 
Variants defined by constraints on the weights

- **LeCun et al 10’** : Convolutional networks and applications in vision. Use convolution matrices with windowing. That is \( W_r \) with many zero coefficients and many other coefficients which are the same, plus pooling performed component wise

\[
\max(a, b) = a + R(b - a)
\]

- **Mallat 12’** : Group invariant scattering. If \( W_r^t W_r \leq I \) (that is \( \| W_r \|_2 \leq 1 \)) then the transformation is non expansive

\[
\| f(x) - f(\hat{x}) \|_2 \leq \| x - \hat{x} \|_2.
\]

Isometric embedding : \( n > m \), complex valued matrices \( W_r \), activation \( | \cdot | \) and convenient parametrization, then all \( \leq \) become \( \approx \).

- **Salimans-Kingma 16’** : Weight Normalization : A Simple Reparameterization to Accelerate Training of Deep Neural Networks. Improve stability (and accuracy) forcing

\[
\| W_r \|_\infty \leq 1 \text{ or } \| W_r \|_\infty \approx 1.
\]
Training is the off-line calculation with softwares of the weights

\[ W = (W_{p+1}, b_{p+1}, W_{p-1}, b_{p-1}, \ldots, W_0, b_0) \in \mathcal{W} \]

where dim(\mathcal{W}) = \sum_{r=0}^{p+1} (a_r + 1)a_{r+1}.

Minimization of a cost functional \( J : \mathcal{W} \rightarrow \mathbb{R} \)

\[ J(W) = \sum_{(x, y) \in \mathcal{D}} \varphi(f_W(x), y) \geq 0. \]

Among many difficulties:

- \( \text{card} (\mathcal{D}) >> 1 \)
- \( J \) non convex for \( p \geq 1 \),
Stochastic gradient (card(\(\mathcal{D}\)) \(\gg\) 1)

Iterations \(k\) are called **epochs**.

Do a **batch** decomposition decided at random

\[
\mathcal{D} = \bigcup_{r=1}^{b} B_r \quad \text{and} \quad J(W) = \sum_{r=1}^{b} J_r(W).
\]

**Stochastic gradient** is a gradient descent algorithm with splitting

\[
W^{k+\frac{r}{b}} = W^{k+\frac{r-1}{b}} - \sigma_k \nabla J_r(W^{k+\frac{r-1}{b}}), \quad 1 \leq r \leq b,
\]

Make "convenient quadratic simplifications": 

\[
J(W) = \frac{1}{2}(A(W - W^*), W - W^*),
\]

and 

\[
J_r(W) = \frac{1}{2}(A_r(W - W_r^*), W - W_r^*) \quad \text{with} \quad A_r > 0 \quad \text{for all batches}.
\]

**Lemma**

Assume \(\lim_{k \to \infty} \sigma_k = 0\) and \(\sum_k \sigma_k = \infty\).

Then \(W^k \to W^*\) (and this is independent of the batches).
Moment methods ($J$ non convex)

Replace the (continuous) gradient descent $W'(t) = -\nabla J(W(t))$ by a moment method with Nesterov acceleration $\nu \geq 0$

$$t > 0 : \begin{cases} W'(t) = Z(t), \\
Z'(t) = -\nabla J(W(t) + \nu Z(t)). \end{cases}$$

One has

$$\frac{d}{dt} \left( \frac{1}{2} |Z(t)|^2 + J(W(t) + \nu Z(t)) \right) = -\nu |\nabla J(W(t) + \nu Z(t))|^2 \leq 0.$$ 

A popular variant is:

Convergence of one layer networks

G. Cybenko 89’, Approximation by Superpositions of a Sigmoidal Function

Theorem

Finite sums of the form \( G(x) = \sum_{j=1}^{N} \alpha_j \sigma (y_j^t x + \theta_j) \) are dense in \( C^0([0, 1]^m) \) equipped with the norm of the maximum.

Constructive proof for ReLU $R$

Let $f^{\text{obj}} \in C_0^2(\mathbb{R})$ (with compact support). Then
\[
  f^{\text{obj}}(x) = \int_{-\infty}^{x} (f^{\text{obj}})'(y) dy
\]
\[
  = \int_{\mathbb{R}} (f^{\text{obj}})'(y) H(x - y) dy = \int_{\mathbb{R}} (f^{\text{obj}})''(y) R(x - y) dy.
\]
Approximation of the integral with a Riemann sum ($\Delta y = h > 0$) yields
\[
  f(x) = \sum_j \alpha_j R(x - jh), \quad \alpha_j = (f^{\text{obj}})''(jh) h.
\]

In higher dimensions $x \in \mathbb{R}^m$, a Boltzmann-like formal representation is
\[
  f^{\text{obj}}(x) = \int_{|\Omega| = 1} \int_{b \in \mathbb{R}} z(\Omega, b) H(\Omega \cdot x - b) d\Omega db + C
\]
with
\[
  z(\Omega, b) = \frac{i}{2\pi} \int_{\mu > 0} \mu^m \hat{f}^{\text{obj}}(\mu \Omega) e^{i\mu b} d\mu, \quad i^2 = -1.
\]
1D mesh \( \cdots < x_{j-1} < x_j < x_{j+1} < \cdots \) with \( h_{j+\frac{1}{2}} = x_{j+1} - x_j \) and adaptive Finite Element \( P_1 \) function \( \varphi_j \)

\[
\varphi_j(x) = R\left(\frac{1}{h_{j-\frac{1}{2}}} R(x - x_{j-1}) - \left(\frac{1}{h_{j-\frac{1}{2}}} + \frac{1}{h_{j+\frac{1}{2}}}\right) R(x - x_j)\right),
\]

\[
\varphi_j(x) = R\left(\frac{1}{h_{j+\frac{1}{2}}} R(x_{j+1} - x) - \left(\frac{1}{h_{j-\frac{1}{2}}} + \frac{1}{h_{j+\frac{1}{2}}}\right) R(x - x_j)\right),
\]

\[
\varphi_j(x) = R^a \left( R^a \left( \frac{x - x_{j-1}}{h_{j-\frac{1}{2}}} \right) + R^a \left( \frac{x_{j+1} - x}{h_{j+\frac{1}{2}}} \right) - 1 \right), \quad a = 1.
\]

Take \( f(x) = \sum_{j \in \mathbb{Z}} f^{\text{obj}}(x_j) \varphi_j(x) \).
Let \( g : [0, 1] \rightarrow [0, 1] \) be the hat function in dimension one

\[
g(x) = 1 - |1 - 2x| = 2R(x) - 4R \left( x - \frac{1}{2} \right), \quad 0 \leq x \leq 1.
\]

Starting from \( g_1 = g \), one defines \( g_{r+1} = gog_r \)

Set \( h_r = \sqrt{3} \left( g_r - \frac{1}{2} \right) \).

Then

\[
\int_0^1 h_r(x)h_s(x)dx = \delta_{rs}
\]

and

\[
\int_0^1 h'_r(x)h'_s(x)dx = 3(2^r)^2 \delta_{rs}.
\]
Takagi 1901': An example of the continuous function without derivative

\[ \sum_{r \geq 1} \frac{1}{2r} g_r(x) = f^{\text{Tak}}(x). \]

By definition \( f^{\text{Tak}} \notin H^1(a, b) \) for all \( 0 \leq a < b \leq 1 \).

Yarostky 17' (and many others):

\[ x - \sum_{r \geq 1} \frac{1}{4^r} g_r(x) = x^2. \]

So

\[ \left\| x^2 - \left( x - \sum_{r=1}^{p} \frac{1}{4^r} g_r(x) \right) \right\|_{L^\infty(0,1)} \leq \sum_{p+1 \leq n} \frac{1}{4^r} = \frac{1}{3 \times 4^p}. \]

The multiplication \( x \rightarrow x^2 \) is obtained on a dense neural network. **Depth \( p \), width \( N = 3 \):** the accuracy is \( \varepsilon = O(4^{-p}) \)

\[ \text{Cost} = O(3 \times p) \approx C|\log \varepsilon|. \]

The same for \( x \rightarrow x^3 = \frac{1}{4}(x + x^2)^2 - \frac{1}{4}(x - x^2)^2 \), and for \( x \rightarrow x^4, x^5, \ldots \).

The same for \((x, y) \rightarrow xy = \frac{1}{4}(x + y)^2 - \frac{1}{4}(x - y)^2\), …
Set \[ F_{n,d} = \left\{ f \in W^{n,\infty}([0, 1]^d), \| f \|_{W^{n,\infty}([0,1]^d)} \leq 1 \right\}. \]

**Theorem**

For any \( d, n \) and \( \varepsilon \in (0, 1) \), there is a ReLU network architecture that

- is capable to express any function in \( F_{n,d} \) with error \( \varepsilon \)
- has the depth at most \( c (\log 1/\varepsilon + 1) \) and at most \( c \varepsilon^{-d/n} (\log 1/\varepsilon + 1) \) weights and computation units, with some constant \( c = c(d, n) \).
Tensorflow : Adam, $k_{\text{max}} = 100$

**Lemma**

Take $p \geq 1$ and $(W_r^0, b_r^0) = 0$ for all $r$. Then $\nabla_{W^0} f = 0$ for all $r$ and $\nabla_{b_r^0} f = 0$ for all $r \geq 1$.

Proof for $p = 1$ : immediately comes from $f(x) = W_1R(W_0x + b_0) + b_1$.

$\implies$ Reference manuals : initialize weights and biases at random !!
A question raised by Ron DeVore (Une journée autour de Ron DeVore 19’) was the stability of Takagi based neural networks.

Take $f^{\text{obj}}(x) = x^2$

$W^0 = \text{exact and } f^0(x) \approx x^2$

$f^k(x) \approx x^2$

One observes practical stability.
Minimization of $H^1$ norm

$$f^{\text{obj}}(x) = x - x^2/2 - x^3/2 = (x - x^2) + \frac{x^2 - x^3}{2}.$$ 

The best results below is obtained with the cost functional $\approx H^1$ norm.

Here error $\approx O\left(\#\text{DOF}^{-2}\right)$, like $P^1$ FEM!!
Volume of Fluid algorithms

Essential for fluid-structure interaction, bubble flows, multiphase flows, ... (inspired by Hesthaven 18', Zaleski 19')

Model equation is advection

$$\partial_t \alpha + \mathbf{a} \cdot \nabla \alpha = \partial_t \alpha + \nabla \cdot (\mathbf{a} \alpha) = 0, \quad \mathbf{a} = (1, 1).$$

Needs: a) maximum principle, b) conservative, c) super-efficient for

$$\alpha_0(x) = I_\omega(x)$$ an indicatrix function \(\in BV\).
Idea of VOF

Horizontal direction $\partial_t \alpha + \partial_x \alpha = 0$ with $a = 1$

- $\Omega_{ij} = [(i - 1)\Delta x, i\Delta x] \times [(j - 1)\Delta x, j\Delta x]$
- Swept $:= [(1 - \beta)\Delta x < x < \Delta x]$
- $\beta = a \frac{\Delta t}{\Delta x}$ the horizontal measure of the Swept region

$$\alpha_{ij} = \frac{|\Omega_{ij} \cap \omega|}{|\Omega_{ij}|} \in [0, 1], \quad \alpha_* = \frac{|\Omega_{00} \cap \text{Swept} \cap \omega|}{|\Omega_{00} \cap \text{Swept}|} \in [0, 1]$$

Consider $f^{\text{obj}} : \mathbb{R}^m \rightarrow \mathbb{R}^n$ with $m = 5^2 + 1$ and $n = 1$

$$f^{\text{obj}} (\alpha_1, \ldots, \alpha_{25}, \beta) = \alpha_*.$$
Construction of the dataset

Implement with numerical quadratures

\[ D : \mathbb{R}^{\text{par}} \rightarrow \mathbb{R}^{(2N+1)^2+1}, \]
\[ (\theta, r, \ldots : \beta) \rightarrow (\alpha, \beta). \]

\[ E : \mathbb{R}^{\text{par}} \rightarrow \mathbb{R}, \]
\[ (\theta, r, \ldots : \beta) \rightarrow \alpha_* . \]

- \( f^{\text{obj}} = E_0 D^{-1} \) (inverse problem)

- Datasets \( \{z_i\}_{i=1,\ldots} \in \mathbb{R}^{\text{par}} \) \( x_i = D(z_i) \) and \( y_i = E(z_i) \) for \( i = 1, \ldots \)

- Finally the dataset for ML is defined as

\[ \mathcal{D} = \{(x_i, y_i)\}_{i=1,\ldots} \]
Accuracy of the VOF-ML flux for arcs of circle

\[ \mathcal{D}_{\text{tot}} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}, \quad \text{batchsize} = 10000. \]

<table>
<thead>
<tr>
<th>dim</th>
<th>par</th>
<th>m</th>
<th>n</th>
<th>\mathcal{D}<em>{\text{train}}/\mathcal{D}</em>{\text{test}}</th>
<th>loss</th>
<th>val</th>
<th>\mathcal{L}^2 \text{ er.}</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>1395790/349490</td>
<td>4.2e-5</td>
<td>4e-5</td>
<td>6.3e-3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>26</td>
<td>1</td>
<td>1395422/349858</td>
<td>3.1e-5</td>
<td>4.2e-5</td>
<td>6.5e-3</td>
</tr>
</tbody>
</table>

\( p = 4; \quad a_0 = 26, \quad a_1 = 80, \quad a_2 = 40, \quad a_3 = 20, \quad a_4 = 10, \quad a_5 = 1. \)
Advection of Zalezak profile

Framework
Accuracy and adaptive
Finite Element approximation

VOF
$$L^1$$-convergence table (CFL=0.5, optimal dataset)

Indicatrix function = ellipse.  
Dataset made with arcs of circles.  
Errors in $$L^1$$ norm : VOF-ML, Upwind and Lagoutière=SLIC/Downwind.

<table>
<thead>
<tr>
<th>block</th>
<th>$$N_{1D}$$</th>
<th>VOF-ML</th>
<th>Upwind</th>
<th>SLIC/Downwind</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 x 5</td>
<td>30</td>
<td>0.002364</td>
<td>0.6633</td>
<td>0.0527</td>
</tr>
<tr>
<td>5 x 5</td>
<td>60</td>
<td>0.001265</td>
<td>0.4895</td>
<td>0.0352</td>
</tr>
<tr>
<td>5 x 5</td>
<td>120</td>
<td>0.001132</td>
<td>0.3565</td>
<td>0.0195</td>
</tr>
<tr>
<td>5 x 5</td>
<td>240</td>
<td>0.000713</td>
<td>0.2551</td>
<td>0.0108</td>
</tr>
<tr>
<td>5 x 5</td>
<td>480</td>
<td>0.000460</td>
<td>0.1818</td>
<td>0.0051</td>
</tr>
<tr>
<td></td>
<td>≈ 0.5</td>
<td>≈ 0.5</td>
<td>≈ 1</td>
<td></td>
</tr>
</tbody>
</table>


Model : \[
\begin{align*}
\partial_t (\rho c_1) + \nabla \cdot (\rho c_1 u) &= 0, \\
\partial_t (\rho c_2) + \nabla \cdot (\rho c_2 u) &= 0, \\
\partial_t (\rho u) + \nabla \cdot (\rho u \otimes u) + \nabla p &= 0, \\
\partial_t (\rho e) + \nabla \cdot (\rho ue + pu) &= 0,
\end{align*}
\]

plus \[
\begin{align*}
1 &= c_1 + c_2, \\
\tau &= c_1 \tau_1 + c_2 \tau_2, \\
e &= c_1 \epsilon_1 + c_2 \epsilon_2 + \frac{1}{2} u^2.
\end{align*}
\]

and perfect gas \[
\begin{align*}
p_1 &= (\gamma_1 - 1) \rho_1 \epsilon_1, \\
p_2 &= (\gamma_2 - 1) \rho_2 \epsilon_2.
\end{align*}
\]
Conclusion/open problems

- Spectral accuracy for Takagi functions wrt depth $p$
- Explain better the efficiency for large $m$
- Non probabilistic proofs of convergence of training with batches
- Minimization/training theory for ReLU functions: in particular $\nabla J$ is only piecewise-$C^0$
- Convergence theory of VOF-ML schemes $\Delta x \to 0$
- Speed up training: computer scientists work on it.
- Ongoing with M. Ancellin (post-doc): 3 phases and more, 3D, . . .
Framework
Accuracy and adaptive
Finite Element approximation
VOF

**CPU cost:** \( \mu s \) per cell \( \times \) cycle

**Complexity of the algorithm**

\[
T_{\text{total}} = T_{\text{Euler2D}} + \frac{\lambda_{\text{interface}}}{N_{1D}} T_{\text{VOF-ML}}, \quad \lambda_{\text{interface}} = \frac{N_{\text{interface}}}{N_{1D}} \approx \text{constant.}
\]

<table>
<thead>
<tr>
<th>block</th>
<th>layers</th>
<th>( T_{\text{Euler2D}} )</th>
<th>( T_{\text{VOF-ML}} )</th>
<th>rescaled ( T_{\text{VOF-ML}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 ( \times ) 3</td>
<td>2</td>
<td>0.958</td>
<td>9.78</td>
<td>0.271</td>
</tr>
<tr>
<td>3 ( \times ) 3</td>
<td>3</td>
<td>0.958</td>
<td>31.9</td>
<td>0.886</td>
</tr>
<tr>
<td>3 ( \times ) 3</td>
<td>4</td>
<td>0.9807</td>
<td>90.8</td>
<td>2.52</td>
</tr>
<tr>
<td>3 ( \times ) 3</td>
<td>5</td>
<td>0.977</td>
<td>255</td>
<td>7.08</td>
</tr>
<tr>
<td>5 ( \times ) 5</td>
<td>2</td>
<td>0.946</td>
<td>15.5</td>
<td>0.432</td>
</tr>
<tr>
<td>5 ( \times ) 5</td>
<td>3</td>
<td>0.928</td>
<td>44.7</td>
<td>1.24</td>
</tr>
<tr>
<td>5 ( \times ) 5</td>
<td>4</td>
<td>0.962</td>
<td>113</td>
<td>3.134</td>
</tr>
<tr>
<td>5 ( \times ) 5</td>
<td>5</td>
<td>1.007</td>
<td>304</td>
<td>8.44</td>
</tr>
</tbody>
</table>
National Lab Taps AI Machine With Massive Chip to Fight Coronavirus

Scientists at Argonne National Laboratory hope the system's calling card, a chip measuring over 64 square inches, will speed up drug discovery.

By Agam Shah
Updated May 6, 2020 6:11 am ET

A national lab is turning to an AI machine the size of a dorm-room refrigerator to speed up the development of treatment and vaccines for coronavirus.

Argonne National Laboratory is using the CS-1 computer, from AI chip startup Cerebras Systems, to run learning models aimed at discovering compounds that can be used in existing or new drugs to help create a vaccine or treatments.

The system's calling card is a massive chip, measuring 8.5 inches by 8.5 inches. Putting the neural network on the chip, instead of dispersing it across a system, enables researchers to load the Intel chips with the training and tuning necessary to get the machine running.