When Big Data and Machine Learning meet Partial Differential Equations

Marc Schoenauer
Michele Alessandro Bucci
Equipe TAU, INRIA, LRI, UP-Sud et CNRS,
Université Paris-Saclay

Statistical Modeling and Machine Learning in Meteorology and Oceanography
Brest, Feb. 10., 2020
AI/Big Data is (Deep) Machine Learning
AI/Big Data is (Deep) Machine Learning

FALSE
AI/Big Data is (Deep) Machine Learning

What has changed:
- Data Deluge
- Moore law
- New algorithms

although ...

avent of WWW or continuation
or better understanding of old ones
Learning from examples
● Supervised: all examples are labelled
● Semi-supervised: some examples are labelled
● Unsupervised: no example is labelled

Reinforcement Learning

Machine Learning

Recognition tasks
 sequential decision making
Machine Learning

Learning from examples
● **Supervised**
  all examples are labelled
● Semi-supervised
  some examples are labelled
● Unsupervised
  no example is labelled

Reinforcement Learning
  sequential decision making
Supervised Learning

A toy case-study

- One example = \((x_1, x_2, y)\), where \(y\) is the label (red or blue here)
- **Goal**: find a model \(f(x_1, x_2)\) that separates the labels
- allowing to correctly label future unlabelled example from \((x_1, x_2)\)
Supervised Learning

A zoology of models

- Linear and Polynomial
- Bayésiens Networks
- Decision trees and Random Forests
- Support Vector Machine (kernel machines)
- **Artificial Neural Networks**

One neuron

A network of neurons

Parameters are the **weights** $w_{ij}$
Deep Supervised Learning

Learning Phase

Gradient back-propagation aka Stochastic Gradient Descent

- Present the examples 1 by 1
  - or mini-batch by mini-batch
- **Forward** pass: Compute the Loss
  e.g., \( L = \sum |y(x_1, x_2) - NN(x_1, x_2)|^2 \)
- **Backward** pass: Compute \( \nabla_w L \) (chain rule)
- Modify the weights \( w_{ij} \) from \( \nabla_w L \) to decrease of the loss
- Loop

Recognition Phase aka Inference

Present an unlabelled example, the output of the network is the predicted label
Neural Networks zoology

Almost complete list of neural network architectures
Good Old Computer Vision

Hand-made features → Learned classifier → CAT
End-to-end Learning

Features and Classifier are learned together

CAT

inputs

classifier output
State-of-the-art

- Many datasets available
  - ImageNet: 14+M examples, 1000 classes
- (pre-trained) networks with numerous layers
  - up to 152!
- Millions to billions weights
  - hundreds of GPU mandatory for learning
- Several tricks of the trade
  - Dropout, residual layers, ensembles, ...

He et al., 2015
Deep Learning
Better than human learning

ILSVRC Top 5 Error on ImageNet

- CV
- Deep Learning
- Human
Deep Learning

Outstanding performances … in well-focused domains

- Supervised learning
  - Image/videos
  - NLP / translation
- Reinforcement/sequential learning
  - Games (AlphaZero)
  - Dynamical systems (Fluid/Structure)
- Representation learning
  - generative models (GANs)
  - domain transfer (DANNs)
- Toward Differentiable Programming

many successes, and fun applications, but …
DL configuration: more of an art

A very high number of hyperparameters to tune

- Cost function
- Topology of the network
  - nblayers, nb neurons, residual or not residual, ...
- Activation functions (sigmoid, tanh, ReLU, …)
- Batch size (and curriculum)
- Optimizer (SGD, Momentum, Adam, Adagrad, …)
  - and its parameters (e.g., learning rate)
- Initialization
- Dropout, Batch Normalisation, ...
- ...

Empirical rules, or meta-optimization (meta-costly)
Mathematically grounded representation of phenomena

- Poisson equation
  \[ \Delta \varphi = \nabla^2 \varphi = \left( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \right) \varphi(x, y, z) = f(x, y, z) \]

- Heat equation
  \[ \frac{\partial u}{\partial t} = \alpha \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) \]

Generally no analytical solution

Numerical solutions by discretisation of the domain

- Finite differences, finite elements, finite volumes, ...
Partial Differential Equations

Mathematically grounded representation of phenomena: exemple

- Generalized Poisson equation

\[
\nabla \cdot (\sigma(x) \nabla u(x)) = 0, \quad x \in \Omega \subset \mathbb{R}^2
\]

\[
u(x) = u_0(x), \quad x \in \partial \Omega.
\]

Conductivity $\sigma$

Sample mesh

Computed current
But does “The Data Deluge Make the Scientific Method Obsolete” (*)?

The answer is of course “no” ...

This talk: survey of synergies between Machine Learning and PDE solving.

ML and PDEs: Agenda

- Simulation is fine, but huge output data
  - DL for Data Analytics (Climate, Particle Physics, …)
- Simulation is fine, but
  - DL as surrogate model (whole simulation, or sub-components)
  - Physic Informed Deep Learning
  - Deep Galerkin Method (high dimensions)
  - PDE-NET
- Inverse problems / calibration
  - Often ill-posed
- Mechanistic model is unknown
  - Learn an analytical model from data
  - Learn a black-box model from data
ML and PDEs: Agenda

- Simulation is fine, but huge output data
  - DL for Data Analytics (Climate, Particle Physics, …)
- Simulation is fine, but
  - DL as surrogate model (whole simulation, or sub-components)
  - Physic Informed Deep Learning
  - Deep Galerkin Method (high dimensions)
  - PDE-NET
- Inverse problems / calibration
  - Often ill-posed
- Mechanistic model is unknown
  - Learn an analytical model from data
  - Learn a black-box model from data
Data Analytics for Simulation Results

- Experiments, either real or simulated, produce loads of data
  - Manual interpretation not possible any more
- Particle physics experiments: several challenges
  - The Higgs boson Challenge (2014)
    - Largest Kaggle challenge (1785 entries)
    - < 100 Boson events per year ($10^{10}$ events)
    - DL winner … unpractical for physicists
  - The TrackML challenge (2018-2019)
    - 10-100 Billion events/year
    - Reconstruct particle trajectories
    - from sparse point-wise traces
    - Winner is not a machine learner
Recognize forthcoming Tropical Cyclones in simulation outputs

- Public Data: Community Atmosphere Model (CAM5)
- 16 variables
- 1152x768 spatial grid, 3 hours time step
- 100 years are available, ~63000 hires samples
- labeled with supervised expert heuristics
- 3 classes: TCs (0.1%), Atmospheric Rivers (1.7%) and Background

T. Kurth et al., “Exascale Deep Learning for Climate Analytics”, *Super Computing* 2018
Deep Learning for Climate Analytics

- Specific architecture DeepLabV3+
- High-speed parallel data staging
  - 27 360 GPUs, 999 PF/s
  - distribution using Horovod (MPI)
  - and all_reduced procedure
- Ran on Summit, first in TOP500

- Weighted loss 1/sqrt(frequency)
- Layer-wise adaptive learning rate control

A new era for extreme weather prediction!

Data Analytics for Simulation Results

- Analysis of simulation results can also be used as feedback to the simulation itself
  - to detect numerical instabilities before they take place
  - and take appropriate counter-measures (e.g., locally modify diffusion parameter, adaptative discretization).

BUT

- ... need some expert labeling
Simulation is fine, but huge output data
- DL for Data Analytics (Climate, Particle Physics, …)

Simulation is fine, but
- DL as surrogate model (whole simulation, or sub-components)
- Physic Informed Deep Learning
- Deep Galerkin Method (high dimensions)
- PDE-NET

Inverse problems / calibration
- Often ill-posed

Mechanistic model is unknown
- Learn an analytical model from data
- Learn a black-box model from data
A basic example: Regression of the complete solution

- Poisson equation: \( \nabla \cdot (\epsilon(x)\nabla \phi(x)) = -\rho(x), \ x \in D \)
  \( \phi|_{\partial D} = 0 \)

- **Inputs**: permittivity \( \epsilon \) and source distribution \( \rho \)
- **Outputs**: dielectric potential \( \phi \)

- Finite differences on a fixed Cartesian grid
- Inputs assumed constant in each pixel/voxel

- “Standard” CNN architecture
- Regularized MSE loss

\[
\|\log_{10}(\phi) - \log_{10}(\hat{\phi})\|^2 + \frac{\lambda}{2n} \sum w^2
\]

Regression of the complete solution (2)

- 64x64 (x64) grids
- 8000/2000 examples (finite differences)
- Error: 1.5% in 2D, 3% in 3D
- from 16s to 0.13s for 2000 simulations in 2D
- from 292s to 1.2s for 5 simulations in 3D

BUT
- No scaling study
- Linear model
- Fixed grid/mesh
Accelerating simulations

Learning sub-scale phenomena

● Global climate modeling
  ○ 2° horizontal resolution, 30 altitude levels
  ○ 30mn time step
● needs to solve CRMs (Cloud Resolving Models)
  ○ turbulence + cloud convection + ...
  ○ in each column (4km-wide),
  ○ at each time-step (20s)
● Train a DNN
● Inputs(z): temperature, humidity, wind profile, … (→ dim 94)
● Outputs(z): heating, moistening, radiative fluxes, … (→ dim 65)
● 9-layers fully connected NN, 256 neurons/layer
● Training data: one-year SPCAM simulations - 140M example

Accelerating simulations

Learning sub-scale phenomena (2)

Results
- 20x speedup
- Means and variability OK
- Good conservation of energy (though not prescribed)
- Good interpolation between extreme values of train data
- Poor extrapolation beyond train data

BUT
- Learning from observations only? Too few available

S. Rasp et al. “Deep learning to represent sub-grid processes in climate models”. PNAS 2018
Soft Body Deformation

Need patient-specific real-time simulation of laparoscopy

- Liver is hyper viscoelastic and anisotropic
- Several complex PDEs for soft tissues
  - anyway an approximation
- Material identification
- Patient-specific geometry
  - Not a big issue, but time consuming
- Boundary conditions are essential
  - but difficult to obtain from images

- Need less than 3mm error
- in less than 50ms per image

Soft Body Deformation

Bottleneck: real-time simulation

Replace FEM simulation by a Deep Network
Supervised learning (regression) of simulation results

Soft Body Deformation

Inspired by the U-net architecture* for image segmentation

- Fully convolutional network
- Encoding: transforms input to reduced space
- Decoding: expands to original dimensions
- # steps and # channels control the accuracy.
- Each convolution kernel isolates different characteristics of u

Very similar to POD approach!

Results

Soft Body Deformation

FEM solution

U-mesh solution

900 ms

3 ms

8.9% relative error

3.4% relative error

3.2% relative error

Differences below 10%, computations 300x faster

ML and PDEs: Agenda

- Simulation is fine, but huge output data
  - DL for Data Analytics (Climate, Particle Physics, …)
- Simulation is fine, but
  - DL as surrogate model (whole simulation, or sub-components)
    - Physic Informed Deep Learning
    - Deep Galerkin Method (high dimensions)
    - PDE-NET
- Inverse problems / calibration
  - Often ill-posed
- Mechanistic model is unknown
  - Learn an analytical model from data
  - Learn a black-box model from data
Physics Informed Deep Learning

Data-driven solution of PDEs

- Define

\[ u_t = \mathcal{N}(t, x, u, u_x, u_{xx}, \ldots) \]

\[ f := u_t - \mathcal{N}(t, x, u, u_x, u_{xx}, \ldots) \]

- and minimize

\[
\frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2 + \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2
\]

- initial and boundary training data

\[ \{t_u^i, x_u^i, u^i\}_{i=1}^{N_u} \]

- collocation training points

\[ \{t_f^i, x_f^i\}_{i=1}^{N_f} \]

Physics Informed Deep Learning

Automatic differentiation

- Thanks to Differentiable Programming
- e.g., in TensorFlow

```python
def u(t, x):
    u = neural_net(tf.concat([t, x], 1), weights, biases)
    return u

def f(t, x):
    u = u(t, x)
    u_t = tf.gradients(u, t)[0]
    u_x = tf.gradients(u, x)[0]
    u_xx = tf.gradients(u_x, x)[0]
    f = u_t + u_x - (0.01/tf.pi)*u_xx
    return f
```

Physics Informed Deep Learning

A meshless approach

Physics Informed Deep Learning

Burger’s equation

\[ u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1, 1], \quad t \in [0, 1] \]
\[ u(0, x) = -\sin(\pi x), \]
\[ u(t, -1) = u(t, 1) = 0. \]

Data: training boundary points \( X \) + 10000 collocation points (not shown).
Predicted dynamics, \( \text{MSE} = 6.7 \times 10^{-4} \).

Physics Informed Deep Learning

Burger’s equation

Exact vs predicted

\[
\begin{array}{c|ccccccc}
N_u & N_f & 2000 & 4000 & 6000 & 7000 & 8000 & 10000 \\
\hline
20 & 2.9e-01 & 4.4e-01 & 8.9e-01 & 1.2e+00 & 9.9e-02 & 4.2e-02 \\
40 & 6.5e-02 & 1.1e-02 & 5.0e-01 & 9.6e-03 & 4.6e-01 & 7.5e-02 \\
60 & 3.6e-01 & 1.2e-01 & 1.7e-01 & 5.9e-03 & 1.9e-03 & 8.2e-03 \\
80 & 5.5e-03 & 1.0e-03 & 3.2e-03 & 7.8e-03 & 4.9e-02 & 4.5e-03 \\
100 & 6.6e-02 & 2.7e-01 & 7.2e-03 & 6.8e-04 & 2.2e-03 & 6.7e-04 \\
200 & 1.5e-01 & 2.3e-03 & 8.2e-04 & 8.9e-04 & 6.1e-04 & 4.9e-04 \\
\end{array}
\]

Influence of amount of data on accuracy

Simulation is fine, but huge output data
  - DL for Data Analytics (Climate, Particle Physics, …)
Simulation is fine, but
  - DL as surrogate model (whole simulation, or sub-components)
  - Physic Informed Deep Learning
  - Deep Galerkin Method (high dimensions)
  - PDE-NET

Inverse problems / calibration
  - Often ill-posed

Mechanistic model is unknown
  - Learn an analytical model from data
  - Learn a black-box model from data
Deep Galerkin Method

- **Original motivations**: High-dim PDEs, parameterized PDEs
- **Context**: parabolic PDEs

\[
\begin{align*}
\partial_t u(t, x) + Lu(t, x) &= 0, \quad (t, x) \in [0, T] \times \Omega \\
u(0, x) &= u_0(x), \quad x \in \Omega \\
u(t, x) &= g(t, x), \quad x \in [0, T] \times \partial \Omega,
\end{align*}
\]

- **Baseline**: minimize

\[
\|\partial_t f + Lf\|_2^2_{[0,T] \times \Omega} + \|f - g\|_2^2_{[0,T] \times \partial \Omega} + \|f(0, \cdot) - u_0\|_2^2_{\Omega}
\]

- **Convergence**: the solution learned by the 1-layer NN converges toward the solution of the PDF when \#neurons \(\to \infty\)
- **Derivatives**: all \(\partial_t\) and \(\partial_x\) can be computed analytically
- **Specific trick** to avoid computing all 2nd order derivatives
  - based on Monte-Carlo approximation
  - but introduces bias and variance

Deep Galerkin Method

The Neural Network

- **Inputs**: $x = (t, x)$
- **Outputs**: $f(x)$
- **Loss**: for $(t_n, x_n)$:
  - no regularization, “infinite” sample set
- **Architecture**: inspired by LSTM topology
  - $L$ hidden layers (typically 4),
    - 4 sub-layers per layer,
    - $x$ fed into all sublayers
  - $M$ units per layer (typically 50)
  - $tanh$ transfer functions
- **Parallel** asynchronous stochastic gradient on 5 GPUs
- 100000 iterations, 1000 samples on each GPU per iter.
  - 500 M points altogether
- Adam optimizer, with complex **ad hoc schedule** to decrease the learning rate

Deep Galerkin Method: Results

Results

<table>
<thead>
<tr>
<th>Number of dimensions</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.05%</td>
</tr>
<tr>
<td>20</td>
<td>0.03%</td>
</tr>
<tr>
<td>100</td>
<td>0.11%</td>
</tr>
<tr>
<td>200</td>
<td>0.22%</td>
</tr>
</tbody>
</table>

- “American Options” PDE with semi-analytic solution

- Parameterized Burger equation
  - Inputs: \((t, x, a, b, \alpha, \nu)\)
  - Results: Indistinguishable from the finite differences solutions

- … without analytical solutions:
  - theoretical bounds
  - results are within bounds

\[
\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} - \alpha u \frac{\partial u}{\partial x}, \quad (t, x) \in [0, 1] \times [0, 1],
\]
\[
u(t, x = 0) = a,
\]
\[
u(t, x = 1) = b,
\]
\[
u(t = 0, x) = g(x), \quad x \in [0, 1].
\]
ML and PDEs: Agenda

- Simulation is fine, but huge output data
  - DL for Data Analytics (Climate, Particle Physics, …)
- Simulation is fine, but
  - DL as surrogate model (whole simulation, or sub-components)
  - Physic Informed Deep Learning
  - Deep Galerkin Method (high dimensions)
  - PDE-NET
- Inverse problems / calibration
  - Often ill-posed
- Mechanistic model is unknown
  - Learn an analytical model from data
  - Learn a black-box model from data
PDE-NET

- **Original motivations:** learn PDEs improve flexibility
- **Context:** 2D convection-diffusion equation
  \[ u_t(t, x, y) = F(x, y, u, u_x, u_y, u_{xx}, u_{xy}, u_{yy}, \ldots) \]
  \[ \begin{cases} \frac{\partial u}{\partial t} = a(x, y)u_x + b(x, y)u_y + cu_{xx} + du_{yy} \\ u|_{t=0} = u_0(x, y) \end{cases} \]
- **Baseline:** minimize
  \[ L = \sum_{i,j} l_{ij}, \text{where } l_{ij} = \|u_j(t_{i+n}, \cdot) - \tilde{u}_j(t_{i+n}, \cdot))\|_2^2 \]
- **Idea:** use convolution neural network to learn derivative operators
  \[ \tilde{u}(t_{i+1}, \cdot) = D_0 u(t_i, \cdot) + \Delta t \cdot F(x, y, D_{00} u, D_{10} u, D_{01} u, D_{20} u, D_{11} u, D_{02} u, \ldots). \]
  \[ D_{ij} u \approx \frac{\partial^{i+j} u}{\partial x^i \partial y^j}. \]
- **Specific trick** to constrain convolution operation
  - filters constrained using their associate moment matrices
  - multiple filters to approximate a given differential operator


Figure 1: The schematic diagram of a $\delta t$-block.

Figure 2: The schematic diagram of the PDE-Net: multiple $\delta t$-blocks.
PDE-NET

\[
\begin{aligned}
\frac{\partial u}{\partial t} &= c\Delta u + f_s(u) \\
\left. u \right|_{t=0} &= u_0(x, y),
\end{aligned}
\]

with \((t, x, y) \in [0, 0.2] \times \Omega,\)

\[f_s(u) = 15 \sin(u)\]

- 50 x 50 mesh
- 7 x 7 filters
- 1,2k parameters in each \(\delta t\)-block

Simulation is fine, but huge output data
  - DL for Data Analytics (Climate, Particle Physics, …)
Simulation is fine, but
  - DL as surrogate model (whole simulation, or sub-components)
  - Physic Informed Deep Learning
  - Deep Galerkin Method (high dimensions)
  - PDE-NET
Inverse problems / calibration
  - Often ill-posed
Mechanistic model is unknown
  - Learn an analytical model from data
  - Learn a black-box model from data
Coupling forward and inverse problems

- Generalized Poisson equation: \( \mathcal{L}u = 0, \quad x \in \Omega \)
  \( u(x) = u_0(x), \quad x \in \partial\Omega. \)

  with
  \[ \mathcal{L}u = \partial_i (a^{ij}(x) \partial_j u) + b^i(x) \partial_j u + c(x) u, \quad i, j = 1, \ldots, \]

- **Forward problem:** find \( u \) knowing \( a, b, c \)
- **Inverse problem:** find \( a (b, c) \) knowing a few measurements of \( u \)
- Use two neural networks, one for \( u \) and one for \( a \)
- input \( x \), standard fully connected
- **Losses** (meshless approach):
  
  \[
  \begin{align*}
  F(u) &= \lambda \| \mathcal{L}u \|_2^2 + \mu \| \mathcal{L}u \|_\infty + \| u - u_0 \|_{1, \partial\Omega} + R^F(u) \\
  I(a^{ij}) &= \lambda \| \mathcal{L}u \|_2^2 + \mu \| \mathcal{L}u \|_\infty + \| a^{ij} - a_0^{ij} \|_{1, \partial\Omega} + R^I(a^{ij})
  \end{align*}
  \]

Inverse problems

Coupling forward and inverse problems: results

- Example: Electrical Impedance Tomography (2D), Poisson equation
- Data (forward): random points on $\Omega$ for $\sigma$, on $\partial\Omega$ for $u_0$
- 4 layers (26, 26, 26, 10), Adam, bs 1000, decay learning rate
- Discretized loss:

\[
\mathcal{F}(u(x; w_u)) = \frac{\lambda}{N_S} \sum_{i=1}^{N_S} |\mathcal{L}_i|^2 + \frac{\mu}{K} \sum_{k \in \text{top}_K(\{|\mathcal{L}_i|\})} |\mathcal{L}_k| \\
+ \frac{1}{N_b} \sum_{b=1}^{N_b} |u(x_b) - u_0(x_b)| + \alpha \|w_u\|_2^2.
\]

- Parameters: $N_S=45000$, $N_b=1200$, $\lambda=0.01$, $\alpha=10^{-8}$, $K=40$, $\mu=10^{-2}$
- Excellent results. In particular

$\text{MSE} = (1.72 \times 10^{-3}, 1.22 \times 10^{-3}, 2.35 \times 10^{-4})$ for 1, 2 or 3 boundary cond.

Inverse problems

Coupling forward and inverse problems: results

- **Example**: Electrical Impedance Tomography (2D), Poisson equation
- Data (inverse): random points on $\Omega$ for $u$, on $\partial\Omega$ for $u_0$ and $\sigma_0$
- Discretized loss:

$\mathcal{I}(\sigma(x; w_\sigma)) = \frac{\lambda}{N_s} \sum_{i=1}^{N_s} |\mathcal{L}_i|^2 + \frac{\mu}{K} \sum_{k \in \text{top}_K(|\mathcal{L}_i|)} |\mathcal{L}_k|$

$\quad + \frac{1}{N_b} \sum_{b=1}^{N_b} \left| \sigma(x_b) - \sigma_0(x_b) \right| + \alpha \| w_\sigma \|_2^2 + \frac{\beta}{N_s} \sum_{i=1}^{N_s} |\nabla \sigma(x_i)|^p$

- Parameters: same, except $\beta=10^{-3}$, $\mu=10^{-2}$
- MSE = $(1.72\ 10^{-3},\ 1.22\ 10^{-3},\ 2.35\ 10^{-4})$ with 1, 2 or 3 measurements

Inverse problems

Coupling forward and inverse problems: results

- **Example**: Electrical Impedance Tomography (2D), Poisson equation
- Data (inverse): random points on $\Omega$ for $u_0$ and $\sigma_0$ on $\partial \Omega$
- Discretized loss: (Inverse)

ML and PDEs: Agenda

- Simulation is fine, but huge output data
  - DL for Data Analytics (Climate, Particle Physics, …)
- Simulation is fine, but
  - DL as surrogate model (whole simulation, or sub-components)
  - Physic Informed Deep Learning
  - Deep Galerkin Method (high dimensions)
- Inverse problems / calibration
  - Often ill-posed
- Mechanistic model is unknown
  - Learn an analytical model from data
  - Learn a black-box model from data
Data-driven Parametric Model Identification

Dictionary-based Learning

\[ u_t = N(t, x, u, u_x, u_{xx}, \ldots) \]

- Sparse optimisation from a set of primitives

\[
N(t, x, u, u_x, u_{xx}, \ldots) = a_{0,0} + a_{1,0}u + a_{2,0}u^2 + a_{3,0}u^3 + \\
a_{0,1}u_x + a_{1,1}uu_x + a_{2,1}u^2u_x + a_{3,1}u^3u_x + \\
a_{0,2}u_{xx} + a_{1,2}uu_{xx} + a_{2,2}u^2u_{xx} + a_{3,2}u^3u_{xx} + \\
a_{0,3}u_{xxx} + a_{1,3}uu_{xxx} + a_{2,3}u^2u_{xxx} + a_{3,3}u^3u_{xxx}
\]

- Pros: intelligibility
- Cons:
  - Numerical differentiation is unstable
  - Completeness of the set of primitives

<table>
<thead>
<tr>
<th>PDE</th>
<th>Form</th>
<th>Error (no noise, noise)</th>
<th>Discretization</th>
</tr>
</thead>
<tbody>
<tr>
<td>KdV</td>
<td>$u_t + 6uu_x + u_{xxx} = 0$</td>
<td>$1% \pm 0.2%, 7% \pm 5%$</td>
<td>$x \in [-30, 30], n=512, t \in [0, 20], m=201$</td>
</tr>
<tr>
<td>Burgers</td>
<td>$u_t + uu_x - \epsilon u_{xx} = 0$</td>
<td>$0.15% \pm 0.06%, 0.8% \pm 0.6%$</td>
<td>$x \in [-8, 8], n=256, t \in [0, 10], m=101$</td>
</tr>
<tr>
<td>Schrödinger</td>
<td>$iu_t + \frac{1}{2}u_{xx} - \frac{x^2}{2}u = 0$</td>
<td>$0.25% \pm 0.01%, 10% \pm 7%$</td>
<td>$x \in [-7.5, 7.5], n=512, t \in [0, 10], m=401$</td>
</tr>
<tr>
<td>NLS</td>
<td>$iu_t + \frac{1}{2}u_{xx} +</td>
<td>u</td>
<td>^2u = 0$</td>
</tr>
<tr>
<td>KS</td>
<td>$u_t + uu_x + u_{xx} + u_{xxx} = 0$</td>
<td>$1.3% \pm 1.3%, 70% \pm 27%$</td>
<td>$x \in [0, 100], n=1024, t \in [0, 100], m=251$</td>
</tr>
<tr>
<td>Reaction</td>
<td>$u_t = 0.1\nabla^2 u + \lambda(A)u - \omega(A)v$</td>
<td>$0.02% \pm 0.01%, 3.8% \pm 2.4%$</td>
<td>$x, y \in [-10, 10], n=256, t \in [0, 10], m=201$ subsample 1.14%</td>
</tr>
<tr>
<td>Diffusion</td>
<td>$v_t = 0.1\nabla^2 v + \lambda(A)u + \lambda(A)v$</td>
<td>$A^2 = u^2 + v^2, \omega = -\beta A^2, \lambda = 1 - A^2$</td>
<td>$x, y \in [-10, 10], n=256, t \in [0, 10], m=201$ subsample 1.14%</td>
</tr>
<tr>
<td>Navier</td>
<td>$\omega_t + (u \cdot \nabla)\omega = \frac{1}{Re} \nabla^2 \omega$</td>
<td>$1% \pm 0.2%, 7% \pm 6%$</td>
<td>$x \in [0, 9], n_x=449, y \in [0, 4], n_y=199, t \in [0, 30], m=151$, subsample 2.22%</td>
</tr>
</tbody>
</table>

Data-driven Free-form Model Identification

Genetic Programming

- An Evolutionary Algorithm
- evolving tree structures
- representing programs, functions, …

- Can explore huge unstructured search spaces
- and discover innovative solutions
- without a template
Identification of 1-D rheological models

- Data: Strain measures at discrete time steps

- Use of GP for rheological models

\[ \sigma_{sim}(t) = F(\varepsilon(t), \dot{\varepsilon}(t); k_1, k_2, k_3, \eta, \sigma_S) \]

**Difficulties**

- Need an *interpreter* of rheological models
- Computational cost: gradually take into account the experiments

Data-driven Free-form Model Identification

Identification of 1-D rheological models: Results

- → **Active Learning**: lack of creep in the experiment
- Also, identification of 3D hyper-elastic law

Frequently identified model with its “experimental” curve

Learning Dynamical Systems with Genetic Programming

- Direct identification of dynamical systems from time series
- with several computational tricks
  - Partitioning the variables
  - Snipping (anti-bloat)
  - Active learning (unrealistic)
- Good results on synthetic and real systems
- but many trials were unsuccessful

Data-driven Free-form Model Identification


<table>
<thead>
<tr>
<th>System</th>
<th>Target Equation</th>
<th>Best Model Equation</th>
<th>Median Model Equation</th>
</tr>
</thead>
</table>
| Single pendulum   | \( \frac{d\theta}{dt} = \omega \)  
                   \( \frac{d\omega}{dt} = -9.8\sin(\theta) \)  
                   \( \frac{d\theta}{dt} = \omega \)  
                   \( \frac{d\omega}{dt} = -9.79987\sin(\theta) \)  
                   \( \frac{d\theta}{dt} = \omega \)  
                   \( \frac{d\omega}{dt} = -9.8682\sin(\theta) \)  |
| Lotka–Volterra    | \( \frac{dx}{dt} = 3x - 2xy - x^2 \)  
                   \( \frac{dy}{dt} = 2y - xy - y^2 \)  
                   \( \frac{dx}{dt} = 3.0014x - 2xy - x^2 \)  
                   \( \frac{dy}{dt} = 2.0001y - xy - y^2 \)  
                   \( \frac{dx}{dt} = 2.9979x - 2.0016xy - x^2 \)  
                   \( \frac{dy}{dt} = 1.999y - 0.917xy - 1.005y^2 \)  |
Data-driven Free-form Model Identification

- Identifies invariants from videos of experiments

- To avoid trivial invariants
  - check partial derivatives
  - for all pairs of variables
  - w.r.t. numerical derivatives

- Generate Pareto front
  - accuracy vs parsimony

- Keep best-looking equations :-)

Impressive results
Units of constants by varying the parameters (e.g., mass)
But requires human knowledge
  - Choice of variables
  - Choice of operators
  - Choice of Pareto solution
and GP poorly scales up

ML and PDEs: Agenda

● Simulation is fine, but huge output data
  ○ DL for Data Analytics (Climate, Particle Physics, …)
● Simulation is fine, but
  ○ DL as surrogate model (whole simulation, or sub-components)
  ○ Physic Informed Deep Learning
  ○ Deep Galerkin Method (high dimensions)
● Inverse problems / calibration
  ○ Often ill-posed
● Mechanistic model is unknown
  ○ Learn an analytical model from data
  ○ Learn a black-box model from data
Identifying both the model and the solution from

Data-driven Black-box Model Identification

Identifying both the model and the solution

Data-driven Model Identification

Identifying both the model and the solution (2)

\[ u_t = \mathcal{N}(t, x, u, u_x, u_{xx}, \ldots) \]

- A Deep Network for both \( u \) and \( \mathcal{N} \)
- Deduced DN for
- Sample (many) data points \((t_i, x_i, u_i)\) \(i=1,\ldots, N\)
- Goal: minimize

\[
\sum_{i=1}^{N} \left( |u(t^i, x^i) - u^i|^2 + |f(t^i, x^i)|^2 \right)
\]

Data-driven Model Identification

Identifying both the model and the solution (3)

**Example**: Burgers’ equation

- $\text{DN}_u$: 5 layers, 50 neurons/layer, sine activation function
- $\text{DN}_N$: 2 layers, 100 neurons/layer, sine a.f.

- “Exact” solution from 4\textsuperscript{th} order Runge-Kutta, time step $10^{-4}$
- **Examples**: 201 snapshots in time
- Train set: 10000 random points for $t \in [0,6.67]$
- Tested for $t \in [6.67,10]$

$$u_t = -uu_x + 0.1u_{xx}$$

Data-driven Model Identification

Assessing the learned dynamics

On the learned equation

\[ u(0, x) = -\sin(\pi x/8) \]

with a different initial condition

\[ u(0, x) = -\exp(-(x + 2)^2) \]

Data-driven Model Identification

Assessing the learned dynamics (2)

with a different initial condition

\[ u(0, x) = - \sin\left(\frac{\pi x}{8}\right) \]

On the learned equation

\[ u(0, x) = - \exp(- (x + 2)^2) \]

From PDE to NNs

- **ResNet module:** $x$ is progressively modified by the residual $f(x, \theta)$
  
  $x_{t+1} = x_t + f(x_t, \theta_t)$
  
  for small $h$ this is the forward Euler scheme


- **ODEnet:**

  Optimization problem:
  
  $\min_{\theta} L(F(x, \theta), y)$ s.t. $\dot{x} = F(x(t), \theta)$
  
  use Lagrangian optimization

  $L(F(x(T), \theta), y) - \int_0^T \lambda(t)(\dot{x} - F(x(t), \theta))$
Conclusions

● Some impressive results (even if on small regular problems)
  ○ Synergy with HPC
  ○ Surrogate modeling
  ○ Meshless simulations

● Still underexploited
  ○ The generative power of DNNs (GANs)
  ○ Transfer learning and domain adaptation (DANNs)
  ○ Graph networks

● Open issues
  ○ Where do the data come from?
  ○ How noisy are they?
  ○ Small data: PDEs as constraints?

How to hybridize Machine Learning and Mechanistic Knowledge?