Methods and Applications of Analysis.
Vol. 26, No. 3, pp. 281–290, September 2019

PARAMETER IDENTIFICATION OF A FLUID-STRUCTURE SYSTEM BY DEEP-LEARNING WITH AN EULERIAN FORMULATION*

OLIVIER PIRONNEAU†

Proceedings of the Conference in Hong-Kong in June 2017
in Honor of Roland Glowinski's 80th Birthday

Abstract. A simple fluid-structure problem is considered as a test to assess the feasibility of deep-learning algorithms for parameter identification. Tensorflow by Google is used and as it is a stochastic algorithm, provision must be made for the robustness of the large displacement fluid-structure simulator with respect to a wide range of values for the Lamé coefficients and the density of the solid. Hence an Eulerian monolithic solver is introduced. The numerical tests validate the deep-learning approach.

Key words. Parameter identification, Fluid-Structure Interaction, Genetic Algorithm, Neural Network, Deep Learning.

Mathematics Subject Classification. 65N30,68T01, 90C15, 35Q30.

1. Introduction.

Tensorflow is an open-source software by Google. It is the result of ten years experience of deep-learning algorithms initially motivated by recognition and classification of images. Two well-known tests are used to rank the various neural networks and formulations: MINIST and CIFAR10 (see for example [9]).

Identification of parameters from the images produced by numerical simulation is an interesting idea worth testing, different from earlier approaches using neural networks like [6].

This means then dealing with a large collection of images generated by computer simulations with a wide range of values for the parameters of the model.

Here we consider a simple fluid-structure problem: a cavity filled with a Newtonian fluid with a lid made of a thick deformable structure. The bottom boundary of the fluid box slides as in the driven cavity problem, thus generating a large eddy which induces a deformation of the lid. Can we recover the parameters of the structure (the lid) by observing its deformation?

The main parameters are the two Lamé coefficients and the density of the material. If all 3 coefficients are sought then a standard least square approach gives rise to an optimal control problem which can be solved by any optimization method, differentiable or stochastic. But Deep-Learning offers an interesting alternative: it allows the recovery of only one of the 3 coefficients without prior knowledge of the two others.

Stochastic optimization algorithms require a robust numerical solver of the PDE system because the parameters are given wild values by the software and the solver should not crash. It turns out that Euler formulations for fluid-structure interaction (FSI) are very robust and it is for this reason that we have coupled it in this study with CMAES and Tensorflow.

*Received December 25, 2017; accepted for publication July 30, 2019.

†University of Paris VI (UPMC), Laboratoire J.-L. Lions (LJLL), Boite 187, place Jussieu, F-75252 Paris cedex 5, France (olivier.pironneau@gmail.com).

Hence the paper has two parts. In the first part we recall our work on the Eulerian formulation of FSI (see also [2]) and in the second part we apply Deep-Learning for the recovery of the parameters.

2. Eulerian Fluid-Structure Systems. In a Lagrangian framework the laws of elasticity are written for the motion $\mathbf{X}(x_0,t)$ of a point x_0 in the initial domain Ω_0^s versus time t. Hence the position of the structure at time t is $\Omega_t^s = \mathbf{X}(\Omega_0,t)$. The displacement of the structure is $\mathbf{d}_0(x_0,t) = \mathbf{X}(x_0,t) - x_0$.

In an Eulerian framework the partial differential equations of elasticity are written in Ω_t^s and $\mathbf{d}$ is now a function of $x \in \Omega_t^s$ and t. Since $x_0 = \mathbf{X}^{-1}(x,t)$, we have that the Eulerian displacement is $\mathbf{d}(x,t) = \mathbf{d}_0(\mathbf{X}^{-1}(x,t),t)$.

The main advantage is that we can write a single variational equation for the fluid and the solid, a so-called monolithic formulation. For incompressible neo-Hookean Mooney-Rivlin material it is:

Find $\mathbf{u}, p, \Omega_t$ (velocity, pressure and domain), such that for all $\hat{\mathbf{u}}, \hat{p}$

$$\boxed{\begin{aligned}&\int_{\Omega_t}[\rho\mathbb{D}_t\mathbf{u}\cdot\hat{\mathbf{u}} - p\nabla\cdot\hat{\mathbf{u}} - \hat{p}\nabla\cdot\mathbf{u}]\mathrm{d}x + \int_{\Omega_t^f}\frac{\mu^f}{2}\mathrm{D}\mathbf{u}:\mathrm{D}\hat{\mathbf{u}}\mathrm{d}x\\ &\quad+\int_{\Omega_t^s}c_1(\mathrm{D}\mathbf{d} - \nabla\mathbf{d}\nabla^T\mathbf{d}):\mathrm{D}\hat{\mathbf{u}}\mathrm{d}x = \int_{\Omega_t} f\hat{\mathbf{u}}\mathrm{d}x\end{aligned}}$$

where the displacement of the solid $\mathbf{d}$ is related to its velocity by

$$\mathbb{D}_t\mathbf{d} := \partial_t\mathbf{d} + \mathbf{u}\cdot\nabla\mathbf{d} = \mathbf{u}$$

The notations are

- $\mathrm{D}u = \nabla\mathbf{u} + \nabla^T\mathbf{u}$,
- ρ is the density, given by $\rho = \rho^s\mathbf{1}_{\Omega_t^s} + \rho^f\mathbf{1}_{\Omega_t^f}$ and ρ^s, ρ^f are the densities – assumed constant initially in the solid Ω_0^s and in the fluid Ω_0^f.
- μ^f is the viscosity of the fluid
- c_1 is the Mooney-Rivlin Neo-Hookean coefficient
- $\mathbf{f}$ is the field of volumic external forces

This formulation is valid only in 2D for incompressible materials. For 3D problems there is an additional nonlinear term (see [1]). Compressible materials are discussed below.

2.1. Discretization by the Characteristic-Galerkin Method. Given a time step δt, denoting by u^n the value of u at $t = n\delta t$ and assuming smoothness, a Taylor expansion easily shows that

$$(\partial_t u + a\cdot\nabla u)|_{x,(n+1)\delta t} = \frac{u^{n+1} - u^n o\mathbb{Y}}{\delta t}|_x + O(\delta t)$$

where $\mathbb{Y}(x)$ approximates $\mathcal{X}_{a^n}(n\delta t)$, the solution at $\tau = n\delta t$ of $\frac{d\mathcal{X}}{d\tau}(\tau) = a^n(\mathcal{X}(\tau)),\quad \mathcal{X}((n+1)\delta t) = x$.

A first order approximation is $\mathbb{Y}(x) = x - a^n(x)\delta t$. (see [7] for details). This leads to

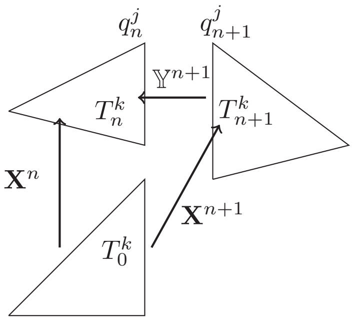


FIG. 1. *Sketch to understand that* $\mathbf{X}^n = \mathbb{Y}^{n+1} \circ \mathbf{X}^{n+1}$*; with* P^1 *elements for all triangles* $T_n^k = \mathbb{Y}^{n+1}(T_{n+1}^k)$.

PROBLEM. Find $\mathbf{u}^{n+1} \in \mathbf{H}_0^1(\Omega_{n+1}), p^{n+1} \in L_0^2(\Omega_{n+1})$, $\Omega_{n+1}^r \subset \mathcal{R}^2$, $r = s, f$, such that $\Omega_{n+1} = \Omega_{n+1}^f \cup \Omega_{n+1}^s$, $\forall \hat{\mathbf{u}} \in \mathbf{H}_0^1(\Omega_{n+1})$, $\forall \hat{p} \in L_0^2(\Omega_{n+1}) = L^2(\Omega_{n+1})/\mathcal{R}$,

$$\int_{\Omega_{n+1}} \Big[\rho_{n+1} \frac{\mathbf{u}^{n+1} - \mathbf{u}^n \circ \mathbb{Y}^{n+1}}{\delta t} \cdot \hat{\mathbf{u}} - p^{n+1} \nabla \cdot \hat{\mathbf{u}} - \hat{p} \nabla \cdot \mathbf{u}^{n+1} + \int_{\Omega_{n+1}^f} \frac{\mu^f}{2} \mathrm{D}\mathbf{u}^{n+1} : \mathrm{D}\hat{\mathbf{u}}$$

$$+ c_1 \int_{\Omega_{n+1}^s} (\mathrm{D}\mathbf{d}^{n+1} - \nabla \mathbf{d}^{n+1} (\nabla \mathbf{d}^{n+1})^T) : \mathrm{D}\hat{\mathbf{u}}\Big] = \int_{\Omega_{n+1}} \mathbf{f} \cdot \hat{\mathbf{u}},$$

$$\mathbf{d}^{n+1} = \mathbf{d}^n \circ \mathbb{Y}^{n+1} + \delta t \mathbf{u}^{n+1},$$

$$\Omega_{n+1} = (\mathbb{Y}^{n+1})^{-1}(\Omega_n) = \{x : \ \mathbb{Y}^{n+1}(x) := x - \mathbf{u}^{n+1}(x)\delta t \in \Omega_n\}.$$

Notice that $\mathbf{d}^{n+1}$ disappears from the formulation and the unknowns are $\mathbf{u}^{n+1}, p^{n+1}$, Ω_{n+1} at each time step.

Discretization in space by the finite element method is done as usual, by choosing compatible approximations for velocities and pressures. Because of the relation between $\mathbf{d}$ and $\mathbf{u}$ it is natural to take for $\mathbf{d}^n$ the same spatial approximation as that of $\mathbf{u}^n$.

Consequently, a large non-linear system is to be solved at each time step. It can be solve by a fixed point algorithm on $\mathbf{u}^{n+1}, p^{n+1}$ with the rest given and updated after each fixed point iteration by their new values. Because of the energy estimates below it can be argued that the fixed-point method converges; our experience shows that 2 iterations are enough. For more details and numerical tests see [4]. We recall the energy inequality obtained in [4] in terms of the Kirchhoff potential of the incompressible material, $\Psi = c_1 \nabla \mathbf{X} : \nabla \mathbf{X} := c_1 \mathrm{tr}(\nabla \mathbf{X} \nabla (\mathbf{X})^T) = c_1 \sum_{ij} (\partial_i \mathbf{X}_j)^2$

THEOREM 1.

$$\int_{\Omega_n} \frac{\rho^n}{2} |\mathbf{u}^n|^2 + \delta t \sum_{k=1}^{n} \int_{\Omega_k^f} \frac{\mu^f}{2} |\mathrm{D}\mathbf{u}^k|^2 + \int_{\Omega_0^s} \Psi^n \leq \int_{\Omega_0} \frac{\rho^0}{2} |\mathbf{u}^0|^2 + \int_{\Omega_0^s} \Psi^0$$

The proof uses the important property of the algorithm – $\mathbf{X}^n = \mathbf{X}^{n+1} \circ \mathbb{Y}^{n+1}$ – which allows by a change of variables to relate the integral of Ψ^n on Ω_n^s with the integral of $\Psi^{n+1} \circ \mathbb{Y}^{n+1}$ on Ω_n^s. This property is preserved by a $P^1 - P^1$-stabilized finite element discretization because triangles are transformed into triangles by the map $\mathbb{Y}^n$ (see Figure 1).

2.2. Saint Venant-Kirchhoff Material. Everything said so far extends to compressible materials with a convex Kirchhoff potential. For hyper-elastic Saint-Venant-Kirchhoff materials, the Kirchhoff potential is : $\Psi = \frac{\lambda^s}{2}(\mathrm{tr}\mathbf{E})^2 + \mu^s \mathrm{tr}\mathbf{E}^2$ with $\mathbf{E} = \frac{1}{2}(\nabla\mathbf{F}^T\nabla\mathbf{F} - \mathbf{I})$. The Eulerian formulation is (see [8]): find $\mathbf{u}, p, \Omega_t$ such that $\forall\hat{\mathbf{u}}, \hat{p}$,

$$\left\{\begin{array}{l} \displaystyle\int_{\Omega_t^f}\Big[\rho\mathbb{D}_t\mathbf{u}\cdot\hat{\mathbf{u}} - p\nabla\cdot\hat{\mathbf{u}} - \hat{p}\nabla\cdot\mathbf{u} + \frac{\mu^f}{2}\mathrm{D}\mathbf{u} : \mathrm{D}\hat{\mathbf{u}}\Big] \\ \displaystyle + \int_{\Omega_t^s}\rho\delta t\Big[b(\mathrm{D}\mathbf{u} - \nabla\bar{\mathbf{d}}\nabla^T\mathbf{u} - \nabla\mathbf{u}\nabla^T\bar{\mathbf{d}}) : \mathrm{D}\hat{\mathbf{u}} + \lambda^s\nabla\cdot\mathbf{u}\ \nabla\cdot\hat{\mathbf{u}}\Big] \\ \displaystyle + \int_{\Omega_t^s}\rho\Big[\mathbb{D}_t\mathbf{u}\cdot\hat{\mathbf{u}} + b(\mathrm{D}\bar{\mathbf{d}} - \nabla\bar{\mathbf{d}}\nabla^T\bar{\mathbf{d}}) : \mathrm{D}\hat{\mathbf{u}} + (c + \lambda^s\nabla\cdot\bar{\mathbf{d}})\nabla\cdot\hat{\mathbf{u}}\Big] = \int_{\Omega_t} f\cdot\hat{\mathbf{u}} \\ \mathbb{D}_t\mathbf{d} = \mathbf{u},\ \bar{\mathbf{d}} = \mathbf{d} - \delta t\mathbf{u},\ \rho = \rho_0^s \mathrm{det}_{\mathbf{I}-\nabla\mathbf{d}}\mathbf{1}_{\Omega_t^s} + \rho^f\mathbf{1}_{\Omega_t^f}, \end{array}\right. \tag{1}$$

with

$$\begin{aligned} &\gamma = (2 - 2\nabla\cdot\mathbf{d} + |\nabla\mathbf{d}|^2)J^2,\ \tilde{\gamma} = \gamma J^{-2} \\ &c = \lambda^s(\frac{1}{2}\gamma - 1)(\tilde{\gamma} - 1) + \mu^s(\gamma - J^2 - 1)\tilde{\gamma} - \lambda^s\nabla\cdot\mathbf{d}, \\ &b = \frac{1}{2}(\frac{\lambda^s}{2} + \mu^s)(\gamma - 1) - \frac{\lambda^s}{4} \end{aligned}$$

Here linear elasticity is visible because $b = \frac{\mu^s}{2} + O(u)$ and $c = O(u)$.

2.3. FSI with Linear Elasticity on a Fixed Domain. We shall study a bidimensional fluid-structure problem where the fluid is initially in a square container $(0, 10) \times (0, 10)$ with no-slip conditions on the walls and the solid is an elongated rectangle $(0, 10) \times (10, 11)$ above the fluid like a lid (Figure 2). The computational domain Ω is the union of the solid and fluid domains. The fluid-solid interface is the upper boundary of the square. The lower boundary of the liquid container slides horizontally as for the driven cavity problem; it generates an eddy inside the square cavity which deforms the lid due to viscous and pressure effects on the fluid-solid interface. If we neglect the time dependency of Ω and if we keep only the first order terms in $\mathbf{u}$, we find that $\mathbf{u}$ and p are given by:

$$\left\{\begin{array}{l} \displaystyle\int_{\Omega^f}\Big[\rho\mathbb{D}_t\mathbf{u}\cdot\hat{\mathbf{u}} - p\nabla\cdot\hat{\mathbf{u}} - \hat{p}\nabla\cdot\mathbf{u} + \frac{\mu^f}{2}\mathrm{D}\mathbf{u} : \mathrm{D}\hat{\mathbf{u}}\Big] \\ \displaystyle + \int_{\Omega^s}\rho\delta t\Big[\frac{\mu^s}{2}\mathrm{D}\mathbf{u} : \mathrm{D}\hat{\mathbf{u}} + \lambda^s\nabla\cdot\mathbf{u}\ \nabla\cdot\hat{\mathbf{u}}\Big] \\ \displaystyle = -\int_{\Omega^s}\rho\Big[\mathbb{D}_t\mathbf{u}\cdot\hat{\mathbf{u}} + \frac{\mu^s}{2}\mathrm{D}\mathbf{d} : \mathrm{D}\hat{\mathbf{u}} + \lambda^s\nabla\cdot\bar{\mathbf{d}}\nabla\cdot\hat{\mathbf{u}}\Big] + \int_{\Omega} f\cdot\hat{\mathbf{u}} \\ \mathbb{D}_t\mathbf{d} = \mathbf{u},\ \rho = \rho_0. \end{array}\right. \tag{2}$$

In the sections bellow, when we refer to “linear elasticity” we solve the above linear elastic formulation on a fixed rectangular domain. Yet even though Ω is fixed, an illusion of its motion is created by moving it with the velocity $\mathbf{u}$. This is what is done below; It is fast because it involves only the solution of a linear system at each step; but it is valid only for small displacements.

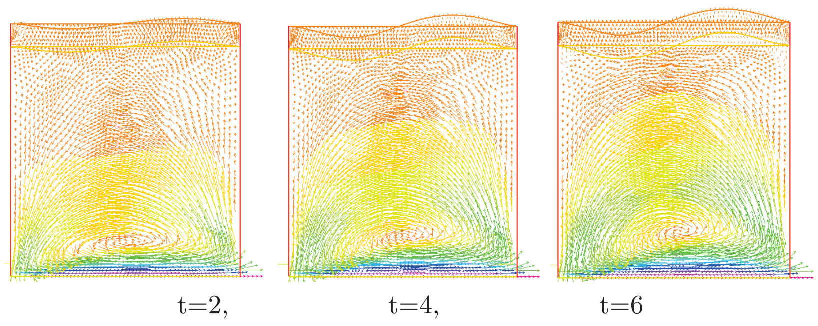


FIG. 2. *A test case showing oscillations of the top boundary of the lid driven by the large eddy, itself created by the sliding bottom boundary. The velocity vectors are shown, colored by their modulus*

3. Inverse Problems. We observe $x,t \in \partial\Omega \times [0,T] \mapsto d(x,t)$; can we recover λ^s, μ^s, ρ^s ? Recall that λ^s and μ^s are related to the Young modulus E and the Poisson ration σ by

$$\mu^s = \frac{E}{2(1+\sigma)}, \quad \lambda^s = \frac{E\sigma}{(1+\sigma)(1-2\sigma)};$$

All simulations below are done with FreeFem++ [3] and $\rho^f = 1$, $\mu^f = 0.25$. Initially the system is at rest and it is simulated up to time T=20 with 50 time steps. The square is discretized uniformly with 400 vertices. The lid is discretized with 100 vertices. The lower boundary of the liquid box slides horizontally with velocity 0.5. The gravity is chosen to be -0.01. All these numbers have been chosen somewhat arbitrary to obtain reasonably moderate yet visible oscillations.

3.1. Stochastic Optimization with CMAES. CMAES is a stochastic global optimizer written by N. Hansen [5]; it is fairly easy to use as it requires only a function which returns the criteria to be minimized, given the parameters.

3.1.1. Linear Elasticity. First it is done with linear elasticity (2). To understand the method let us first comment the results of a simple simulation with parameters $E = 210$, $\sigma = 0.049$, $\rho_s = 0.1$. The results are shown on Figure 2 at time 2, 4, and 6. A quasi periodic regime is established up to time T=20.

So by observing the upper boundary of the lid $\partial\Omega^{top}$ for all x and t can we recover the parameters of the problem?

From a mathematical stand-point the answer is of course yes, because the parameter space is $\mathcal{R}^3$.

Let us begin with CMAES. We chose $E = 250$, $\sigma = 0.0245$, $\rho^s = 0.1$ and ran a simulation with `FreeFem++` to generate a surface: $x,t \in \partial\Omega^{top} \times [0,T] \mapsto \mathbf{d}(x,t)$-exact. Then the parameters are initialized (randomly) at $E = 4.9505$, $\sigma = 0.0485149$, $\rho^s = 0.04$ and `FreeFem++` generates another surface $x,t \in \partial\Omega^{top} \times [0,T] \mapsto \mathbf{d}(x,t)$. CMAES is called to minimized the L^2 time and space distance between $\mathbf{d}$ and $\mathbf{d}$-exact. Hence at each iteration of CMAES whenever the parameters E, σ, ρ^s are changed `FreeFem++` is called to generate a new $\mathbf{d}(x,t)$. This is implemented by writing a function $E, \sigma, \rho^s \mapsto \{x,t \in \partial\Omega^{top} \times [0,T] \mapsto \mathbf{d}(x,t)\}$.

An instruction in Freefem++ like

```
    real minimum = cmaes(J,vv,stopTolFun=0.1e-2,stopMaxFunEval=100000,stopMaxIter=50);
```

generates the following output:

```
Exact solution  E=250  sigma= 0.0245 rhos= 0.1
==============================================
Initial start  E=4.9505  sigma= 0.0485149 rhos= 0.04

CMA@-ES(mu_eff=2.3), Ver="3.11", dimension=3, diagonalIter=0, randomSeed=417174088
Intermediate cost= 4.56954
...
Intermediate cost= 0.000340451
...
Intermediate cost= 1.70237e-07
Intermediate cost= 0.000896846
Stop : TolFun: function value differences 8.22e-03 < stopTolFun=1.00e-02

Number of fitness evalution(s) : 42   minimum= 1.70237e-07

E=249.154  sigma= 0.0485479 rhos= 0.0750725
============================================================
```

The initial values of the parameters were taken at random, far from the optimum values.

The results are barely acceptable unless a large number of iterations are made; other tests where one or two parameters are fixed at their desired values and two or one determined by CMAES give better results but convergence is slow for the identification of all 3 parameters at once. A possible explanation is that the Poisson coefficient σ does not influence much $\mathbf{d}$ in 2 dimensions, so the problem is stiff.

3.1.2. Large displacement elasticity. A similar exercise has been done with system (1) with a moving geometry. The results are as follows:

```
Exact solution E=508 sigma= 0.02 rhos= 2.5
=========================================
initial start E=800 sigma= 0.02 rhos= 4.80769
===========================================
Intermediate cost= 4.62204
(3,7)-CMA-ES(~=2.3), ~= 11 3.11.00.beta 11 , dimension=3,
diagonalIterations=0, randomSeed=491032307 (Oct14 09:22:00 2017)
Intermediate cost= 4.49854
Intermediate cost= 4.44218
....
Intermediate cost= 4.46141
Intermediate cost= 4.34254
Stop : MaxFunEvals: conducted function evaluations 21 >= 20
Number of fitness evaluation(s): 21
minimum= 3.92605
Solution found E=691.599 sigma= 0.0106398 rhos= 4.82827
==================================================
```

As before the precision is not very good, indicating that the problem is stiff.

3.2. Inverse Problem with Tensorflow. To simplify the problem, we cast it into the framework of MINST. MINST is a character recognition test were 50000 images each containing a single hand written digit, 0..,9, are used to train a convolutional deep neural network. The images are black&white with a low resolution 28x28 and labeled by their content, namely an index $j \in (0, .., 9)$. Once trained the neural network is tested against 10000 new images. The algorithm guesses the content which is compared with the index of the image. The percentage of right answers measures the quality of the algorithm.

To use this technology, we need to pass from the drawn surface $x, t \in [0, 10] \times (0, 20) \mapsto \mathbf{d}(x, t)$ to a 28x28 B&W image. This is done by generating a color jpeg image of the above surface with the gnuplot keywords

`splot matrix with image`.

Then this color picture is downgraded into a 28x28 B&W .png image by the following `python` script:

```
import os
from PIL import Image
for i in range(10):
    path='../tensorflow/ffdata/'+str(i)+'/'
    os.chdir(path)
    for filename in os.listdir(path):
        if filename.endswith(".jpg"):
            Im = Image.open(filename).convert('LA')
            size = 28,28
            Im_resized = Im.resize(size, Image.ANTIALIAS)
            thefilename, thefile_extension
                                = os.path.splitext(filename)
            thefilename += '.png'
            Im_resized.save(thefilename)
```

To give a label to each image we have divided the range of the parameter to be identified into 10 subintervals, labeled, 0,..9. In Figure 3, 3 examples are given.

Thus 990 simulations were performed indexed by 2 integers $j \in (0, 1, ..., 9)$, $k \in (1, .., 99)$; k is used to generate random values for σ and ρ^s. The parameters of the simulations are functions of j and k as follows:

$$H(x) = \frac{x^2}{1+x^2}, \quad E = 2100H(0.1(j+1)), \quad \sigma = 0.49H(0.1k^2), \quad \rho^s = H(0.1k^2)$$

Integer j is used to index the images. 990 minus m images were used to train the network and m images to assess the performance of the algorithm. We chose m randomly around 70. Note that in MNIST the images are stored in compact form and the MINST test programs assumes that the images are compacted in two files, one for the training and one for the evaluation. We complied with this obligation. Furthermore, we used the `keras` environment within `Anaconda` to call the convolution neural network function `cnn` of `tensorflow` (see Figure 4).

The deep neural network is the one given as an example with Keras, unmodified. It has an input layer, 2 convolutions layers separated by two maxpool layers followed by a flatten layer, a dense layer, a drop-out layer and finally a last dense output layer. The total number of parameters is 843658. The configurations of the layers are visible on fig 4. Most of the time 50 iterations (epochs) were used.

This structure is certainly unecessarily rich but the purpose of this exercise was not to optimize the network but rather to play with a network that works in a reasonable amount of CPU time; this is the case, no run took more than a minute.

3.2.1. Identification of the Young Modulus using (2). First let us apply this strategy to the identification of the Young modulus E. Here $m = 73$. The results are shown on Figure 5. They show that the label that gives E is recovered with 96.6% probability. This means that E is recovered with 10% precision with this probability without knowing anything about the Poisson modulus σ and the density ρ^s.

The cause of this strikingly good result is evident on Figure 3: it is E which influences the period of the oscillations and at the level of the reduced B&W images this is still clear and seemingly only mildly dependent of σ and ρ^s.

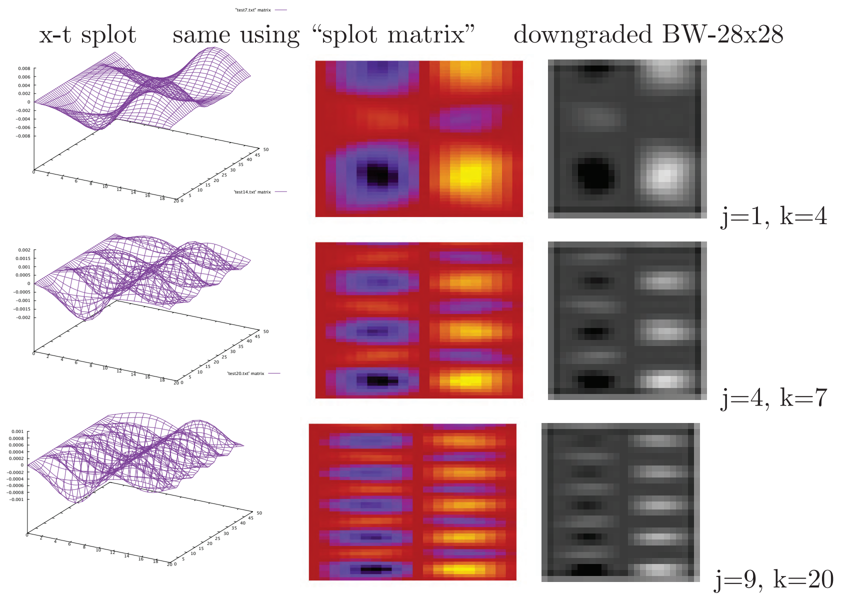
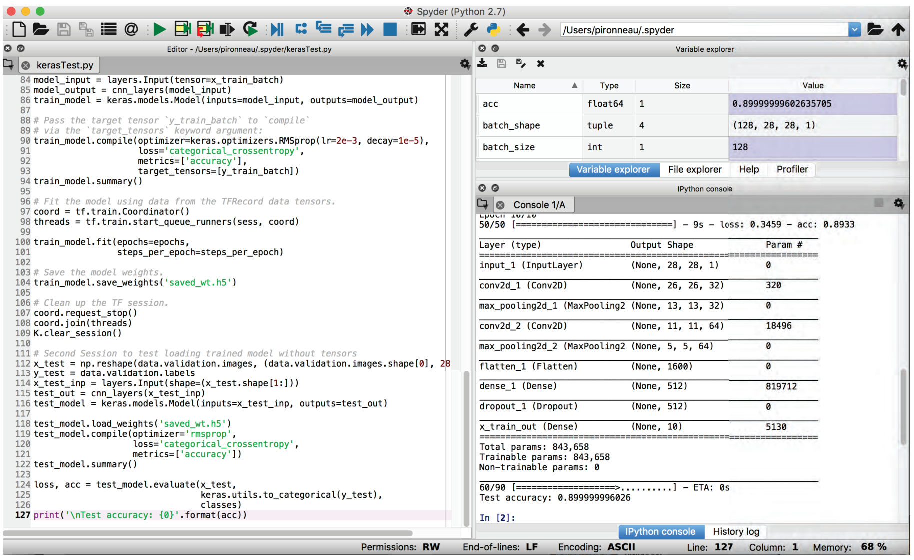


FIG. 3. *From surfaces $x, t \mapsto d(x,t)$ to 28x28 black&white images*



```
model_input = layers.Input(tensor=x_train_batch)
model_output = cnn_layers(model_input)
train_model = keras.models.Model(inputs=model_input, outputs=model_output)

# Pass the target tensor `y_train_batch` to `compile`
# via the `target_tensors` keyword argument:
train_model.compile(optimizer=keras.optimizers.RMSprop(lr=2e-3, decay=1e-5),
                    loss='categorical_crossentropy',
                    metrics=['accuracy'],
                    target_tensors=[y_train_batch])
train_model.summary()

# Fit the model using data from the TFRecord data tensors.
coord = tf.train.Coordinator()
threads = tf.train.start_queue_runners(sess, coord)

train_model.fit(epochs=epochs,
                steps_per_epoch=steps_per_epoch)

# Save the model weights.
train_model.save_weights('saved_wt.h5')

# Clean up the TF session.
coord.request_stop()
coord.join(threads)
K.clear_session()

# Second Session to test loading trained model without tensors
x_test = np.reshape(data.validation.images, (data.validation.images.shape[0], 28
y_test = data.validation.labels
x_test_inp = layers.Input(shape=(x_test.shape[1:]))
test_out = cnn_layers(x_test_inp)
test_model = keras.models.Model(inputs=x_test_inp, outputs=test_out)

test_model.load_weights('saved_wt.h5')
test_model.compile(optimizer='rmsprop',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])
test_model.summary()

loss, acc = test_model.evaluate(x_test,
                                keras.utils.to_categorical(y_test),
                                classes)
print('\nTest accuracy: {0}'.format(acc))
```

Name	Type	Size	Value
acc	float64	1	0.8999999602635705
batch_shape	tuple	4	(128, 28, 28, 1)
batch_size	int	1	128

```
Epoch 10/10
50/50 [==============================] - 9s - loss: 0.3459 - acc: 0.8933
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               819712
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
x_train_out (Dense)          (None, 10)                5130
=================================================================
Total params: 843,658
Trainable params: 843,658
Non-trainable params: 0
_________________________________________________________________
60/90 [===================>..........] - ETA: 0s
Test accuracy: 0.899999996026

In [2]:
```

```
Permissions: RW   End-of-lines: LF   Encoding: ASCII   Line: 127   Column: 1   Memory: 68 %
```

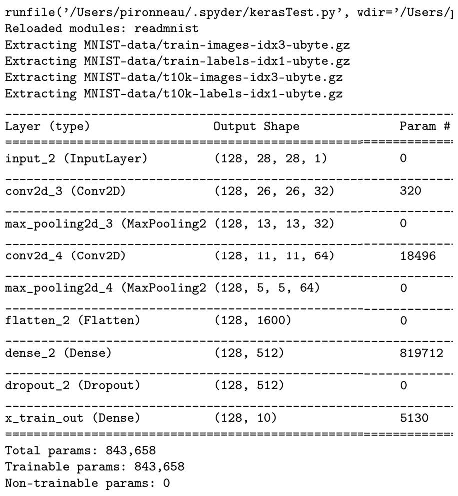
FIG. 4. *Using* `tensorflow` *from with* `keras` *within* `Anaconda+spyder`

3.2.2. Identification of the density. Using (2), let the images generated with the following parameters

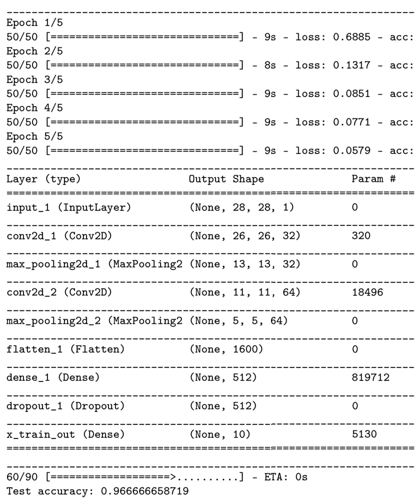
$$E = 2100H(0.1k), \quad \sigma = 0.49H(0.1k^2), \quad \rho^s = H(0.1(j+1)^2)$$

The results are just a little better than uniform randomness with a probability of success of 15%(see Figure 7); evidently varying the density does not create striking differences in the images.

Finally we do the same test but using the full large displacement formulation

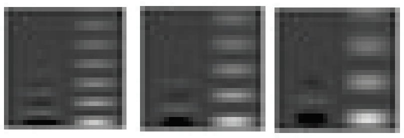


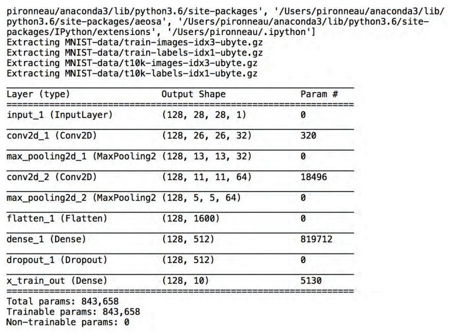
```
runfile('/Users/pironneau/.spyder/kerasTest.py', wdir='/Users/
Reloaded modules: readmnist
Extracting MNIST-data/train-images-idx3-ubyte.gz
Extracting MNIST-data/train-labels-idx1-ubyte.gz
Extracting MNIST-data/t10k-images-idx3-ubyte.gz
Extracting MNIST-data/t10k-labels-idx1-ubyte.gz
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (128, 28, 28, 1)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (128, 26, 26, 32)         320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (128, 13, 13, 32)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (128, 11, 11, 64)         18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (128, 5, 5, 64)           0
_________________________________________________________________
flatten_2 (Flatten)          (128, 1600)               0
_________________________________________________________________
dense_2 (Dense)              (128, 512)                819712
_________________________________________________________________
dropout_2 (Dropout)          (128, 512)                0
_________________________________________________________________
x_train_out (Dense)          (128, 10)                 5130
=================================================================
Total params: 843,658
Trainable params: 843,658
Non-trainable params: 0
```



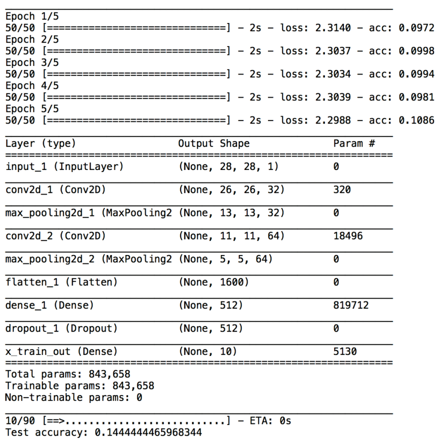
```
_________________________________________________________________
Epoch 1/5
50/50 [==============================] - 9s - loss: 0.6885 - acc: 0.791
Epoch 2/5
50/50 [==============================] - 8s - loss: 0.1317 - acc: 0.965
Epoch 3/5
50/50 [==============================] - 9s - loss: 0.0851 - acc: 0.977
Epoch 4/5
50/50 [==============================] - 9s - loss: 0.0771 - acc: 0.978
Epoch 5/5
50/50 [==============================] - 9s - loss: 0.0579 - acc: 0.984
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               819712
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
x_train_out (Dense)          (None, 10)                5130
=================================================================

_________________________________________________________________
60/90 [===================>..........] - ETA: 0s
Test accuracy: 0.966666658719
```

FIG. 5. *Identification of the Young modulus using tensorflow and linear elasticity (2)*





```
pironneau/anaconda3/lib/python3.6/site-packages', '/Users/pironneau/anaconda3/lib/
python3.6/site-packages/aeosa', '/Users/pironneau/anaconda3/lib/python3.6/site-
packages/IPython/extensions', '/Users/pironneau/.ipython']
Extracting MNIST-data/train-images-idx3-ubyte.gz
Extracting MNIST-data/train-labels-idx1-ubyte.gz
Extracting MNIST-data/t10k-images-idx3-ubyte.gz
Extracting MNIST-data/t10k-labels-idx1-ubyte.gz
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (128, 28, 28, 1)          0
_________________________________________________________________
conv2d_1 (Conv2D)            (128, 26, 26, 32)         320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (128, 13, 13, 32)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (128, 11, 11, 64)         18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (128, 5, 5, 64)           0
_________________________________________________________________
flatten_1 (Flatten)          (128, 1600)               0
_________________________________________________________________
dense_1 (Dense)              (128, 512)                819712
_________________________________________________________________
dropout_1 (Dropout)          (128, 512)                0
_________________________________________________________________
x_train_out (Dense)          (128, 10)                 5130
=================================================================
Total params: 843,658
Trainable params: 843,658
Non-trainable params: 0
```



```
_________________________________________________________________
Epoch 1/5
50/50 [==============================] - 2s - loss: 2.3140 - acc: 0.0972
Epoch 2/5
50/50 [==============================] - 2s - loss: 2.3037 - acc: 0.0998
Epoch 3/5
50/50 [==============================] - 2s - loss: 2.3034 - acc: 0.0994
Epoch 4/5
50/50 [==============================] - 2s - loss: 2.3039 - acc: 0.0981
Epoch 5/5
50/50 [==============================] - 2s - loss: 2.2988 - acc: 0.1086
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               819712
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
x_train_out (Dense)          (None, 10)                5130
=================================================================
Total params: 843,658
Trainable params: 843,658
Non-trainable params: 0
_________________________________________________________________
10/90 [==>...........................] - ETA: 0s
Test accuracy: 0.1444444465968344
```

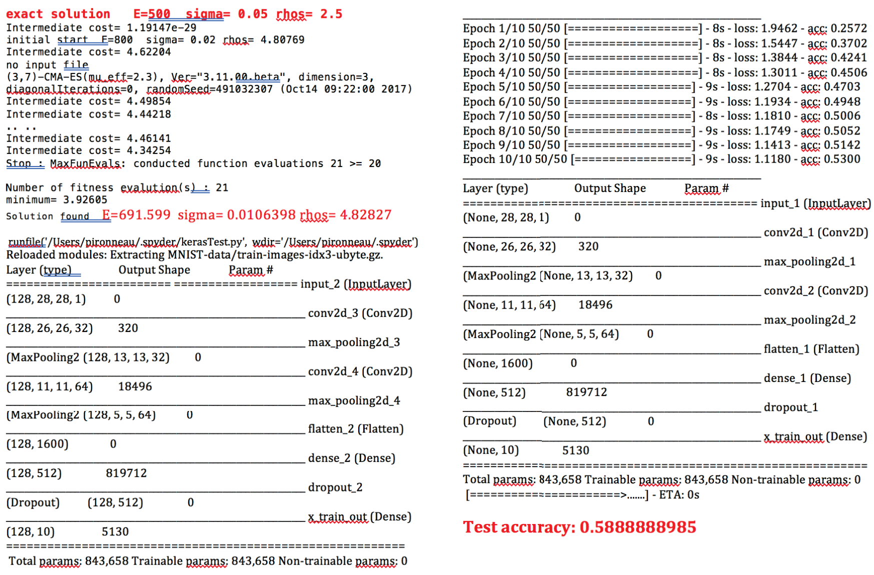
FIG. 6. *Identification of the density of the solid with tensorflow and linear elasticity (2)*

using (1). The results, in Figure 6, show that the images generated with this Eulerian model are much more sensitive to ρ^s, yielding a 59% probability of identifying the solid density with 10% precision. The performance of CMAES on this test is shown also on Figure 7.

REFERENCES

[1] C.-Y. CHIANG, O. PIRONNEAU, T. W. H. SHEU, AND M. THIRIET, *Numerical Study of a 3D Eulerian Monolithic Formulation for Incompressible Fluid-Structures Systems*, Fluids 2017, 2(2), 34; MDPI. Basel.

[2] TH. DUNNE AND R. RANNACHER, *Adaptive Finite Element Approximation of Fluid-Structure Interaction Based on an Eulerian Variational Formulation.* In "Fluid-Structure Interaction: Modelling, Simulation, Optimization". Lecture Notes in Computational Science and Engineering, vol 53., pp. 110–146, H-J. Bungartz, M. Schaefer eds. Springer 2006.



```
exact solution    E=500  sigma= 0.05 rhos= 2.5
Intermediate cost= 1.19147e-29
initial start  E=800  sigma= 0.02 rhos= 4.80769
Intermediate cost= 4.62204
no input file
(3,7)-CMA-ES(mu_eff=2.3), Ver="3.11.00.beta", dimension=3,
diagonalIterations=0, randomSeed=491032307 (Oct14 09:22:00 2017)
Intermediate cost= 4.49854
Intermediate cost= 4.44218
.. ..
Intermediate cost= 4.46141
Intermediate cost= 4.34254
Stop : MaxFunEvals: conducted function evaluations 21 >= 20

Number of fitness evalution(s) : 21
minimum= 3.92605
Solution found  E=691.599  sigma= 0.0106398 rhos= 4.82827

runfile('/Users/pironneau/.spyder/kerasTest.py', wdir='/Users/pironneau/.spyder')
Reloaded modules: Extracting MNIST-data/train-images-idx3-ubyte.gz.
Layer (type)          Output Shape          Param #
========================= =================== input_2 (InputLayer)
(128, 28, 28, 1)      0
___________________________________________________ conv2d_3 (Conv2D)
(128, 26, 26, 32)      320
___________________________________________________ max_pooling2d_3
(MaxPooling2 (128, 13, 13, 32)      0
___________________________________________________ conv2d_4 (Conv2D)
(128, 11, 11, 64)      18496
___________________________________________________ max_pooling2d_4
(MaxPooling2 (128, 5, 5, 64)      0
___________________________________________________ flatten_2 (Flatten)
(128, 1600)       0
___________________________________________________ dense_2 (Dense)
(128, 512)        819712
___________________________________________________ dropout_2
(Dropout)      (128, 512)        0
___________________________________________________ x_train_out (Dense)
(128, 10)         5130
=============================================================
Total params: 843,658 Trainable params: 843,658 Non-trainable params: 0
```

```
_____________________________________________________
Epoch 1/10 50/50 [====================] - 8s - loss: 1.9462 - acc: 0.2572
Epoch 2/10 50/50 [====================] - 8s - loss: 1.5447 - acc: 0.3702
Epoch 3/10 50/50 [====================] - 8s - loss: 1.3844 - acc: 0.4241
Epoch 4/10 50/50 [====================] - 8s - loss: 1.3011 - acc: 0.4506
Epoch 5/10 50/50 [===================] - 9s - loss: 1.2704 - acc: 0.4703
Epoch 6/10 50/50 [===================] - 9s - loss: 1.1934 - acc: 0.4948
Epoch 7/10 50/50 [===================] - 8s - loss: 1.1810 - acc: 0.5006
Epoch 8/10 50/50 [===================] - 9s - loss: 1.1749 - acc: 0.5052
Epoch 9/10 50/50 [===================] - 9s - loss: 1.1413 - acc: 0.5142
Epoch 10/10 50/50 [==================] - 9s - loss: 1.1180 - acc: 0.5300
_____________________________________________________
Layer (type)          Output Shape          Param #
============================================= input_1 (InputLayer)
(None, 28, 28, 1)      0
_____________________________________________________ conv2d_1 (Conv2D)
(None, 26, 26, 32)      320
_____________________________________________________ max_pooling2d_1
(MaxPooling2 (None, 13, 13, 32)      0
_____________________________________________________ conv2d_2 (Conv2D)
(None, 11, 11, 64)      18496
_____________________________________________________ max_pooling2d_2
(MaxPooling2 (None, 5, 5, 64)      0
_____________________________________________________ flatten_1 (Flatten)
(None, 1600)       0
_____________________________________________________ dense_1 (Dense)
(None, 512)        819712
_____________________________________________________ dropout_1
(Dropout)      (None, 512)        0
_____________________________________________________ x_train_out (Dense)
(None, 10)         5130
=============================================================
Total params: 843,658 Trainable params: 843,658 Non-trainable params: 0
[======================>......] - ETA: 0s

Test accuracy: 0.5888888985
```

FIG. 7. *Top left: Identification of all parameters using (1); the results are poor. Below: Identification of the density of the solid only without information on E and σ, with tensorflow and large displacement elasticity (1) can be made with 58.8% probability.*

[3] F. HECHT, *New development in FreeFem++*, J. Numer. Math., 20 (2012), pp. 251–265. `http://www.FreeFem.org`.

[4] F. HECHT AND O. PIRONNEAU, *An energy stable monolithic Eulerian fluid-structure finite element method*, Int. J. Numer. Methods in Fluids, 85:7 (2017), pp. 430-446.

[5] NIKOLAUS HANSEN, *The CMA Evolution Strategy: A Comparing Review*, 2006 in `http://www.lri.fr/~hansen/cmaesintro.html`.

[6] Y. M. A. HASHASH, S. JUNG, AND J. GHABOUSSI, *Numerical implementation of a neural network based material model in finite element analysis*, IJNME 2004.

[7] O. PIRONNEAU, *Finite Element Methods for Fluids*, Wiley, 1989.

[8] O. PIRONNEAU, *An Eulerian Monolithic Fluid-Structure Formulation*, Anniversary volume in honor of P.G. Ciarlet. Li-Ta-Tsien ed. 2018. Also in `http://hal.archives-ouvertes.fr/hal-01348648`.

[9] I. GOODFELLOW, Y. BENGIO, AND A. COURVILLE, *Deep Learning*, MIT-Bradford, 2016.