NPAC computing projects: Teaching a neural network to predict a nuclear potential energy surface

D. Regnier, R.-D. Lasseri

March 11, 2020

Contents

1 Context on Potential Energy Surfaces (PES)

Potential energy surfaces are a key ingredient to our understanding of many multidimensional physics phenomena. They are particularly used in quantum chemistry to give the total energy of the electrons of a molecule as a function of the positions of the atoms. Similarly, they have been used for decades to investigate the dynamics of large collective motions in atomic nuclei. Typical examples of collective motion of the nucleons are the giant dipole resonance or the nuclear fission.

In this context, it is meaningful to look at the total energy of a nuclear system at rest as a function of several variables related to its deformation. The figure 1 highlights such a PES calculated in the context of fission studies. From this PES, we can already determine qualitatively the behavior of the fissioning nucleus. Starting at low deformation in the first potential well, we expect that a sufficiently excited nucleus will mostly follow a path of least potential toward fission. Therefore, we can already deduce from the PES that it will elongate and preferably go toward an asymmetric fission that produces a small and a big fragments, on the contrary to a symmetric fission for which both fragments have the same size.

To determine a PES, the standard approach is to (i) choose a regular mesh for the deformation variables (ii) for each point of this mesh, call a nuclear structure code that

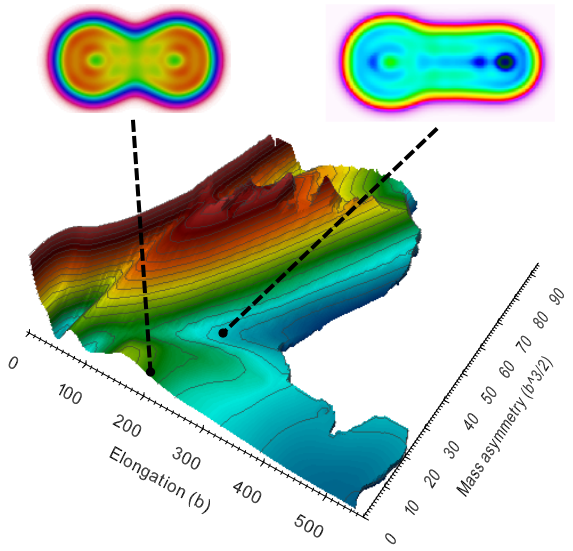

Figure 1: Potential Energy Surface (PES) of the ^{240}Pu as a function of the elongation (quadrupol moment) and the left/right asymmetry (octupol moment). This PES was obtained by $\simeq$ 40000 constrained Hartree-Fock-Bogoliubov calculations in roughly 7.10^4h.cpu.

will give the energy at this deformation. The state of the art studies rely on a constrained Hartree-Fock-Bogoliubov calculation performed at each deformation. Such a calculation typically takes 5-10 min.cpu. Producing a 2-dimensional PES that represents the energy of a nucleus as a function of its elongation and its left/right asymmetry (roughly 40000 points) requires the substantial numerical cost of $\simeq 7.10^3$h.cpu. In our age of supercomputing, this is not such a big deal (it is only a one night calculation on a cluster of 700 cores). Nonetheless, we seek to reduce as much as possible this numerical cost at least for two reasons:

1. To better understand the nucleosynthesis, we would like to compute the PESs of more than 1000 nuclei.

2. The quadrupol β_2 and octupol β_3 moments are not sufficient to describe some exotic shapes of the nuclei. We would like to determine PES as a function of β_2, β_3, *and* some additional variables like β_4, β_5,... This means computing 3D, 4D, etc, potential energy surfaces. The numerical cost increases exponentially with the number of variables.

To face these challenges, the big question is the following one.

Can you build an algorithm that would give the best estimation of the potential energy surface of a nucleus with the minimal numerical cost, *i.e* a minimal number of calls to the nuclear structure code that provides the energy at a given deformation ?

2 Objectives of the project

The goal of this project is to build a neural network that is capable of making a fast and accurate estimation of a PES. Ultimately, your algorithm should be capable of producing automatically an approximatiton of any PES to an arbitrary precision with the minimum number of calls to a nuclear reaction code.

To reach this objective, we strongly advise you to follow these steps:

1. First generate a reference 2-dimensional potential energy surface of the ^{236}U with the microscopic-macroscopic code BARRIER. Take a moment to interpret the physics of your results. Is the PES consistent with the fact that ^{235}U is a fissile isotope. What kind of fragmentation do you expect the most ?

2. Build a simple neural network (one layer perceptron) with the Keras library and train it to reproduce the reference PES. As usual in machine learning, the idea is first to train the neural network with a randomly chosen fraction of points of the reference PES. Then, we test its quality by comparing its prediction to another fraction of points of the PES not used for the training.

3. Optimize the architecture of your neural network. To do so you will have to perform extensive studies of the variation of your results with a set of hyperparameters such as the number of neurons in each layer.

4. Program an active learning procedure based on a committee of neural networks. The committee of neural network should then determine by itself which points are to be used in the training.

5. If you go this far, you may then explore how your algorithm is performing for a 3-dimensional PES, try other active learning algorithms based on the Hessian matrix calculation, find a way to estimate on the fly the uncertainty associated with you neural network prediction and/or solve the same problem based on Gaussian processes instead of neural network. Apart from the numerical aspects, you may also want to focus on producing pretty movies of the active learning procedure for your presentation... At this point, any additional and original idea will be rewarded.

In what follows, we give you some indications related to these steps.

3 BARRIER: a microscopic-macroscopic nuclear structure code

For this project, you will use the code BARRIER to compute the energy of the nucleus as a function of several shape variables. This is an open access Fortran code that can be downloaded from the Computer Physics Communications Program Library at `www.cpc.cs.qub.ac.uk/cpc/summaries/aclx`. The code itself is described in the article [3].

In this code, the atomic nucleus is mostly pictured as a uniformly charged liquid drop with a sharp suface. The shape of the drop is parametrized based on the family of Cassini ovaloids as detailed in Ref. [5]. The Cassinian ovals form a family of shapes that are parametrized by one parameter ϵ that can be seen as the elongation toward fission (cf. Fig. 2). To obtain a richer family of states (that includes for instance an asymmetry between the left and right pre-fragments), the Cassinian ovals are deformed according to a a series of Legendre polynomials (see Eq. (3) of Ref. [5]). At the end, the shape of the

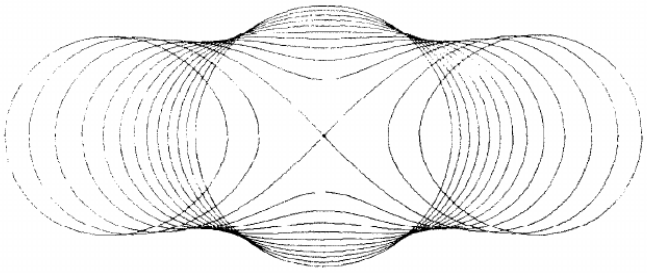

Figure 2: The family of Cassinian oval obtained by varying the parameter ϵ. See Ref. [5] for more details.

nuclear density is parametrized by 7 parameters $(\epsilon, \alpha_1, \cdots, \alpha_6)$. The parameter ϵ defines which Cassinian oval will used. The parameters α_i corresponds to multipolar deformations of the Cassinian oval. For instance α_3 corresponds to an octupol deformation.

For a given shape, the code BARRIER computes the total energy of the system in a semi-classical way.

$$E(\epsilon, \alpha_1, \cdots, \alpha_6) = E_{LD} + E_{sh} + E_{\text{pairing}} \tag{1}$$

The liquid drop part E_{LD} gives the classical energy of a charged deformed system with a part coming from the Coulomb interaction and one coming from the surface tension. The shell correction part is determined by solving the Shrodinger eigen-equation for single nucleons in a mean field determined from the shape. Finally, the pairing correction E_{pairing} is obtained by solving a BCS equation from the previously determined single particle states.

In practice, the BARRIER code has two modes. The first mode computes the energy of one deformation point. We give here an exemple of input for a 236U.

```
&INTDAT IARRAY=0,IPRINT=68,CASSIN=.t.,
  KEYSPE=1,OUTSPE=.T.,TYPERUN=1 /END
&DYNAMC IZ=92,IN=144,ICHOIC=1,IENDD=1,
epsil=0.5,alpha1=0.00,alpha2=0.00,alpha3=0.1,alpha4=-.16D0 /END
```

The calculated energy is called 'barrier' and is produced in a file 'pash.dat'. A second mode directly computes a two dimensional potential energy surface. We give here an example of input for the variables ϵ and α_4.

```
 &INTDAT IARRAY=0,IPRINT=68,CASSIN=.t.,
  KEYSPE=1,OUTSPE=.T.,TYPERUN=2 /END
&DYNAMC IZ=92,IN=144,ICHOIC=2,IENDD=1,epsil=0.10D0,,alpha3=-.00D0,
   vareps=.t.,beg1v=-0.100,step1v=0.020,end1v=0.900,
   varp4=.t., beg2v=0.000,step2v=0.005,end2v=0.100 /END
```

As a first step, compute and plot the potential energy surface of uranium 236 in the ϵ, α_3 space and a fixed value of $\alpha_4 = -0.16$. The typicall range of interest for the octupol deformation is $\alpha_3 < 0.2$. An final but important remark is that the BARRIER code commes with a small bug when downloaded. Don't panic ! Just correct it and compile...

4 Regression with a small neural network

Once you have your reference potential energy surface with a good resolution, the goal is to build and train a simple neural network to emulate the function

$$(\epsilon, \alpha_3) \rightarrow E(\epsilon, \alpha_3). \tag{2}$$

Using the KERAS API of the TensorFlow library, build a simple perceptron network with 500 neurons in the hidden layer. To train this network, you may randomly choose a sample of 50% of the points of your reference PES. This 50% is your training set whereas you will use the other 50% as a validation set to assess the quality of the neural network. With your selected training set, train this simple network. Visualize and interpret the learning curve, the final root mean square error (RMS) on the validation set as well as the prediction of the PES obtained from the neural network.

After this first attempt, you can go for a more complex architecture given by the three hidden layers with 100-100-100 neurons. You should typically reach a RMS of a few 100 keV. What is responsible for such an improvement ?

5 Optimization of the hyperparameters

Now that you have a working neural network, go and try to improve its performance by modifiying its hyperaparameters (i.e its architecture). The hyperparameters you should considere in your study could be (but are not limited to):

- The number of hidden layers
- The number of neurons per layers
- The batch size
- The number of epochs
- The activation functions
- The loss optimizer

Become familiar with the effect of these parameters on your results. How do they affect the learning curves of your neural network ? Can you find some plateau conditions that stabilize the final RMS ? What is the best architecture you retain for this kind of problem ?

6 Active learning

Did you try to perform several time the same training with the same neural network ? It may happen that doing so you find different final RMS. Can you observe and explain this behavior ?

To remedy this standard issue with neural networks one possible solution is to build a committee of neural networks. The idea of a committe is that you have N neural networks that are trained using essentially the same training data. Once this is done, the committee can predict the fitted function as follows:

$$E_{committee}(\epsilon, \alpha_3) = \frac{1}{N} \sum_i E_i(\epsilon, \alpha_3) \tag{3}$$

where E_i is the result obtained by the ith neural network of the committee. The benefit of a committe is that (i) the prediction becomes independant of the fluctuations observed on individual NN, (ii) it can give you an estimate of the variance of the result. As a first to the active learning algorithm, build a committee of 5 identical neural networks and train it on your reference PES. Give an unbiaised estimate of the variance of the committe at each point of the PES as well as the standard deviation associated to the prediction Eq. 3[1] Show that the standard deviation on the estimation of the energy decreases with the number of members of the committee.

At this point you have noticed that the variance of the committe is not uniformly spread on the PES. Some area are more uncertain than others. The idea of active learning is to leverage the fact that the committee gives you a hint of where to put to the efforts to improve. Starting from a very small number of training points for which you computed the energy, the active learning algorithm is the following one:

1. Train you committee from scratch on your current training set.
2. Estimate the variance of the committee members as a function of (ϵ, α_3).
3. Select automatically N_p new points in the areas where the variance is maximal.
4. Compute the energy of these points and add them to the training set.
5. Go back to the first step up to the point where you are satisfyed with the quality of the committee.

Your task is now to build such an algorithm. We suggest to take $N_p = 2\%$ of your data set. The new points can at first be randomly selected among the 20% points of your mesh having the highest variance. You may stop the active learning procedure when the validation RMS reaches roughly 200 keV. How many points did you have to compute to reach such a precision ? How can you improve this number ? In particular, can you find more efficient ways to select the new points after each learning cycle ?

The whole point of active learning is that you can at first only compute the energy of a few points. Then the algorithm tells by itself which points to compute next for the learning to be the most efficient !

A Some notes on neural networks

The purpose of this section is to give a crash course on feed forward neural networks. We will only present here some very basic aspects a very short amount of words. For an introduction on how to code your first neural networks with Keras you should definitely have a look at the book by F. Chollet, *Deep learning with Python* [2]. An open source version of this book is available at https://github.com/veritone/ds-transcription-capstone/blob/master/Deep%20Learning%20With%20Python%20Chollet.pdf. For more formal introductions to the concepts of machine learning we advise the two textbooks Refs. [1, 4].

Mathematicaly speaking, a neural network is simply a function that takes as input a vector of real numbers in $\Re^{N_i}$ and gives as an output another vector of real numbers in $\Re^{N_o}$. This function is built as the successive applications of several simple functions f_i as shown in Fig. 3.

[1] Hint: this is a standard Monte Carlo estimation of the variance of an estimator.

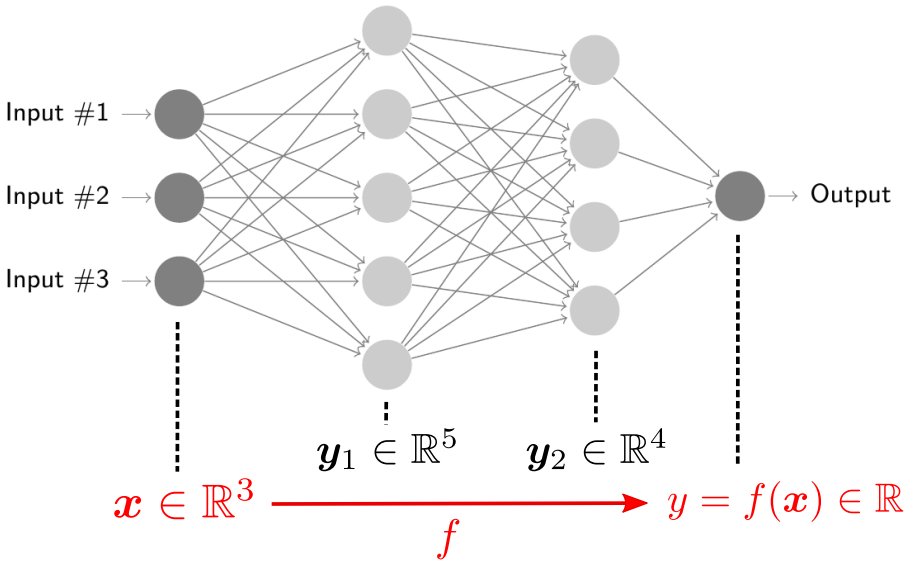

Figure 3: Illustration of a simple neural network. The circles represent real numbers. This particular networks takes as an input 3 reals and gives only one real as an output. Each intermediate vector $\boldsymbol{y}_i$ is determined as a function of the previous layer through a simple function f_i.

The functions $\boldsymbol{y}_i = f_i(\boldsymbol{y_{i-1}})$ are defined as follows:

$$\begin{aligned} \boldsymbol{y}_1 &= f_1(\boldsymbol{x}) && = A_1(W_1 \cdot \boldsymbol{x} + \boldsymbol{b}_1) \\ \boldsymbol{y}_2 &= f_2(\boldsymbol{y}_1) && = A_2(W_2 \cdot \boldsymbol{y}_1 + \boldsymbol{b}_1) \\ y &= f_3(\boldsymbol{y}_2) && = A_3(W_3 \cdot \boldsymbol{y}_2 + \boldsymbol{b}_2) \\ y &= f(\boldsymbol{x}) && = f_3 \circ f_2 \circ f_1(\boldsymbol{x}) \end{aligned}$$

where the W_i and $\boldsymbol{b}_i$ are some matrices and vectors of real parameters and the functions A_i are called activation functions. To put it in a nutshell, the 'layer' of neurons $\boldsymbol{y}_i$ is obtained from the previous layer by (i) first performing a linear transformation of $\boldsymbol{y}_{i-1}$, (ii) then passing the result of this transformation to an arbitrary non linear function that acts the same on all elements of the vector. There are many possible choices for the activation function. A commonly used one is the ReLU (Rectified Linear Unit. We show in Fig. 4 an example of Exponential Linear Unit (ELU).

The number of hidden layers (*i.e.* internal layers), the number of neurons per layer, the choice of activation functions and a few other parameters are called the 'architecture' of the neural network. We also refere to these parameters as the hyperparameters. Once you made the choice of their values, you will have to determine the 'weights' of your neural network which are the W_i matrices and the $\boldsymbol{b}_i$. This is called the training of your network. To train a network, you need a training data set which is an ensembles of pairs $(\boldsymbol{x}_k, y_k)$. With this set, the idea is to fit the weights of the neutwork so that

$$\forall k : \quad f(\boldsymbol{x}_k) \simeq y_k. \tag{4}$$

Training the network is in other words a fit to an established known dataset. It is a regression task. To perform, the fit, the standard approach is:

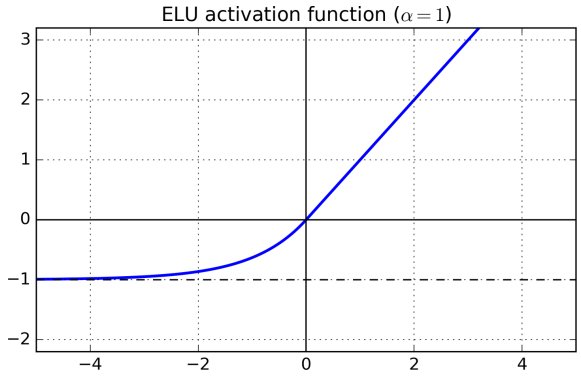

Figure 4: An Exponential Linear Unit (ELU) function. This function can be used as activation function on each element of a vector of real.

1. Define a loss function that estimates the ability of your network to reproduce the points in the training set. For a regression task on continuous outputs we usually use a chi square:
$$L = \frac{1}{K}\sum_{k=1}^{K}(f(\boldsymbol{x}_k) - y_k)^2 \tag{5}$$

2. Optimize all the real numbers in W_i and $\boldsymbol{b}_i$ so to minimize the loss function. This is done by powerful optimizer algorithms adapted to this large dimension problem (it could be as large as several million dimensions). Most of the algorithm are based on improved variants of a stochastic gradient descent. Starting from a random initial condition, the derivatives of the loss function with respect to all the parameters are computed. From this information a small displacement in the parameter space is performed. Then the loss and its derivatives are computed again, and this continues iteratively up to reaching a sufficiently stable local minimum.

After the training phase, the neural network is completely defined. It is now a function that will associate to any input a given output. It is ready to be used and to make predictions.

References

[1] C. M. Bishop. *Pattern Recognition And Machine Learning*. Springer-Verlag New York Inc., New York, 1st ed. 2006. corr. 2nd printing 2011 edition, Aug. 2006.

[2] F. Chollet. *Deep Learning with Python*. Manning Publications, Shelter Island, New York, Nov. 2017.

[3] F. Garcia, O. Rodriguez, J. Mesa, J. D. T. Arruda-Neto, V. P. Likhachev, E. Garrote, R. Capote, and F. Guzmán. BARRIER code: Calculation of fission barriers. *Computer Physics Communications*, 120(1):57–70, July 1999.

[4] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, Nov. 2016.

[5] V. V. Pashkevich. On the asymmetric deformation of fissioning nuclei. *Nuclear Physics A*, 169(2):275–293, July 1971.