Machine learning for bounce calculation

Ryusuke Jinno (IBS-CTPU)

Based on 1805.12153
Aug. 28th, 2018 @ COSMO, Daejeon
MAIN IDEA

- Machine learning (ML) is widely used for image recognition

![Image of cat and dog with machine learning symbols](image)

- In particle cosmology, we often calculate quantities from scalar potentials

![Image of scalar potential and bounce action](image)

**e.g.**

Bounce action 
(\(\sim\) tunneling rate)
Once we regard potentials as images (imagine equal-height contours), we can make machine learning the relation between potentials & quantity.
TALK PLAN

1. Machine learning: lightning introduction
2. Machine learning meets tunneling in QFT
3. Summary
MACHINE LEARNING:
LIGHTNING INTRODUCTION

- Terminology?

**Artificial intelligence (AI)**
- Machines that can perform tasks that are characteristic of human intelligence [J. McCarthy]

**Machine learning (ML)**
- A way of achieving AI: learning without being explicitly programmed

**Neural network (NN)**
- Machine learning with artificial neurons (→ next)

```
x_1 \rightarrow w_1
x_2 \rightarrow w_2
\vdots
x_n \rightarrow w_n
```

**Deep learning**
- Neural network with deep (= many) layers of neurons
1. Each **neuron** collects electric signals through **synapses**
2. When the total signal exceeds a threshold, electric signal is sent to next **neuron** through **axon**
Artificial neuron mimics biological neuron

Diagramatic notation

Equation \[ z = f \left( \sum x_i w_i + b \right) \]

- \( w_i \): weight
- \( b \): bias
- \( f \): ReLU (rectified linear unit)

\[ f(y) \]

\[ y \]
NEURAL NETWORK?

- Neural network = network of artificial neurons

\[
\text{input layer} \quad \text{hidden layer 1} \quad \ldots \quad \text{hidden layer N} \quad \text{output layer}
\]

\[
\begin{align*}
x_{\text{in}} & \quad W_1 & \quad f & \quad x_1 \\
& \quad \vdots & \quad \vdots & \quad \vdots \\
& \quad x_N & \quad f & \quad W_{\text{out}} & \quad x_{\text{out}}
\end{align*}
\]
How to train the neural network with "supervised learning"

- Suppose we have many data of \((x_{\text{in}}, x_{\text{out}}^{(\text{true})})\)

- Then we can define "how poorly the machine predicts"

Error function \(E = \sum_{\text{data}} \sum_{i: \text{component}} \left| (x_{\text{out}})_i - (x_{\text{out}}^{(\text{true})})_i \right| \)

- Training of neural network = update of weights \(W\) and biases \(b\) using \(E\)

\[
W \rightarrow W - \alpha \frac{\partial E}{\partial W} \quad b \rightarrow b - \alpha \frac{\partial E}{\partial b}
\]

\(\alpha\) : constant

Note: there are more sophisticated algorithms, e.g. AdaGrad, Adam, ...
TALK PLAN

✔ 1. Machine learning: lightning introduction

2. Machine learning meets tunneling in QFT

3. Summary
TUNNELING PROBLEM IN QFT

- Quantum tunneling in vacuum in 1+3 dim.  
  [Coleman '77]

  - Nucleation rate $\Gamma$ is dominantly determined by "bounce configuration" $\bar{\phi}$

    \[ \Gamma \propto e^{-S_E[\bar{\phi}]}, \quad S_E[\bar{\phi}] = \int dt_E \int d^3x \left[ \frac{1}{2} (\partial_E \bar{\phi})^2 + V(\bar{\phi}) \right] \]

  - Bounce configuration $\bar{\phi}$: solution of EOM with inverted potential $-V$

    \[ \frac{d^2 \bar{\phi}}{dr^2} + \frac{3}{r} \frac{d \bar{\phi}}{dr} - \frac{dV}{d\phi} = 0 \quad \text{w/ boundary conditions} \quad \frac{d \bar{\phi}}{dr}(r = 0) = 0, \quad \bar{\phi}(r = \infty) = 0 \]

rate $\Gamma$  \quad \text{inverted}  \quad \begin{cases} V \quad \text{rate} \\ \Gamma \end{cases}

$\rightarrow r = 0$  \quad \begin{cases} -V \quad \text{r} = \infty \end{cases}
Calculation of $\bar{\phi}$ requires many times of iterations

Note: there are many approaches, e.g.
[ Duncan et al. '92, Dutta et al. '12, Guada et al. '18 ]
[ Kusenko '95, Moreno et al. '98 ] [ Cline et al. '99, Wainwright '11 ]
[ Konstandin et al. '06 ] [ Masoumi et al. '16 ] [ Espinosa '18 ]

- Many researchers have calculated $S_E[\bar{\phi}]$ for similar potentials...

Can we avoid re-calculating it again and again?
MACHINE LEARNING MEETS TUNNELING IN QFT

- Machine-learning approach
  - Can we construct a machine which gives $S_E$ for input potential $V$?
  - Advantages: 1. faster than any other method / 2. we can share the trained machine
  - Such a machine does not have to solve EOM:
    cat-dog classifier does not have to recognize them as humans do
DATA TAKING

- We use 3 classes of potentials C1-C3:

\[ V(\phi) = \sum_{n=1}^{7} a_n^{(1)} \phi^{n+1} \]

- Class 1 (C1) : \( V(\phi) = \sum_{n=1}^{7} a_n^{(2)} \phi^{2n} \)

- Class 2 (C2) : \( V(\phi) = a_1^{(3)} \phi^2 + \sum_{n=2}^{7} a_n^{(3)} \phi^{2n-1} \)

- Coefficients \( a_n^{(i)} \) are generated "randomly"

- Each class contains 10,000 sets of potential and bounce action

- Bounce action is calculated with traditional overshoot/undershoot

\[ x_{\text{out}}^{(\text{true})} = \ln S_4^{(\text{true})} \]
MACHINE SETUP

- **Input**: sampled values of potential & its derivatives

\[
\mathbf{x}_{\text{in}} = \left\{ V(\phi_{\text{sample}}) \mid \phi_{\text{sample}} = \frac{1}{16}, \cdots, \frac{15}{16} \right\} \\
\oplus \left\{ V'(\phi_{\text{sample}}) \mid \phi_{\text{sample}} = \frac{1}{16}, \cdots, \frac{15}{16} \right\} \oplus \left\{ V''(\phi_{\text{sample}}) \mid \phi_{\text{sample}} = \frac{0}{16}, \cdots, \frac{16}{16} \right\}
\]

- **Output**: logarithmic bounce action \( \mathbf{x}_{\text{out}} = \ln S_4 \)

- **Number of hidden layers**: \( N = 2 \)

- **Implementation**: TensorFlow (r1.17)
RESULTS

- Result: works with sub-% even for < 1min training

Training: 24,000 data from C1+C2+C3 / Test: 6,000 data from C1+C2+C3
Application: 30,000 data from C1+C2+C3

Scatter plot for machine’s performance

Average of 10 times trial

| Training & Test | Applied to | $\langle |\Delta S_4/S_4| \rangle$ |
|-----------------|------------|-------------------------------|
| $C_1 + C_2 + C_3$ | $C_1 + C_2 + C_3$ | 0.00503 |
SUMMARY

- Calculation of quantities from scalar potential can be regarded as image recognition process.

- We proposed using machine learning technique for such calculations, and demonstrated its usefulness in one-dimensional transition.
Backup
Data taking & Training
DATA TAKING

- We use 3 classes of potentials C1-C3:

\[
\begin{align*}
\text{Class 1 (C1)} : & \quad V(\phi) = \sum_{n=1}^{7} a_n^{(1)} \phi^{n+1} \\
\text{Class 2 (C2)} : & \quad V(\phi) = \sum_{n=1}^{7} a_n^{(2)} \phi^{2n} \\
\text{Class 3 (C3)} : & \quad V(\phi) = a_1^{(3)} \phi^{2} + \sum_{n=2}^{7} a_n^{(3)} \phi^{2n-1}
\end{align*}
\]

- Coefficients \( a_n^{(i)} \) are generated "randomly"

- Each class contains 10,000 sets of potential and bounce action

- Bounce action is calculated with traditional overshoot/undershoot

\[ x_{\text{out}}^{(\text{true})} = \ln S_4^{(\text{true})} \]
DETAILS ABOUT POTENTIAL GENERATING PROCESS

- **Random seeds generation** \((V_{\text{max}}, \phi_0, \phi_{1-}, \phi_{1+}, \phi_2)\)
  - 4 numbers are generated in \([0, 1]\), and identified with
    \[
    \phi_{1+} < \phi_0 < \phi_2 < \phi_{1-} \quad \text{or} \quad \phi_{1+} < \phi_2 < \phi_0 < \phi_{1-} \quad \text{(probability 0.5 for each)}
    \]
  - \(V_{\text{max}}\) is sampled from \(10^{-2} \leq V_{\text{max}} \leq 10^{-0.5}\) (flat distribution in log space)

- **Coefficients** \(a_{n}^{(i)}\) are determined so that
  - \(V\) takes local
    \[
    \left\{ \begin{array}{l}
    \text{maximum } V_{\text{max}} \\
    \text{minimum } 0 \text{ or } -1
    \end{array} \right\} @ \left\{ \begin{array}{l}
    \phi = \phi_0 \\
    \phi = 0 \text{ or } \phi = 1
    \end{array} \right\}
    \]
  - \(V'\) takes local
    \[
    \left\{ \begin{array}{l}
    \text{maximum } \phi_{1+} \\
    \text{minimum }
    \end{array} \right\} @ \left\{ \begin{array}{l}
    \phi = \phi_{1+}
    \end{array} \right\}
    \]
  - \(V''\) takes local minimum @ \(\phi = \phi_2\)

- Added to data if there is no local maximum/minimum other than \(\phi = \phi_0, 0, 1\)
MACHINE SETUP

We use a simple machine: $N = 2$
TRAINING & TEST & APPLICATION DATASET

- We construct training & test & application dataset

- Training dataset: used for training (→ next slide)
  - 8,000 data from C1

- Test dataset: used to check that there is no overfitting
  - 2,000 data from C1

- Application dataset: machine is finally applied to this
  - 10,000 data from C1
TRAINING & TEST & APPLICATION DATASET

- We construct training & test & application dataset

Step 1
- Training & Test

Step 2
- Application

- Training dataset: used for training (→ next slide) 8,000 data from C2
- Test dataset: used to check that there is no overfitting 2,000 data from C2
- Application dataset: machine is finally applied to this 10,000 data from C2
TRAINING & TEST & APPLICATION DATASET

- We construct training & test & application dataset

- Training dataset: used for training (→ next slide)
  - E.g.: 8,000 data from C3

- Test dataset: used to check that there is no overfitting
  - E.g.: 2,000 data from C3

- Application dataset: machine is finally applied to this
  - E.g.: 10,000 data from C3
TRAINING & TEST & APPLICATION DATASET

- We construct training & test & application dataset

- Training dataset: used for training (→ next slide)  
  24,000 data from C1+C2+C3

- Test dataset: used to check that there is no overfitting  
  6,000 data from C1+C2+C3

- Application dataset: machine is finally applied to this  
  30,000 data from C1+C2+C3
TRAINING & TEST & APPLICATION DATASET

- We construct training & test & application dataset

- Training dataset: used for training (→ next slide)
  - 16,000 data from C2+C3

- Test dataset: used to check that there is no overfitting
  - 4,000 data from C2+C3

- Application dataset: machine is finally applied to this
  - 10,000 data from C1
We construct training & test & application dataset

- Training dataset: used for training (→ next slide) 16,000 data from C3+C1
- Test dataset: used to check that there is no overfitting 4,000 data from C3+C1
- Application dataset: machine is finally applied to this 10,000 data from C2
We construct training & test & application dataset

- Training dataset: used for training (→ next slide)
  16,000 data from C1+C2

- Test dataset: used to check that there is no overfitting
  4,000 data from C1+C2

- Application dataset: machine is finally applied to this
  10,000 data from C3
MACHINE SETUP

- Input: sampled values of potential & its derivatives

\[
x_{in} = \left\{ V(\phi_{sample}) \mid \phi_{sample} = \frac{1}{16}, \cdots, \frac{15}{16} \right\} \oplus \left\{ V'(\phi_{sample}) \mid \phi_{sample} = \frac{1}{16}, \cdots, \frac{15}{16} \right\} \oplus \left\{ V''(\phi_{sample}) \mid \phi_{sample} = \frac{0}{16}, \cdots, \frac{16}{16} \right\}
\]

- Output: predicted value of logarithmic bounce action \( x_{out} = \ln S_4^{(pred)} \)

- Note: implicit rescaling of input & output

  - In the following, \( x_{in} \) & \( x_{out} \) are understood as rescaled

\[
(x_{in})_i \rightarrow \frac{(x_{in})_i - \langle (x_{in})_i \rangle}{\sigma(x_{in})_i} \quad x_{out} \rightarrow \frac{x_{out} - \langle x_{out} \rangle}{\sigma x_{out}}
\]

  - \( \langle \rangle \) & \( \sigma \) : mean & variance calculated over training & test dataset
TRAINING PROCESS

- **Error function** = how poorly the machine predicts

\[
E = \frac{1}{\text{(\# of data passed to the machine)}} \sum_{\text{data}} \left| x_{\text{out}} - x_{\text{out}}^{\text{(true)}} \right|
\]

\[
x_{\text{out}} = \ln S_4^{\text{(pred)}} : \text{predicted value of logarithmic bounce action}
\]

\[
x_{\text{out}}^{\text{(true)}} = \ln S_4^{\text{(true)}} : \text{true value of logarithmic bounce action}
\]

- **Training** = update of weights and biases using error function

\[
W \rightarrow W - \alpha \frac{\partial E}{\partial W} \quad b \rightarrow b - \alpha \frac{\partial E}{\partial b}
\]

Note: In the actual training we use a slightly more sophisticated algorithm Adam
DETAILS OF TRAINING PROCESS

- **Mini-batch training**
  - We feed the machine with 1/10 of the training data (= mini-batch) for one time
  - 10 times of this process use the whole training data = 1 epoch
  - We train the machine for 10,000 epochs

- **Implementation**
  - Above process is implemented with TensorFlow (r1.17)
DETAILS OF TRAINING PROCESS

- **Mini-batch training**
  - We feed the machine with 1/10 of the training data (= mini-batch) for one time
  - 10 times of this process use the whole training data = 1 epoch
  - We train the machine for 10,000 epochs

- **Implementation**
  - Above process is implemented with TensorFlow (r1.17)
DETAILS OF TRAINING PROCESS

- **Mini-batch training**
  - We feed the machine with 1/10 of the training data (= mini-batch) for one time
  - 10 times of this process use the whole training data = 1 epoch
  - We train the machine for 10,000 epochs

- **Implementation**
  - Above process is implemented with TensorFlow (r1.17)
DETAILS OF TRAINING PROCESS

- **Mini-batch training**
  - We feed the machine with 1/10 of the training data (= mini-batch) for one time
  - 10 times of this process use the whole training data = 1 epoch
  - We train the machine for 10,000 epochs

- **Implementation**
  - Above process is implemented with TensorFlow (r1.17)
DETAILS OF TRAINING PROCESS

- Mini-batch training
  - We feed the machine with 1/10 of the training data (= mini-batch) for one time
  - 10 times of this process use the whole training data = 1 epoch
  - We train the machine for 10,000 epochs

- Implementation
  - Above process is implemented with TensorFlow (r1.17)
Results
RESULTS

- Case A: 1 class for training & test & application

  Training: 8,000 data from C1 / Test: 2,000 data from C1
  Application: 10,000 data from C1

Scatter plot for machine's performance

| Training & Test | Applied to | $\langle |\Delta S_4/S_4| \rangle$ |
|-----------------|------------|-------------------------------|
| $C_1$           | $C_1$      | 0.00607                       |
RESULTS

- Case A : 1 class for training & test & application

  Training : 8,000 data from C2 / Test : 2,000 data from C2
  Application : 10,000 data from C2

  Scatter plot for machine’s performance

  Average of 10 times trial

| Training & Test | Applied to | $\langle |\Delta S_4/S_4| \rangle$ |
|-----------------|------------|----------------------------------|
| $C_2$           | $C_2$      | 0.00423                          |
RESULTS

- Case A: 1 class for training & test & application

  Training: 8,000 data from C3 / Test: 2,000 data from C3
  Application: 10,000 data from C3

Scatter plot for machine’s performance

Average of 10 times trial

| Training & Test | Applied to | $\langle|\Delta S_4/S_4|\rangle$ |
|-----------------|------------|-------------------------------|
| $C_3$           | $C_3$      | 0.00418                       |
RESULTS

- Case B: mixture of 3 classes

Training: 24,000 data from C1+C2+C3 / Test: 6,000 data from C1+C2+C3
Application: 30,000 data from C1+C2+C3

Scatter plot for machine’s performance

Average of 10 times trial

| Training & Test | Applied to | $\langle |\Delta S_4/S_4| \rangle$ |
|-----------------|------------|-----------------------------|
| $C_1 + C_2 + C_3$ | $C_1 + C_2 + C_3$ | 0.00503 |
RESULTS

- Case C: training & test over 1 class / application to other 2 classes

| Training & Test: $C_2 + C_3$ | Applied to: $C_1$ | $\langle |\Delta S_4/S_4| \rangle$ |
|-----------------------------|------------------|-----------------|
| $C_2 + C_3$                 | $C_1$            | 0.0248          |
RESULTS

- Case C: training & test over 1 class / application to other 2 classes

  Training: 16,000 data from C3+C1 / Test: 4,000 data from C3+C1
  Application: 10,000 data from C2

Scatter plot for machine’s performance

Average of 10 times trial

| Training & Test | Applied to | $\langle|\Delta S_4/S_4|\rangle$ |
|-----------------|------------|-------------------------------|
| $C_3 + C_1$     | $C_2$      | 0.0128                        |
RESULTS

- Case C: training & test over 1 class / application to other 2 classes

Training: 16,000 data from C1+C2 / Test: 4,000 data from C1+C2
Application: 10,000 data from C3

Scatter plot for machine’s performance

Average of 10 times trial

| Training & Test | Applied to | \( \langle |\Delta S_4/S_4| \rangle \) |
|-----------------|------------|----------------------------------|
| \( C_1 + C_2 \) | \( C_3 \)   | 0.00903                          |
Discussion
DISCUSSION

- How much precision can we expect in practical use?

  Potential shapes in particle physics are not that many

  → If we train with such potentials, the resulting precision will be $C_1+C_2+C_3$ or better

- How much is the speedup?

  - Overshoot/undershoot typically takes $O(1-10)$ sec in my code

  - Other approaches take e.g. $O(10^{-2})$ sec \[\text{[Guada et al. '18, "Polygonal bounces" (private communication)]}\]

  - Our machine takes $O(10)$ sec for training,

    while after training it takes $O(10^{-4})$ sec to calculate the bounce
DISCUSSION

- Generalizations?
  - Different spacetime dimensions → trivial
  - Multidimensional transitions → needs good ideas

  e.g. 1) 2 dim.: convolutional neural network (CNN) may help
  2) ML may be used for 1 dim. part in existing multidimensional public codes
  3) ML may also be used for "initial position suggestor" in such public codes
      by identifying the output as the initial position
DISCUSSION

- Generalizations?
  - Different spacetime dimensions
  - Multidimensional transitions

  e.g.
  1) 2 dim.: convolutional neural network (CNN) may help
  2) ML may be used for 1 dim. part in existing multidimensional public codes
  3) ML may also be used for "initial position suggestor" in such public codes
     by identifying the output as the initial position
DISCUSSION

- **Generalizations?**
  - Different spacetime dimensions
  - Multidimensional transitions

  *e.g.*
  1) **2 dim.**: convolutional neural network (CNN) may help
  2) ML may be used for 1 dim. part in existing multidimensional public codes
  3) ML may also be used for "initial position suggestor" in such public codes

  by identifying the output as the initial position
DISCUSSION

- Generalizations?
  - Different spacetime dimensions
  - Multidimensional transitions →

  e.g. 1) 2 dim.: convolutional neural network (CNN) may help

  2) ML may be used for 1 dim. part in existing multidimensional public codes

  3) ML may also be used for "initial position suggestor" in such public codes
   by identifying the output as the initial position
Others
NEURAL NETWORK: IMAGE RECOGNITION

- Image classifier
NEURAL NETWORK: IMAGE RECOGNITION

- Image classifier: input = image / output = label

Note: precisely, output layer is log-odds $\log \frac{P(\text{cat})}{P(\text{dog})}$

Note: actual image recognition is not that simple, e.g. CNN