Detection of phase transition via convolutional neural network

Akinori Tanaka (RIKEN iTHES)

arXiv:1609.09087
w/ Akio Tomiya (CCNU)

Deep learning and physics @Osaka Univ.
Today’s talk

1. Introduction
2. What is NN?
3. Our work
1. Introduction

Main field: Mathematical physics

- Recent publications

"Remarks on a SUSY exact action in 3D supergravity" (Phys. Rev. D93 no. 10, (2016) 105029.)
"Varieties of Abelian mirror symmetry on RP2 × S1" (JHEP 02 (2016) 088.)
"Exact Path Integral for 3D Quantum Gravity II" (Phys. Rev. D93 no. 6, (2016) 064014.)

Why ML?
**Bottleneck of DL**: Data preparation

- "supervised learning" needs large data

  Input $\rightarrow$ Machine(NN) $\rightarrow$ Output

  $D = \{(\text{Input}, \text{Answer})\}$

"MNIST" (handwritten numbers): $|D| = O(10^4)$

奎 386
3620
• Other data?
  
  • Extended MNIST: handwritten alphabets + numbers
  • Caltech 101: Images (airplanes, motorcycles, ...)
  • Imagenet: Images
  
  ...

• What we want
  
  • Drawing anime-characters?
  • Automation in factories?
  • Automatic driving?
  
  ...

You should prepare training data by yourself! → difficult.
• Quantum mechanics

Feynman's path integral

→ Suitable for ML training data.
• Statistical physics / finite temp QFT

```python
In [*]: #matplotlib notebook
temps, mags, configs, T = [], [], [], 500
np_state = np.random.choice(dof, l**2).reshape(l, l); h_p = hamiltonian_square(r
fig = plt.figure(figsize=(10, 4)); al, ar = fig.add_subplot(1, 2, 1), fig.add_subplot
plot_ising(al, ar, np_state, h_p, T)
```
Today’s talk

1. Introduction
2. What is NN?
3. Our work
2. What is NN?

- $(\text{Linear} + \text{Non-linear}) \circ (\text{Linear} + \text{Non-linear}) \circ (\cdots$

$$\tilde{y} = W_2 \sigma \left( W_1 \tilde{x} + \tilde{b}_1 \right) + \tilde{b}_2$$

“Deep Neural Network”
● **NN training**

Tuning parameters inside, $W$, $b$, as

\[ \vec{y} = W_2 \sigma \left( W_1 \vec{x} + \vec{b}_1 \right) + \vec{b}_2 \]

equal as possible

\[ \vec{y}_1 \leftrightarrow \vec{y}_{\text{ans}1} \]

\[ \vec{y}_2 \leftrightarrow \vec{y}_{\text{ans}2} \]

\[ \vdots \leftrightarrow \vdots \]
- Example: Prime number detection :p (Keras)

\[
x[n] = \begin{pmatrix}
    n \mod 2 \\
    n \mod 3 \\
    n \mod 5 \\
    n \mod 7 \\
    n \mod 11 \\
\end{pmatrix}
\]

\[
y[n] = \begin{cases}
    1 & n: \text{prime} \\
    0 & \text{otherwise}
\end{cases}
\]

- import data(n=0,1,2,...150)

```
In [9]:
import prime; x, y = prime.get_xy();
for i in range(5):
    n = np.random.randint(10, 100)
    print("n={}").format(n), blue("x[n]={}".format(x[n])), red("y[n]={}".format(y[n]))
```

n=36, x[n]=[0 0 1 1 3], y[n]=0
n=42, x[n]=[0 0 2 0 9], y[n]=0
n=19, x[n]=[1 1 4 5 8], y[n]=1
n=53, x[n]=[1 2 3 4 9], y[n]=1
n=42, x[n]=[0 0 2 0 9], y[n]=0

- training
n = 36, x[n] = [0 0 1 1 3], y[n] = 0
n = 42, x[n] = [0 0 2 0 9], y[n] = 0
n = 19, x[n] = [1 1 4 5 8], y[n] = 1
n = 53, x[n] = [1 2 3 4 9], y[n] = 1
n = 42, x[n] = [0 0 2 0 9], y[n] = 0

- training

In [3]:

```python
# training by 2, 3, 5, ..., 99
prime_NN.fit(x[2:100], y[2:100], epochs=500, batch_size=50)
```

Epoch 491/500
98/98 [==========================================] - 0s - loss: 0.3773
Epoch 492/500
98/98 [==========================================] - 0s - loss: 0.3770
Epoch 493/500
98/98 [==========================================] - 0s - loss: 0.3765
Epoch 494/500
98/98 [==========================================] - 0s - loss: 0.3761
Epoch 495/500
98/98 [==========================================] - 0s - loss: 0.3757
Epoch 496/500
98/98 [==========================================] - 0s - loss: 0.3753
Epoch 497/500
98/98 [==========================================] - 0s - loss: 0.3748
Epoch 498/500
98/98 [==========================================] - 0s - loss: 0.3744
Epoch 499/500
98/98 [==========================================] - 0s - loss: 0.3740
Epoch 500/500
98/98 [==========================================] - 0s - loss: 0.3736

Out[3]: `<keras.callbacks.History at 0x11db4ba8>`
• Can NN find prime numbers > 100?

```python
In [5]:
predictions = []
answers = []
for n in range(100, 150):
    if prime_NN.predict(x[n].reshape(1,5))>0.5:
        predictions.append(n)
    if y[n] == 1:
        answers.append(n)
print('Predictions :', predictions)
print('True primes :', answers)
```

Predictions : [101, 113, 137, 143, 149]
True primes : [101, 103, 107, 109, 113, 127, 131, 137, 139, 149]

So far, so good!

Deep learning and **Physics**
Today's talk

1. Introduction
2. What is NN?
3. Our work
3. Our work

"Hello World" of "Deep Learning and physics"
• 2D ferromag Ising model

  • System and Energy

  \[ H = -\sum_{i,j} \left( \sigma_{ij}\sigma_{(i+1)j} + \sigma_{ij}\sigma_{i(j+1)} \right) \]

  • Probability

  \[ P \left( \begin{array}{ccc} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{array} \right) \propto e^{-\frac{H}{T}} \]
• Phase transition

\[ \langle \sigma \rangle \neq 0 \quad \rightarrow \quad \langle \sigma \rangle = 0 \]

Order parameter

L. Onsager (1944) etc.
- **Idea**: Ising configurations $\rightarrow$ NN

NN can classify 90~%!

But, already submitted ;-(

by *(Nature Physics 13, 431–434 (2017))*
- Hot / Cold?

\[ T_c = 2.27... \text{ was given!} \]

Can NN find 2.27... itself?
• What we did (simplified ver.)

In [4]: ani

Out[4]:

![Diagram of a discretized neural network with operations like ReLU and Softmax.](image-url)
**Demo**

```
In [5]: x, y = training_data(T, configs, temps)
    : Ising_NN.compile(optimizer='adam', loss='categorical_crossentropy', m

In [6]: Ising_NN.fit(x, y, epochs=100, batch_size=10)
```

```
500/500 [============================] - 0s - loss: 0.9082 - acc: 0
  .7160
Epoch 95/100
500/500 [============================] - 0s - loss: 0.9053 - acc: 0
  .7060
Epoch 96/100
500/500 [============================] - 0s - loss: 0.9061 - acc: 0
  .7160
Epoch 97/100
500/500 [============================] - 0s - loss: 0.8993 - acc: 0
  .7160
Epoch 98/100
500/500 [============================] - 0s - loss: 0.9030 - acc: 0
  .7080
Epoch 99/100
500/500 [============================] - 0s - loss: 0.8964 - acc: 0
  .7160
Epoch 100/100
500/500 [============================] - 0s - loss: 0.9042 - acc: 0
  .7180
```

```
Out[6]: <keras.callbacks.History at 0x1282aba20>
```
- Heatmap for $W$

```python
In [7]: from matplotlib import inline
W = Ising_NN.get_weights()
fig = plt.figure(figsize=(12, 4.8)); ay = fig.add_subplot(1,2,1); ax = set_W(W, ax, ay)
```

$T_c \sim 2.27...$
• In our paper: #data=10,000, tensorflow

\[ \rightarrow: 1/\text{temperature}, \quad 1/2.27\ldots \sim 0.44\ldots \]
• How to get quantitative value?

• Lattice size dependence

<table>
<thead>
<tr>
<th>System size</th>
<th>$\beta_c$ (CNN)</th>
<th>$\beta_c$ (FC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8×8</td>
<td>0.478915</td>
<td>0.462494</td>
</tr>
<tr>
<td>16×16</td>
<td>0.448562</td>
<td>0.433915</td>
</tr>
<tr>
<td>32×32</td>
<td>0.451887</td>
<td>0.415596</td>
</tr>
</tbody>
</table>

$L \to \infty$ $\beta_c^{\text{Exact}} \sim 0.440686$
Without any order parameters!
Summary

- What we did
  - Propose NN detecting PT.
  - Ising PT can be identified.

- Future
  - Other systems? (wo SSB)
  - Other tasks? (e.g. MCMC)
Sub slides
Our setup
A trial for no SSB case

- XY model

\[ H = -\sum_{i,j} \left( \cos(\theta_{ij} - \theta_{(i+1),j}) + \cos(\theta_{ij} - \theta_{i,(j+1)}) \right) \]

- Snapshots
• Phase transition

J. Kosterlitz & D. J. Thouless (1973)

Critical

$1/T_c \sim 1.1$

• Heat map of $W$ (with dropout)