Machine learning for bounce calculation

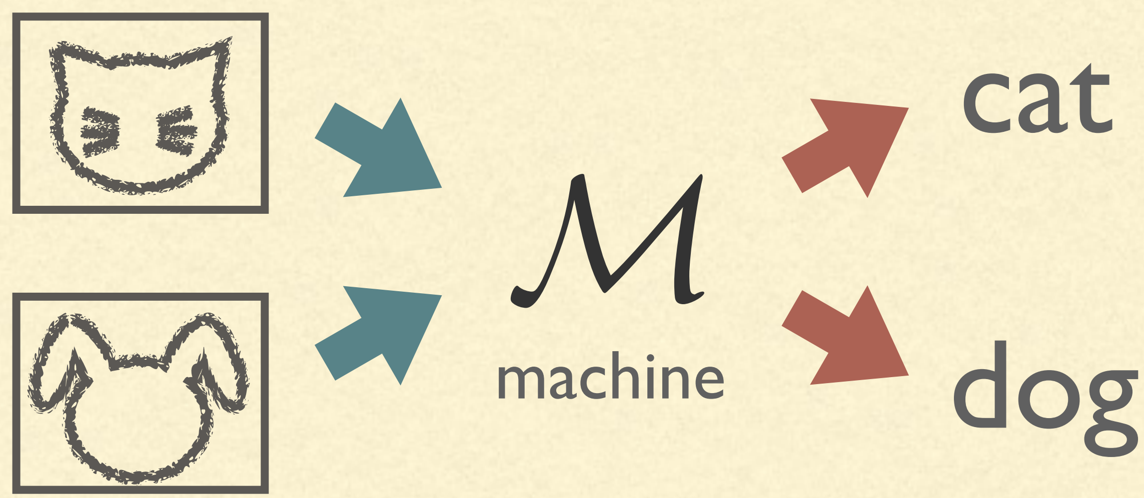
Ryusuke Jinno (IBS-CTPU)

기초과학연구원
Institute for Basic Science

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

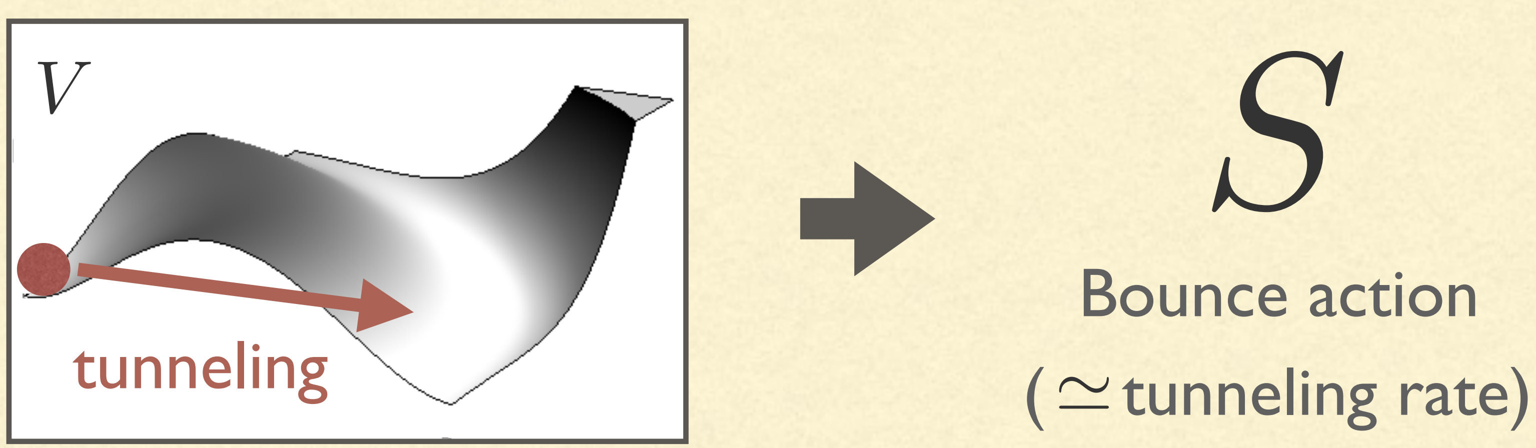
MAIN IDEA

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



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

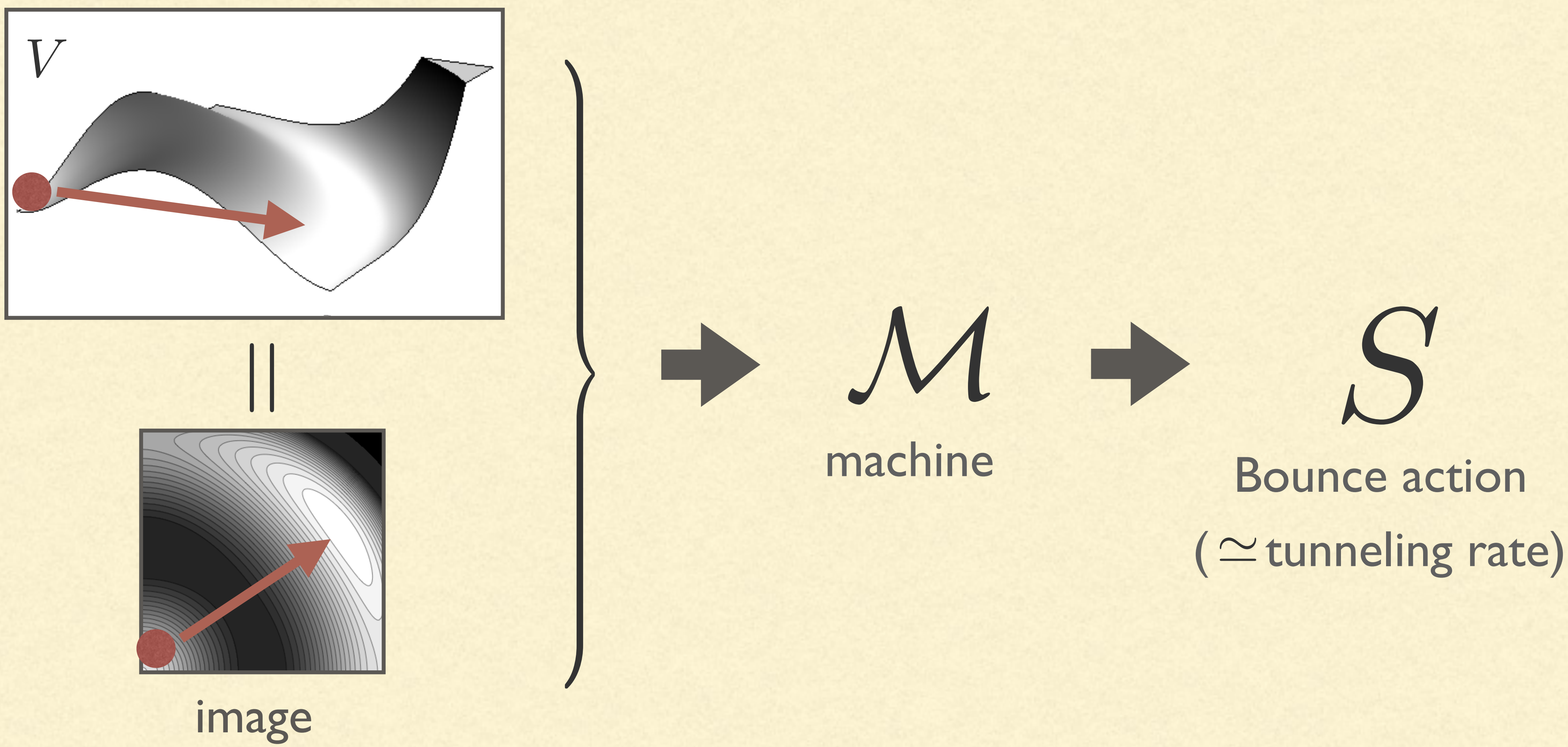
e.g.



MAIN IDEA

- Once we regard potentials as images (imagine equal-height contours),

  we can make machine learning the relation between potentials & quantity

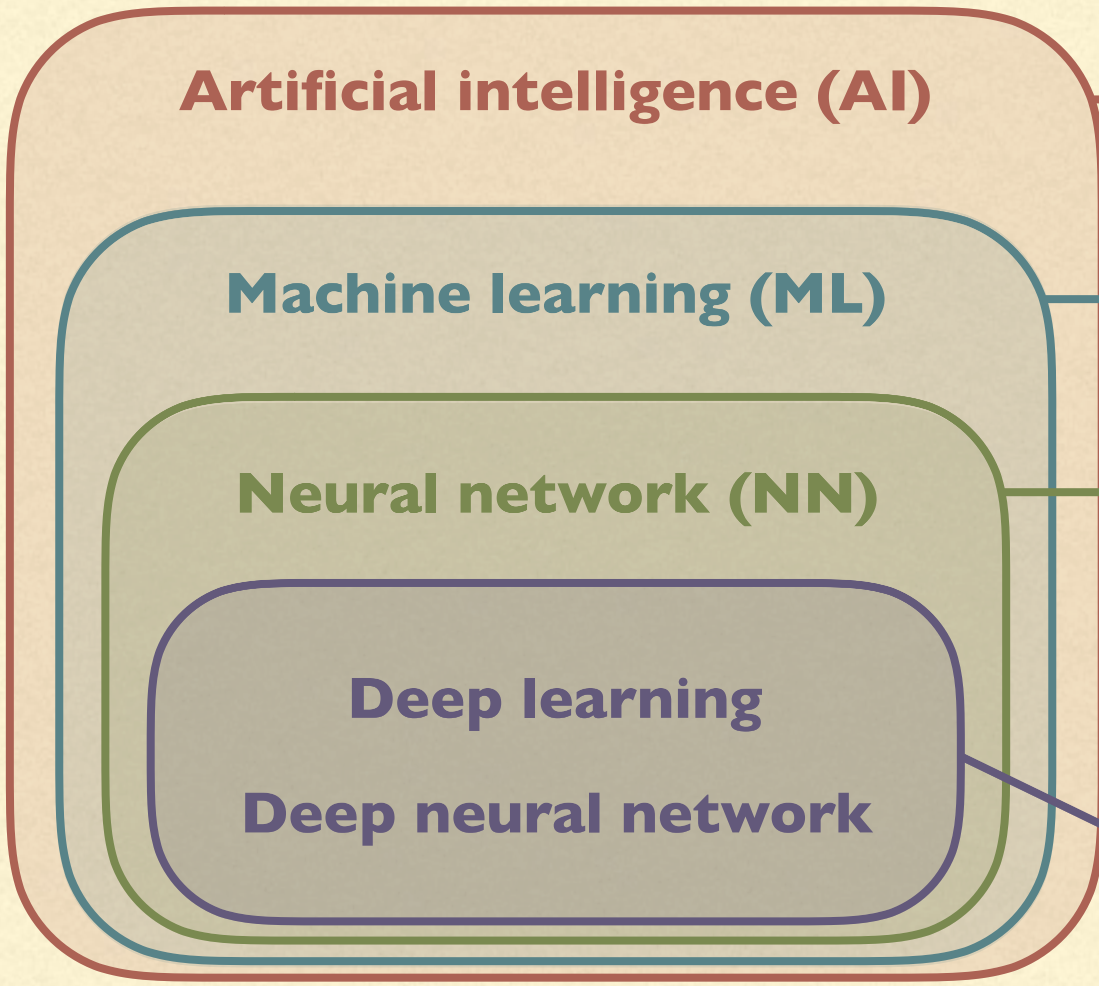


TALK PLAN

1. Machine learning : lightning introduction
2. Machine learning meets tunneling in QFT
3. Summary

MACHINE LEARNING: LIGHTNING INTRODUCTION

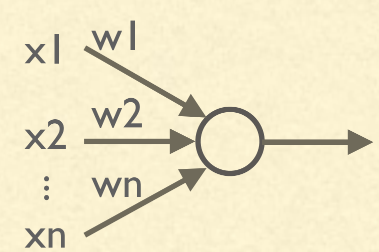
- Terminology?



Machines that can perform tasks that are characteristic of human intelligence [J. McCarthy]

A way of achieving AI: lerning without being explicitly programmed

Machine learning with artificial neurons **(→ next)**

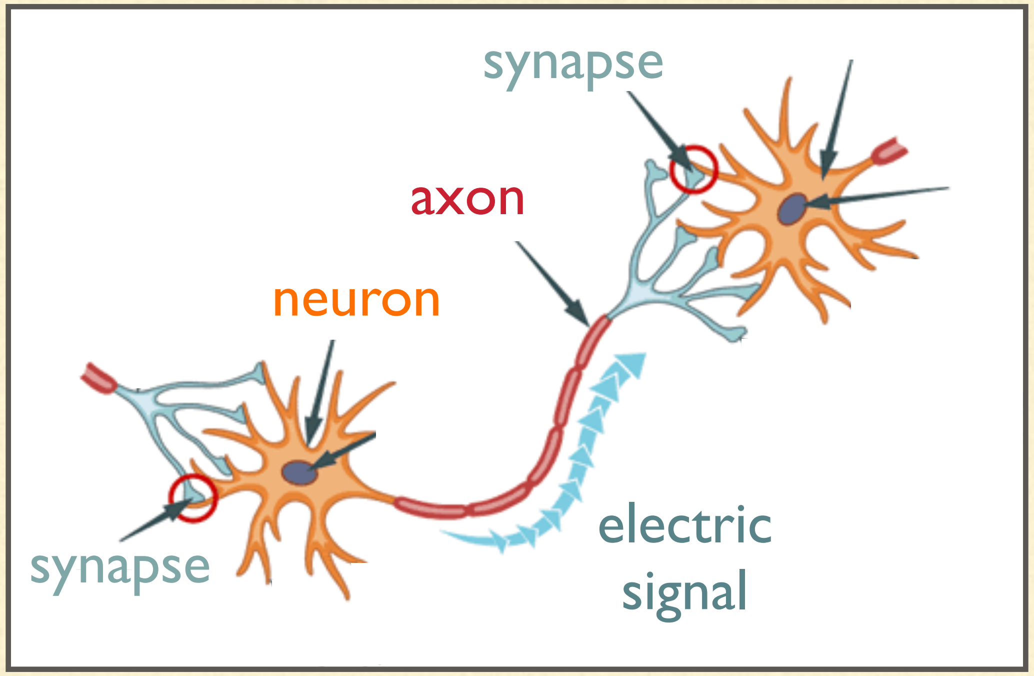


Neural network with deep (= many) layers of neurons

NEURAL NETWORK ?

- Biological neuron

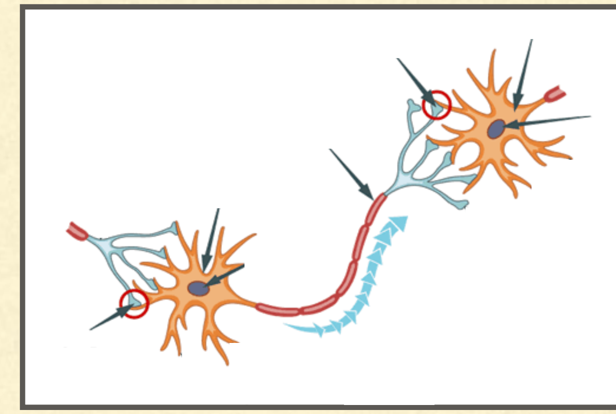
[https://medium.com/autonomous-agents/mathematical-foundation-for-activation-functions-in-artificial-neural-networks-a51c9dd7c089]



1. Each neuron collects electric signals through synapses
2. When the total signal exceeds a threshold,

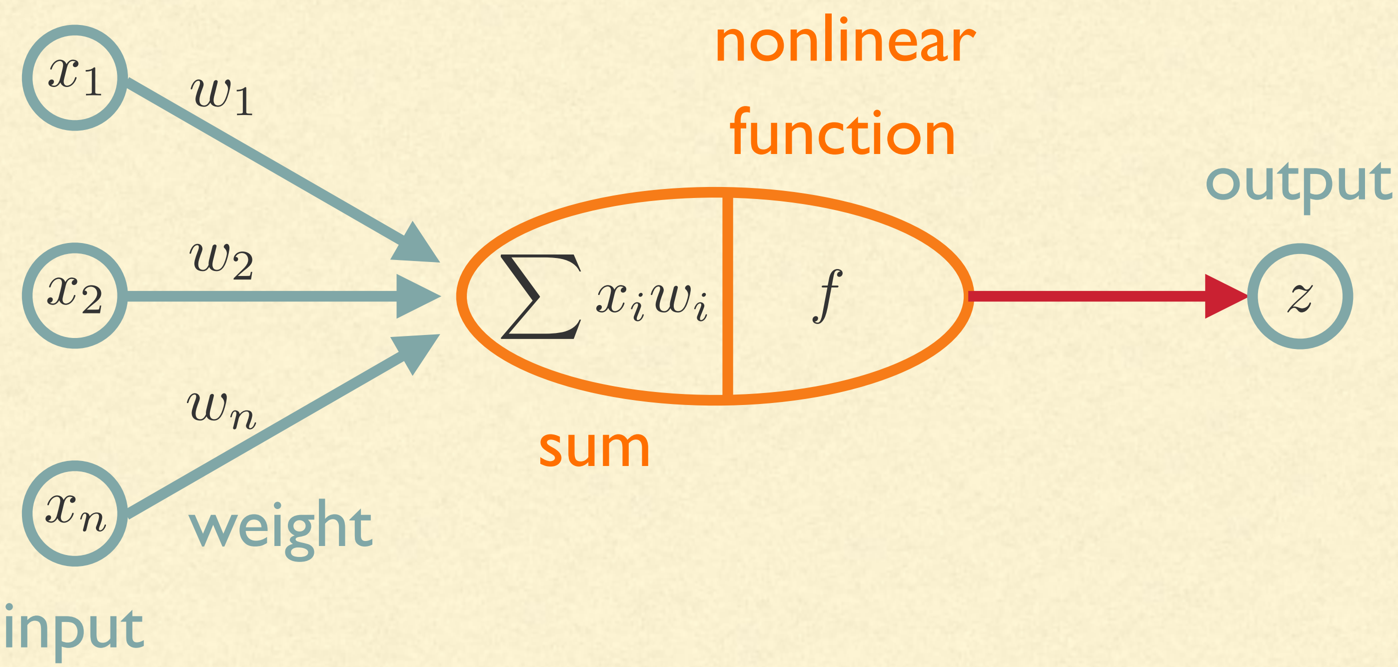
   electric signal is sent to next neuron through axon

NEURAL NETWORK ?



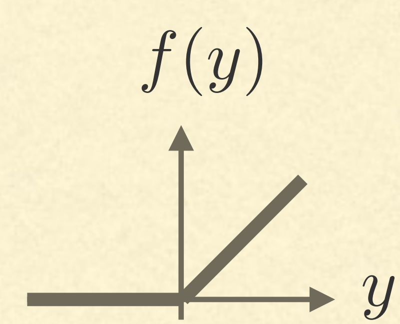
- Artificial neuron mimics biological neuron

Diagramatic notation



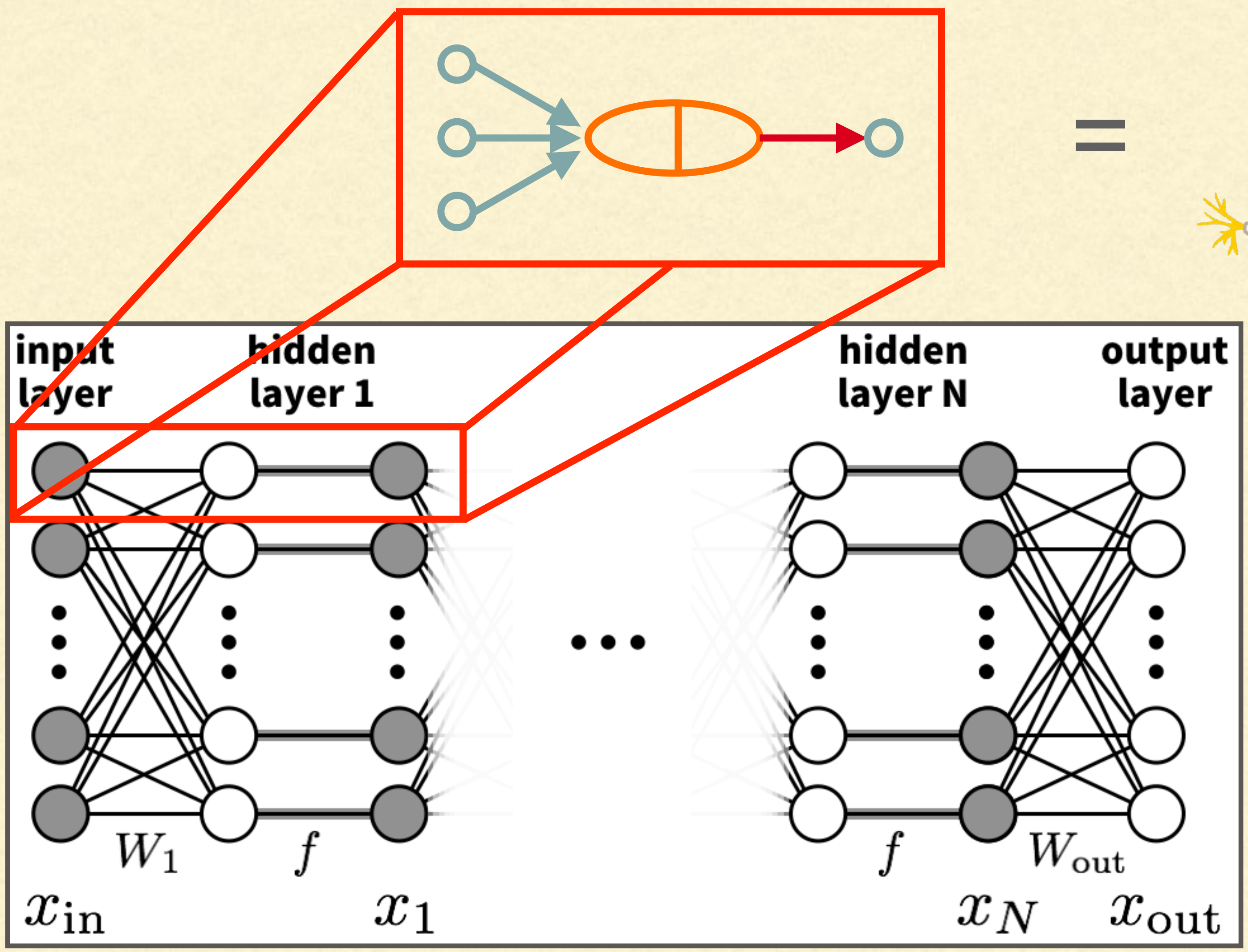
Equation $z = f\left(\sum x_i w_i + b\right)$

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

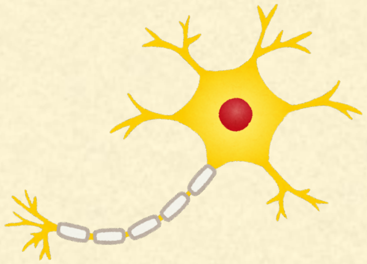


NEURAL NETWORK ?

- Neural network = network of artificial neurons



=

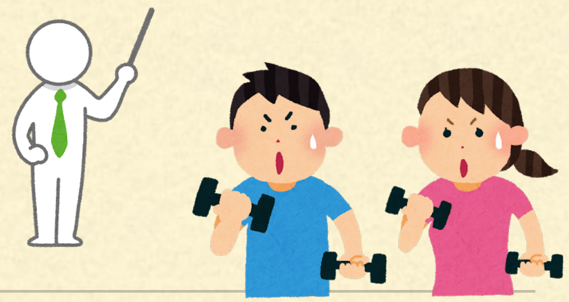


neuron

=

neural network

NEURAL NETWORK: SUPERVISED LEARNING



- How to train the neural network with "supervised learning"

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

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

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

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

$$W \to W - \alpha \frac{\partial E}{\partial W} \qquad b \to b - \alpha \frac{\partial E}{\partial b}$$

α : constant

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

TALK PLAN

✔. Machine learning : lightning introduction

2. Machine learning meets tunneling in QFT

3. Summary

TUNNELING PROBLEM IN QFT

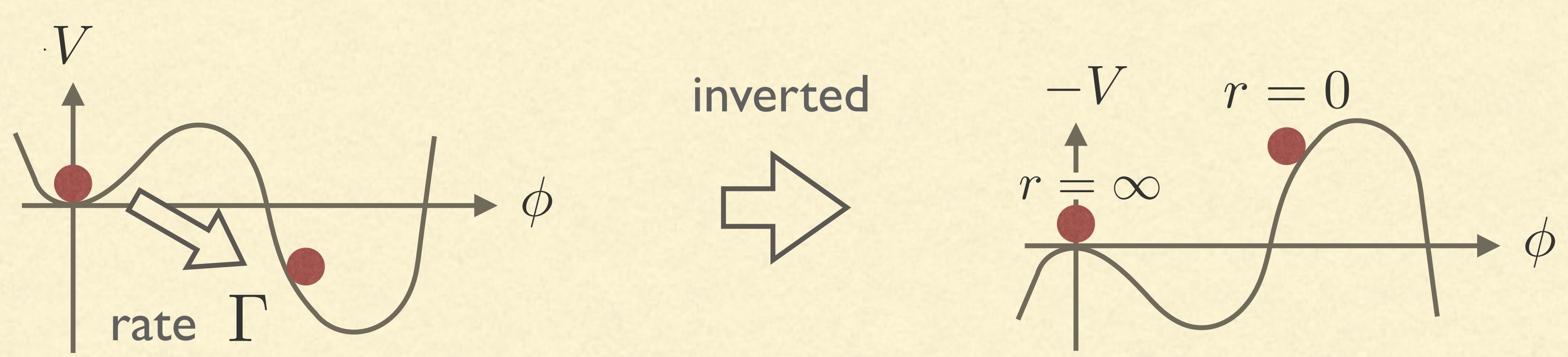
- Quantum tunneling in vacuum in 1+3 dim. [Coleman '77]

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

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

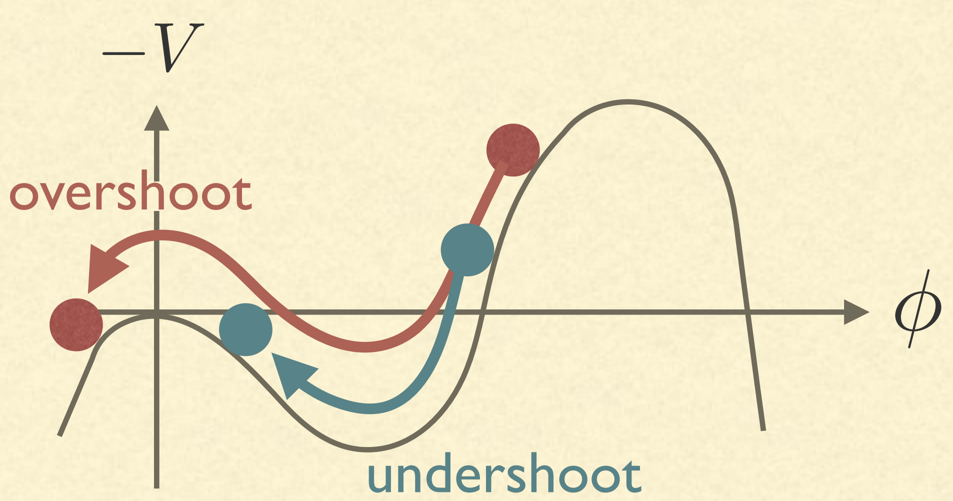
  - Bounce configuration $\bar{\phi}$: solution of EOM with inverted potential $-V$

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



MACHINE LEARNING MEETS TUNNELING IN QFT

- 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

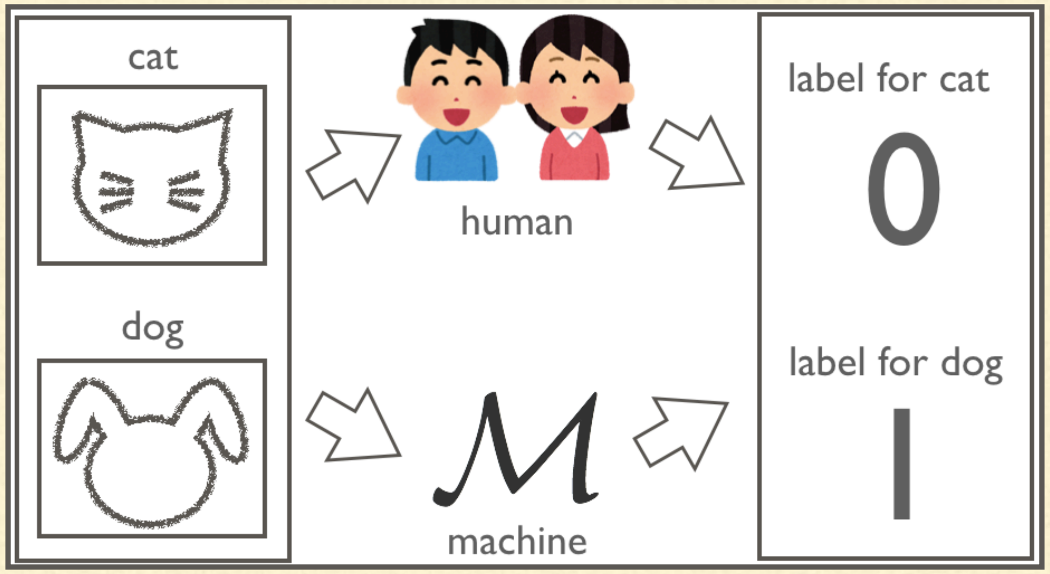
- Machine-learning approach

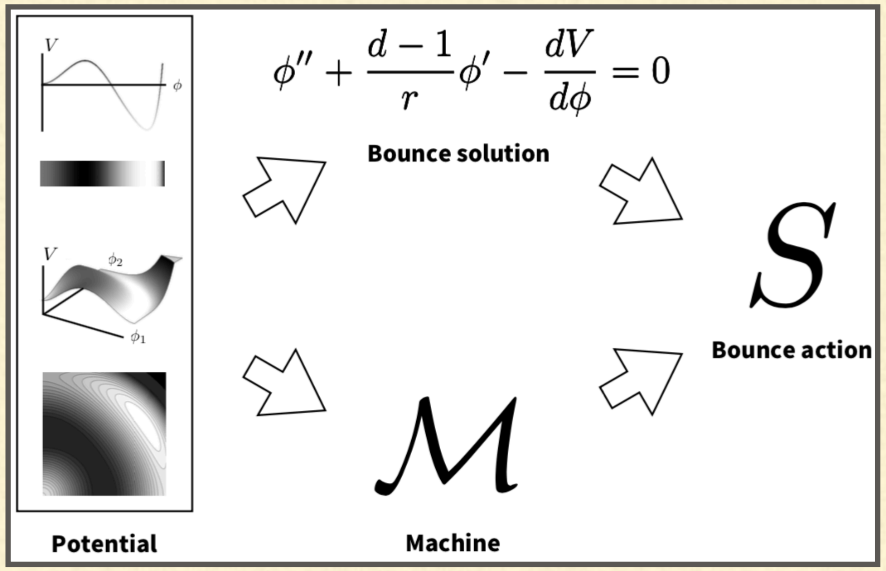
  - Can we construct a machine which gives S_E for input potential V ?

  - 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

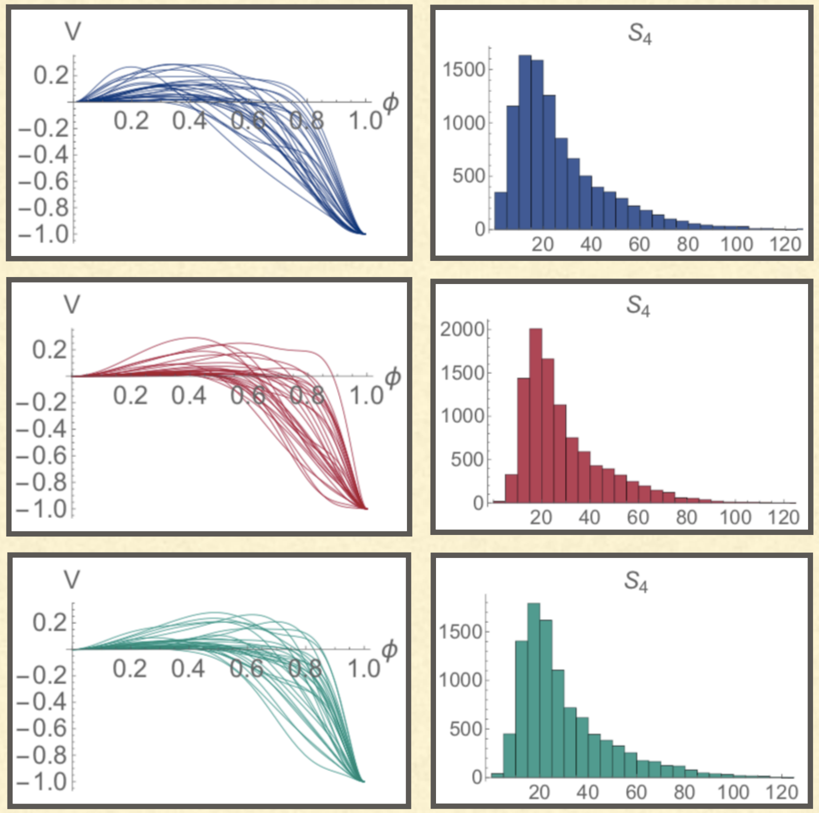
- We use 3 classes of potentials C1-C3:

$$
\begin{cases}
\text{Class 1 (C1)} : V(\phi) = \sum_{n=1}^{7} a_n^{(1)} \phi^{n+1} \\
\text{Class 2 (C2)} : V(\phi) = \sum_{n=1}^{7} a_n^{(2)} \phi^{2n} \\
\text{Class 3 (C3)} : V(\phi) = a_1^{(3)} \phi^2 + \sum_{n=2}^{7} a_n^{(3)} \phi^{2n-1}
\end{cases}
$$

C1

C2

C3



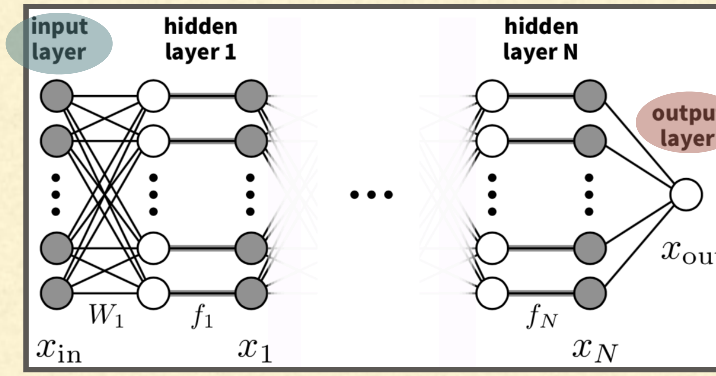
Potential   Bounce action

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

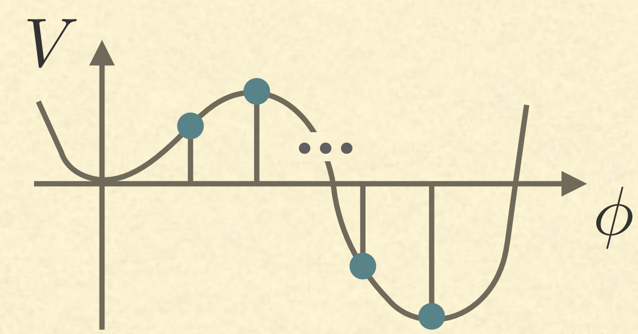
- Each class contains 10,000 sets of potential and bounce action $x_{\text{out}}^{(\text{true})} = \ln S_4^{(\text{true})}$

- Bounce action is calculated with traditional overshoot/undershoot

MACHINE SETUP



- Input : sampled values of potential & its derivatives



$$x_{\rm in} = \left\{ V(\phi_{\rm sample}) \middle| \phi_{\rm sample} = \frac{1}{16}, \cdots, \frac{15}{16} \right\}$$

$$\oplus \left\{ V'(\phi_{\rm sample}) \middle| \phi_{\rm sample} = \frac{1}{16}, \cdots, \frac{15}{16} \right\} \oplus \left\{ V''(\phi_{\rm sample}) \middle| \phi_{\rm sample} = \frac{0}{16}, \cdots, \frac{16}{16} \right\}$$

- Output : logarithmic bounce action $x_{\rm out} = \ln S_4$
- Number of hidden layers : N = 2
- Implementation: TensorFlow (r1.17)

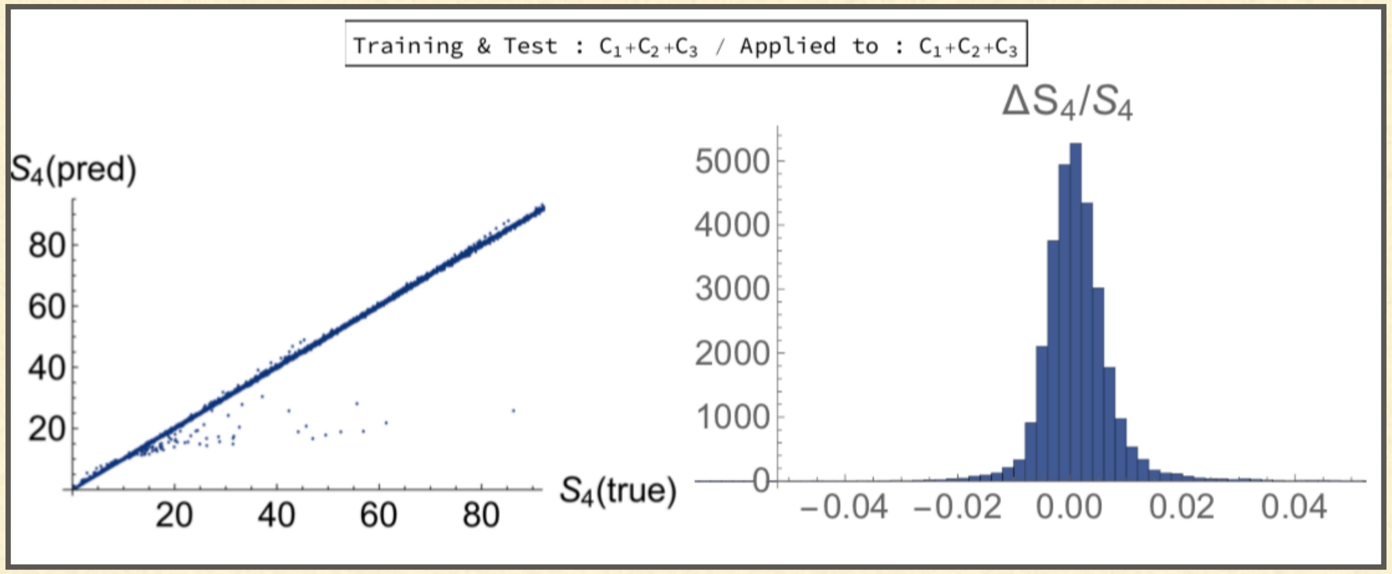
RESULTS

- 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

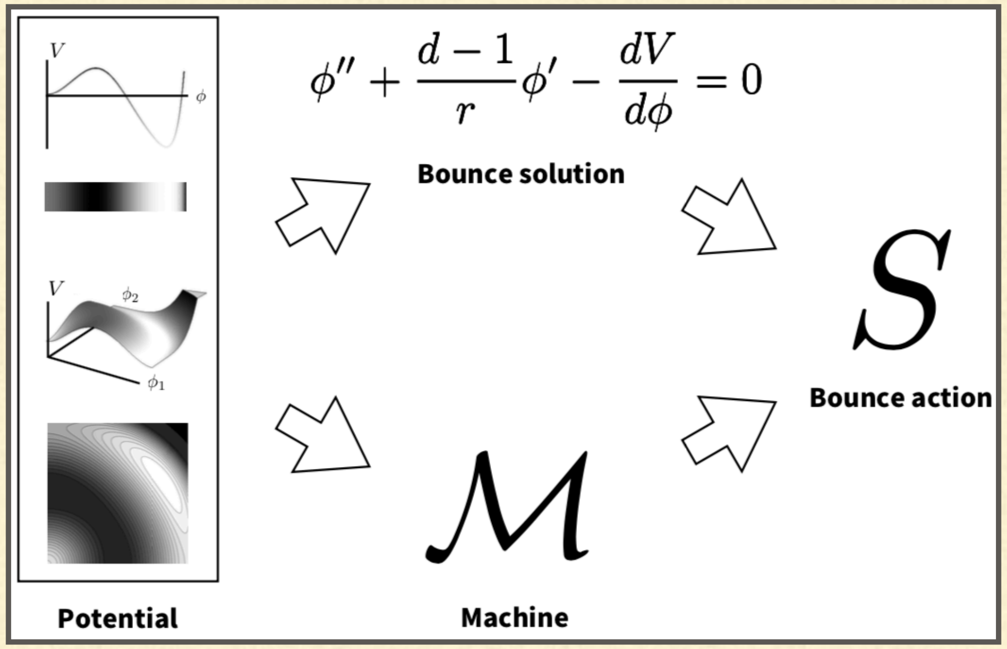


Average of 10 times trial

Training & Test	Applied to	$\langle\langle\lvert\Delta S_4/S_4\rvert\rangle\rangle$
$C_1+C_2+C_3$	$C_1+C_2+C_3$	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

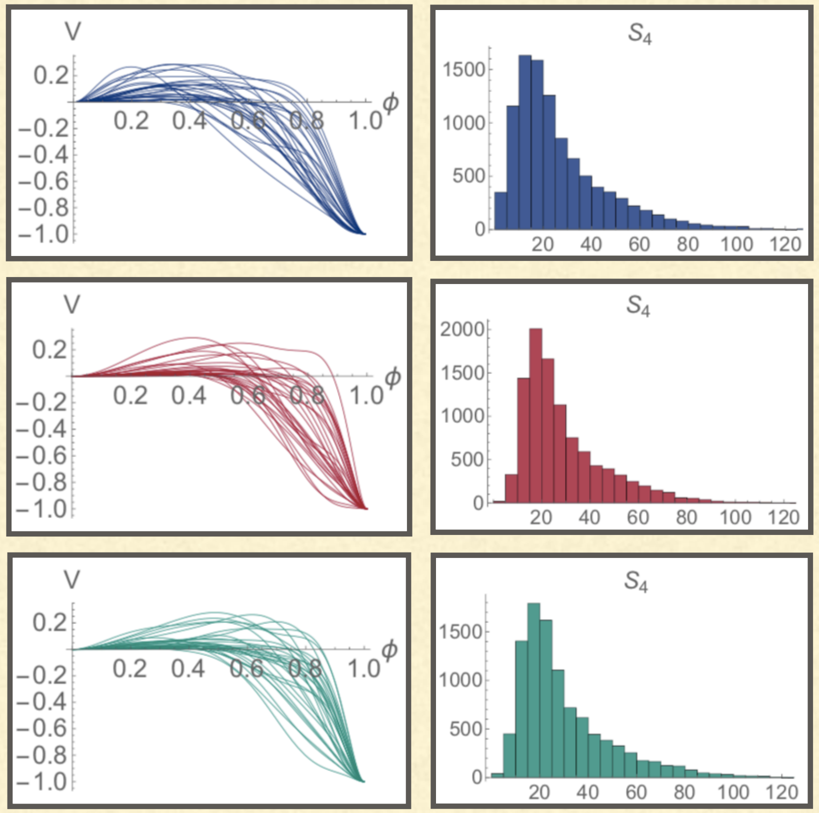
- We use 3 classes of potentials C1-C3:

$$
\begin{cases}
\text{Class 1 (C1)} : V(\phi) = \sum_{n=1}^{7} a_n^{(1)} \phi^{n+1} \\
\text{Class 2 (C2)} : V(\phi) = \sum_{n=1}^{7} a_n^{(2)} \phi^{2n} \\
\text{Class 3 (C3)} : V(\phi) = a_1^{(3)} \phi^2 + \sum_{n=2}^{7} a_n^{(3)} \phi^{2n-1}
\end{cases}
$$

C1

C2

C3



Potential Bounce action

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

- Each class contains 10,000 sets of potential and bounce action $x_{\rm out}^{\rm (true)} = \ln S_4^{\rm (true)}$

- Bounce action is calculated with traditional overshoot/undershoot

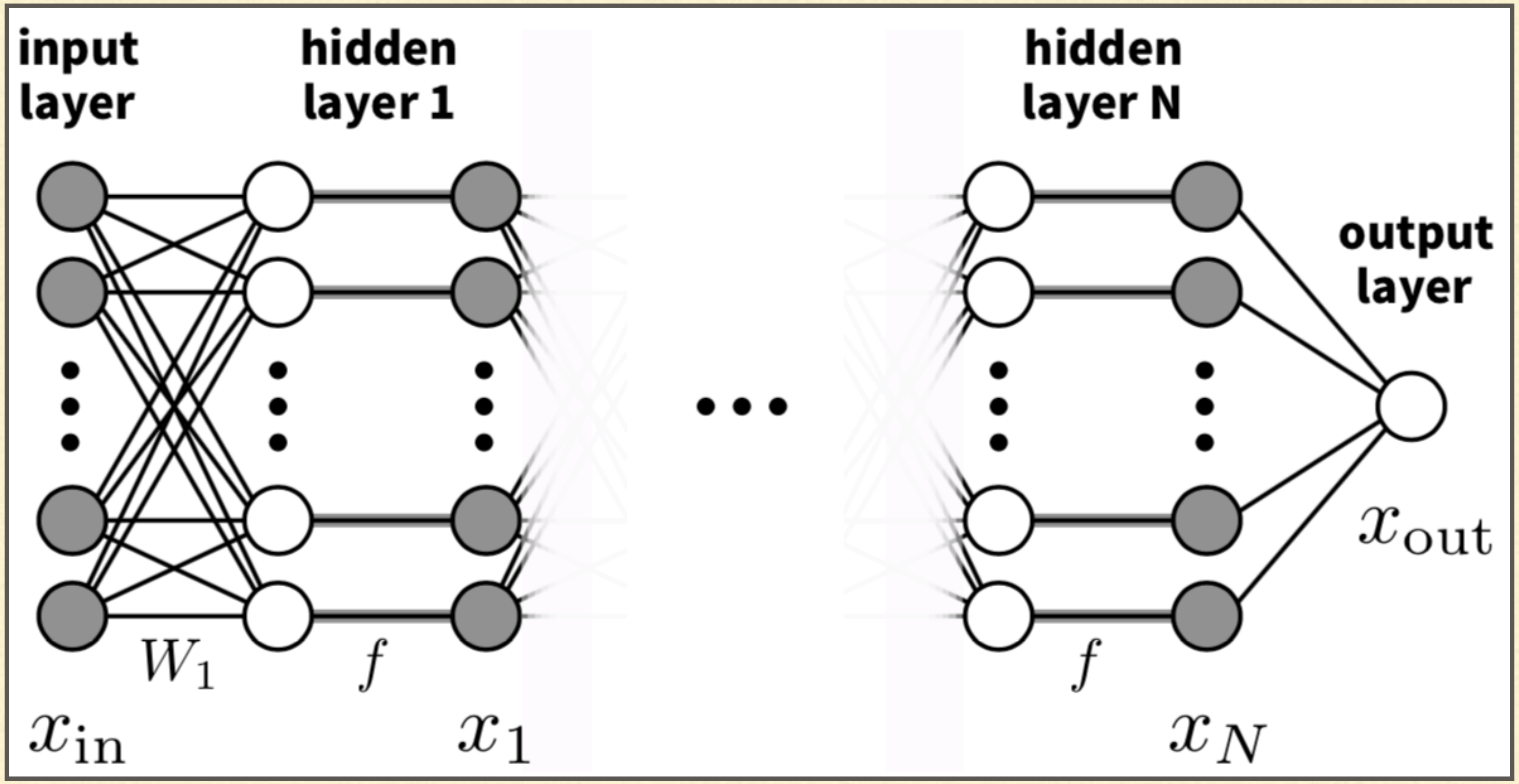
DETAILS ABOUT POTENTIAL GENERATING PROCESS

- Random seeds generation $(V_{\max}, \phi_0, \phi_{1-}, \phi_{1+}, \phi_2)$
  - 4 numbers are generated in $[0, 1]$, and identified with

    $\phi_{1+} < \phi_0 < \phi_2 < \phi_{1-}$ or $\phi_{1+} < \phi_2 < \phi_0 < \phi_{1-}$ (probability 0.5 for each)
  - $V_{\max}$ is sampled from $10^{-2} \leq V_{\max} \leq 10^{-0.5}$ (flat distribution in log space)
- Coefficients $a_n^{(i)}$ are determined so that
  - V takes local $\left\{ \begin{array}{l} \text{maximum } V_{\max} \\ \text{minimum } 0 \text{ or } -1 \end{array} \right\}$ @ $\left\{ \begin{array}{l} \phi = \phi_0 \\ \phi = 0 \text{ or } \phi = 1 \end{array} \right\}$
  - V' takes local $\left\{ \begin{array}{l} \text{maximum} \\ \text{minimum} \end{array} \right\}$ @ $\left\{ \begin{array}{l} \phi = \phi_{1+} \\ \phi = \phi_{1-} \end{array} \right\}$
  - 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

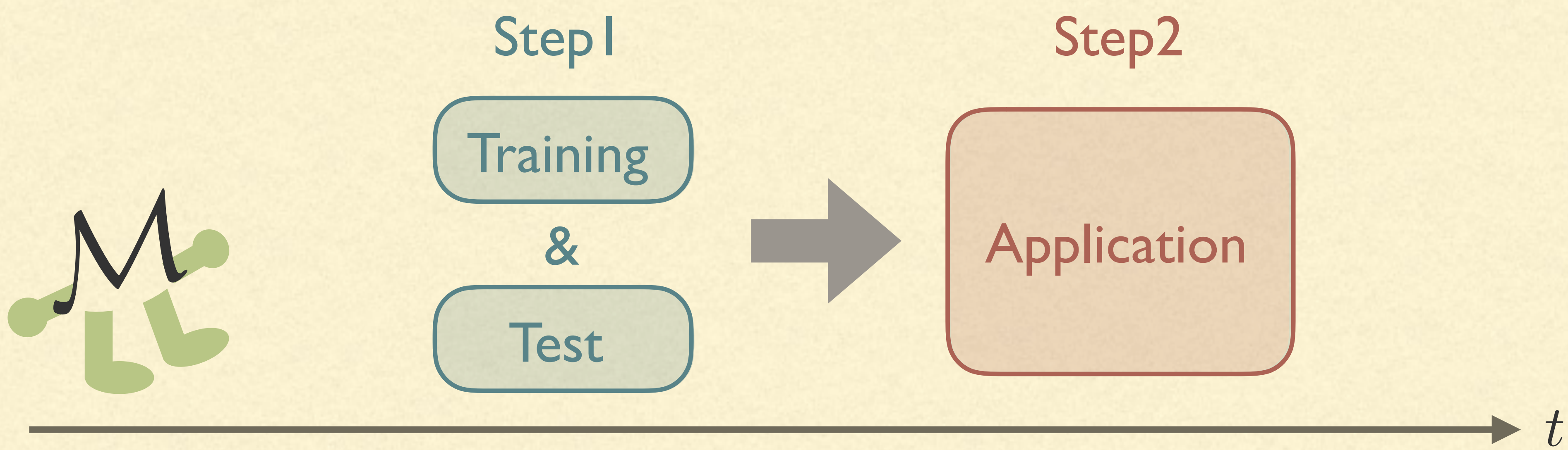
$\mathcal{M}$



We use a simple machine : $N = 2$

TRAINING & TEST & APPLICATION DATASET

- We construct training & test & application dataset

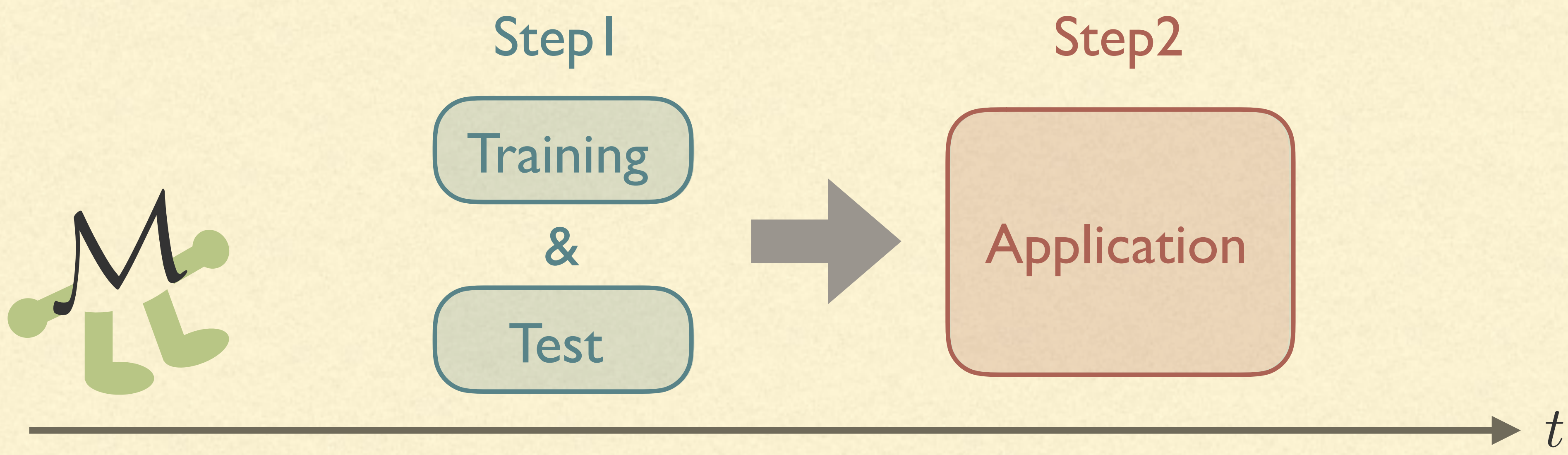


e.g.

- 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



e.g.

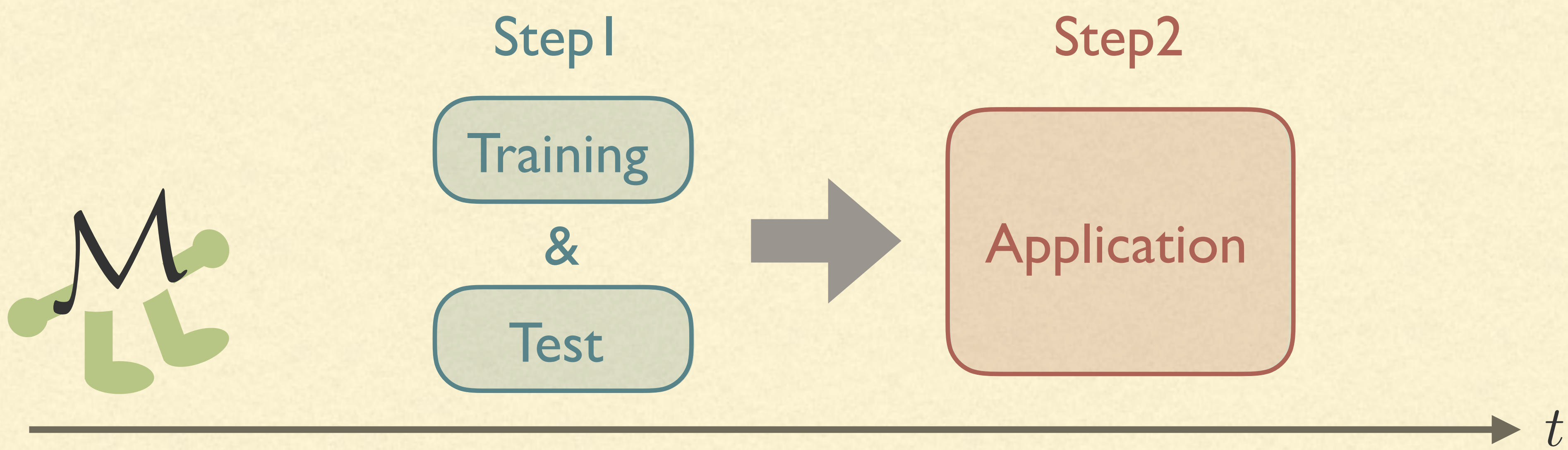
- Training dataset : used for training (→ next slide) 8,000 data from C2

- 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

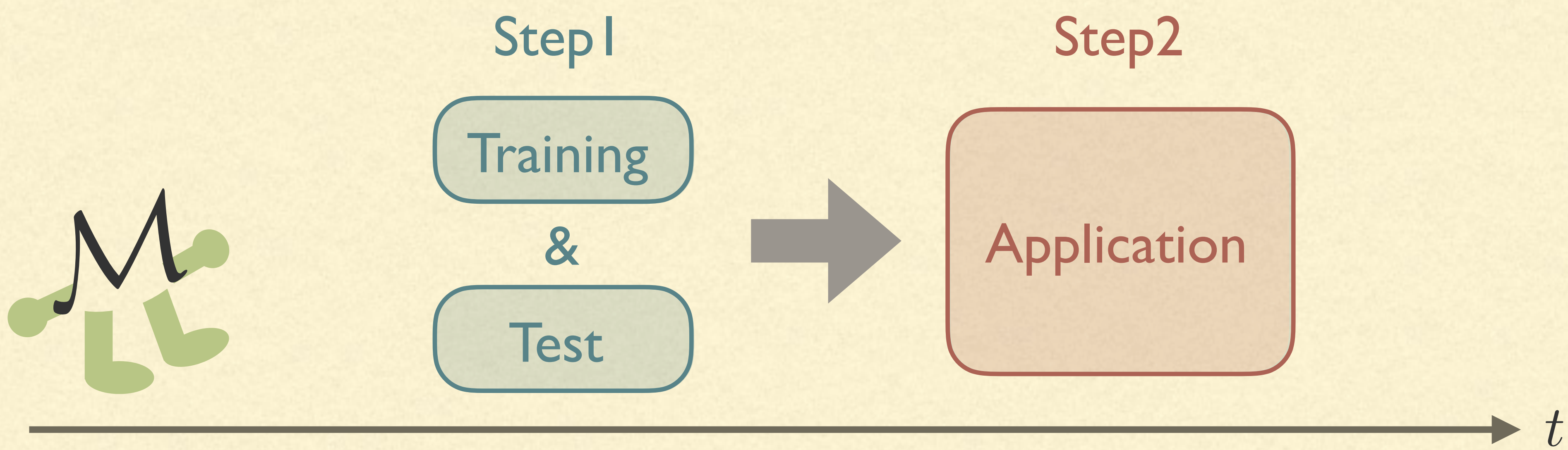


e.g.

- Training dataset : used for training (→ next slide) — 8,000 data from C3
- Test dataset : used to check that there is no overfitting — 2,000 data from C3
- Application dataset : machine is finally applied to this — 10,000 data from C3

TRAINING & TEST & APPLICATION DATASET

- We construct training & test & application dataset



e.g.

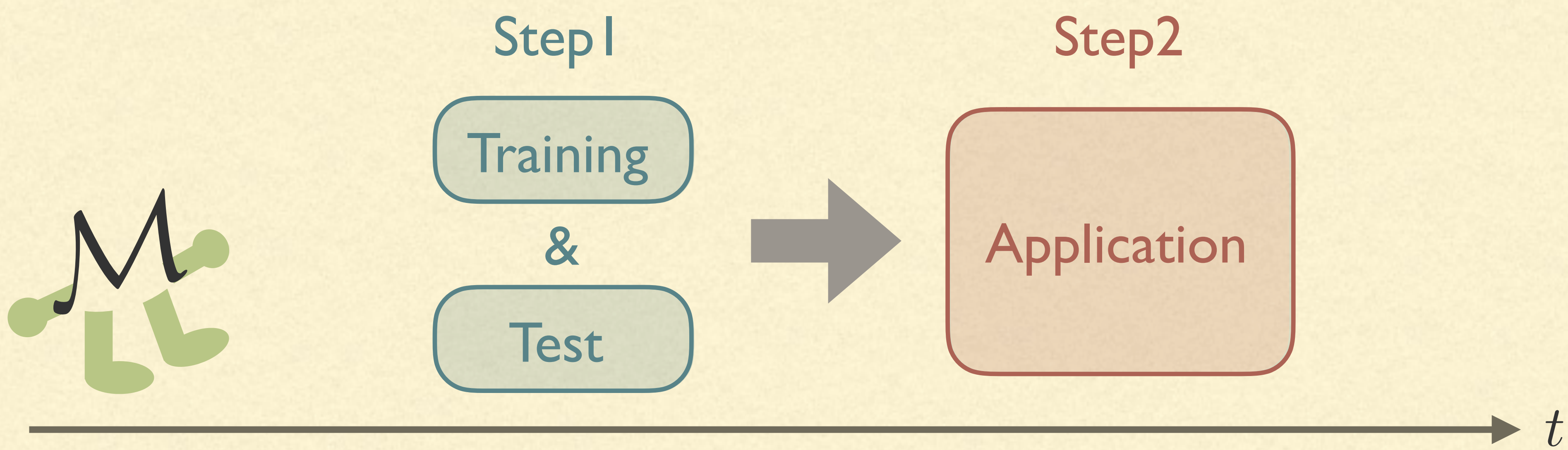
- Training dataset : used for training (→ next slide) 24,000 data from C1+C2+C3

- 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



e.g.

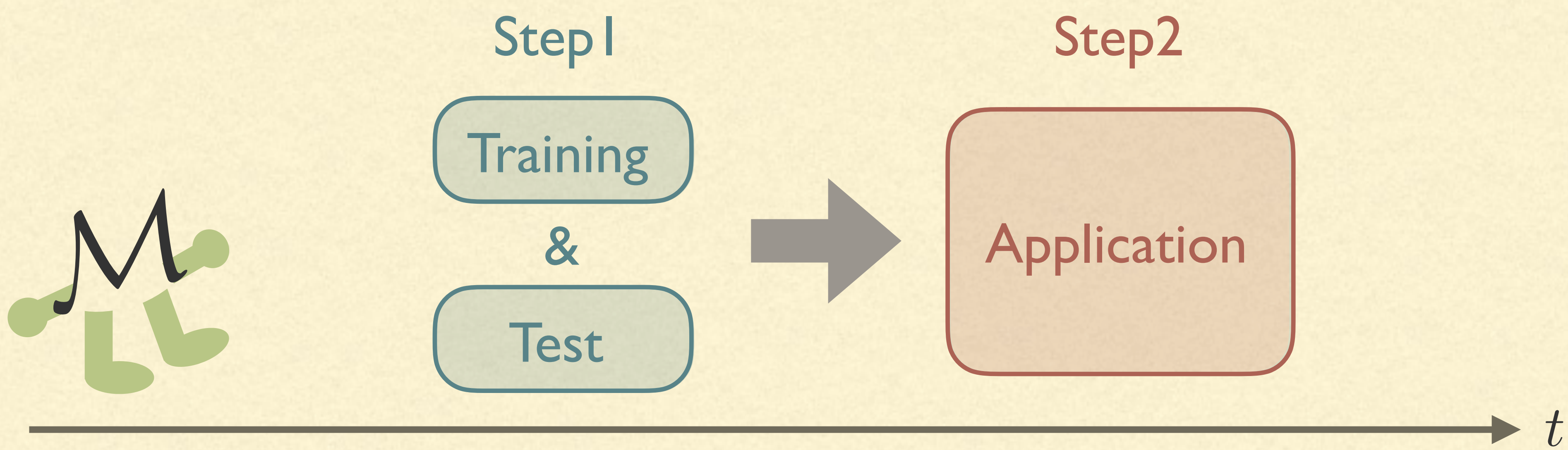
- Training dataset : used for training (→ next slide) 16,000 data from C2+C3

- 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

TRAINING & TEST & APPLICATION DATASET

- We construct training & test & application dataset



e.g.

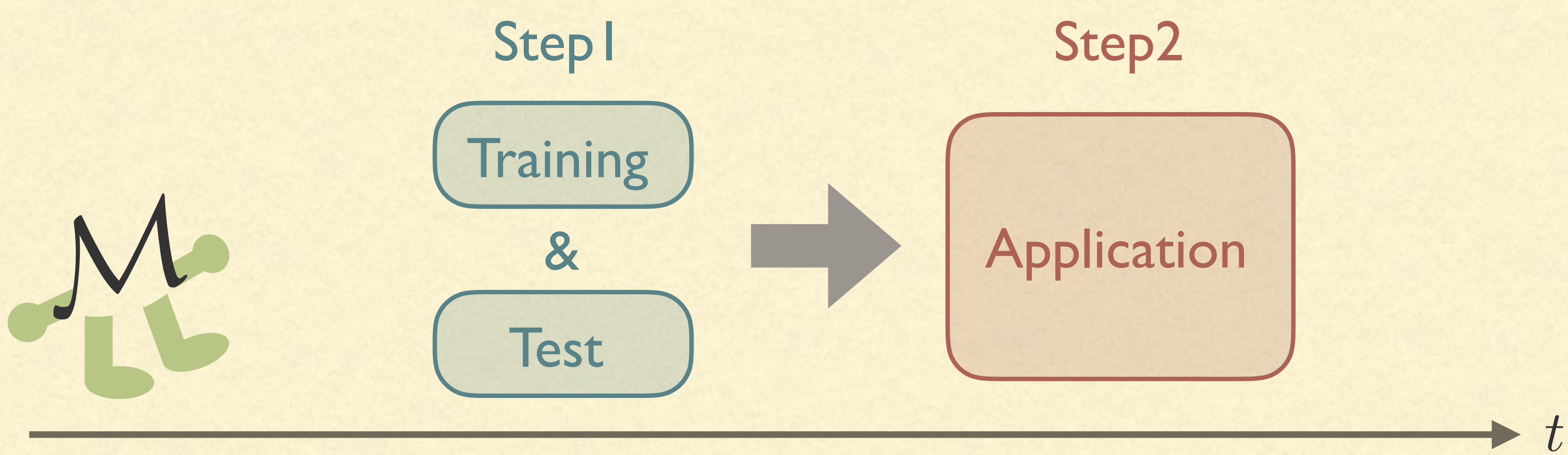
- Training dataset : used for training (→ next slide) 16,000 data from C3+C1

- 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

TRAINING & TEST & APPLICATION DATASET

- We construct training & test & application dataset



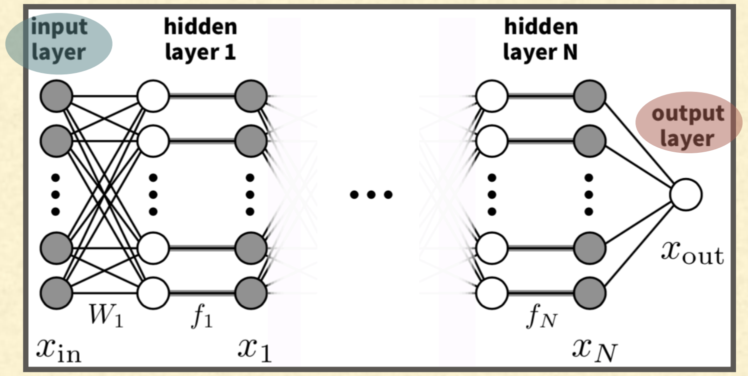
e.g.

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

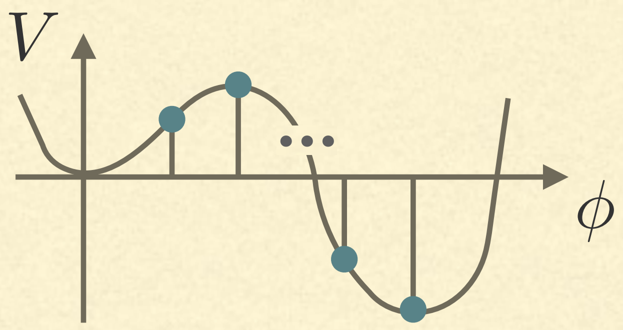
- Test dataset : used to check that there is no overfitting 4,000 data from C1+C2

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

MACHINE SETUP



- Input : sampled values of potential & its derivatives



$$x_{\rm in} = \left\{ V(\phi_{\rm sample}) \middle| \phi_{\rm sample} = \frac{1}{16}, \cdots, \frac{15}{16} \right\}$$
$$\oplus \left\{ V'(\phi_{\rm sample}) \middle| \phi_{\rm sample} = \frac{1}{16}, \cdots, \frac{15}{16} \right\} \oplus \left\{ V''(\phi_{\rm sample}) \middle| \phi_{\rm sample} = \frac{0}{16}, \cdots, \frac{16}{16} \right\}$$

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

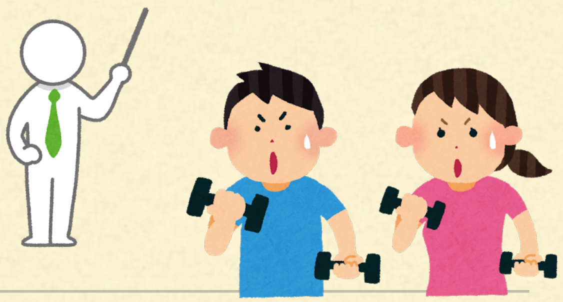
- Note : implicit rescaling of input & output

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

$$(x_{\rm in})_i \to \frac{(x_{\rm in})_i - \langle (x_{\rm in})_i \rangle}{\sigma_{(x_{\rm in})_i}} \qquad x_{\rm out} \to \frac{x_{\rm out} - \langle x_{\rm out} \rangle}{\sigma_{x_{\rm out}}}$$

  - $\langle\rangle$ & σ : mean & variance calculated over training & test dataset

TRAINING PROCESS



- Error function = how poorly the machine predicts

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

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

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

- Training = update of weights and biases using error function

$$W \to W - \alpha \frac{\partial E}{\partial W} \qquad b \to b - \alpha \frac{\partial E}{\partial b}$$

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

training dataset

- Implementation

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

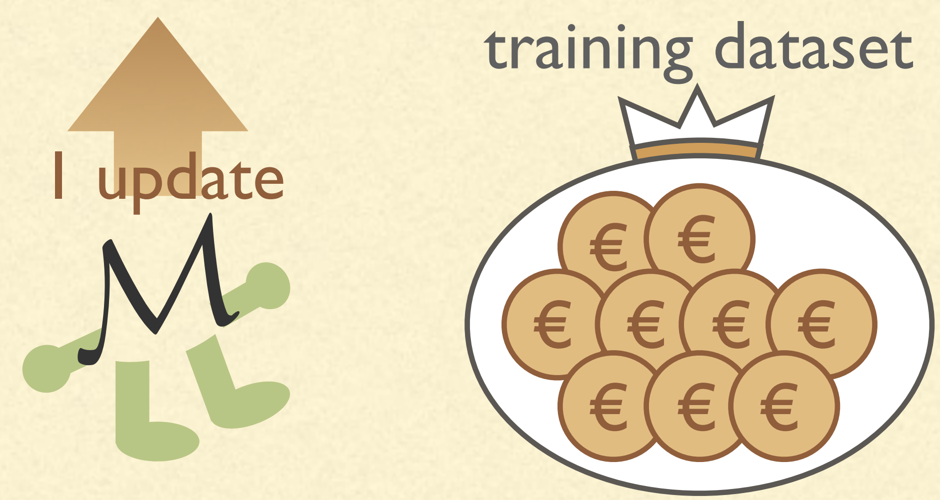
DETAILS OF TRAINING PROCESS

- 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)

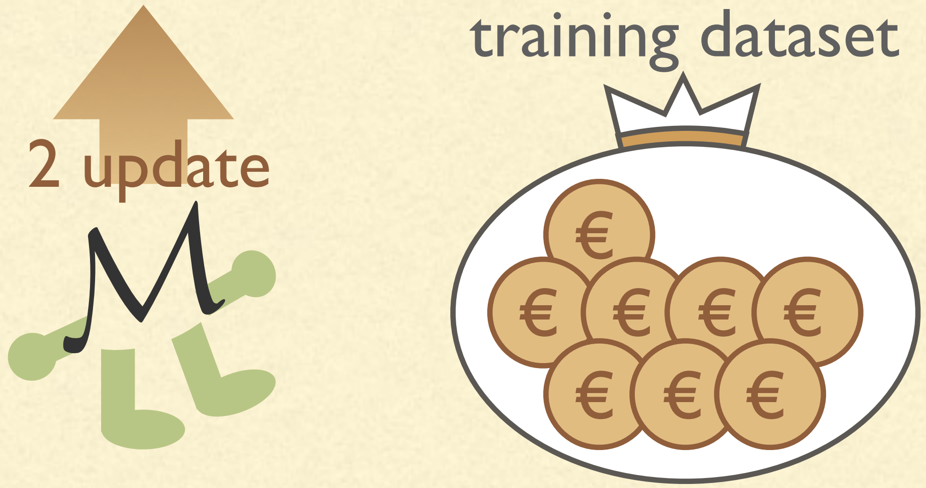
DETAILS OF TRAINING PROCESS

- 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)

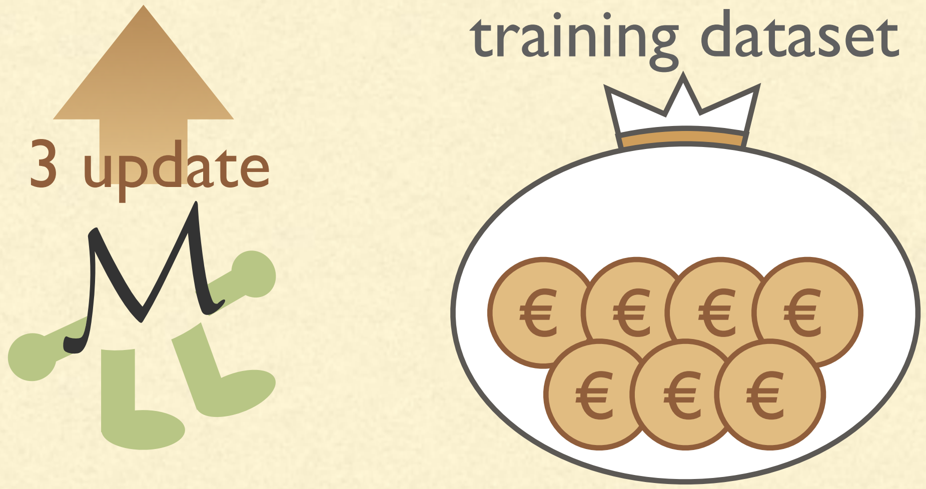
DETAILS OF TRAINING PROCESS

- 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)

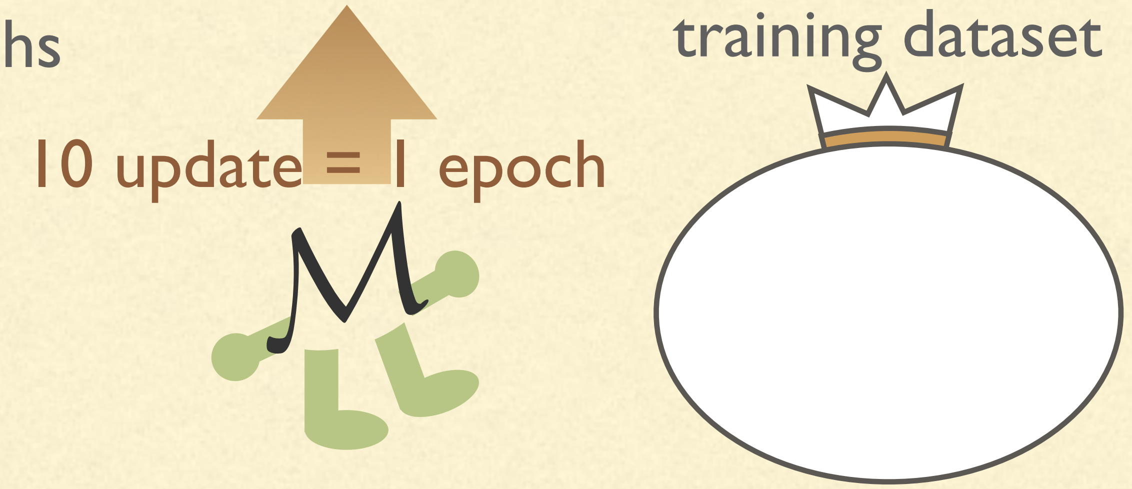
DETAILS OF TRAINING PROCESS

- 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

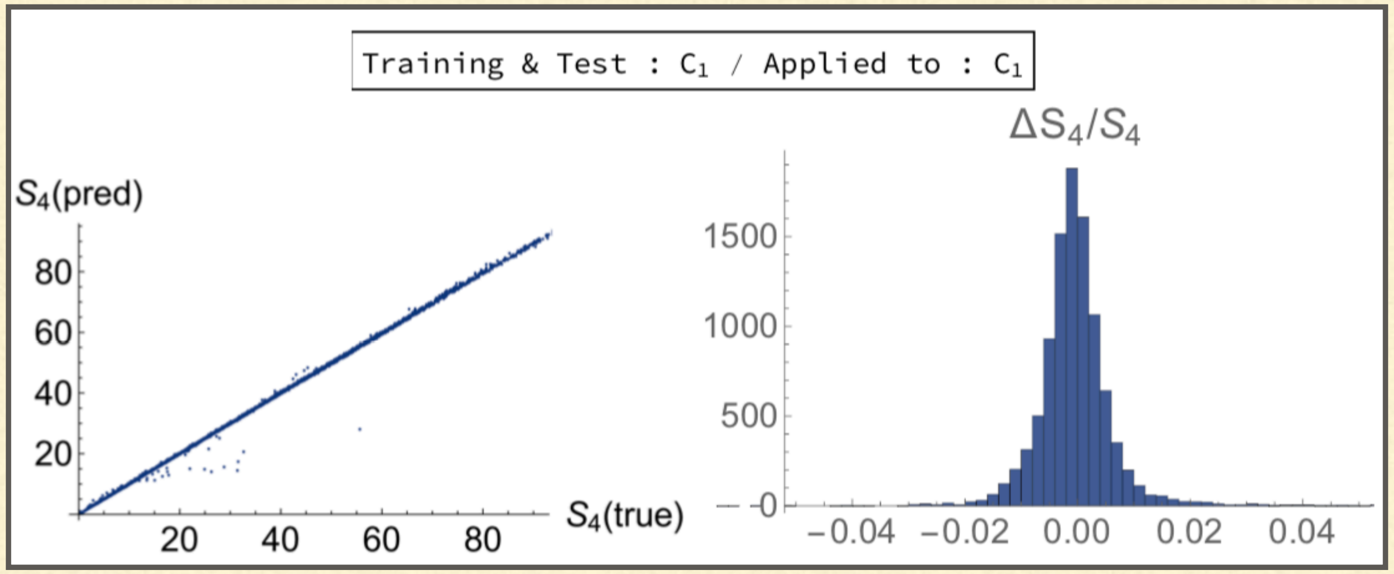
RESULTS

- 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



Average of 10 times trial

Training & Test	Applied to	$\langle\langle \lvert \Delta S_4/S_4 \rvert \rangle\rangle$
C_1	C_1	0.00607

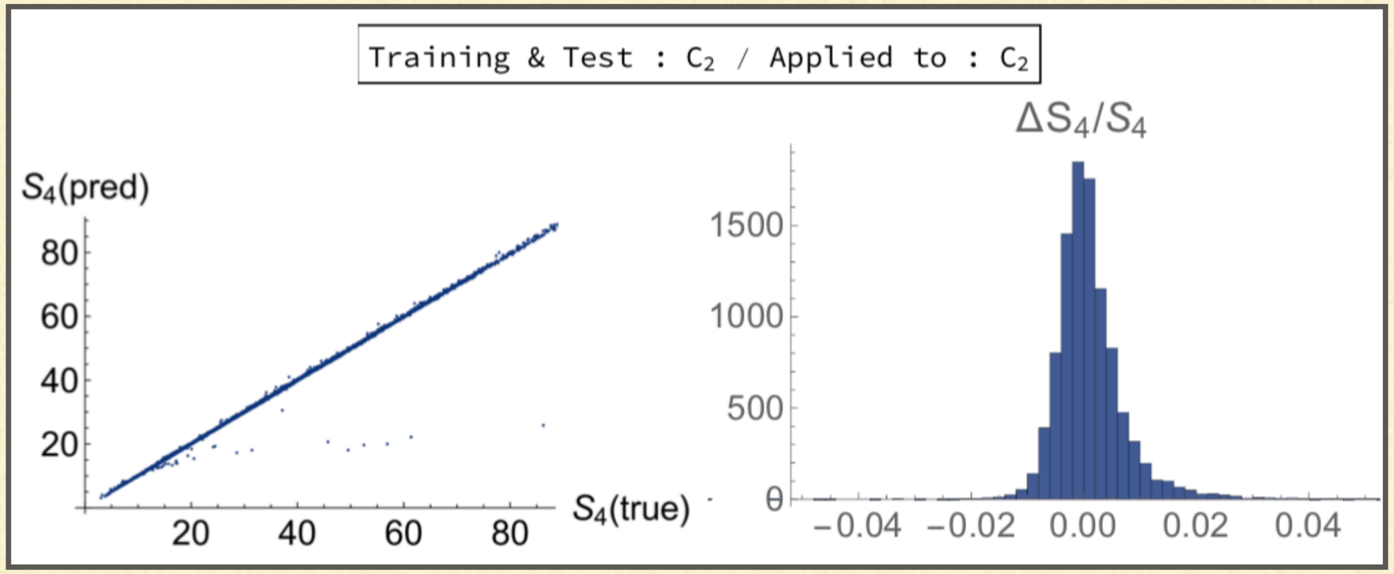
RESULTS

- 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	$\langle\langle\lvert\Delta S_4/S_4\rvert\rangle\rangle$
C_2	C_2	0.00423

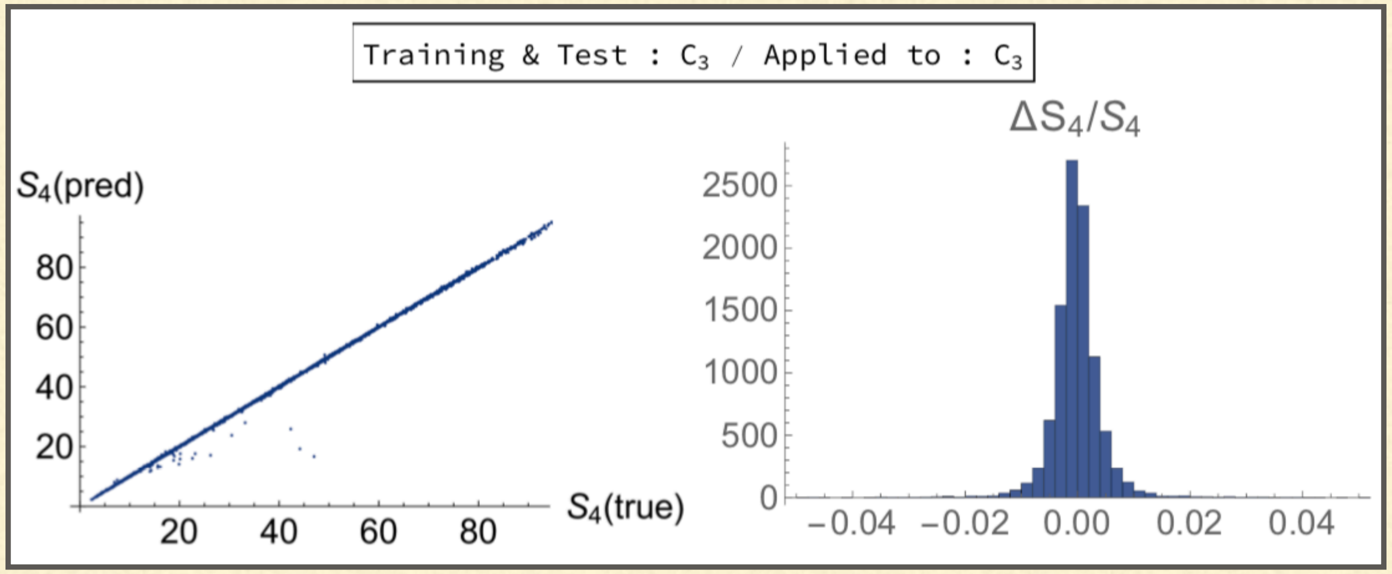
RESULTS

- 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	$\langle\langle \lvert\Delta S_4/S_4\rvert \rangle\rangle$
C_3	C_3	0.00418

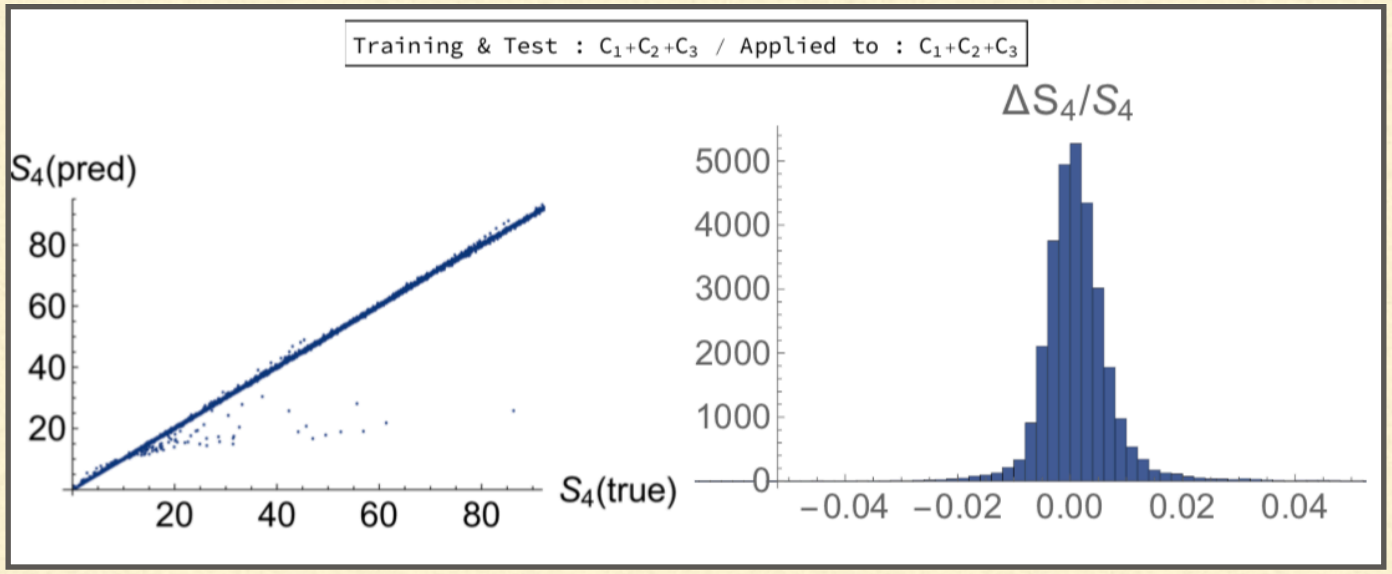
RESULTS

- 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	$\langle\langle\lvert\Delta S_4/S_4\rvert\rangle\rangle$
$C_1+C_2+C_3$	$C_1+C_2+C_3$	0.00503

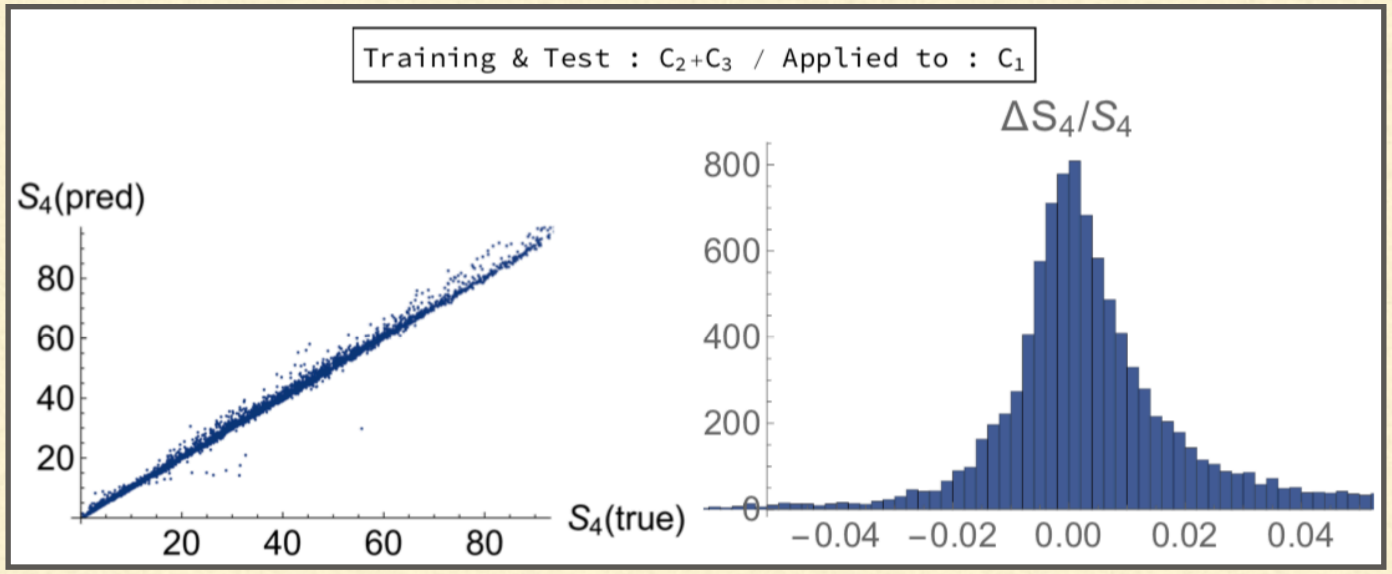
RESULTS

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

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

Application : 10,000 data from C1

Scatter plot for machine's performance



Average of 10 times trial

Training & Test	Applied to	$\langle\langle \lvert\Delta S_4/S_4\rvert \rangle\rangle$
$C_2 + C_3$	C_1	0.0248

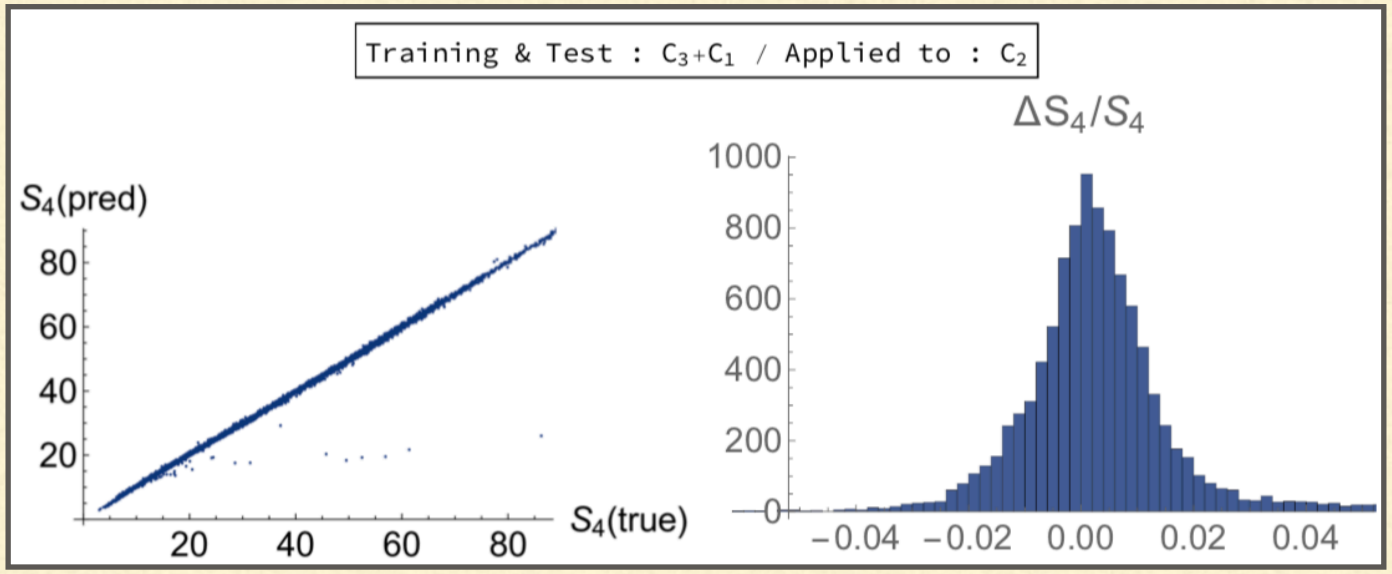
RESULTS

- 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	$\langle\langle\lvert\Delta S_4/S_4\rvert\rangle\rangle$
$C_3 + C_1$	C_2	0.0128

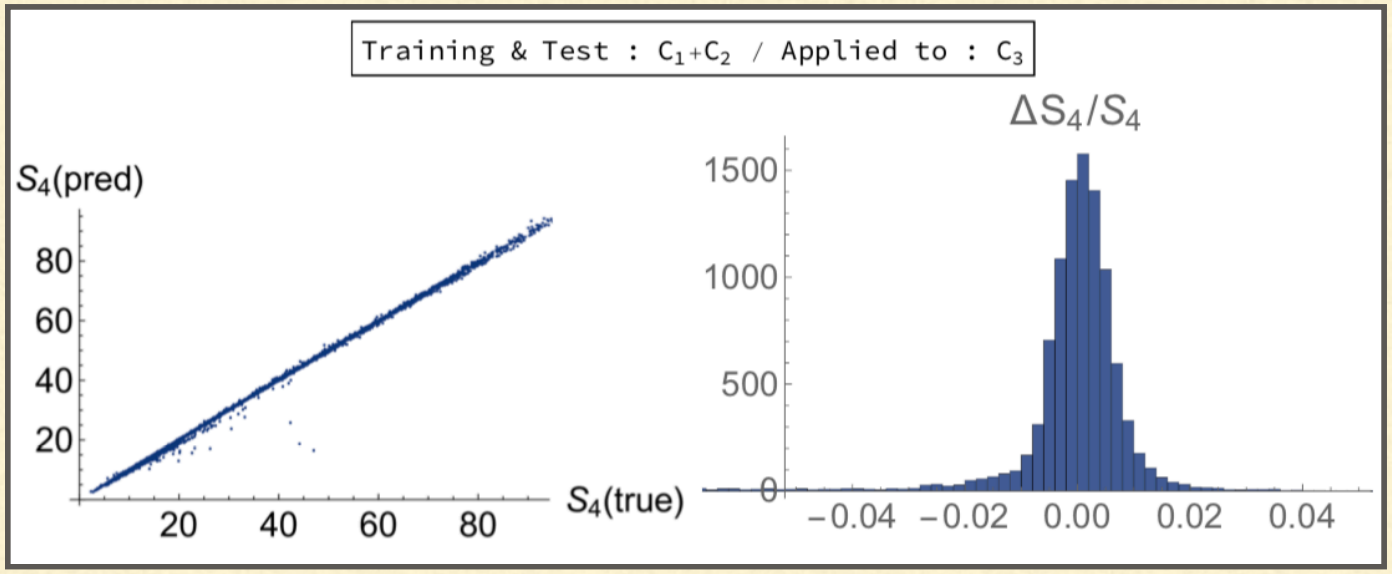
RESULTS

- 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	$\langle\langle\|\Delta S_4/S_4\|\rangle\rangle$
$C_1 + C_2$	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 C1+C2+C3 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 [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

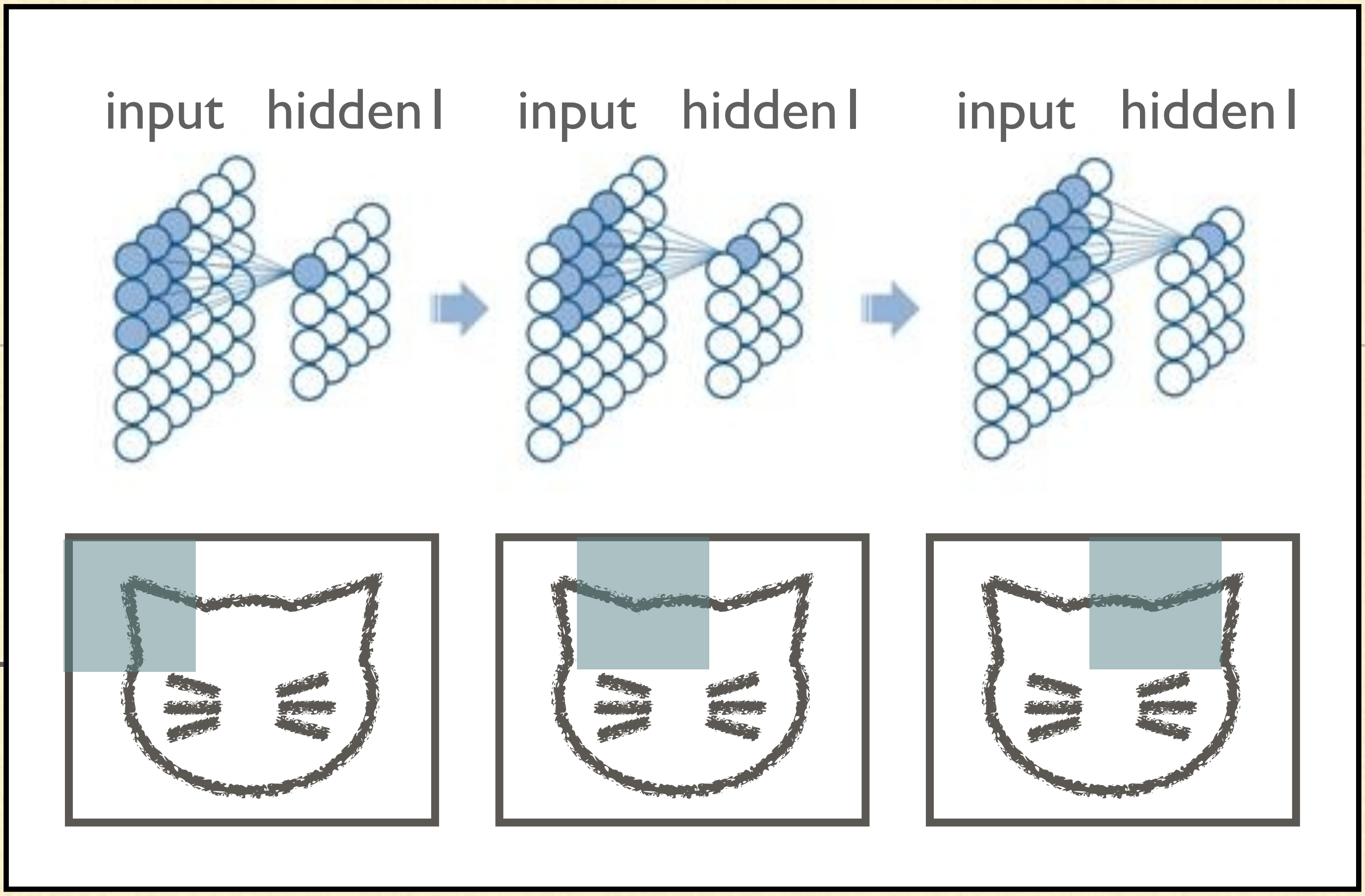
    e.g. 1) 2 dim. : convolutional neural network (CNN) may help

    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

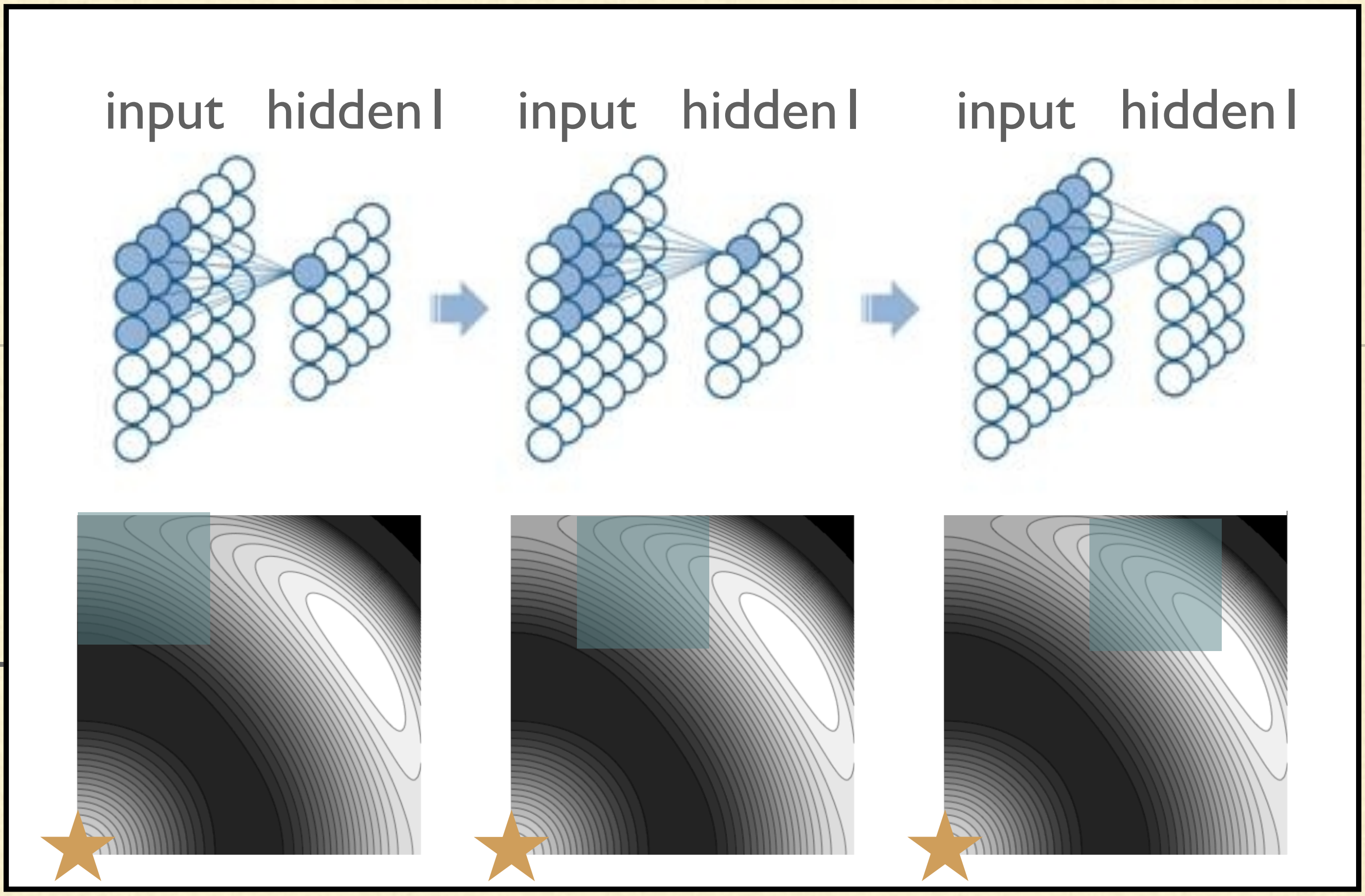
  by identifying the output as the initial position

DISCUSSION

- Generalizations?

  - Different spacetime dimensions →

  - Multidimensional transitions →



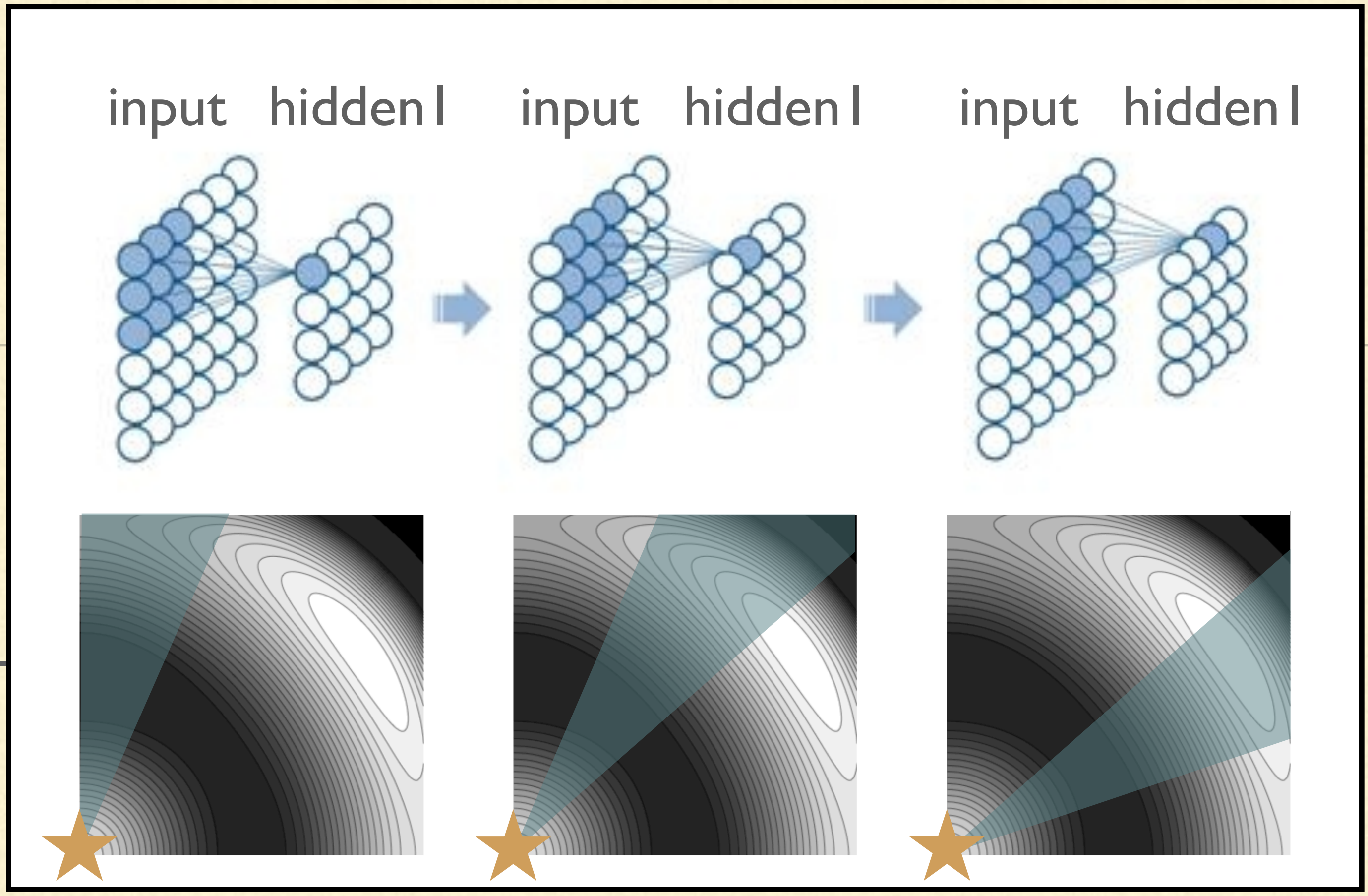
e.g. 1) 2 dim. : convolutional neural network (CNN) may help

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

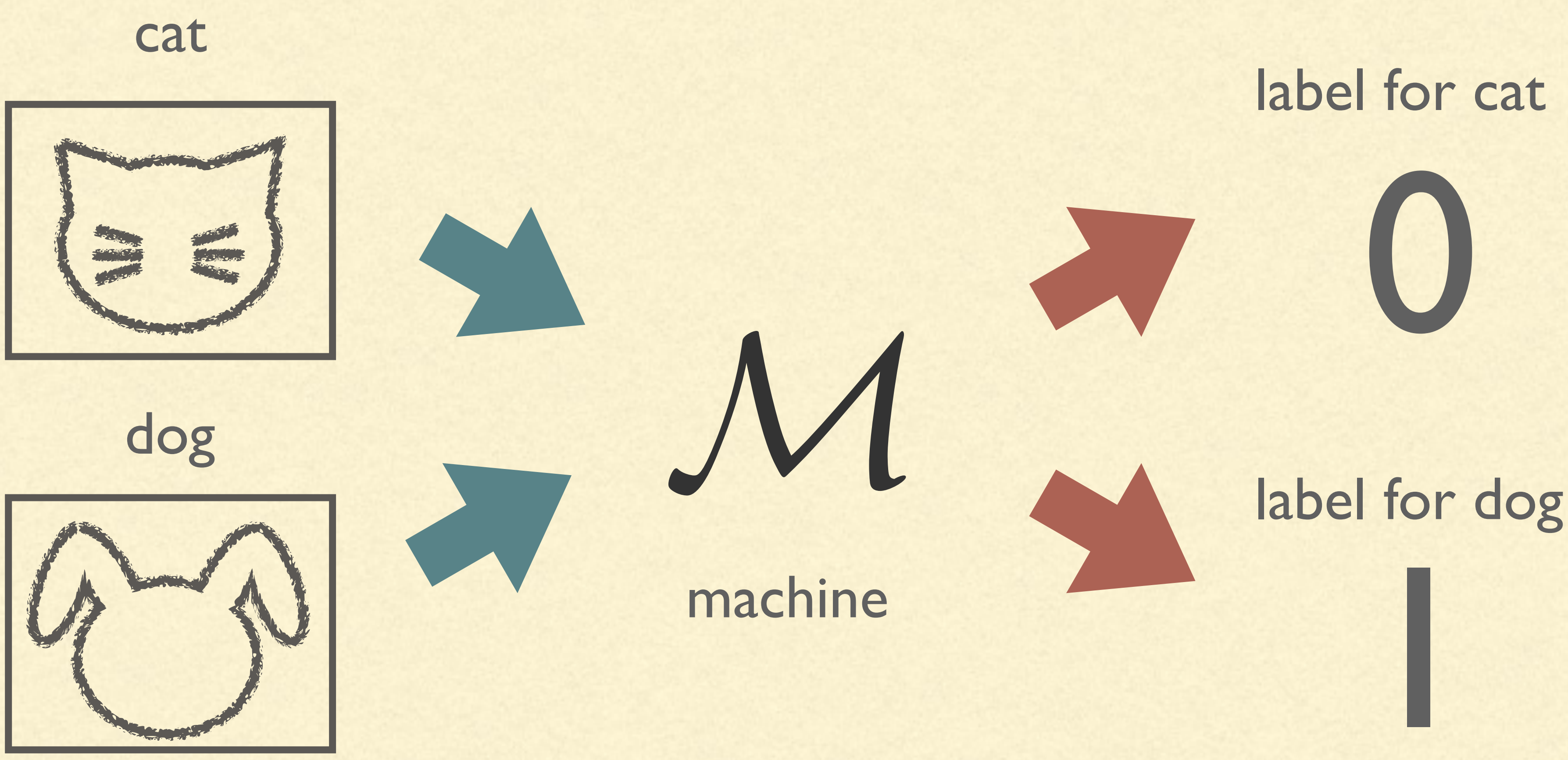
  3) ML may also be used for "initial position suggestor" in such public codes

  by identifying the output as the initial position

Others

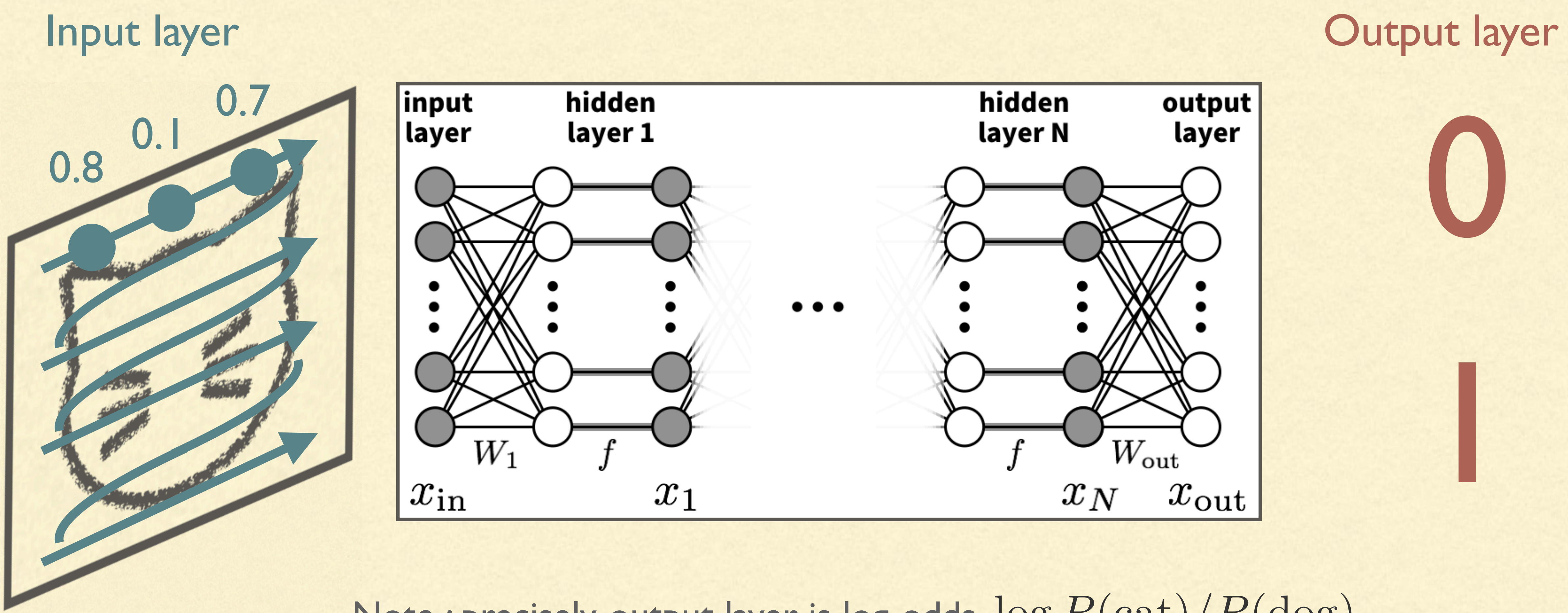
NEURAL NETWORK: IMAGE RECOGNITION

- Image classifier



NEURAL NETWORK: IMAGE RECOGNITION

- Image classifier : input = image / output = label



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

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