

The 4th NUSYS Summer School

Zhuhai, 2024.07.27-08.02

Introduction to machine learning in nuclear physics

Zhongming Niu

School of Physics and Optoelectronic Engineering, Anhui University

2024.07.29-30



Outline

- ① Introduction
- ② Bayesian neural network approach
 - ★ Bayesian approach
 - ★ Neural network (Tensorflow Playground, 1stOpt, PyTorch)
 - ★ Bayesian neural network (flexible Bayesian modeling software)
- ③ Results and discussion
 - ★ Nuclear masses
 - ★ Nuclear β -decay half-lives
 - ★ Some other results
- ④ Summary and perspectives

Outline

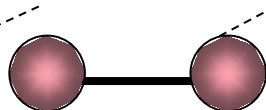
- ① Introduction
- ② Bayesian neural network approach
 - ★ Bayesian approach
 - ★ Neural network
 - ★ Bayesian neural network
- ③ Results and discussion
 - ★ Nuclear masses
 - ★ Nuclear β -decay half-lives
 - ★ Some other results
- ④ Summary and perspectives

Nuclear physics and machine learning

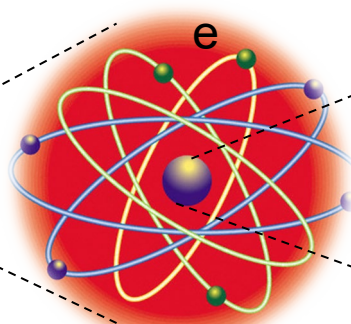
football



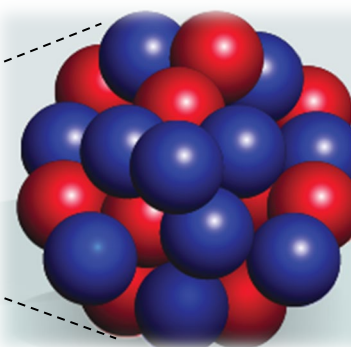
molecule



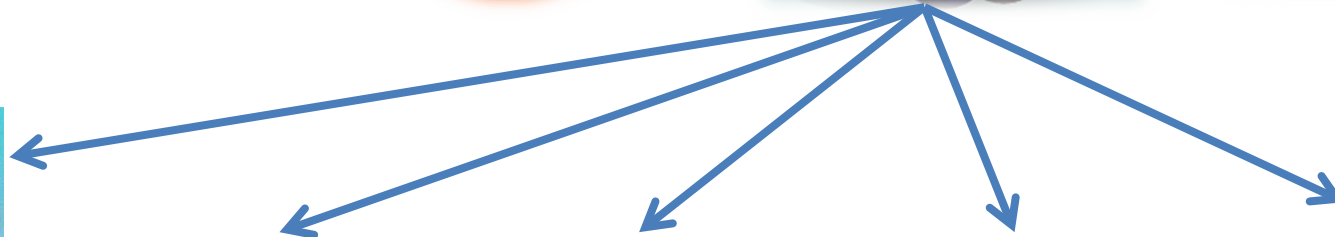
atom



nucleus



nucleon



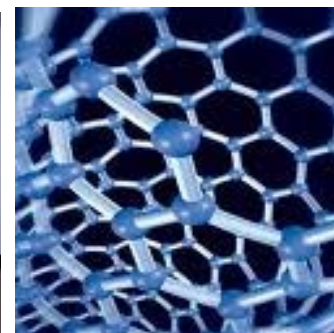
National Defense Construction



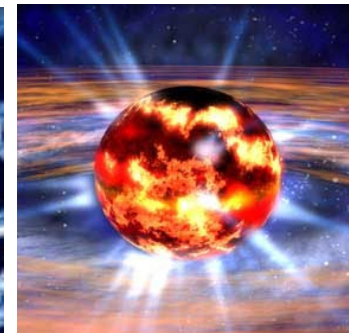
Energy and Power



Medical Health



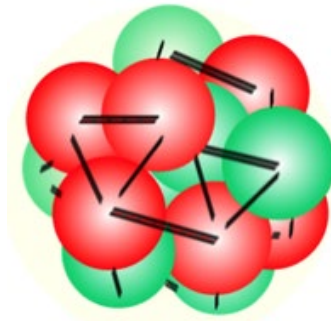
Materials Science



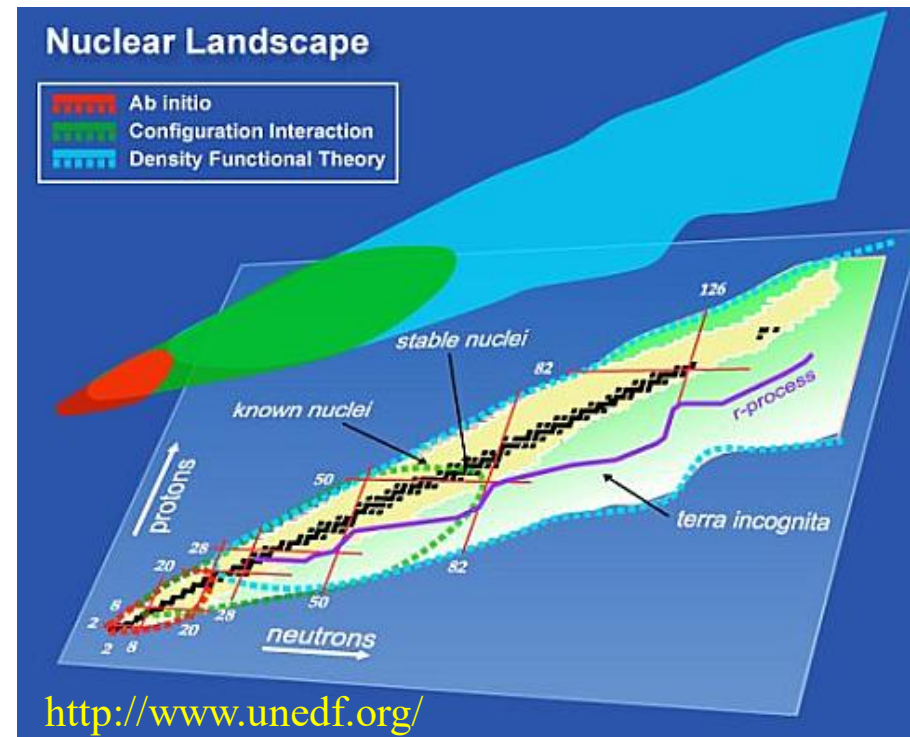
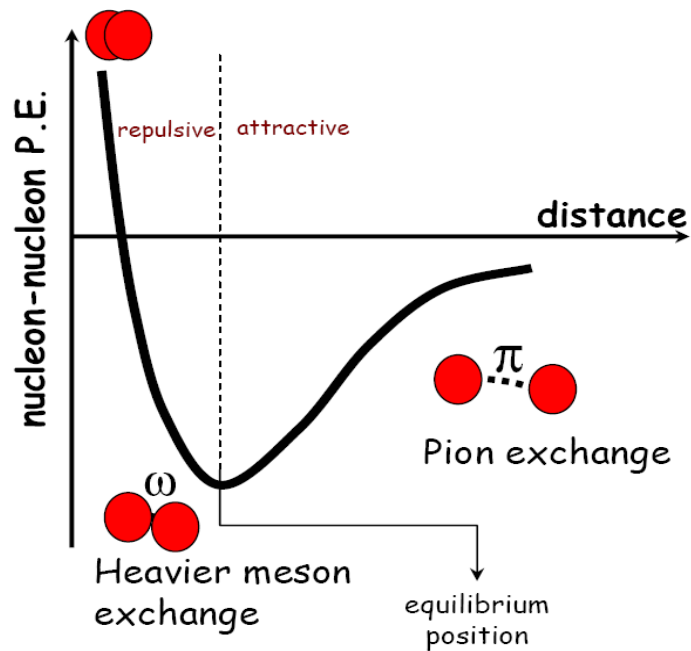
Astrophysics & Cosmology

Nuclear physics

● The nucleus is a **quantum many-body system** consisting of protons and neutrons bounded by **strong interactions**.



- ★ Nuclear force: realistic nuclear force, effective nuclear force
- ★ Nuclear many-body method: ab initio, shell model, density functional theory



Nuclear physics

● Nuclear data:

- Masses: ~2500
AME2020: [Wang2021CPC](#)
- Charge radii: ~1000
[Angeli2013ADNDT](#)
- Low-lying states: ~400, 900, 800, ...
 0_2^+ , 2_1^+ , 4_1^+ , ... <http://www.nndc.bnl.gov/>
- Half-lives: ~3000
NUBASE2020: [Kondev2021CPC](#)
- Nuclear reaction database EXFOR: ~25000
<https://www-nds.iaea.org/exfor/>

Widely used databases

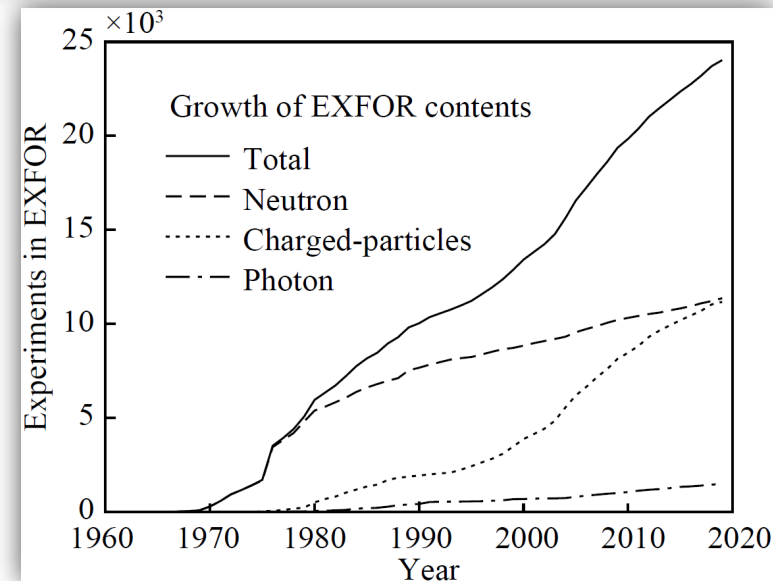
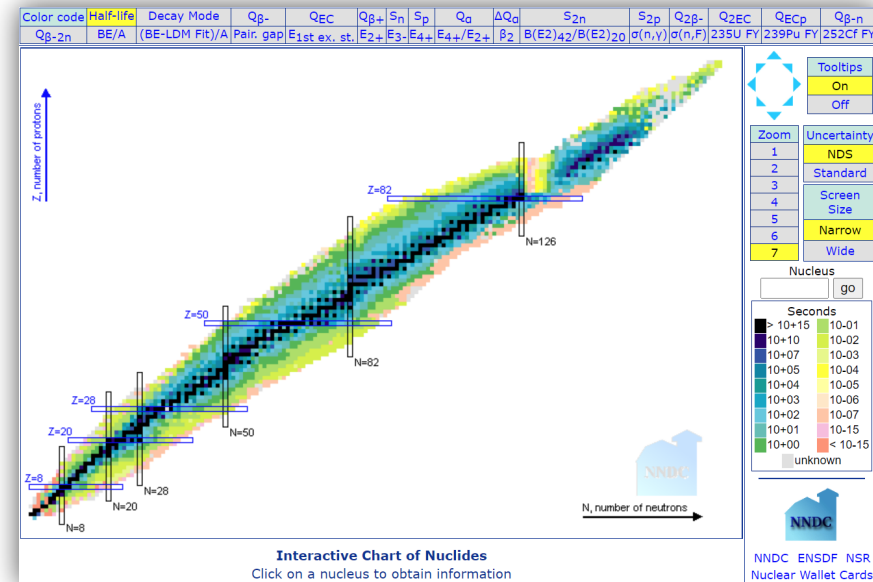
China: CENDL: <http://www.nuclear.csdb.cn/>

USA: ENDF: <http://www.nndc.bnl.gov/>

Japan: JENDL: <http://www.ndc.jaea.go.jp/>

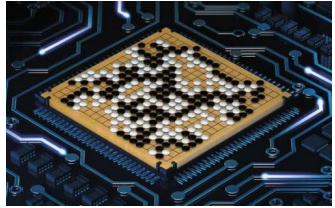
Europe: JEFF: <http://www.oecd-nea.org/>

Russia: BROND: <http://www-nds.iaea.org/>



Wang2021NPR

Machine learning



game of go



computer vision



translation



autonomous car



voice recognition

Machine learning in nuclear physics

- **(D)NN**: (Deep) Neural Network
- **BNN**: Bayesian Neural Network
- **CNN**: Convolutional Neural Network
- **MDN**: Mixture Density Network
- **(B)GP**: (Bayesian) Gaussian Processes
- **CGP**: Constrained Gaussian Processes
- **DT**: Decision Tree
- **NBP**: Naive Bayesian Probability Classifier
- **SVM**: Support Vector Machines
- **RBF**: Radial Basis Function
- **KRR**: Kernel Ridge Regression
- **CLEAN**: CLEAN Image Reconstruction
- ...

Nuclear physics and machine learning

● Nuclear structure:

➤ Masses:

NN: [Gazula1992NPA](#), [Athanasopoulos2004NPA](#), [Bayram2014ANE](#), [Zhang2017JPG](#), [Ming2022NST \(LZU\)](#),
[Yuksel2021IJMPE](#), [Li2022PRC \(CCNU\)](#), [Zeng2024PRC\(BUAA\)](#); DT: [Gao2021NST \(HZU\)](#); NBP: [Liu2021PRC \(UPC\)](#)
 BNN: [Utama2016PRC](#), [Niu2018PLB](#), [Niu2019PRC \(AHU\)](#), [Rodriguez2019EPL](#); BNN (Q_α): [Rodriguez2019JPG](#);
 BNN and BGP ($S_{1n/2n}$): [Neufcourt2018,2020PRC](#), [Neufcourt2019PRL](#)
 SVM: [Clark2006IJMPB](#); CLEAN: [Morales2010PRC](#); KRR: [Wu2020PRC](#), [Wu2021PLB \(PKU\)](#)
 RBF: [Wang2011PRC \(GXNU\)](#), [Niu2013,2016, 2019PRC \(AHU\)](#)

➤ Nuclear spins and parities:

NN: [Gernoth1993PLB](#); SVM: [Clark2006IJMPB](#)

➤ Nucleon density:

NN: [Yang2021PLB \(IMPCAS\)](#); [Shang2022NST\(JLU\)](#)

➤ Magnetic moment:

BNN: [Yuan2021CPC \(JLU\)](#)

➤ Charge radii:

BNN: [Utama2016JPG](#), [Dong2022PRC \(BUAA\)](#); CNN: [Cao2023NST](#), [Su2023Symmetry \(FUDAN\)](#);

NN: [Wu2020PRC \(SCU\)](#); NBP: [Ma2020PRC \(UPC\)](#)

➤ Excited states:

NN ($E(2_1^+)$): [Akkoyuna2020arXiv](#); NN (excitation spectrum): [Lasseri2020PRL](#), [Wang2022PLB \(AHU\)](#)

NN (giant dipole resonance key parameters): [Bai2021PLB \(LZU\)](#), [Wang2021PRC \(SYSU\)](#)

Nuclear physics and machine learning

● Nuclear decays and reactions:

➤ α -decay properties:

DT, NN (T_α): [Saxena2021JPG](#), [Ma2021CPC \(LZU\)](#); GP (T_α and Q_α): [Yuan2022CPC \(TJU\)](#);

DNN (T_α): [Li2022PRC \(CCNU\)](#)

➤ β -decay half-lives:

BNN: [Niu2019PRC \(AHU\)](#); SVM, NN: [Costiris2008arXiv](#); NN: [Costiris2009PRC](#), [Li2022SSPMA \(LZU\)](#)

NN(P_{1n}): [Wu2021PRC \(SCU\)](#)

➤ Fission yields:

BNN (yields): [Wang2019PRL](#), [Qiao2021PRC \(PKU\)](#); MDN(yields): [Lovell2019EPJWC](#); NN (T_{sf}): [Lay2024PRC](#)

➤ Cross-sections in proton induced spallation reactions:

BNN: [Ma2020CPC](#), [Peng2022PRC \(HTU\)](#)

➤ Neutron-nucleus scattering data: KRR: [Huang2022CTP \(PKU\)](#); BNN: [Liang2021Thesis \(GXNU\)](#)

➤ Fusion reaction cross-sections: NN: [Akkoyun2020NIMB](#)

➤ Electron-nucleus cross sections: NN: [Hammal2023PRC](#)

● Others:

➤ Optimization of model parameters: NN: [Scamps2021EPJA](#)

➤ Extrapolation problems in ab initio method:

NN: [Negoita2019PRC](#), [Jiang2019PRC](#); CGP: [Yoshida2020PRC](#)




➤ DFT: NN: [Hizawa2023PRC](#); NN+CNN: [Yang2023PRC](#); KRR: [Wu2022PRCL \(PKU\)](#)

➤ ab initio calculation: [Yang2022PLB](#), [2023PRC \(PKU\)](#)


Nuclear physics and machine learning

REVIEWS OF MODERN PHYSICS, VOLUME 94, JULY-SEPTEMBER 2022


Colloquium: Machine learning in nuclear physics

Amber Boehnlein , Markus Diefenthaler , Nobuo Sato, Malachi Schram , and Veronique Ziegler


Thomas Jefferson National Accelerator Facility,
12000 Jefferson Avenue, Newport News, Virginia 23606, USA

Cristiano Fanelli 

Laboratory for Nuclear Science and Institute for Artificial Intelligence and Fundamental Interactions, Massachusetts Institute of Technology,
Cambridge, Massachusetts 02139, USA

Morten Hjorth-Jensen 



Facility for Rare Isotope Beams and Department of Physics and Astronomy,
Michigan State University, East Lansing, Michigan 48824, USA
and Department of Physics and Center for Computing in Science Education,
University of Oslo, N-0316 Oslo, Norway

Tanja Horn 

Department of Physics, The Catholic University of America, Washington, D.C. 20064, USA
and Thomas Jefferson National Accelerator Facility, 12000 Jefferson Avenue,
Newport News, Virginia 23606, USA

Michelle P. Kuchera


Department of Physics and Department of Mathematics and Computer Science,
Davidson College, Davidson, North Carolina 28035, USA

Dean Lee , Witold Nazarewicz , and Peter Ostroumov


Facility for Rare Isotope Beams and Department of Physics and Astronomy,
Michigan State University, East Lansing, Michigan 48824, USA

Kostas Orginos


Department of Physics, William & Mary, Williamsburg 23185, Virginia, USA
and Thomas Jefferson National Accelerator Facility,
12000 Jefferson Avenue, Newport News, Virginia 23606, USA

Alan Poon  and Xin-Nian Wang

Nuclear Science Division, Lawrence Berkeley National Laboratory,
1 Cyclotron Road, Berkeley, California 94720, USA

Alexander Scheinker 

Accelerator Operations and Technology Division Applied Electrodynamics Group,
Los Alamos National Laboratory, Los Alamos, New Mexico 87544, USA

Michael S. Smith 

Physics Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 37831-6354, USA

Long-Gang Pang 

Key Laboratory of Quark and Lepton Physics, Institute of Particle Physics,
Central China Normal University, Wuhan 430079, China


 (published 8 September 2022)

Eur. Phys. J. A (2021) 57:100

<https://doi.org/10.1140/epja/s10050-020-00290-x>
**THE EUROPEAN
PHYSICAL JOURNAL A**


Review

A.I. for nuclear physics

Paulo Bedaque¹, Amber Boehnlein^{2,3} , Mario Cromaz³, Markus Diefenthaler², Latifa Elouadrhiri², Tanja Horn⁴, Michelle Kuchera⁵, David Lawrence², Dean Lee⁶, Steven Lidia⁶, Robert McKeown², Wally Melnitchouk², Witold Nazarewicz⁶, Kostas Orginos^{2,7}, Yves Roblin², Michael Scott Smith⁸, Malachi Schram⁹, Xin-Nian Wang³

¹ University of Maryland, College Park, MD, USA

² Thomas Jefferson National Accelerator Facility, Newport News, VA, USA

³ Lawrence Berkeley National Laboratory, Berkeley, CA, USA

⁴ Catholic University, Washington D.C., USA

⁵ Davidson College, Davidson, NC, USA

⁶ Michigan State University, East Lansing, MI, USA

⁷ College of William & Mary, Williamsburg, VA, USA

⁸ Oak Ridge National Laboratory, Oak Ridge, TN, USA

⁹ Pacific Northwest National Laboratory, Richland, WA, USA

SCIENCE CHINA

Physics, Mechanics & Astronomy



• Invited Review •

 August 2023 Vol. 66 No. 8: 282001
<https://doi.org/10.1007/s11433-023-2116-0>

Machine learning in nuclear physics at low and intermediate energies

Wanbing He^{1,2*}, Qingfeng Li^{3,4*}, Yugang Ma^{1,2*}, Zhongming Niu^{5*},
Junchen Pei^{6,7*}, and Yingxun Zhang^{8,9*}

¹ Key Laboratory of Nuclear Physics and Ion-beam Application (MOE), Institute of Modern Physics, Fudan University, Shanghai 200433, China;

² Shanghai Research Center for Theoretical Nuclear Physics, NSFC and Fudan University, Shanghai 200438, China;

³ School of Science, Huzhou University, Huzhou 313000, China;

⁴ Institute of Modern Physics, Chinese Academy of Sciences, Lanzhou 730000, China;

⁵ School of Physics and Optoelectronic Engineering, Anhui University, Hefei 230601, China;

⁶ State Key Laboratory of Nuclear Physics and Technology, School of Physics, Peking University, Beijing 100871, China;

⁷ Southern Center for Nuclear-Science Theory (SCNT), Institute of Modern Physics, Chinese Academy of Sciences, Huizhou 516000, China;

⁸ Department of Nuclear Physics, China Institute of Atomic Energy, Beijing 102413, China;

⁹ Guangxi Key Laboratory of Nuclear Physics and Technology, Guangxi Normal University, Guilin 541004, China

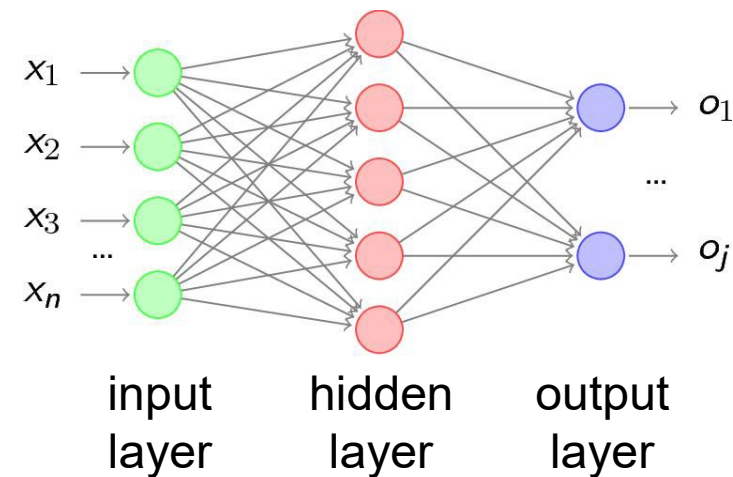
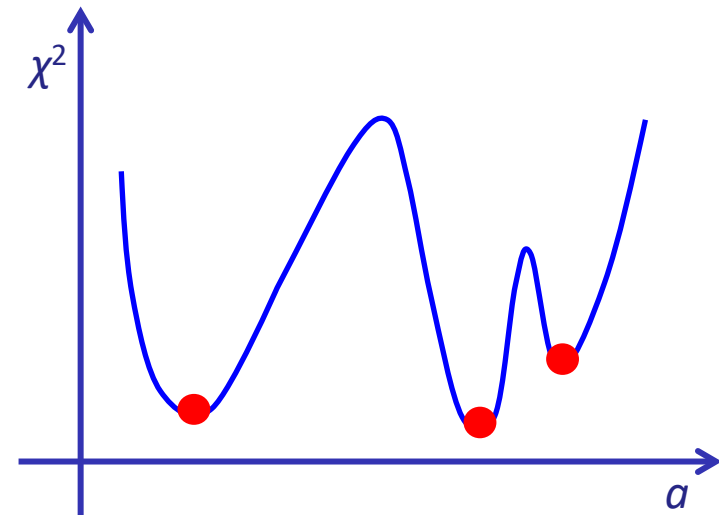
Bayesian neural network

Bayesian neural network:

- ✓ Avoid over fitting by using prior distribution
- ✓ Quantify uncertainty
- ✓ Automatically identify irrelevant inputs

- ✓ Powerful data fitting ability
- ✓ Predict unknown data
- ✓ Simple learning rules and hence easy for computer implementation

■ Black-box algorithms, data hunger, time-consuming



Introduction

Physics Letters B 778 (2018) 48–53

Contents lists available at [ScienceDirect](https://www.sciencedirect.com)

Physics Letters B

www.elsevier.com/locate/physletb



Nuclear mass predictions based on Bayesian neural network approach with pairing and shell effects



Z.M. Niu (牛中明)^{a,b}, H.Z. Liang (梁豪兆)^{b,c,d,*}

PHYSICAL REVIEW C **99**, 064307 (2019)

^a School of Physics
^b Interdisciplinary
^c RIKEN Nishina
^d Department of

Predictions of nuclear β -decay half-lives with machine learning and their impact on r -process nucleosynthesis

Z. M. Niu (牛中明)^{1,2}, H. Z. Liang (梁豪兆)^{3,4,*}, B. H. Sun (孙保华)⁵, W. H. Long (龙文辉)⁶ and Y. F. Niu (牛一斐)^{6,7}

¹School of Physics and Materials Science, Anhui University, Hefei 230601, China

²Institute of Physical Science and Information Technology, Anhui University, Hefei 230601, China

³RIKEN Nishina Center, Wako 351-0198, Japan

PHYSICAL REVIEW C **106**, L021303 (2022)

Letter

⁷ELI-NP,

Nuclear mass predictions with machine learning reaching the accuracy required by r -process studies

Z. M. Niu (牛中明)^{1,*} and H. Z. Liang (梁豪兆)^{b,2,3,†}

¹School of Physics and Optoelectronic Engineering, Anhui University, Hefei 230601, China

²Department of Physics, Graduate School of Science, The University of Tokyo, Tokyo 113-0033, Japan

³RIKEN Nishina Center, Wako 351-0198, Japan

Outline

- ① Introduction
- ② Bayesian neural network approach
 - ★ Bayesian approach
 - ★ Neural network
 - ★ Bayesian neural network
- ③ Results and discussion
 - ★ Nuclear masses
 - ★ Nuclear β -decay half-lives
 - ★ Some other results
- ④ Summary and perspectives

Example: toss coin

- Example: toss a coin of unknown properties;
probability ω of the coin landing heads



- ✓ Choose some criterion, such as maximum likelihood
- ✓ Find the optimal estimator according to this criterion, such as the frequency of heads in past tosses
- ✓ Express this unknown properties using a probability distribution over possible values based on our intuitive believes
- ✓ Update this distribution using the Bayes' theorem as the outcome of each toss becomes known

$$\omega = \frac{N^{\text{head}}}{N^{\text{total}}}$$

$$p(\omega | D) = \frac{p(D | \omega)p(\omega)}{p(D)}$$

Maximum likelihood method

Maximum likelihood method

Suppose N trials are performed and the number of head-ups is k . If the probability of a head-up is ω , then the likelihood function is

$$p(N, k | \omega) = C_N^k \omega^k (1 - \omega)^{N-k}$$

$$\frac{dp(N, k | \omega)}{d\omega} = 0 \Rightarrow \omega = \frac{k}{N}$$

E.g.: $N = 10, k = 6$:

✓ $\omega = 0.5$ 时:

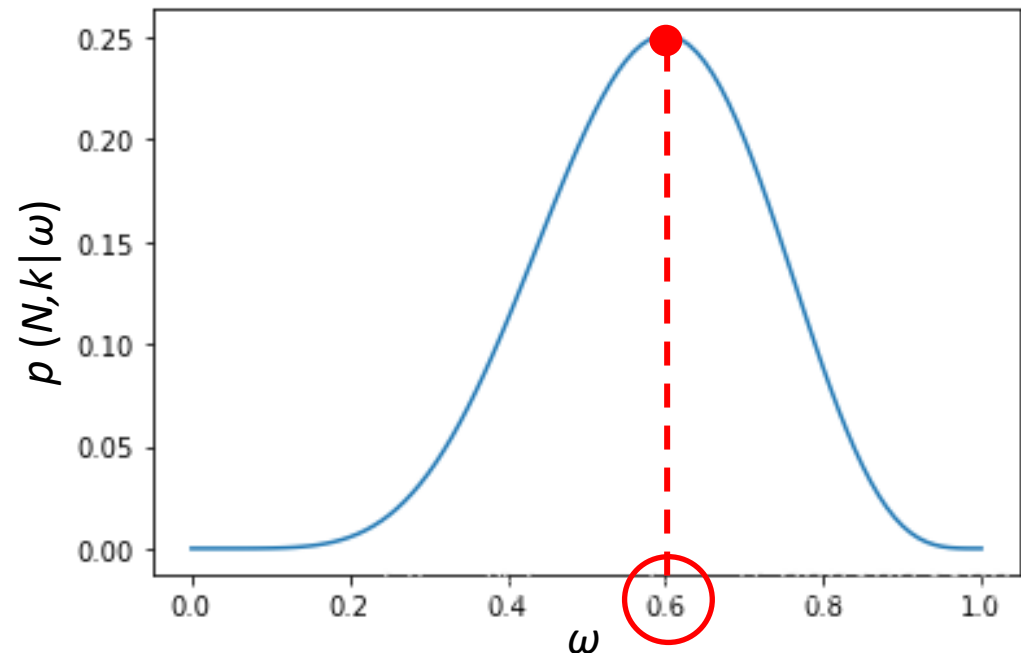
$$p = C_{10}^6 0.5^6 (1 - 0.5)^4 \approx 0.21$$

✓ $\omega = 0.6$ 时:

$$p = C_{10}^6 0.6^6 (1 - 0.6)^4 \approx 0.25$$

✓ others:

see right figure



Bayesian method

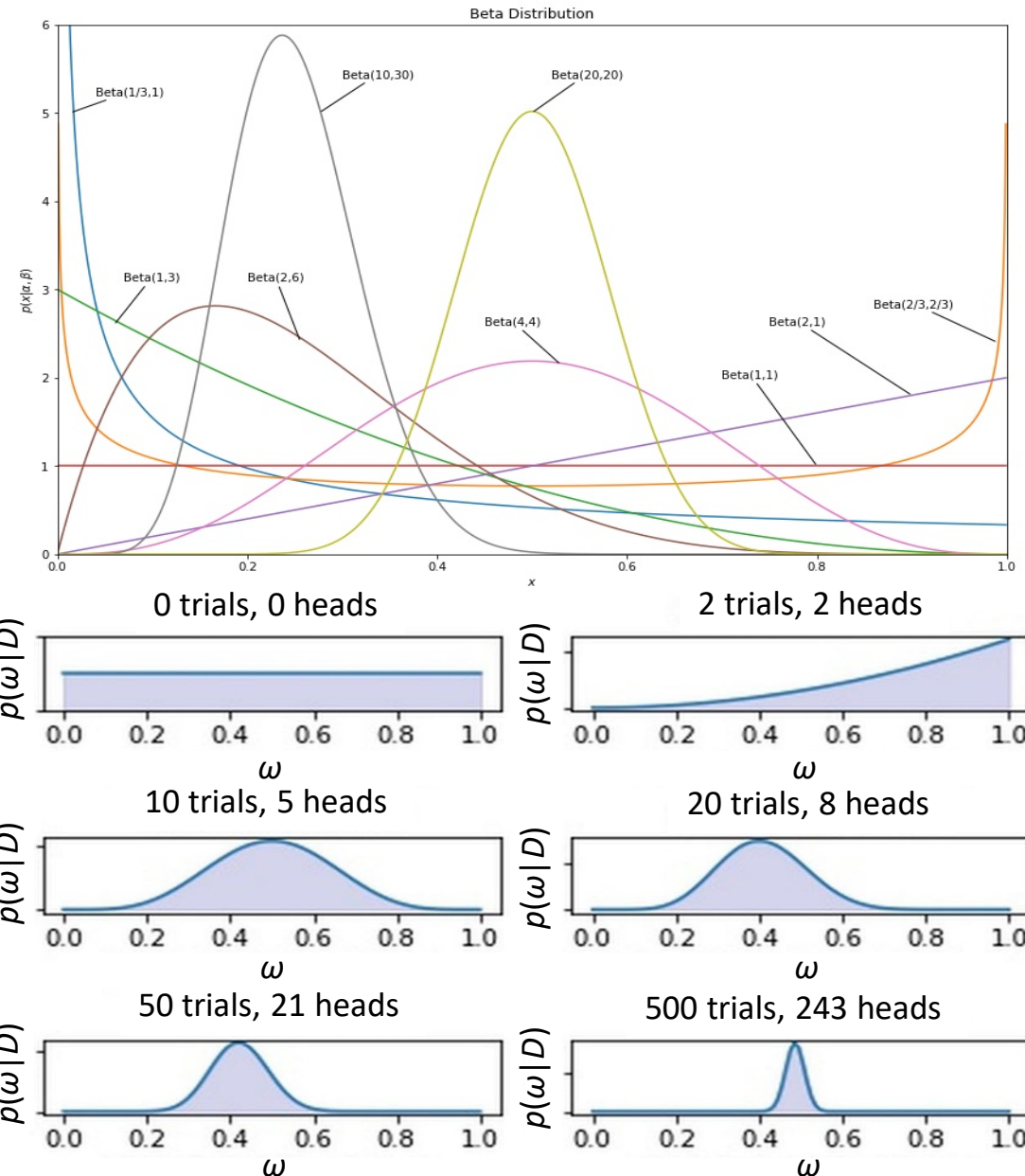
Bayesian method

$$p(\omega | D) = \frac{p(D | \omega) p(\omega)}{p(D)}$$

$$p(\omega) \sim \text{Beta}(\alpha, \beta)$$

E.g.: $\text{Beta}(1,1) \equiv \text{Uniform}(0,1)$

$$p(D | \omega) = P(N, k | \omega) \\ = C_N^k \omega^k (1 - \omega)^{N-k}$$



Bayesian and frequentist (traditional) views

● Differences between Bayesians and frequentists [Bishop2006Springer](#)

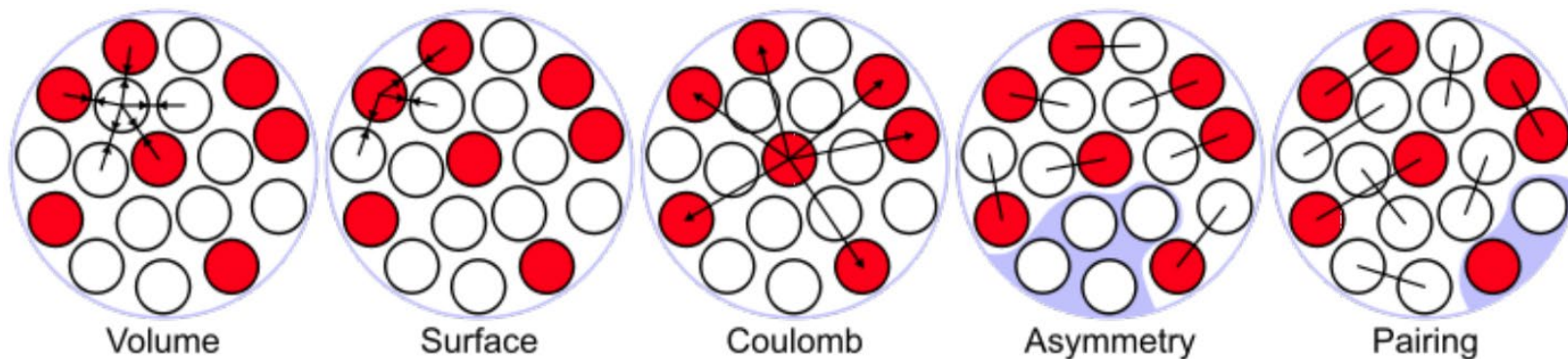
Frequentists:

- ✓ **Data are a repeatable random sample**
 - there is a frequency
- ✓ Underlying parameters remain constant during this repeatable process
- ✓ **Parameters are unknown but fixed**

Bayesians:

- ✓ Data are observed from the realized sample
- ✓ **Parameters are unknown and described probabilistically**
- ✓ **Data are fixed**

Example: LDM



$$B(Z, A) = a_v A - a_{surf} A^{2/3} - a_c \frac{Z^2}{A^{1/3}} - a_{sym} \frac{(N - Z)^2}{A} + a_p \frac{(-1)^Z + (-1)^N}{2\sqrt{A}}$$

➤ Minimize χ^2 or maximize likelihood

$$\chi^2 = \sum_{Z, N \geq 8} \left[\frac{B_{\text{exp}}^{Z, N} - B_{\text{LDM}}^{Z, N}}{\delta_m} \right]^2$$

likelihood $\propto \exp(-\chi^2 / 2)$



实验数据取自 AME2012
CPC 36, 1603 (2012)

a_v	15.5868
a_{surf}	17.0871
a_c	0.7066
a_{sym}	23.1537
a_p	12.2047

Example: LDM

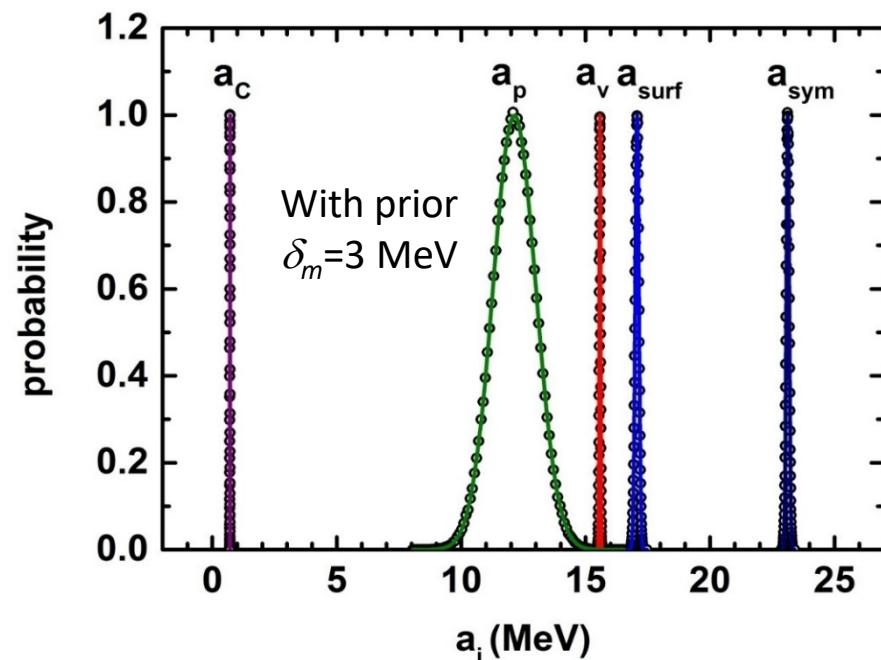
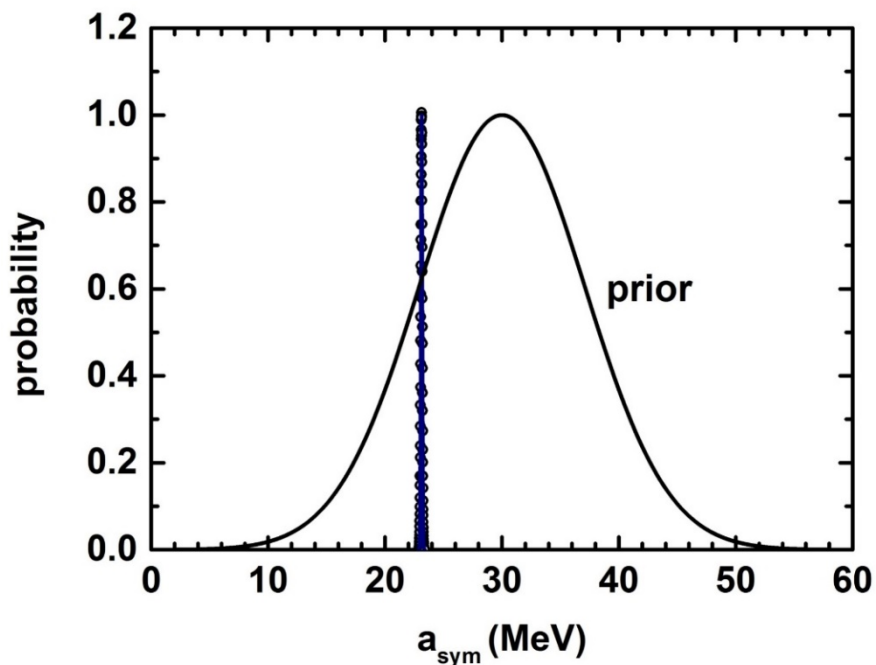
● Bayes' theorem: $p(\omega | D) = p(D | \omega) p(\omega) / p(D)$

$$B(Z, A) = a_v A - a_{surf} A^{2/3} - a_c \frac{Z^2}{A^{1/3}} - a_{sym} \frac{(N-Z)^2}{A} + a_p \frac{(-1)^Z + (-1)^N}{2\sqrt{A}}$$

$$p(\omega) \rightarrow p(a_i) \propto \exp\left[-(a_i - a_{i0})^2 / 2\sigma_{a_{i0}}^2\right] \quad a_{i0} = 16, 20, 0.7, 30, 10; \quad \sigma_{a_{i0}} = 5, 10, 0.5, 10, 5$$

$$p(D | \omega) \rightarrow p(m_{\text{exp}} | a_i) \propto \exp(-\chi^2 / 2), \quad \chi^2 = \sum_{Z, N \geq 8} \left[\frac{B_{\text{exp}}^{Z, N} - B_{\text{LDM}}^{Z, N}}{\delta_m} \right]^2$$

实验数据取自 AME2012
CPC 36, 1603 (2012)



Example: LDM

	With prior ($\delta_m=3$ MeV)		Without prior ($\delta_m=3$ MeV)		With prior ($\delta_m=10$ MeV)		least square method	
a_v	15.5786	± 0.0238	15.5785	± 0.0238	15.5794	± 0.0795	15.5868	± 0.0244
a_{surf}	17.0706	± 0.0742	17.0705	± 0.0742	17.0734	± 0.2474	17.0871	± 0.0758
a_c	0.7054	± 0.0017	0.7054	± 0.0017	0.7054	± 0.0055	0.7066	± 0.0017
a_{sym}	23.125	± 0.0595	23.1249	± 0.0595	23.1275	± 0.1985	23.1537	± 0.0607
a_p	12.1387	± 0.8565	12.2029	± 0.8692	11.6717	± 2.4851	12.2047	± 0.8853

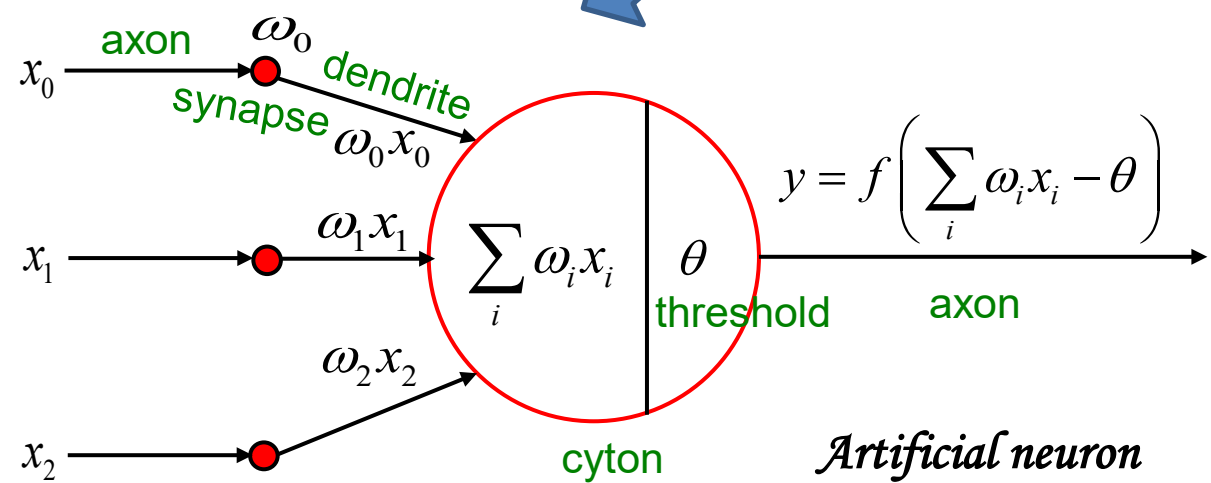
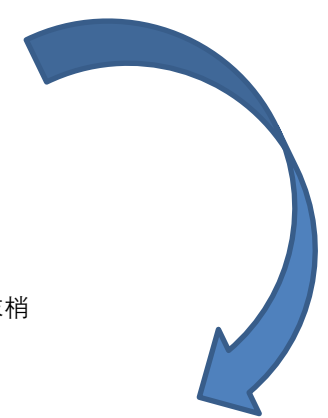
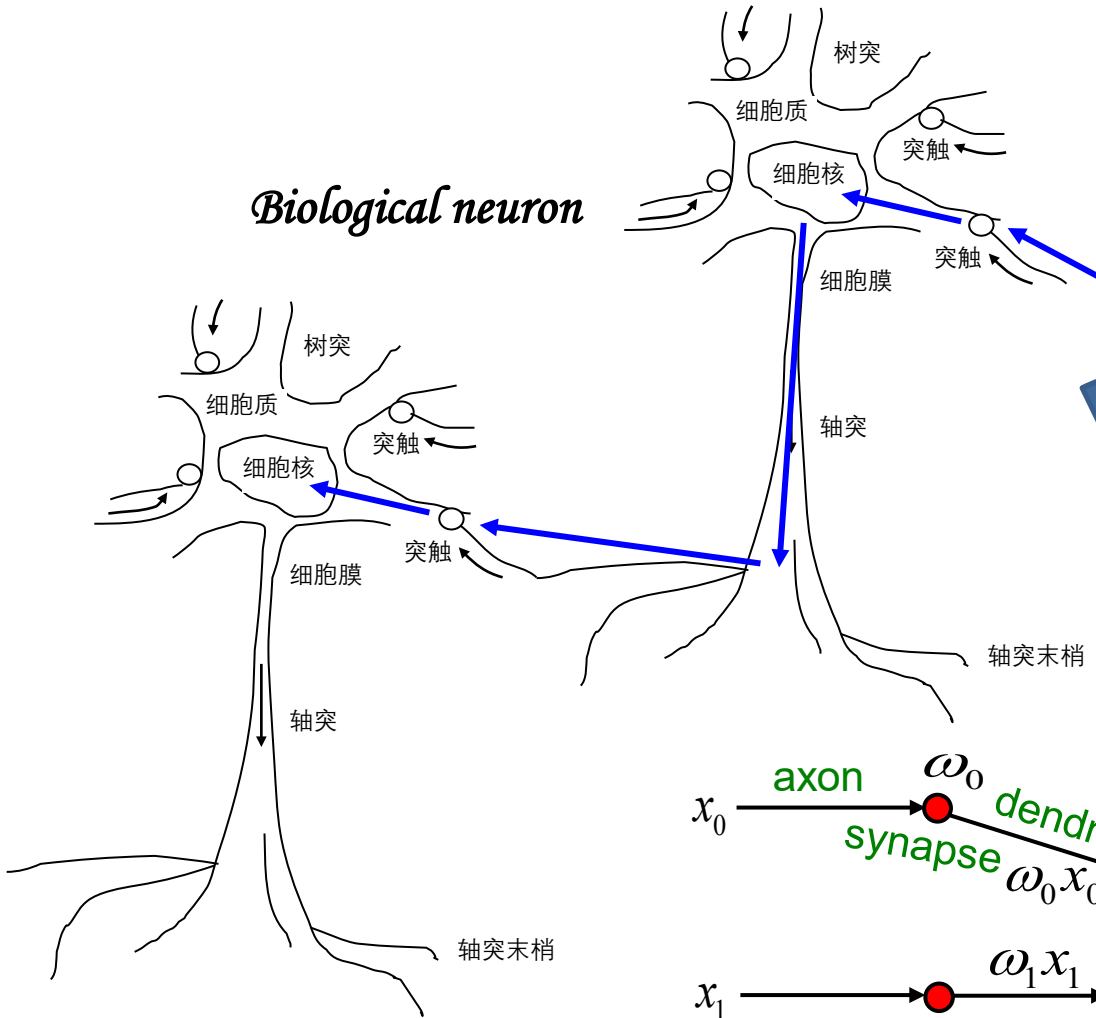
- ✓ When the number of the data is large enough, the influence of prior distribution is almost negligible and the deduced results are similar to those from least square method.
- ✓ If δ_m are very large, the values of a_i are still similar, while their uncertainties would increase.

Outline

- ① Introduction
- ② Bayesian neural network approach
 - ★ Bayesian approach
 - ★ Neural network (Tensorflow Playground, 1stOpt, PyTorch)
 - ★ Bayesian neural network
- ③ Results and discussion
 - ★ Nuclear masses
 - ★ Nuclear β -decay half-lives
 - ★ Some other results
- ④ Summary and perspectives

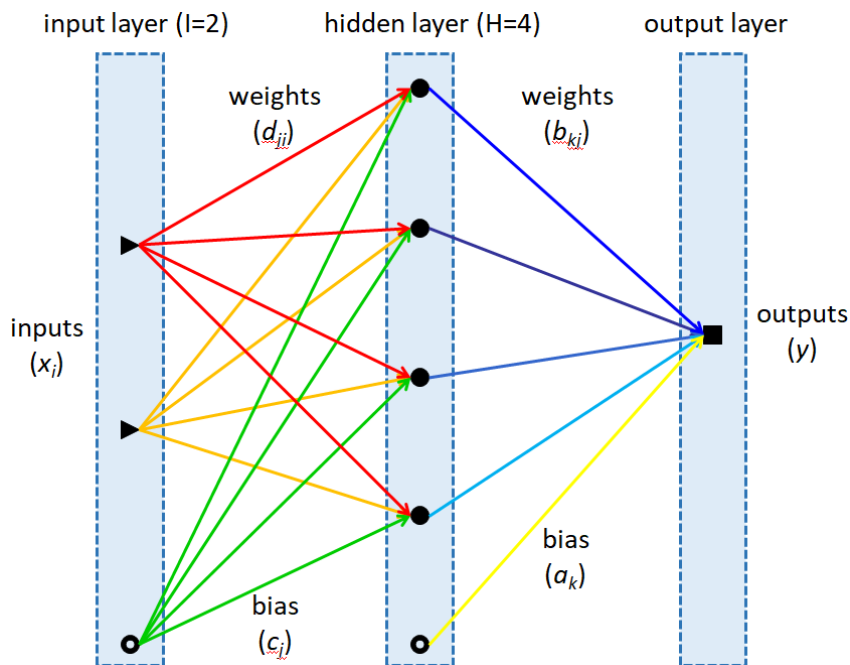
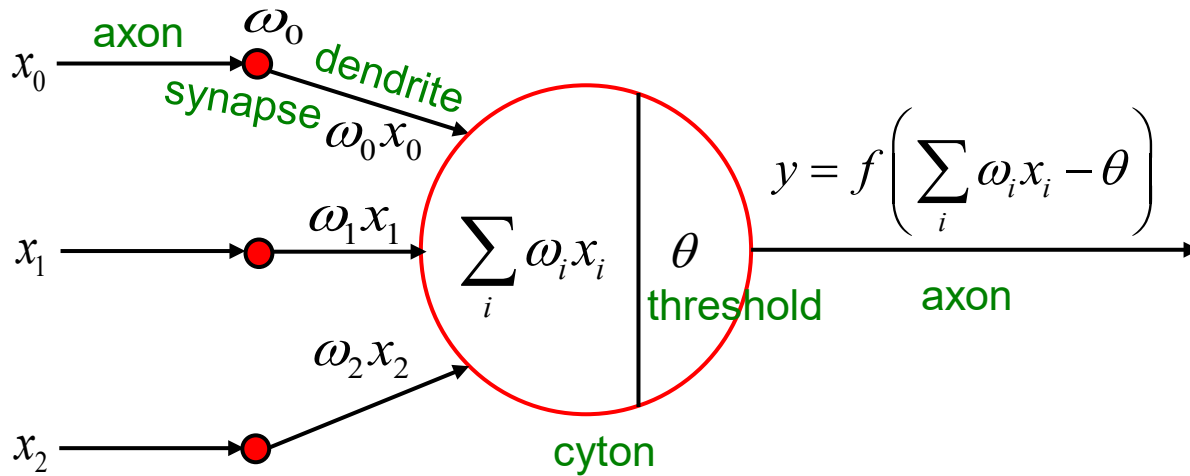
Neural network

Biological neuron



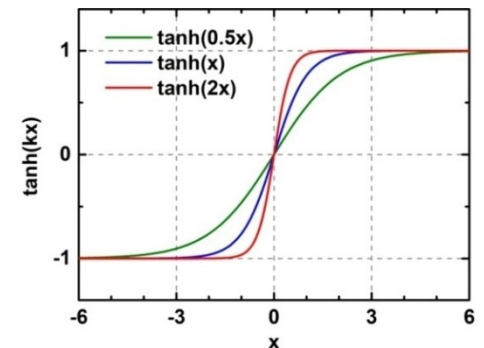
Artificial neuron

Neural network

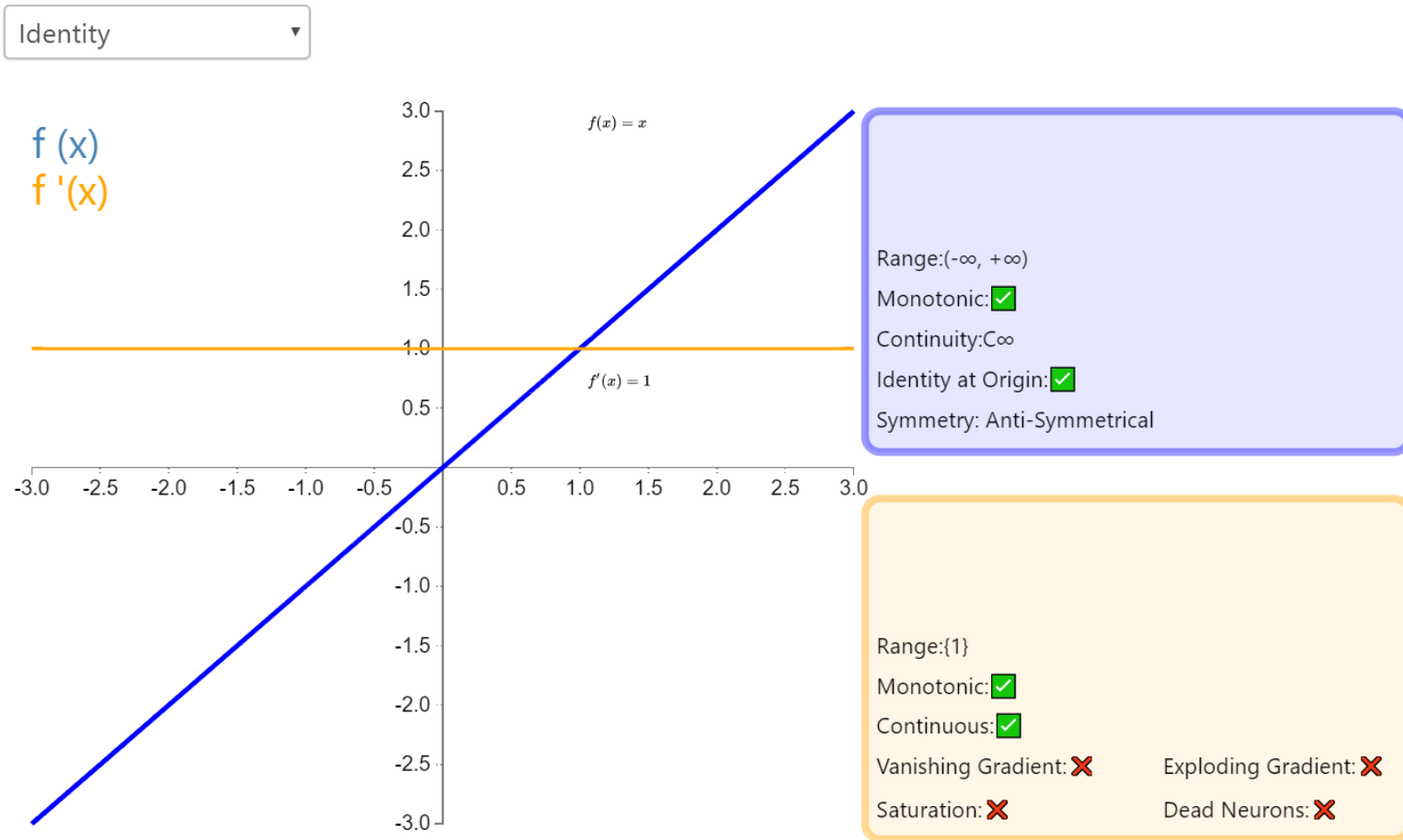


$$y(x, \omega) = a + \sum_{j=1}^H b_j \tanh\left(c_j + \sum_{i=1}^I d_{ji} x_i\right)$$

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

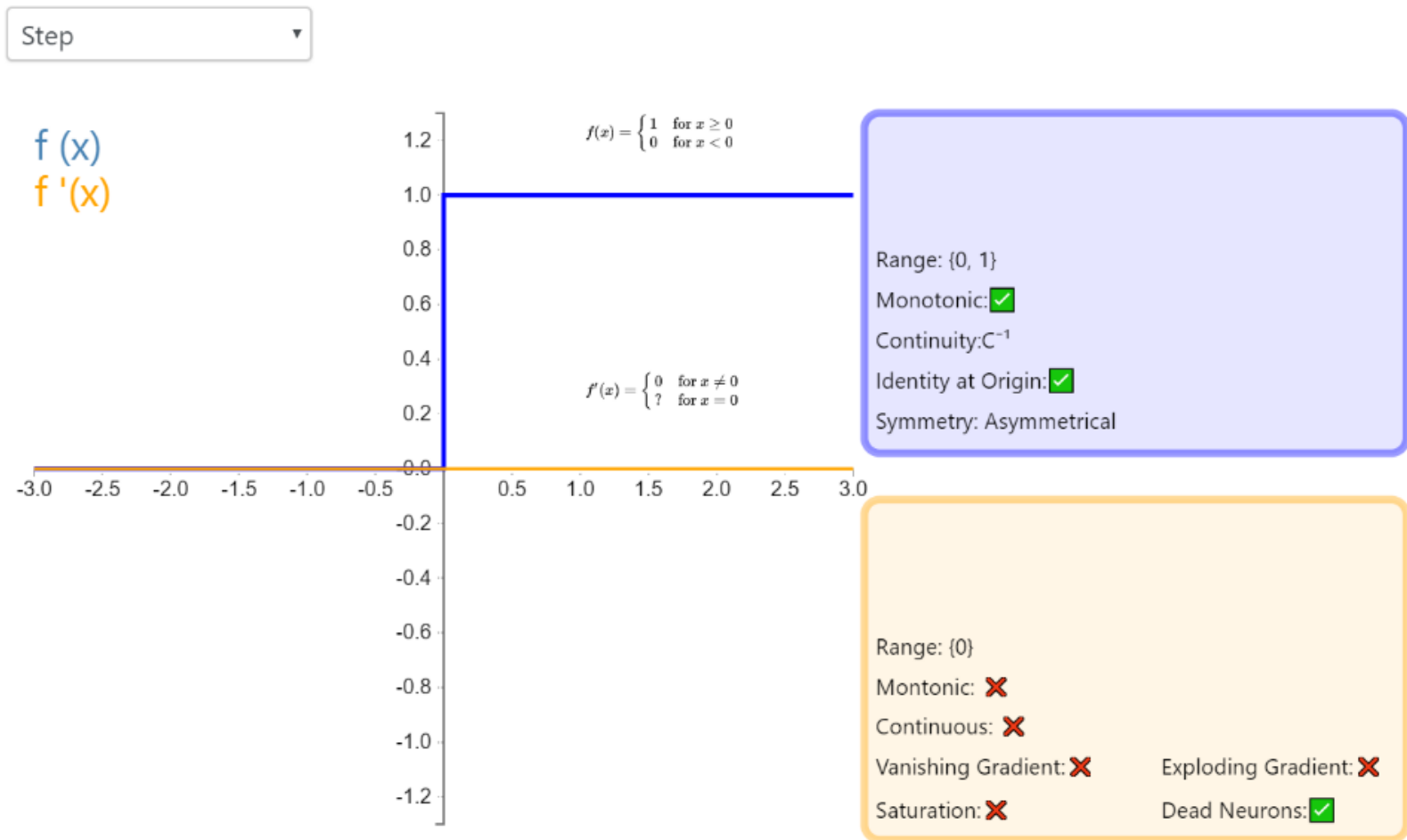


Activation function



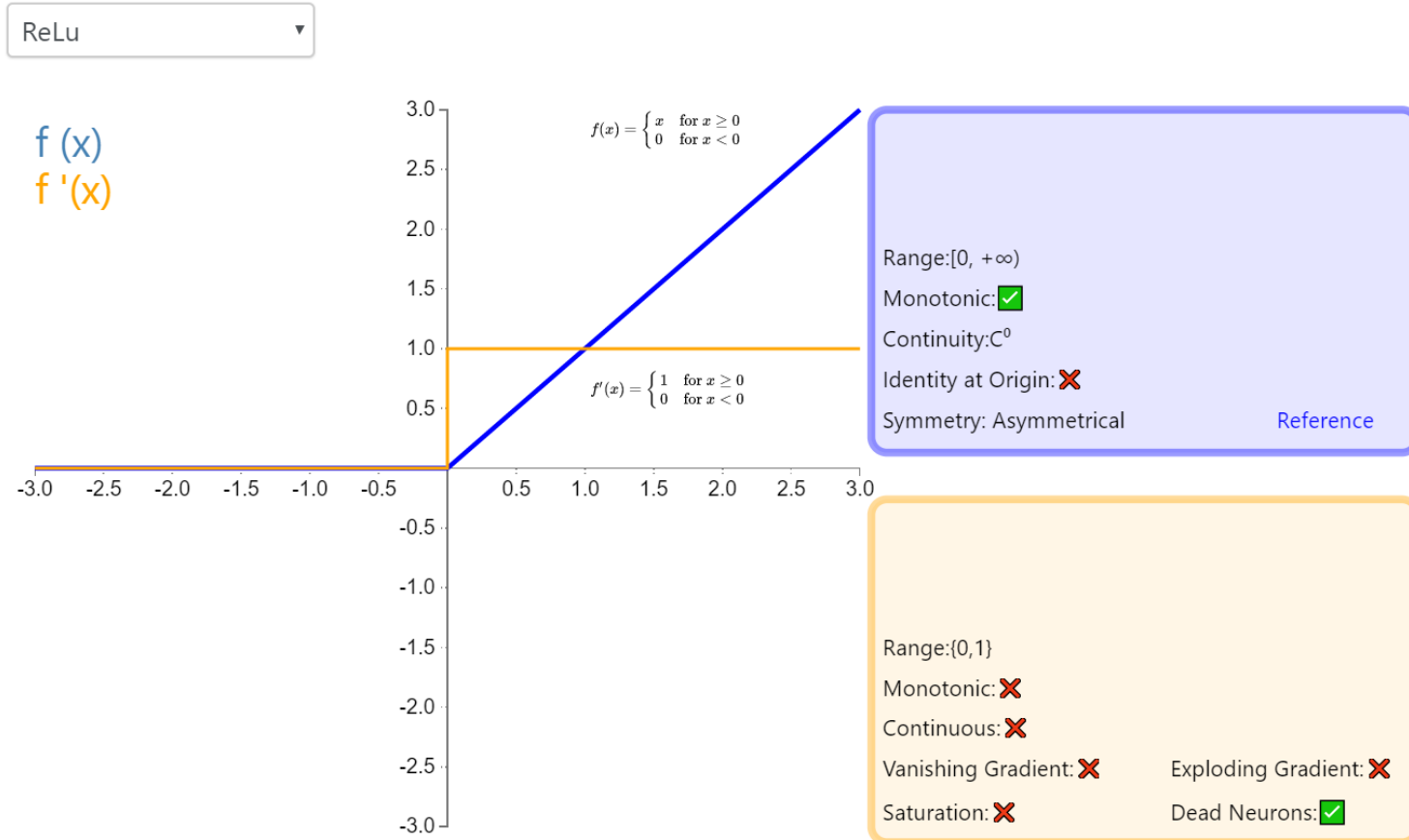
Identity: by this activation function, the input to the node equals the output. It is perfect for tasks where the underlying behavior is linear (similar to linear regression). When there is a nonlinearity, this activation function alone is not sufficient, but it can still be used as an activation function on the final output node for regression tasks.

Activation function



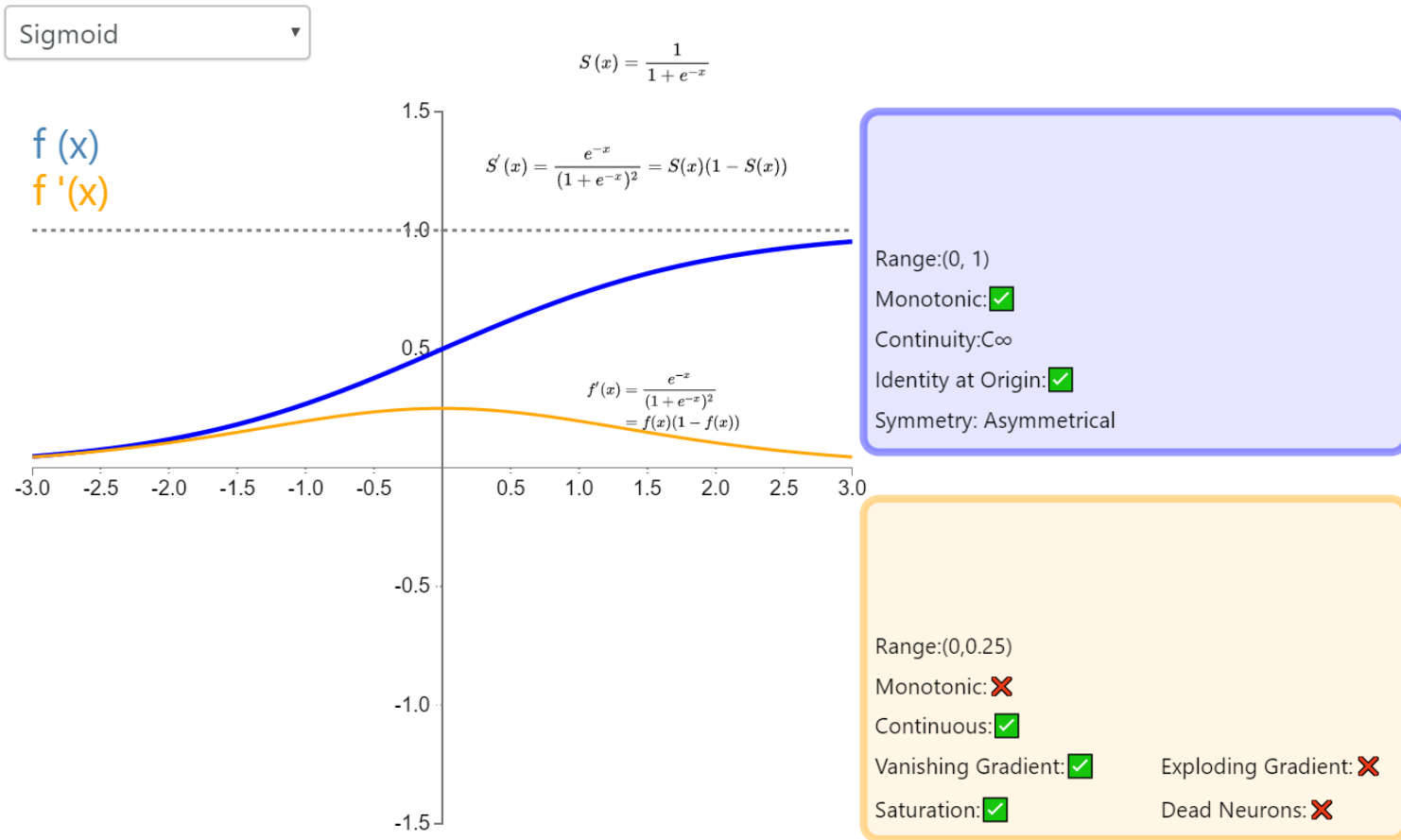
Step: this activation function is more theoretical than practical, **mimicking the all-or-nothing properties of biological neurons**. It cannot be applied to neural networks because its derivative is 0 (except that the zero point derivative is undefined), which means that gradient-based optimization methods are not feasible.

Activation function



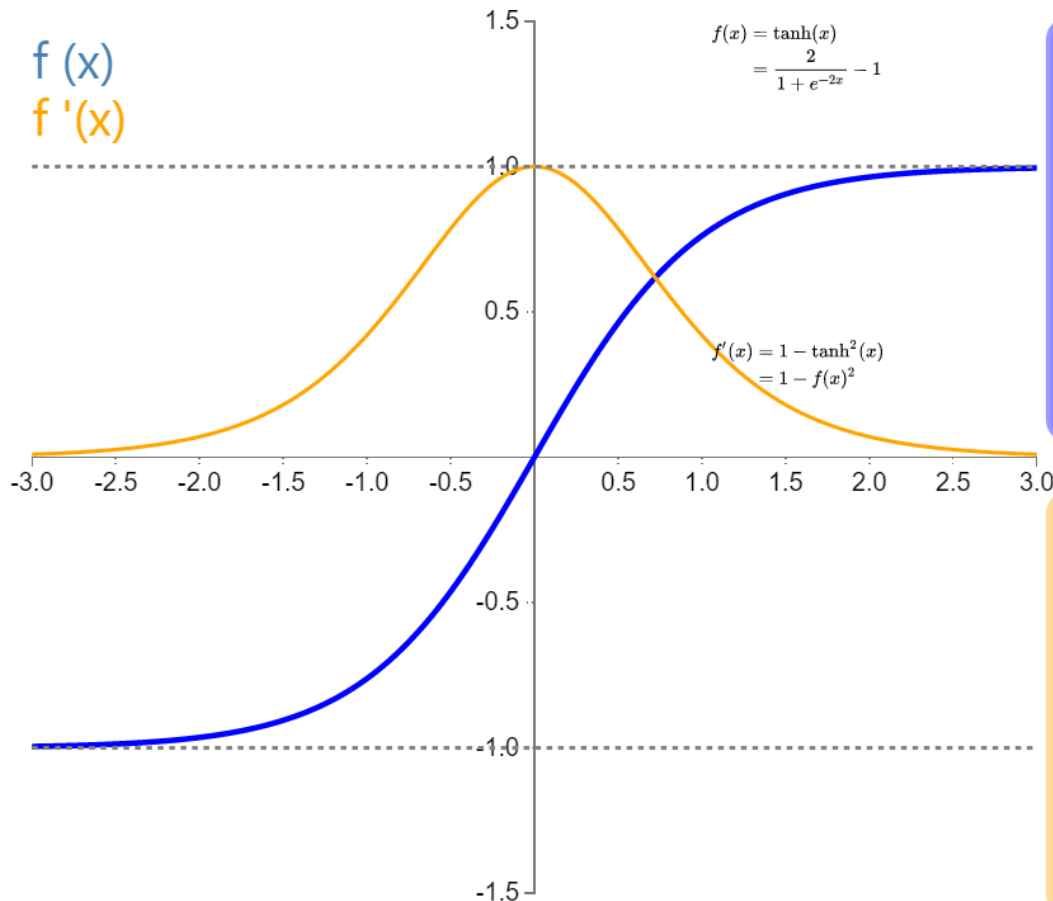
ReLU: rectified linear unit (ReLU) is the most commonly used activation function in neural networks. It retains the biological heuristic of the step function (the neuron activates only when the input exceeds the threshold), but the derivative is not zero when the input is positive, allowing for gradient-based learning. Using this function makes the computation faster, because neither the function nor its derivatives involve complex mathematical operations.

Activation function



Sigmoid: it is well known for its importance in logistic regression and ranges from 0 to 1. The Logistic Sigmoid (or Sigmoid as commonly called) activation function introduces the concept of probability to neural networks. Its derivative is non-zero and is easy to compute. However, in classification tasks, it is gradually being replaced by the Tanh function as the standard activation function, because the latter is an odd function (symmetric regarding the origin).

Activation function



Range: $(-1, 1)$
 Monotonic:
 Continuity: C^∞
 Identity at Origin:
 Symmetry: Anti-Symmetrical

Range: $(0, 1]$
 Monotonic:
 Continuous:
 Vanishing Gradient: Exploding Gradient:
 Saturation: Dead Neurons:

Tanh: in classification tasks, the hyperbolic tangent function (Tanh) has gradually replaced the Sigmoid function as the standard activation function, which has many characteristics favored by neural networks. It is **completely differentiable**, **antisymmetric**, and **has a center of symmetry at the origin**.

Selection of activation function for the output

➤ $y \in [-\infty, +\infty], [0, +\infty], [0, 1], [-1, 1]$

the output variable y do not need to be normalized, the output layer uses the activation function "identity", "ReLU", "sigmoid", and "tanh", respectively.

➤ $y \in [y_{\min}, y_{\max}]$:

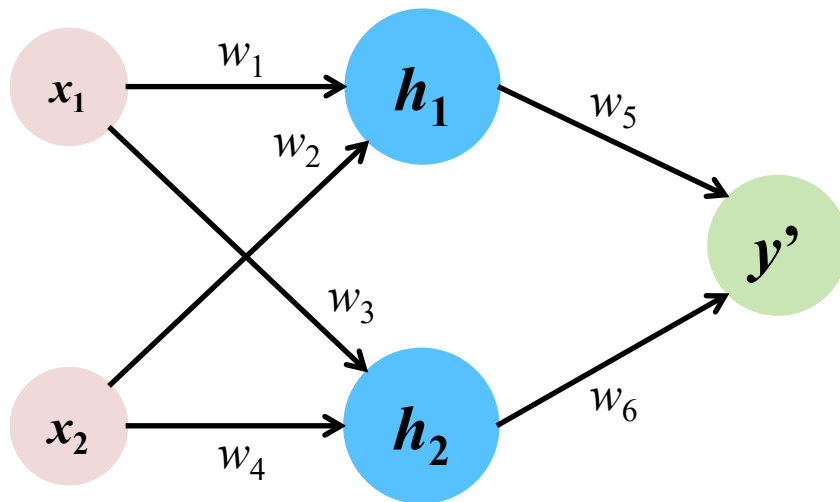
the output variable can be normalized with

$$u = \frac{y - y_{\min}}{y_{\max} - y_{\min}} \in [0, 1] \quad \text{or} \quad u = \frac{y - \frac{y_{\min} + y_{\max}}{2}}{\frac{y_{\max} - y_{\min}}{2}} \in [-1, 1],$$

the output layer uses the activation function "sigmoid" or "tanh".

BP (back propagation) neural network

Take a single hidden layer neural network as an example:



Suppose the characteristics of this sample have two dimensions, each with a value of

$$x_1 = 0.5$$

$$x_2 = 1.0$$

Suppose the true value of this sample is

$$y = 0.8$$

All parameters of the neural network are randomly initialized, assuming that

$$w_1 = 1.0; \quad w_2 = 0.5;$$

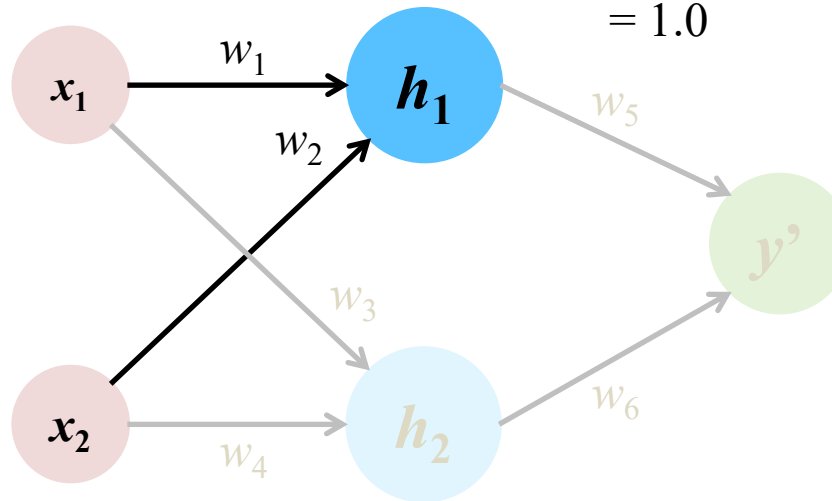
$$w_3 = 0.5; \quad w_4 = 0.7;$$

$$w_5 = 1.0; \quad w_6 = 2.0;$$

Each round has the forward propagation and the back propagation.

BP neural network

● 1st round: Forward propagation



$$\begin{aligned}
 h_1 &= w_1 \cdot x_1 + w_2 \cdot x_2 \\
 &= 1.0 \times 0.5 + 0.5 \times 1.0 \\
 &= 1.0
 \end{aligned}$$

Suppose the characteristics of this sample have two dimensions, each with a value of

$$x_1 = 0.5$$

$$x_2 = 1.0$$

Suppose the true value of this sample is

$$y = 0.8$$

All parameters of the neural network are randomly initialized, assuming that

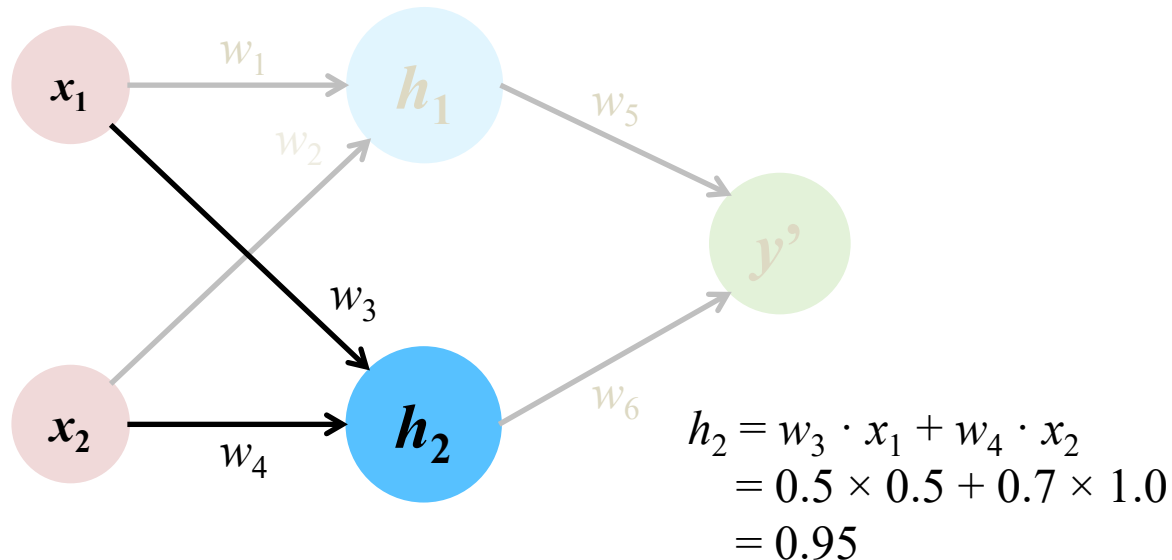
$$w_1 = 1.0; \quad w_2 = 0.5;$$

$$w_3 = 0.5; \quad w_4 = 0.7;$$

$$w_5 = 1.0; \quad w_6 = 2.0;$$

BP neural network

● 1st round: Forward propagation



Suppose the characteristics of this sample have two dimensions, each with a value of

$$x_1 = 0.5$$

$$x_2 = 1.0$$

Suppose the true value of this sample is

$$y = 0.8$$

All parameters of the neural network are randomly initialized, assuming that

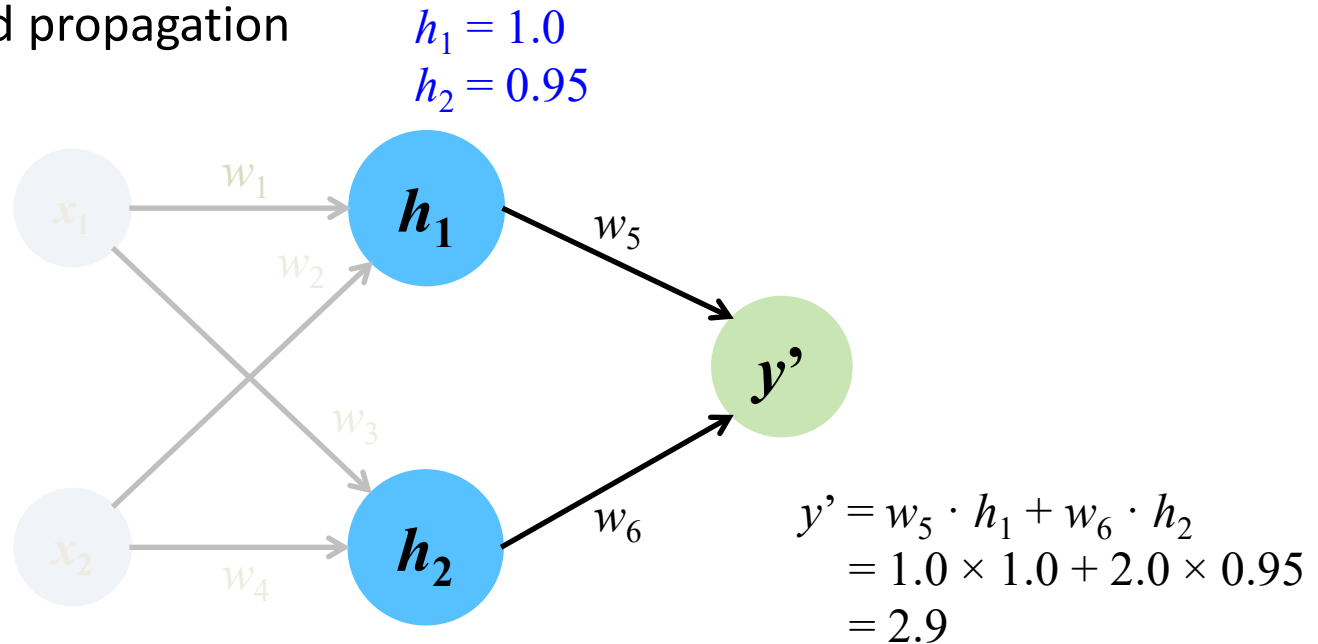
$$w_1 = 1.0; \quad w_2 = 0.5;$$

$$w_3 = 0.5; \quad w_4 = 0.7;$$

$$w_5 = 1.0; \quad w_6 = 2.0;$$

BP neural network

- 1st round: Forward propagation



Suppose the characteristics of this sample have two dimensions, each with a value of

$$x_1 = 0.5$$

$$x_2 = 1.0$$

Suppose the true value of this sample is

$$y = 0.8$$

All parameters of the neural network are randomly initialized, assuming that

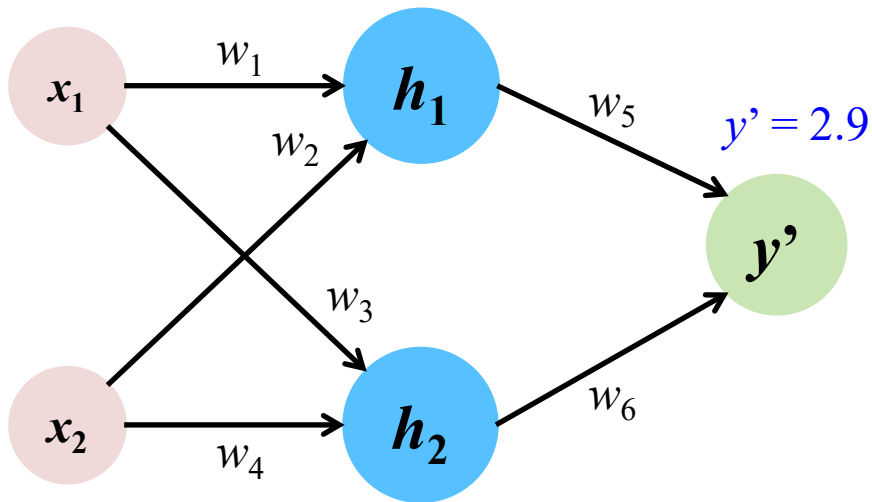
$$w_1 = 1.0; \quad w_2 = 0.5;$$

$$w_3 = 0.5; \quad w_4 = 0.7;$$

$$w_5 = 1.0; \quad w_6 = 2.0;$$

BP neural network

● 1st round: Forward propagation



After this round of forward propagation, the error is

$$\begin{aligned}\delta &= (y - y')^2 / 2 \\ &= 0.5 \times (0.8 - 2.9)^2 \\ &= 2.205\end{aligned}$$

(Calculation of the loss function, which generally varies from task to task)

Suppose the characteristics of this sample have two dimensions, each with a value of

$$x_1 = 0.5$$

$$x_2 = 1.0$$

Suppose the true value of this sample is

$$y = 0.8$$

All parameters of the neural network are randomly initialized, assuming that

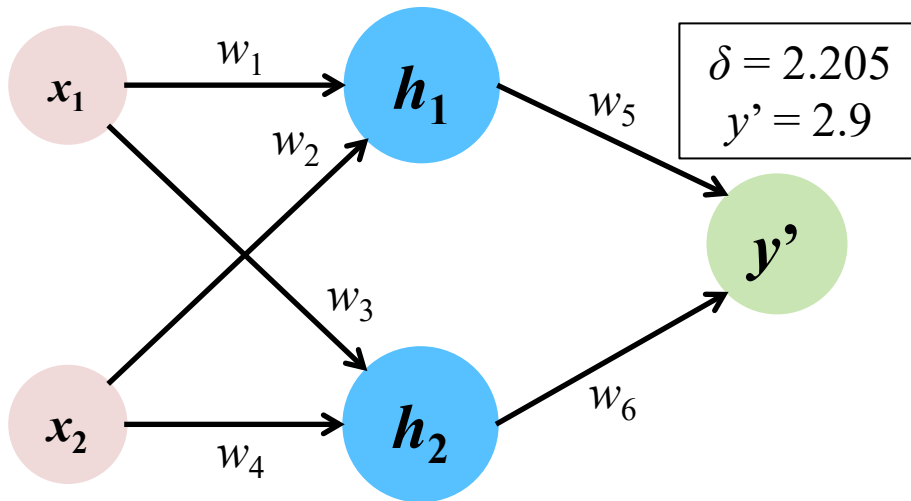
$$w_1 = 1.0; \quad w_2 = 0.5;$$

$$w_3 = 0.5; \quad w_4 = 0.7;$$

$$w_5 = 1.0; \quad w_6 = 2.0;$$

BP neural network

1st round: Back propagation



Take the calculation of w_5 as an example, and w_6 as an analogy

$$\frac{\partial \delta}{\partial w_5} = \frac{\frac{\partial \delta}{\partial y'}}{\frac{\partial y'}{\partial w_5}}$$

$$\begin{aligned} \delta &= \frac{1}{2}(y - y')^2 \\ \frac{\partial \delta}{\partial y'} &= \frac{1}{2} \times 2 \times (y - y') \times (-1) \\ &= y' - y \\ &= 2.9 - 0.8 = 2.1 \end{aligned}$$

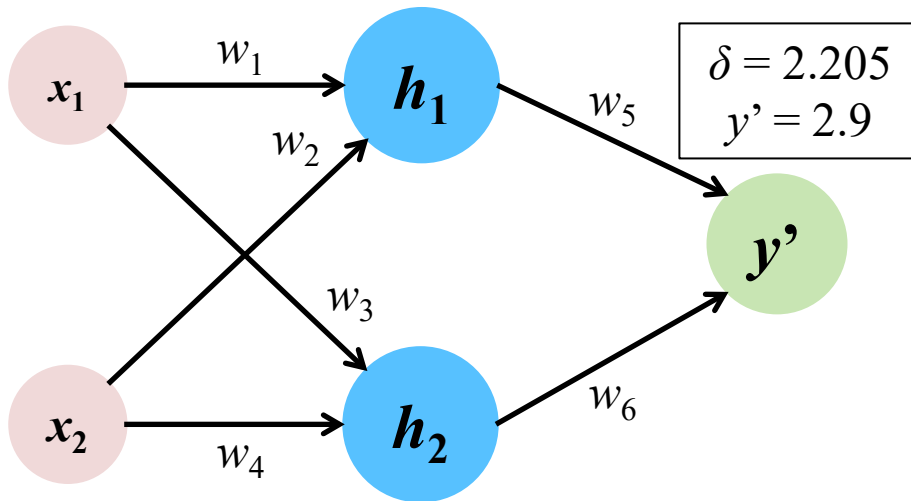
$$\begin{aligned} y' &= w_5 \cdot h_1 + w_6 \cdot h_2 \\ \frac{\partial y'}{\partial w_5} &= h_1 + 0 = 1.0 \end{aligned}$$



$$\frac{\partial \delta}{\partial w_5} = 2.1 \times 1.0 = 2.1$$

BP neural network

1st round: Back propagation



Then the value of w_5 can be updated

Learning rate, in this case 0.1

$$\begin{aligned}
 w_5^{(update)} &= w_5 - \eta \frac{\partial \delta}{\partial w_5} \\
 &= 1.0 - 0.1 \times 2.1 \\
 &= 0.79
 \end{aligned}$$

Similarly, it is possible to obtain

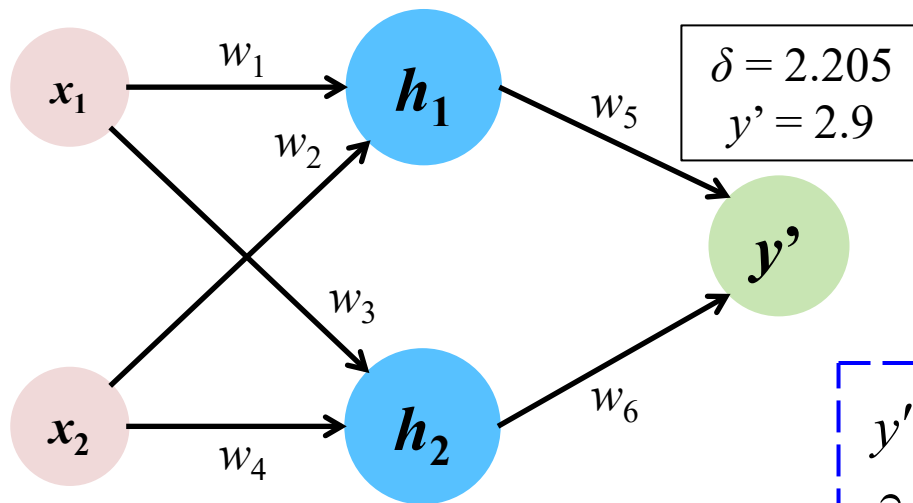
$$w_6^{(update)} = 1.8005$$

We have known that before

$$\frac{\partial \delta}{\partial w_5} = 2.1$$

BP neural network

1st round: Back propagation



$$\delta = \frac{1}{2}(y - y')^2$$

$$\frac{\partial \delta}{\partial y'} = 2 \times \frac{1}{2} \times (y - y')(-1)$$

$$= y' - y$$

$$= 2.9 - 0.8 = 2.1$$

$$y' = w_5 \cdot h_1 + w_6 \cdot h_2$$

$$\frac{\partial y'}{\partial h_1} = w_5 + 0 = 1.0$$

$$h_1 = w_1 \cdot x_1 + w_2 \cdot x_2$$

$$\frac{\partial h_1}{\partial w_1} = x_1 + 0 = 0.5$$

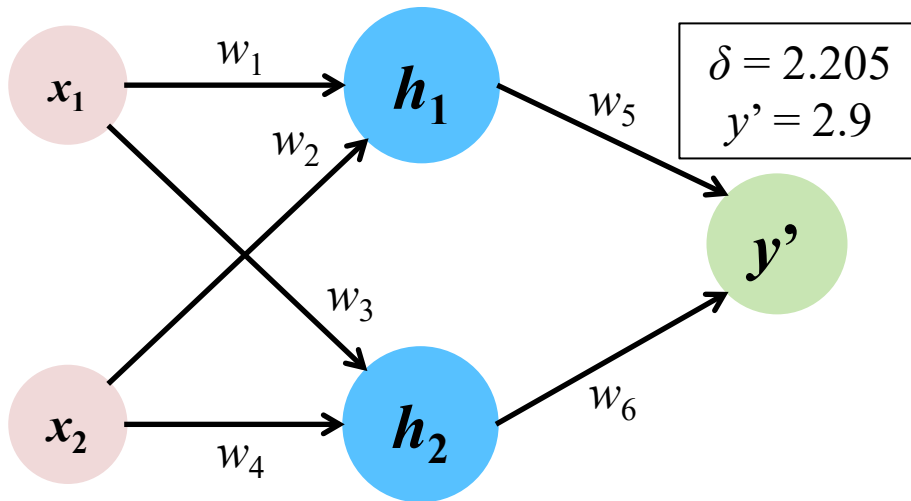
Take the calculation of w_1 as an example, and w_2, w_3, w_4 as an analogy

$$\frac{\partial \delta}{\partial w_1} = \frac{\partial \delta}{\partial y'} \cdot \frac{\partial y'}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1}$$

$$\frac{\partial \delta}{\partial w_1} = 2.1 \times 1.0 \times 0.5 = 1.05$$

BP neural network

● 1st round: Back propagation



We have known that before

$$\frac{\partial \delta}{\partial w_1} = 1.05$$

Then the value of w_1 can be updated

Learning rate, in this case 0.1

$$\begin{aligned} w_1^{(update)} &= w_1 - \eta \cdot \frac{\partial \delta}{\partial w_1} \\ &= 1.0 - 0.1 \times 1.05 \\ &= 0.895 \end{aligned}$$

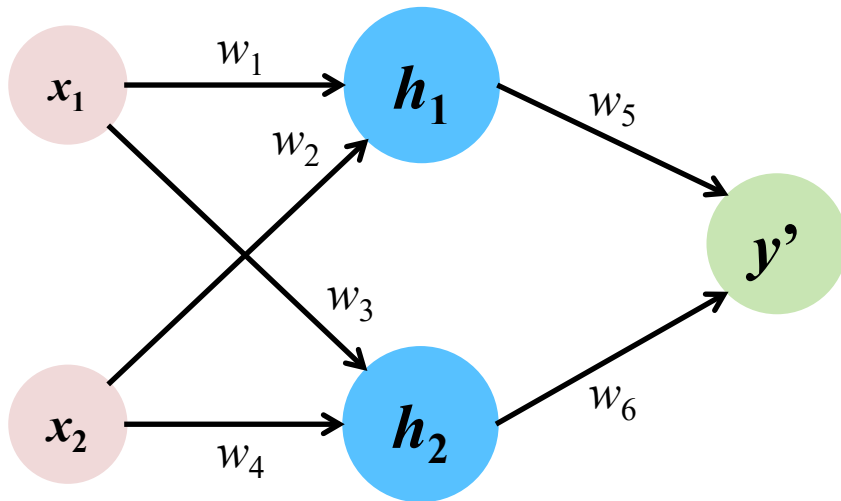
Similarly, it is possible to obtain

$$w_2^{(update)}, w_3^{(update)}, w_4^{(update)}$$

Homework: Please calculate these three values.

BP neural network

● 1st round: Back propagation



Suppose the characteristics of this sample have two dimensions, each with a value of

$$x_1 = 0.5$$

$$x_2 = 1.0$$

Suppose the true value of this sample is

$$y = 0.8$$

Parameters obtained by updating after one round of forward and back propagation

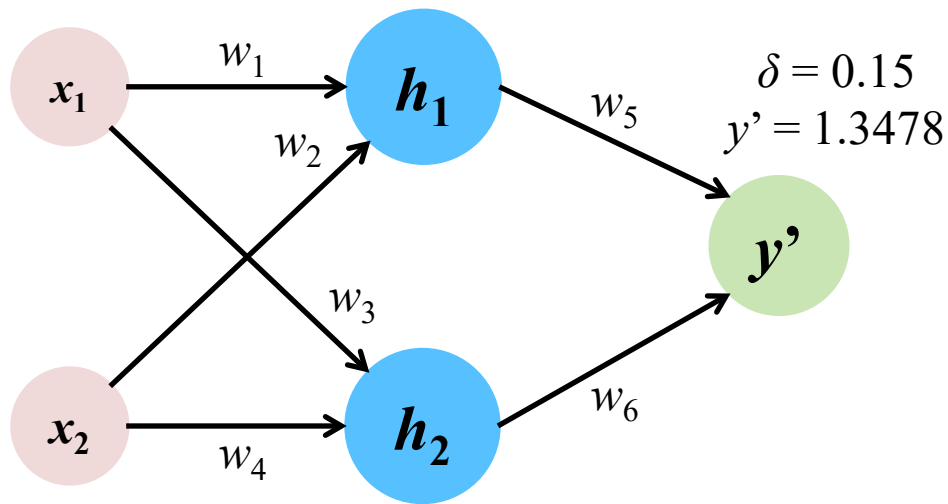
$$w_1 = 0.895; \quad w_2 = \quad ;$$

$$w_3 = \quad ; \quad w_4 = \quad ;$$

$$w_5 = 0.79; \quad w_6 = 1.8005;$$

BP neural network

● 2nd round: Forward propagation



Suppose the characteristics of this sample have two dimensions, each with a value of

$$x_1 = 0.5$$

$$x_2 = 1.0$$

Suppose the true value of this sample is

$$y = 0.8$$

With the updated parameters $w_1, w_2, w_3, w_4, w_5,$ and w_6 :

$$\begin{aligned} h_1 &= w_1 \cdot x_1 + w_2 \cdot x_2 \\ &= 0.895 \times 0.5 + \quad \times 1.0 \\ &= \end{aligned}$$

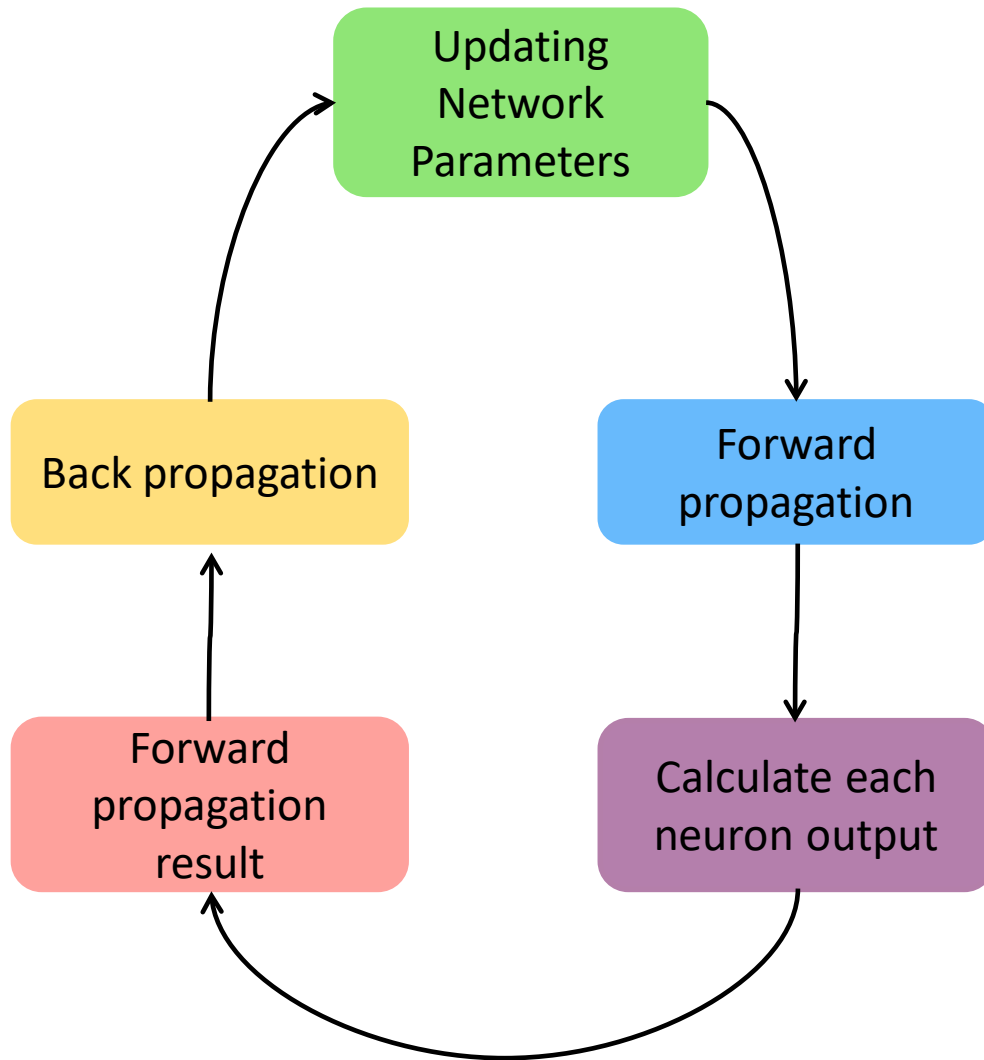
$$\begin{aligned} h_2 &= w_3 \cdot x_1 + w_4 \cdot x_2 \\ &= \quad \times 0.5 + \quad \times 1.0 \\ &= \end{aligned}$$

$$\begin{aligned} y' &= w_5 \cdot h_1 + w_6 \cdot h_2 \\ &= 0.79 \times \quad + 1.8005 \times \quad \\ &= 1.3478 \end{aligned}$$

$$\delta = (y - y')^2 / 2$$

$$\delta_1: 2.205 \rightarrow \delta_2: 0.15$$

BP neural network



Note:

★ In the actual use of neural networks, each neuron is usually given a final value after an **activation function** on top of the above calculations.

★ The derivation must be combined with the activation function in the form of a derivative, but there is no difference in the overall process.

Tensorflow Playground



Epoch
002,216

Learning rate
0.03

Activation
Tanh

Regularization
None

Regularization rate
0

Problem type
Classification

DATA

Which dataset do you want to use?



Ratio of training to test data: 50%



Noise: 0



Batch size: 10



REGENERATE

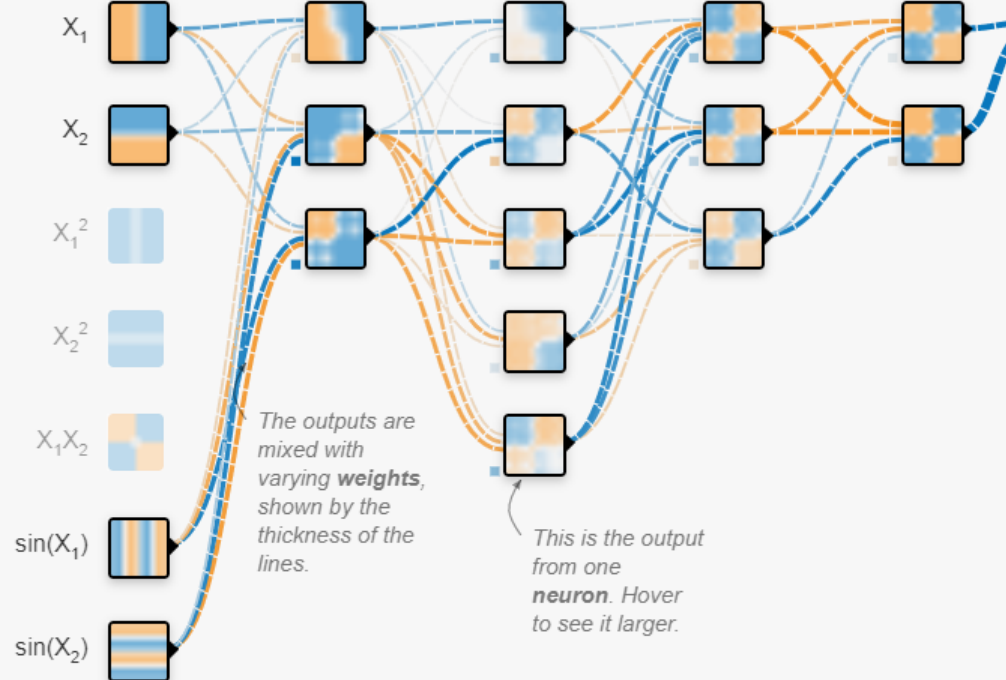
FEATURES

Which properties do you want to feed in?

- X_1
- X_2
- X_1^2
- X_2^2
- $X_1 X_2$
- $\sin(X_1)$
- $\sin(X_2)$

4 HIDDEN LAYERS

- 3 neurons
- 5 neurons
- 3 neurons
- 2 neurons

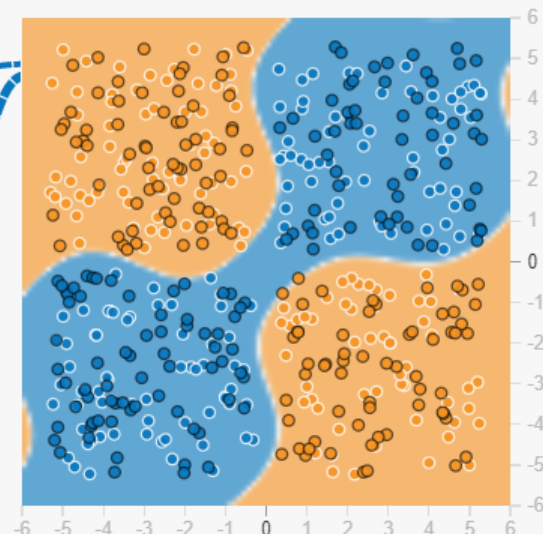


The outputs are mixed with varying weights, shown by the thickness of the lines.

This is the output from one neuron. Hover to see it larger.

OUTPUT

Test loss 0.007
Training loss 0.000



Colors shows data, neuron and weight values.

Show test data Discretize output

<http://playground.tensorflow.org>

1stOpt

七维高科
7D-Soft High Technology Inc.

1stOpt - First Optimization

首页 产品介绍 屏幕截图 案例展示 注册 服务合作 联系我们
English

1stOpt 10.0于2022年11月1日发布。新增功能参见: [链接...](#)

历史纪录:
1: 1stOpt 9.0于2020年10月18日发布。新增功能参见: [链接...](#)

2: 1stOpt 8.0于2018年8月25日发布。新增功能参见: [链接...](#)

3: 1stOpt 7.0于2016年4月20日发布。7.0版改进和新增功能: [链接...](#)

4: 1stOpt 6.0于2014年1月10日发布。享受免费的用户将会在正式发布一月内收到升级安装包。新增功能参见: [链接...](#)

5: 1stOpt 5.0于2012年2月3日发布。 [链接...](#)

6: 1stOpt 4.0于2010年8月6日发布。 [链接...](#)

7: 1stOpt 3.0于2009年5月8日发布。更强大、更稳定、更多功能、更易于使用。 [链接...](#)

8: 1stOpt 2.5于2007年11月1日发布。 [链接...](#)

9: 1stOpt 2.0于2006年10月7日发布。 [链接...](#)

1 领先世界、当今最强大、最易于使用的数值优化分析计算软件平台: 最强大的全局优化算法令优化拟合不再成为难题, 独特的ODE求解器轻松应对任意边值问题。

2 非线性拟合、非线性方程组、参数估算反演、方程求解、微分方程求解、微分方程拟合、非线性规划、混合整数规划... 令你忘却Matlab、Origin、Lingo、Gams等世界品牌使用时的繁琐、低效与不足。

3 遍布世界数十万各科技工作者的选择!

我们的用户

高校: 国内几乎所有“985”和“211”顶尖高校如清华大学、北京大学、浙江大学、上海交通大学、复旦大学、同济大学、北京航空航天大学、中国科学院、西安交大、南京航空航天大学、吉林大学、武汉大学、华中科技大学、华中师范大学、四川大学等, 国外如英国的牛津大学、美国的斯坦福大学、美国加州大学、加拿大卡加利大学等;

科研单位: 国内如中国科学院下属近百个研究所(院)、中国原子能科学研究院、中国物理研究院、中国建筑科学研究院、中国计量科学研究院、中国石化石油工程研究院、中国石化、上海微小卫星工程中心、清华大学清华-富士康纳米科技研究中心、上海石油化工研究院、国防科工委航天二院、国防科工委航天三院等, 国外如美国国家能源部橡树岭国家实验室(Oak Ridge National Laboratory)及再生能源研究所等;

国家级重点实验室: 国内上百家国家级重点实验室及国家级技术研发中心;

企业: 上海宝钢集团、大庆石油集团、中国船舶重工集团公司、日本三洋公司、霍尼韦尔公司(Honeywell)、艾默生公司、中石油、中国石化集团、中国电力工程顾问集团、3M中国有限公司、台达能源技术(上海)有限公司、上海克矿能源科技开发有限公司、山西中阳钢铁有限公司、香港纳米及先进材料研发有限公司、上海市工程设计研究总院(集团)有限公司、苏州纳微科技有限公司等。

论文发表

截至2020年9月31日为止, 已有逾万篇学术性论文采用了1stOpt分析计算软件工具, 发表刊物包括众多国内外SCI、EI、IEEE及核心期刊。

基金项目

包括国家自然科学基金、国家863计划项目、国家973项目、国家重大科技专项和国家“十一五”、“十二五”、“十三五”、“十四五”科技支撑计划等数千项各类科研项目采用了1stOpt软件平台。

用户评价

- 1) 1stOpt是我至今舍不得花钱购买的唯一一款软件, 很高兴也难以令人置信还是国产的!
- 2) 1stOpt相比于Matlab就像Matlab相比于C/C++, 简单、好用、强大, 太适合我们这些非数学和计算机专业的人了。
- 3) 我试用了贵公司的产品, 感觉真很棒, 如何夸赞都不过分。我用过Lingo、Matlab、OriginPro、SPSS、DataFit等软件, 完全知道1stopt的技术含量和价值。真的希望贵公司能够好好开发这个产品, 是中国人的骄傲。
- 4) 我为中国有如此优秀的数值计算软件开发公司感到深深的高兴!
- 5) 最大的两个感觉是: 界面简单, 计算快速准确, 界面简单方面, 和Lingo相似; 而计算速度和精度根据网上相关数据远远优于Lingo, 在参数设置合理的情况下更是优于Matlab。
- 6) Lingo够强, 当初做数学建模时一直在用Lingo和Matlab, Lingo做优化确实好, 不过已经有了更强的, 而且是国产软件, 叫1stopt!
- 7) 简洁的界面, 优秀的算法, 合乎常情的语言结构都给我很深的印象。整个软件系统很容易上手, 当知道是国产软件后, 更是对你们钦佩不已, 并感到特别的自豪。试用了几个例子, 优化效果要好于matlab的优化工具箱, 无论从使用方便还是从计算结果上来说, 你们的软件都是最棒的。你们的全局优化算法和不用赋初值的方法为实际问题带来很多方便, 有了你们的软件我甚至以后不用考虑算法的问题了。

联系电话: 13269987748; QQ: 454715413;
微信: software_7d_1stopt; E-Mail: info@7d-soft.com

京ICP备18022811号-1

版权所有: 北京七维高科科技有限公司, 1997-2022, 京ICP备18022811号-1, (最近更新: 2022年12月12日)

7D Software
7D-Soft High Technology Inc.

1stOpt - First Optimization

Main Introduction Screenshot Cooperation Contact Order About Us
Chinese

State of the Art Optimization Software for Advance Numerical Computation

The world No. 1 software package for nonlinear regression, curve fit, system equation solving, global optimization for various problems!

Have you experienced with struggling to guess and guess the initial start values of the parameters on your optimization/regression models? Do you want to an optimization analysis tool with the features of easy-to-use, lightweight and powerful? 1stOpt is answer!

1stOpt (the former name is Auto2Fit) is a robust, flexible and state-of-the-art optimization package. The unique optimization algorithm of UGO (Universal Global Optimization) makes the 1stOpt the best and the most efficient and effective tool in the area of nonlinear regression worldwide today. 1stOpt is also easier, faster and powerful for solving linear, nonlinear and integer, mixed integer optimization modes, both constrained and unconstrained.

1stOpt can be used in the fields below:

- Nonlinear regression: Linear, nonlinear, ODE regression, regression with complex function and data, multi-nonlinear with no limitations on variable and parameter number, no need to guess initial start values
- Global optimization: find the maximal/minimal value of any function or model. The function or model may be nonlinear, non continuous, non derivable, with constrained conditions.
- Equation Solve: solve any type of equation or system equations
- ODE Equation Solve: both initial value problem solve any type of equation or system equations
- Model Auto-calibration: parameter estimation of any complicated engineer model
- Plot 2-D, 3-D cartesian and 2-D parametric function chart, especial for implicit functions
- and much more...

Tel: 13269987748; QQ: 454715413;
WideChart: software_7d_1stopt; E-Mail: info@7d-soft.com

Copyright: 7D-Soft High Technology Inc., 1997-2020 (Latest Updated: Oct. 18, 2020)

English homepage:
<http://www.7d-soft.com/en/>

Chinese homepage: <http://www.7d-soft.com/>

Niu Zhongming

Machine learning in nuclear physics

Monday, July 29, 2024

43/134

1stOpt

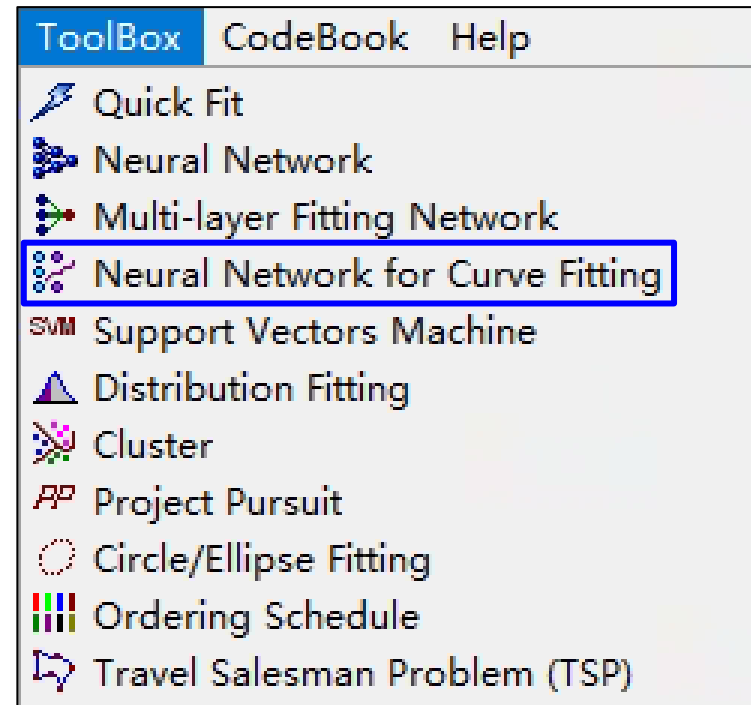
Introduction to 1stOpt (first optimization) software

● Advantage

- ✓ Optimization software for advance numerical computation
- ✓ **Simple interface and easy to use**
- ✓ **Core algorithm: universal global optimization algorithm**
- ✓ Direct support for Basic, Pascal, and Python
- ✓ Can be used in conjunction with C++ and Fortran

● Scope of application

- ✓ **Parameter estimation**
- ✓ linear, nonlinear fitting, regression
- ✓ Solution of nonlinear equations
- ✓ Solution of ordinary differential equations
- ✓ Fitting solution of ordinary differential equation
- ✓ Solution of complex equation, complex nonlinear fitting
- ✓ Extremum solution of arbitrary dimension function
- ✓ Implicit function root solution
- ✓ Linear, nonlinear and integer programming
- ✓ Combinatorial optimization problem



1stOpt: network settings



NNFit - [Untitled]
Neural Network Structure
 Input Layer
 No. of Independent: 2
 Data Normalization
 Importance Analysis
 Hide Layers:

Layers	Node	Trans. Fu...
No.1	3	Sigmoid
No.2	5	Tanh

 Add + Delete - Edit
 Output Layer
 No. of dependent: 1
 Trans. Fun.: Linear
 Data Normalization
Options
Opened File List

Neural Network Design
 Data for Curve Fitting
 Connections
 1: N1L1-N1L2
 2: N1L1-N2L2
 3: N1L1-N3L2
 4: N2L1-N1L2
 5: N2L1-N2L2
 6: N2L1-N3L2
 7: N1L2-N1L3
 8: N1L2-N2L3
 9: N1L2-N3L3
 10: N1L2-N4L3
 11: N1L2-N5L3
 12: N2L2-N1L3
 13: N2L2-N2L3
 14: N2L2-N3L3
 15: N2L2-N4L3
 16: N2L2-N5L3
 17: N3L2-N1L3
 18: N3L2-N2L3
 19: N3L2-N3L3
 20: N3L2-N4L3
 21: N3L2-N5L3
 22: N1L3-N1L4
 23: N2L3-N1L4
 24: N3L3-N1L4
 25: N4L3-N1L4
 26: N5L3-N1L4
 27: B1-N1L2
 28: B1-N2L2
 29: B1-N3L2
 30: B2-N1L3
 31: B2-N2L3
 32: B2-N3L3
 33: B2-N4L3
 34: B2-N5L3
 35: B3-N1L4

Bias →
 Inputs →
 Outputs →

main page of 1stOpt neural network fitting toolbox

Code Command: NNFit(NS=[2-3-5-1], TF=[1,2,4], DN=[1,1], NC=1, CT=1, Code=1, DC=[]);

1stOpt: data settings

Neural Network Structure

Input Layer

No. of Independent: 2

Data Normalization

Importance Analysis

Hide Layers:

Layers	Node	Trans. Fu...
No.1	15	Tanh

Add + Delete - Edit

Output Layer

No. of dependent: 1

Trans. Fun.: Linear

Data Normalization

Options

Opened File List

Neural Network Design

Data for Curve Fitting

	A	B	C	D	E	F	G
	Independent x1	Independent x2	Dependent y	Independent x1	Independent x2	Dependent y	Independent x1
1	-3.000000	0.279415	-0.646512				
2	-2.900000	0.464602	-0.678828				
3	-2.800000	0.631267	-0.579860				
4	-2.700000	0.772764	-0.452795				
5	-2.600000	0.883455	-0.593300				
6	-2.500000	0.958924	-0.267339				
7	-2.400000	0.996165	-0.138229				
8	-2.300000	0.993691	-0.060193				
9	-2.200000	0.951602	0.184503				
10	-2.100000	0.871576	-0.174407				
11	-2.000000	0.756802	0.003048				
12	-1.900000	0.611858	0.005739				
13	-1.800000	0.442520	-0.010562				
14	<						>

Cell[0,0] - [A1] = -3.000000

Training Data / Verification Data

- ✓ The data for the neural network can be directly input in the fitting data table, which has the **"Training data" table** and the **"Verification data" table**.
- ✓ The verification data will only be used for the verification of the training results and will not have any impact on the training results.

1stOpt: output of code

1. Connection Types: Multilayer Feed Forward
2. Code Type: Quick Model
3. Code Format: **Full**
4. Data Types: **CodeBook sheet**
5. New CodeBlock: Check

The screenshot shows the NNFit software interface. On the left, the 'Neural Network Structure' panel has 'Options' set to 'Multilayer Feed Forward'. The 'Output of Code' section is highlighted with a blue box, showing 'Code Type: Quick Model' and 'Code Format: Full'. The 'Data Types' section is also highlighted, showing 'CodeBook Sheet' selected. The main window displays a 'Neural Network Design' diagram with 27 nodes and connections. The 'Connections' list on the left shows a fully connected multilayer structure. The 'Code Command' at the bottom is: `NNFit(NS=[2-15-1], TF=[2,4], DN=[1,0], NC=1, CT=1, Code=3, DC=[], Importance);`

The screenshot shows the 1stOpt software interface. The 'CodeBook Editor' window displays the following code:

```

1 NewCodeBlock "NNFit in Quick Model (2-15-1)";
2 //NNFit(NS=[2-15-1], TF=[2,4], DN=[1,0], NC=1, CT=1, Code=3, DC=[], Importance);
3 Algorithm = UGO1[10];
4 Constant
5   RMin = -1.0, RMax = 1.0, //Scaling 1
6   Minx1 = -10.0, Maxx1 = 10.0, //Min. and
7   Minx2 = -0.99999, Maxx2 = 0.99999; //Min. and
8 ConstStr
9   TFH(v) = (exp(v) - exp(-v)) / (exp(v) + exp(-v)), //Tanh (Tra
10  TFO(v) = v, //Linear (S
11  xx1 = RMin + (RMax - RMin) * (x1 - Minx1) / (Maxx1 - Minx1), //Scaling 1
12  xx2 = RMin + (RMax - RMin) * (x2 - Minx2) / (Maxx2 - Minx2), //Scaling 1
13  s11 = p1*xx1 + p16*xx2 + p46, //Input to
14  h11 = TFH(s11), //Node Outp
15  s12 = p2*xx1 + p17*xx2 + p47, //Input to
16  h12 = TFH(s12), //Node Outp
17  s13 = p3*xx1 + p18*xx2 + p48, //Input to
18  h13 = TFH(s13) //Node Outp

```

The 'CodeBook Sheet' window shows a table with 18 rows and 6 columns (A-F). The first row contains values: -3.000000, 0.279415, -0.648512, -10.000000, -0.912945, -4.242986. The second row contains: -2.900000, 0.464602, -0.678828, -9.900000, -0.813674, -4.399954. The third row contains: -2.800000, 0.631267, -0.579860, -9.800000, -0.681964, -3.935915. The fourth row contains: -2.700000, 0.772764, -0.452795, -9.700000, -0.523066, -3.783998. The fifth row contains: -2.600000, 0.883455, -0.593300, -9.600000, -0.343315, -3.940952. The sixth row contains: -2.500000, 0.958924, -0.267339, -9.500000, -0.149877, -3.336756. The seventh row contains: -2.400000, 0.996165, -0.138229, -9.400000, 0.049536, -3.197399. The eighth row contains: -2.300000, 0.993691, -0.060193, -9.300000, 0.246974, -3.304040. The ninth row contains: -2.200000, 0.951602, 0.184503, -9.200000, 0.434566, -3.097259. The tenth row contains: -2.100000, 0.871576, -0.174407, -9.100000, 0.604833, -3.002656. The eleventh row contains: -2.000000, 0.756802, 0.003048, -9.000000, 0.750987, -2.961848. The twelfth row contains: -1.900000, 0.611858, 0.005739, -8.900000, 0.867202, -2.681241. The thirteenth row contains: -1.800000, 0.442520, -0.010562, -8.800000, 0.948844, -2.863241. The fourteenth row contains: -1.700000, 0.255541, -0.450648, -8.700000, 0.992659, -2.247033. The fifteenth row contains: -1.600000, 0.058374, -0.268406, -8.600000, 0.996900, -2.668829. The sixteenth row contains: -1.500000, -0.141120, -0.322984, -8.500000, 0.961397, -2.596515. The seventeenth row contains: -1.400000, -0.334988, -0.629047, -8.400000, 0.887567, -2.402863. The eighteenth row contains: -1.300000, -0.517777, -0.877777, -8.300000, 0.777777, -2.111111.

1stOpt: code modification

● Neural network code (full code format):

```

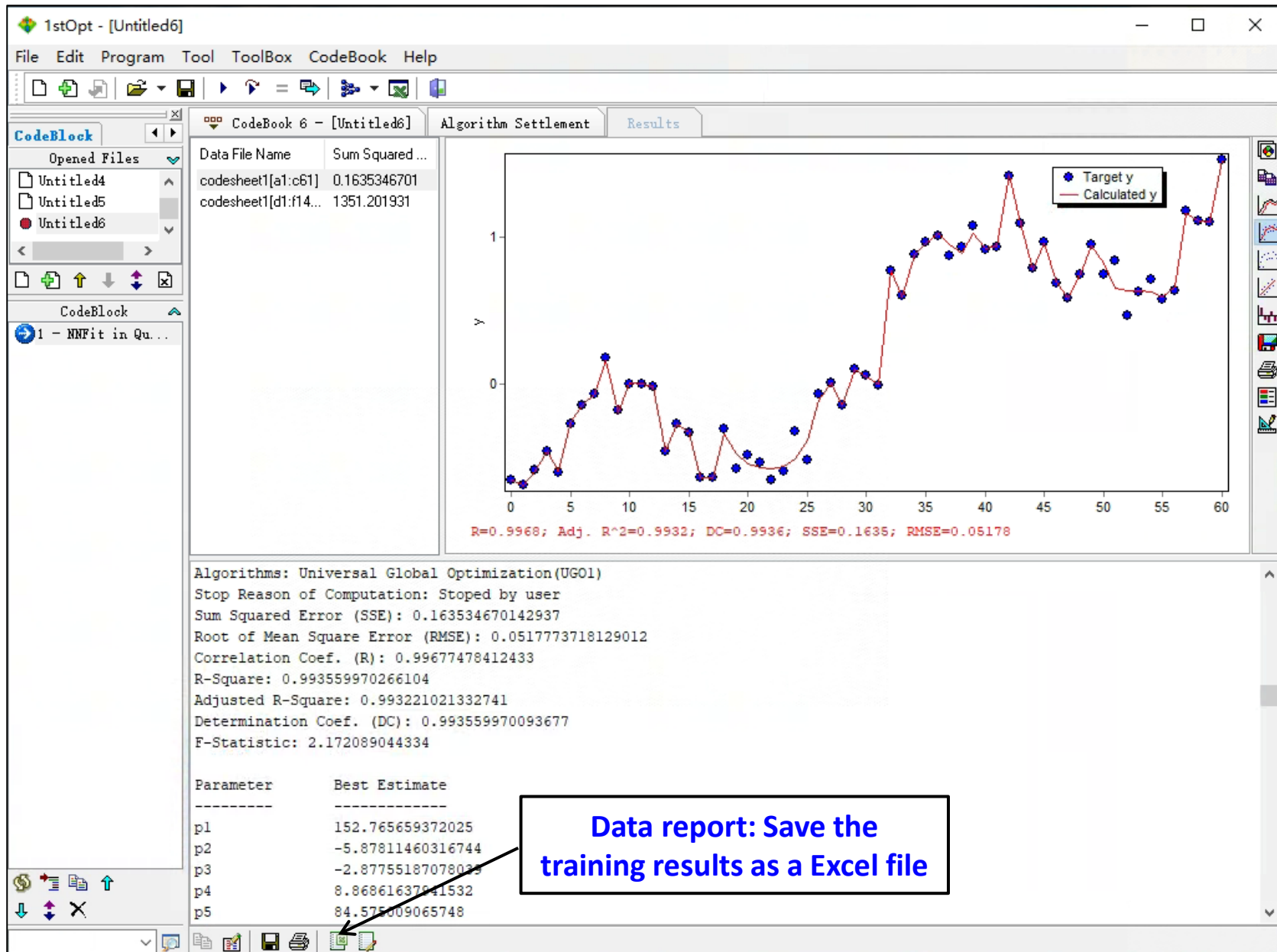
NewCodeBlock "NNFit in Quick Model (2-15-1)";
//NNFit(NS=[2-15-1], TF=[2,4], DN=[1,0], NC=1, CT=1, Code=3, DC=[], Importance);
Algorithm = UGO1[10];
Constant
  RMin = -1.0, RMax = 1.0,           //Scaling range of input layer
  Minx1 = -10.0, Maxx1 = 10.0,      //Min. and Max. values (Variable x1)
  Minx2 = -0.99999, Maxx2 = 0.99999; //Min. and Max. values (Variable x2)
ConstStr
  TFH(v) = (exp(v)-exp(-v))/(exp(v)+exp(-v)), //Tanh (Transfor Function for Hide layer)
  TFO(v) = v, //Linear (Transfor Function for Output layer)
  xx1 = RMin+(RMax-RMin)*(x1-Minx1)/(Maxx1-Minx1), //Scaling range of input layer to [-1,1]
  xx2 = RMin+(RMax-RMin)*(x2-Minx2)/(Maxx2-Minx2), //Scaling range of input layer to [-1,1]
  s11 = p1*xx1+p16*xx2+p46, //Input to Node-1 (Hide Layer-1)
  h11 = TFH(s11), //Node Output (Node-1, Hide Layer-1)
  s12 = p2*xx1+p17*xx2+p47, //Input to Node-2 (Hide Layer-1)
  h12 = TFH(s12), //Node Output (Node-2, Hide Layer-1)
  ....
  s115 = p15*xx1+p30*xx2+p60, //Input to Node-15 (Hide Layer-1)
  h115 = TFH(s115), //Node Output (Node-15, Hide Layer-1)
  s21 = p31*h11+p32*h12+p33*h13+p34*h14+..... +p45*h115+p61, //Input to Node-1 (Output Layer)
  h21 = TFO(s21); //Node Output (Node-1, Output Layer)
PassParameter
.....
Parameter p(61);
DataFileWeight = [1,0];
.....
Variable x1, x2, y;
Function y = h21;
DataFile "CodeSheet1[A1:C61]"; //Training Data
DataFile "CodeSheet1[D1:F140]"; //Verification Data

```

one can change it to any normalized method

one can change it to any activation function

1stOpt: training results



1stOpt: data report for the training results

	A	B	C	D	E	F	G	H
1	codesheet1[a1:c61]		codesheet1[d1:f140]					
2	Target y	Calculated y	Target y	Calculated y	Parameter Name	Parameter Value	Algorithms	Universal Global Optimization(UGO1)
3	-0.64651	-0.64477461	-4.242986	2.984419665	p1	152.7656594	Elapsed Time (Hr:Min:Sec:Msec)	02:01:49:75
4	-0.67883	-0.67716931	-4.399954	2.770477104	p2	-5.878114603	Sum Squared Error (SSE)	0.16353467
5	-0.57986	-0.58441313	-3.935915	2.536055835	p3	-2.877551871	Root of Mean Square Error (RMSE)	0.051777372
6	-0.4528	-0.4529083	-3.783998	-0.32520594	p4	8.868616379	Correlation Coef. (R)	0.996774784
7	-0.5933	-0.58378312	-3.940952	-6.604920107	p5	84.57500907	Adjusted R-Square	0.993221021
8	-0.26734	-0.24656846	-3.336756	-7.753939994	p6	-4.584584789	Determination Coef. (DC)	0.99355997
9	-0.13823	-0.14287653	-3.197399	-1.674842214	p7	4.310917534	Chi-Square	10466.28587
10	-0.06019	-0.0625298	-3.30404	-3.08779343	p8	2.996579121	F-Statistic	2.172089044
11	0.184503	0.167495455	-3.097259	-7.720180457	p9	-5.586598703	SSE (c61)	0.16353467
12	-0.17441	-0.19053886	-3.002656	-8.661918724	p10	5.435688369	SSE (f140)	1351.201931
13	0.003048	0.011867137	-2.961848	-7.635656476	p11	16.33884143	RMSE (c61)	0.051777372
14	0.005739	0.005700268	-2.681241	-7.705407124	p12	4.744747697	RMSE (f140)	3.106677061
15	-0.01056	-0.01114633	-2.863241	-7.786132872	p13	11.69239995	R (c61)	0.996774784
16	-0.45065	-0.45108374	-2.247033	-7.79881306	p14	6.584917608	R (f140)	0.6312612
17	-0.26841	-0.26981661	-2.668829	-7.789985399	p15	24.62032453	Adj. R^2 (c61)	0.993221021
18	-0.32298	-0.32213566	-2.596515	-7.780284173	p16	-106.4970391	Adj. R^2 (f140)	0.385222115
19	-0.62905	-0.62969197	-2.402863	-7.73283383	p17	53.5335669	DC (c61)	0.99355997
20	-0.62616	-0.62573792	-2.618968	-7.649956145	p18	5.318096334	DC (f140)	-0.262733643
21	-0.30127	-0.33373737	-2.680076	-8.420880198	p19	-11.50687776		
22	-0.57055	-0.47116191	-2.870259	-6.984191216	p20	76.38910586		
23	-0.47667	-0.53397232	-2.697166	-2.76905393	p21	1.085126221		
24	-0.52933	-0.55977565	-3.081283	-1.651647649	p22	1.792852818		
25	-0.64688	-0.56607193	-2.731012	-2.297389182	p23	-7.857255001		
26	-0.58464	-0.55352335	-2.606847	-7.464428414	p24	-2.113869152		
27	-0.31729	-0.50512875	-3.121554	2.540891909	p25	2.842323144		
28	-0.51072	-0.37730494	-2.853545	2.861816664	p26	19.2286442		
29	-0.06244	-0.10513991	-2.803326	2.807722881	p27	15.62255013		
30	0.011471	0.014199365	-3.386142	2.740337621	p28	-17.2340948		
31	-0.13905	-0.1378876	-3.351022	2.67419555	p29	18.82810288		
32	0.107544	0.107336437	-2.924784	2.622256409	p30	15.1881583		
33	0.062445	0.057703597	-3.075281	2.595121188	p31	0.172517529		
34	-0.00113	0.000945575	-2.793848	2.597536719	p32	-0.461951952		
35	0.771977	0.772984275	-2.686456	2.626877564	p33	-8.232200315		
36	0.598594	0.592158965	-2.607806	2.673802365	p34	9.493047869		

Example: Neural network in PyTorch

The screenshot shows the PyTorch website homepage. At the top, there is a navigation bar with the PyTorch logo and links for Learn, Ecosystem, Edge, Docs, Blog & News, About, and Become a Member. A search icon is also present. Below the navigation bar, a banner features the PyTorch logo and the text "GET STARTED". Underneath, it says "Choose Your Path: Install PyTorch Locally or Launch Instantly on Supported Cloud Platforms" and includes a "Get started" button. The main content area is divided into three columns: "2024 PYTORCH CONFERENCE" with a "Full details + guidelines" link, "PYTORCH 2.4" with a "Learn More" link, and "MEMBERSHIP AVAILABLE" with a "Join" link. At the bottom, there is a section for "KEY FEATURES & CAPABILITIES" with a "See all Features" button.

<https://pytorch.org/>

Example: Neural network in PyTorch

Import package

```
# Import package  
import os                                ## Import os  
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE" ## Prevents hanging  
import numpy as np                       ## Import numpy, using np for numpy  
import pandas as pd                     ## Import pandas, using pd for pandas  
import torch                             ## Import torch  
import torch.nn as nn                   ## Use nn for torch.nn  
import csv                               ## Import csv  
from torch import optim                 ## From Torch import optim  
import matplotlib.pyplot as plt        ## Import matplotlib  
import math                             ## Import math
```

Example: Neural network in PyTorch

Import the data: here is an example of two inputs and one output.

```

## Import relevant data
train = pd.read_csv(r"Test12Sin.csv", usecols=[0,1,2], header = None)
train = np.array(train)          ## Convert data to numpy arrays
train = torch.FloatTensor(train) ## Turning data into a tensor in pytorch
mydata = pd.read_csv(r"Call2Sin.csv", header = None)
mydata = np.array(mydata)
mydata = torch.FloatTensor(mydata)
nI = 2      ## Number of neural network inputs
nH1 = 5     ## Number of neurons in the first hidden layer of the neural network
nH2 = 10    ## Number of neurons in the second hidden layer of the neural network
nO = 1      ## Number of neural network output
x_train, y_train = torch.split(train,[nI,nO],dim = 1)
## The training set is divided into x_train and y_train, which are used as inputs
and output to train the neural network.
x_data, y_data = torch.split(mydata,[nI,nO],dim = 1)
result = x_data.clone()

```

	A	B	C	D	E
1	-3.0	0.279415	-0.64651		
2	-2.9	0.464602	-0.67883		
3	-2.8	0.631267	-0.57986		
4	-2.7	0.772764	-0.4528		
5	-2.6	0.883455	-0.5933		
6	-2.5	0.958924	-0.26734		
7	-2.4	0.996165	-0.13823		
8	-2.3	0.993691	-0.06019		
9	-2.2	0.951602	0.184503		
10	-2.1	0.871576	-0.17441		
11	-2	0.756802	0.003048		
12	-1.9	0.611858	0.005739		
13	-1.8	0.44252	-0.01056		
14	-1.7	0.255541	-0.45065		
15	-1.6	0.058374	-0.26841		
16	-1.5	-0.14112	-0.32298		
17	-1.4	-0.33499	-0.62905		
18	-1.3	-0.5155	-0.62616		
19	-1.2	-0.67546	-0.30127		
20	-1.1	-0.8085	-0.57055		
21	-1	-0.9093	-0.47667		
22	-0.9	-0.97385	-0.52933		
23	-0.8	-0.99957	-0.64688		
24	-0.7	-0.98545	-0.58464		
25	-0.6	-0.93204	-0.31729		
26	-0.5	-0.84147	-0.51072		
27	-0.4	-0.71736	-0.06244		
28	-0.3	-0.56464	0.011471		
29	-0.2	-0.38942	-0.13905		
30	-0.1	-0.19867	0.107544		
31	0	0	0.062445		
32	0.1	0.198669	-0.00113		
33	0.2	0.389418	0.771977		
34	0.3	0.564642	0.598594		
35	0.4	0.717356	0.884152		
36	0.5	0.841471	0.967547		
37	0.6	0.932039	1.010313		
38	0.7	0.98545	0.871936		

Example: Neural network in PyTorch

Construct neural network: a double hidden layer neural network

```
class net(nn.Module):  ## Define the network, net is the network name
    def __init__(self):
        super(net,self).__init__()
        self.hidden1 = nn.Linear(in_features = nI, out_features = nH1, bias = True)
        ## For example, in_features = 2, out_features = 5 means 2 inputs and 5 outputs.
        self.active1 = nn.Sigmoid()
        ## Define the activation function layer active1, nn.Sigmoid() as the Sigmoid activation function
        self.hidden2 = nn.Linear(in_features = nH1, out_features = nH2, bias = True)
        self.active2 = nn.Tanh()  ## nn.Tanh() as the Tanh activation function
        self.regression = nn.Linear(in_features = nH2, out_features = nO, bias = True)
    def forward(self, x):  ##Forward propagation path of the network
        x = self.hidden1(x)
        x = self.active1(x)
        x = self.hidden2(x)
        x = self.active2(x)
        output = self.regression(x)
        return output
net = net()
```

Example: Neural network in PyTorch

Train the neural network and calculate the loss value

```
## Train on net and compute the loss function
optimizer = optim.Adam(net.parameters(), lr = 0.001, eps = 1e-08, weight_decay = 0)
## Using Adam's Algorithm
## lr: learning rate
## eps: Terms added to the denominator to increase the stability of numerical calculations
## weight_decay: Weight decay (L2 penalty)

loss_func = nn.MSELoss()    ## mean square error
trainloss = torch.tensor([]) ## Setting up a storage list for a set of training set loss values
for epoch in range(10000):
    prediction = net(x_train)
    train_loss = loss_func(prediction, y_train)
    optimizer.zero_grad()   ## Gradient to 0
    train_loss.backward()   ## Backward propagation of the loss to compute the gradient
    optimizer.step()        ## Optimization using gradients
    trainloss = torch.cat((trainloss, torch.tensor([train_loss])), dim = 0)
time = net(x_data)
result = torch.cat((result, time), dim=1)
```

Example: Neural network in PyTorch

Output the parameters of the neural network

```

F = torch.tensor([])    ## weight_hidden1
E = torch.tensor([])    ## bias_hidden1
D = torch.tensor([])    ## weight_hidden2
C = torch.tensor([])    ## bias_hidden2
B = torch.tensor([])    ## weight_regression
A = torch.tensor([])    ## bias_regression
for j in range(nI):
    F = torch.cat((F, net.hidden1.weight[:,j]), dim = 0)
E = torch.cat((E, net.hidden1.bias), dim = 0)
for k in range(nH1):
    D = torch.cat((D, net.hidden2.weight[:,k]), dim = 0)
C = torch.cat((C, net.hidden2.bias), dim = 0)
B = torch.cat((B, torch.squeeze(torch.FloatTensor(net.regression.weight), dim = 0)), dim = 0)
A = torch.cat((A, net.regression.bias), dim = 0)

np.savetxt('hidden1_weight.csv',F.data.numpy(),delimiter = ',')
np.savetxt('hidden1_bias.csv',E.data.numpy(),delimiter = ',')
np.savetxt('hidden2_weight.csv',D.data.numpy(),delimiter = ',')
np.savetxt('hidden2_bias.csv',C.data.numpy(),delimiter = ',')
np.savetxt('regression_weight.csv',B.data.numpy(),delimiter = ',')
np.savetxt('regression_bias.csv',A.data.numpy(),delimiter = ',')

```

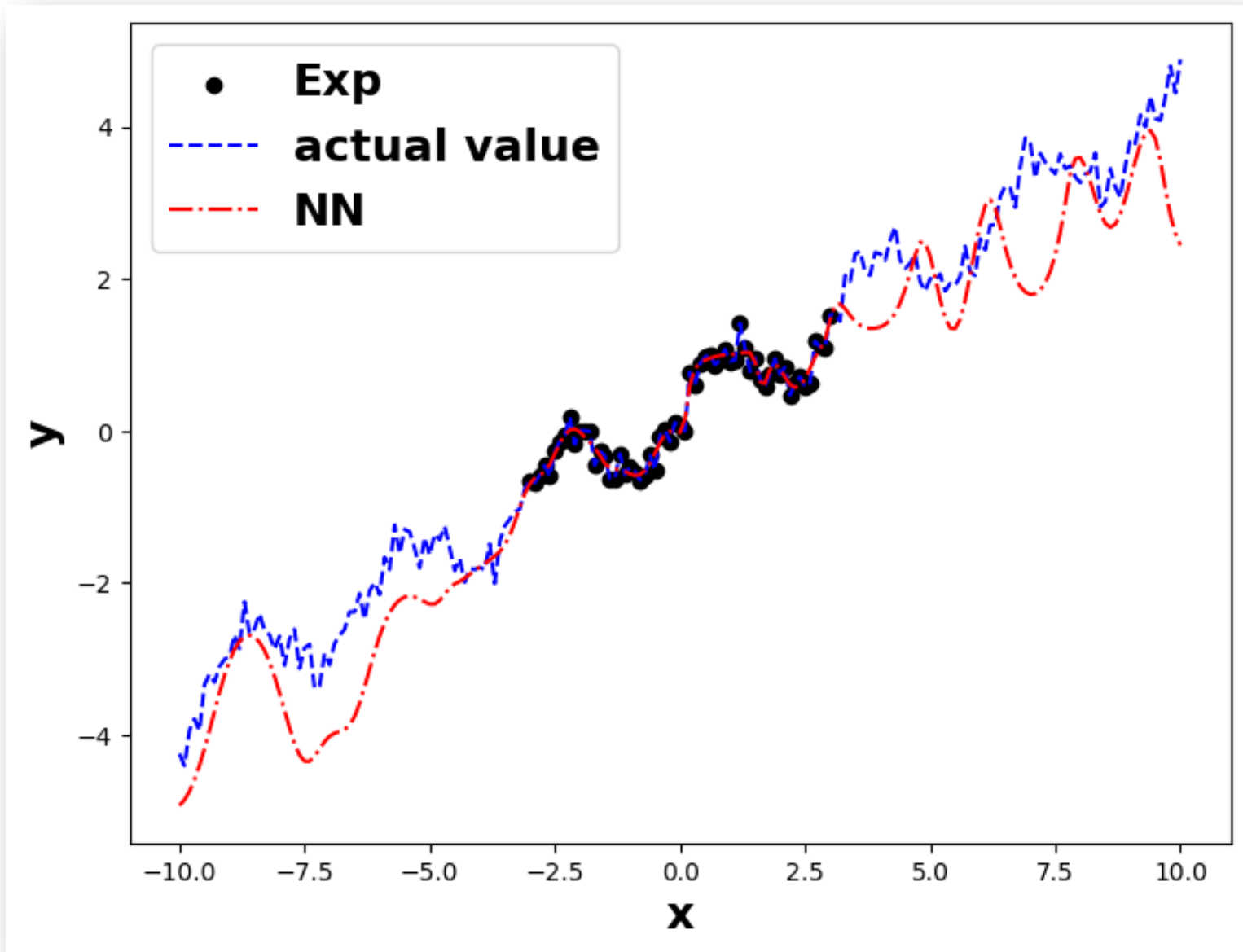

Example: Neural network in PyTorch

Plot figures

```
plt.figure(num = 1,figsize = (8,6))
plt.plot([i for i in range(len(trainloss[0:]))],trainloss[0:],
        'k--',linewidth = 2, markerfacecolor='none',label = 'Train')
plt.legend(loc = 'upper right',prop={'size':18,'weight':'bold'})
plt.xlabel("Epochs",fontsize = 18,fontweight = 'bold')
plt.ylabel("Loss",fontsize = 18,fontweight = 'bold')
plt.yscale('log')

plt.figure(num = 2, figsize =(8,6))
plt.scatter(x_train[:,0], y_train, color = 'black',marker='o',label = 'Exp')
plt.plot(x_data[:,0], y_data,'b--',label = 'actual value')
plt.plot(x_data[:,0], result[:,2].detach().numpy(), 'r-.',label = 'NN')
plt.xlabel("x",fontsize = 18,fontweight = 'bold')
plt.ylabel("y",fontsize = 18,fontweight = 'bold')
plt.legend(loc = 'upper left',prop = {'size':18,'weight':'bold'})
plt.show()
```

Example: Neural network in PyTorch



Outline

- ① Introduction
- ② Bayesian neural network approach
 - ★ Bayesian approach
 - ★ Neural network
 - ★ **Bayesian neural network** (flexible Bayesian modeling software)
- ③ Results and discussion
 - ★ Nuclear masses
 - ★ Nuclear β -decay half-lives
 - ★ Some other results
- ④ Summary and perspectives

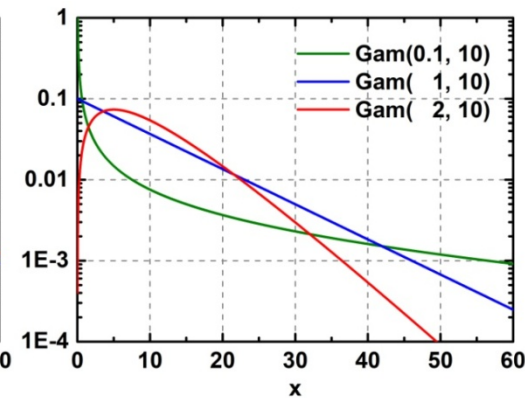
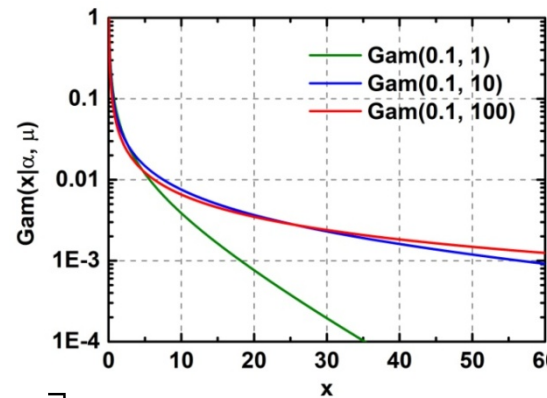
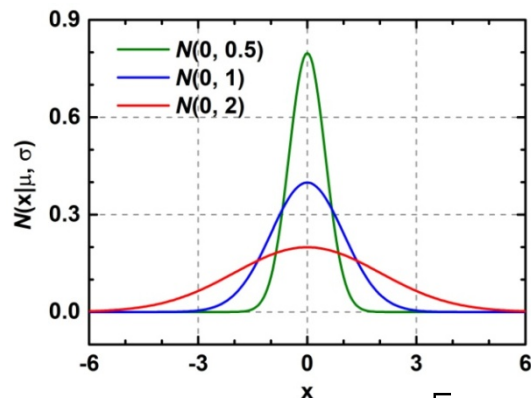
Bayesian approach in regression problem

- Posterior distributions of parameters are [Neal1996Springer](#)

$$p(\omega | D) = \frac{p(D | \omega)p(\omega)}{p(D)} \propto p(D | \omega)p(\omega), \quad D = \{(x_1, t_1), (x_2, t_2), \dots, (x_N, t_N)\}$$

- ✓ prior distribution $p(\omega)$: $p(\omega) = N(\omega | 0, \sigma_\omega)$

$$p(\tau_\omega = 1 / \sigma_\omega^2) = \text{Gam}(\tau_\omega | \alpha_\omega, \mu_\omega); \quad p(\tau_n = 1 / \sigma_n^2) = \text{Gam}(\tau_n | \alpha_n, \mu_n)$$



$$N(x | \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x - \mu)^2}{2\sigma^2}\right]$$

$$\text{Gam}(x | \alpha, \mu) = \frac{(\alpha / \mu)^\alpha}{\Gamma(\alpha)} x^{\alpha-1} \exp\left(-\frac{\alpha x}{\mu}\right)$$

- ✓ likelihood function $p(D | \omega)$

$$p(x, t | \omega) = \exp(-\chi^2 / 2), \quad \chi^2 = \sum_{n=1}^N \left[\frac{t_n - y(x_n, \omega)}{\sigma_n} \right]^2$$

Bayesian approach in regression problem

- Posterior distributions of parameters are [Neal1996Springer](#)

$$p(\omega | D) = \frac{p(D | \omega)p(\omega)}{p(D)} \propto p(D | \omega)p(\omega), \quad D = \{(x_1, t_1), (x_2, t_2), \dots, (x_N, t_N)\}$$

✓ sampling with Markov chain Monte Carlo (MCMC) method

- Make predictions

$$\langle y_n \rangle = \int y(x_n, \omega) p(\omega | D) d\omega = \frac{1}{K} \sum_{k=1}^K y(x_n, \omega_k)$$

$$\Delta y_n = \sqrt{\langle y_n^2 \rangle - \langle y_n \rangle^2}$$

Remark:

- BNN approach can give the joint probability distribution of all parameters, from which we can get the correlations among parameters, so the number of independent parameters may be much less the number of BNN parameters.

Monte Carlo method for calculating definite integrals

- Replace mathematical expectations with arithmetical averages

$$J = \int_a^b h(x) dx = \int_a^b \frac{h(x)}{f(x)} f(x) dx$$

Let $g(x) = \frac{h(x)}{f(x)} \Rightarrow J = \int_a^b g(x) f(x) dx$

Taking random samples x_i based on the probability density function $f(x)$, one can replace the integral value with a mathematical expectation

$$J = \int_a^b g(x) f(x) dx = E(g) = \frac{1}{n} \sum_{i=1}^n g(x_i) = \overline{g(x)}$$

- ✓ If $f(x)$ is selected as the probability density function of $[a, b]$ uniform distribution

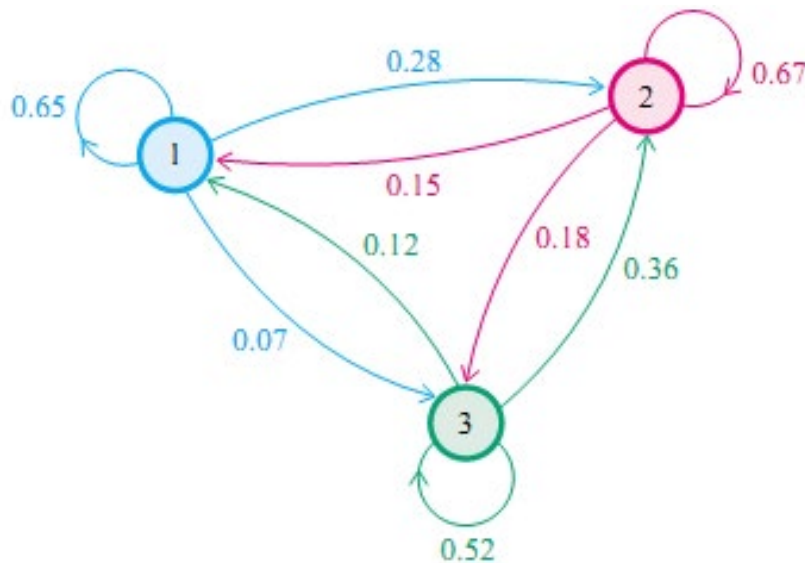
$$f(x) = \frac{1}{b-a} \Rightarrow g(x) = \frac{h(x)}{f(x)} = (b-a)h(x)$$

$$\longrightarrow J = \frac{1}{n} \sum_{i=1}^n g(x_i) = \frac{1}{n} \sum_{i=1}^n (b-a)h(x_i) = \frac{b-a}{n} \sum_{i=1}^n h(x_i)$$

Markov chain

- Sociologist often divides people into three classes according to their economic status

		Children generation		
State		1	2	3
Parent generation	1	0.65	0.28	0.07
	2	0.15	0.67	0.18
	3	0.12	0.36	0.52



Matrix of transition probability

$$Q = \begin{bmatrix} 0.65 & 0.28 & 0.07 \\ 0.15 & 0.67 & 0.18 \\ 0.12 & 0.36 & 0.52 \end{bmatrix}$$

Suppose the proportion of the current generation in the upper, middle and lower classes is

$$p_0 = [p_0(1), p_0(2), p_0(3)]$$

Then the distribution proportion of their children, grandchildren, ..., the n th generation will be

$$p_1 = p_0 Q$$

$$p_2 = p_1 Q = p_0 Q^2$$

...

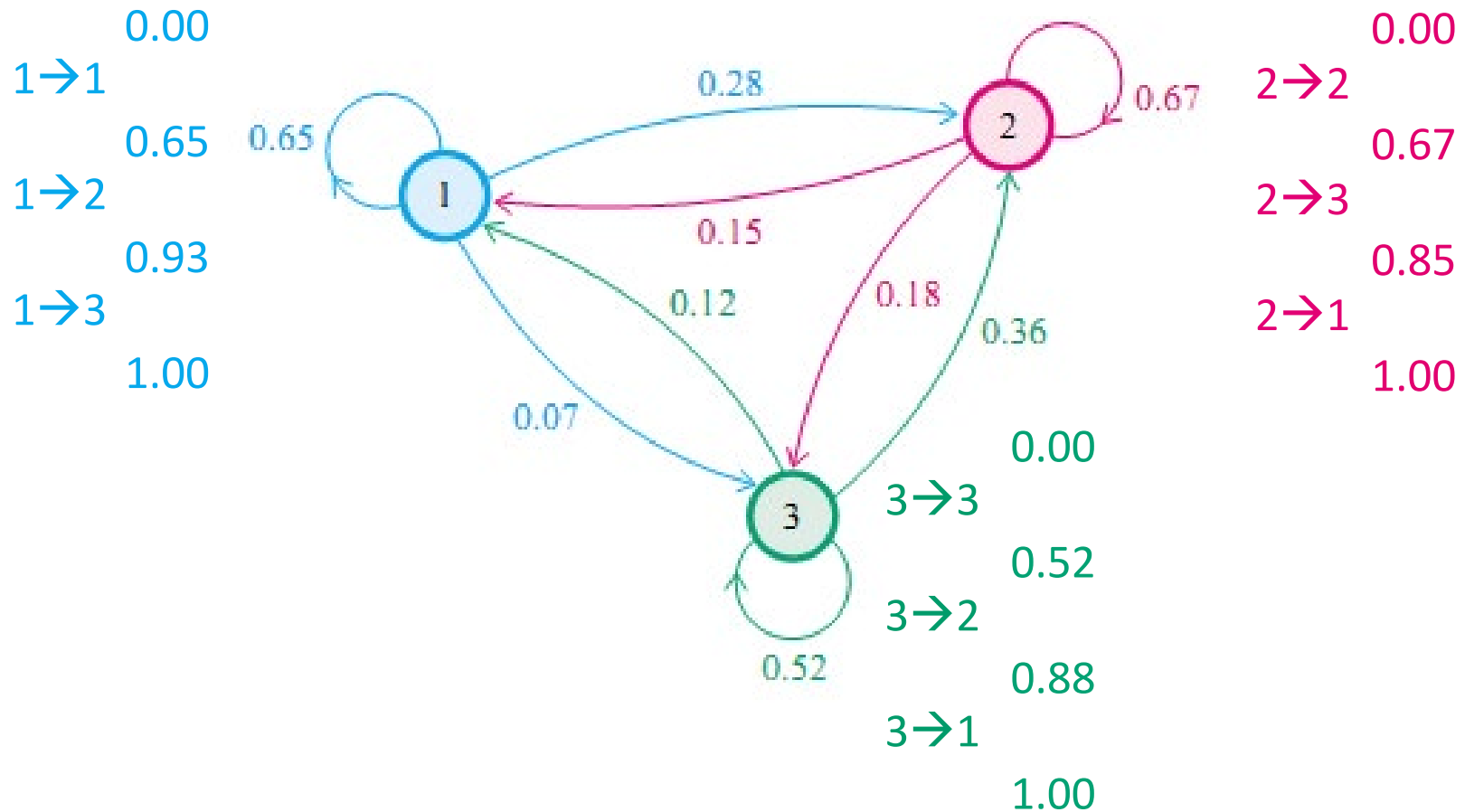
$$p_n = p_{n-1} Q = p_0 Q^n$$

Markov chain

generation	lower	middle	upper	generation	lower	middle	upper
0	0.210	0.680	0.110	0	0.75	0.15	0.1
1	0.252	0.554	0.194	1	0.522	0.347	0.132
2	0.270	0.512	0.218	2	0.407	0.426	0.167
3	0.278	0.497	0.225	3	0.349	0.459	0.192
4	0.282	0.490	0.226	4	0.318	0.475	0.207
5	0.285	0.489	0.225	5	0.303	0.482	0.215
6	0.286	0.489	0.225	6	0.295	0.485	0.220
7	0.286	0.489	0.225	7	0.291	0.487	0.222
8	0.289	0.488	0.225	8	0.289	0.488	0.225
9	0.286	0.489	0.225	9	0.286	0.489	0.225
10	0.286	0.489	0.225	10	0.286	0.489	0.225
...

Even though the initial states are different, when n is large enough, each row of the Q^n matrix converges stably to the probability distribution [0.286, 0.489, 0.225].

Markov chain



1—(0.61)—1—(0.37)—1—(0.61)—1—(0.68)—2—(0.34)—2—(0.61)—2—(0.88)—
 1—(0.97)—3—(0.66)—2—(0.55)—2—(0.87)—1—...

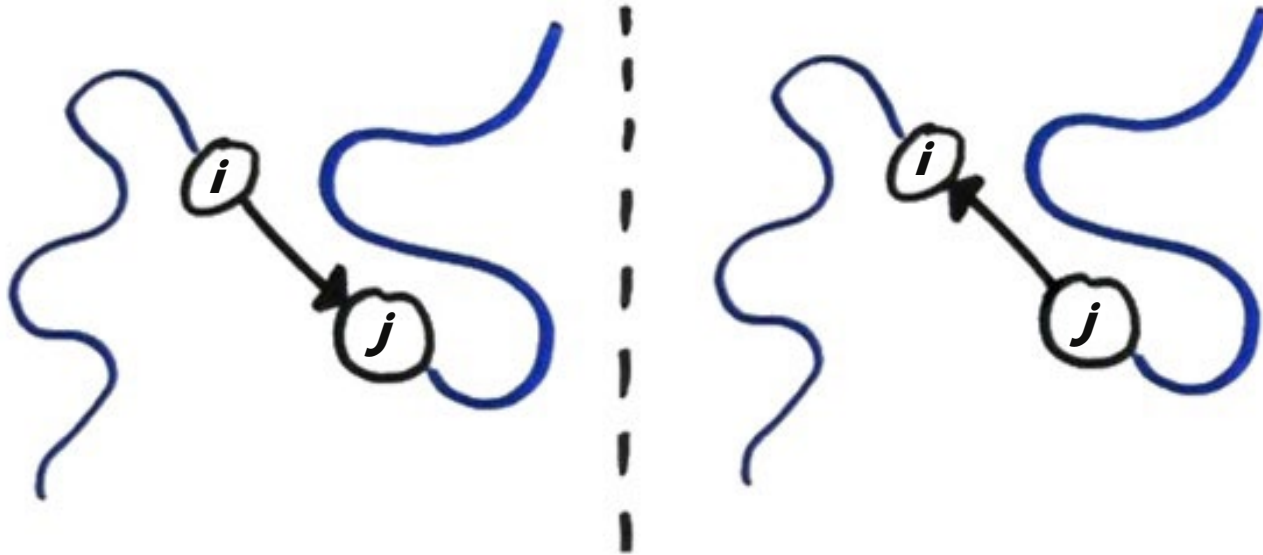
➔ Sample: 1, 1, 1, 1, 2, 2, 2, 1, 3, 2, 2, 1, ... (the probabilities of occurrence of 1, 2, and 3 are 0.286, 0.489, and 0.225, respectively)

Markov Chain Monte Carlo

- **Basic idea of Markov chain Monte Carlo method (MCMC):**
 - ✓ Given the probability distribution $p(x)$ to be sampled
 - ✓ **Construct the transfer matrix Q of the Markov chain** so that the stationary distribution of this Markov chain is $p(x)$
 - ✓ Randomly select the initial state x_0
 - ✓ Perform a Markov process transfer to obtain a series of state values: $\{x_0, x_1, \dots, x_n, x_{n+1}, \dots\}$
 - ✓ Check the number of steps at which the Markov process converges, if it is the n th step, then **$\{x_n, x_{n+1}, \dots\}$ is a sample of the distribution $p(x)$**

Markov Chain Monte Carlo

- Detailed balance condition: If the transfer matrix Q and distribution $p(x)$ of the aperiodic Markov chain satisfy:



$$p(i)q(i, j) = p(j)q(j, i)$$

Then $p(x)$ is the stationary distribution of the Markov chain, matrix element $q(i, j)$ of Q represents the transition probability from state i to j , and the above equation is called detailed balance condition.

Markov Chain Monte Carlo

● **Metropolis-Hastings method:** there is a Markov chain with a transfer matrix of Q , in general:

$$p(i)q(i, j) \neq p(j)q(j, i)$$

that is, the detailed balance condition does not hold, so $p(x)$ cannot be the stationary distribution of this Markov chain. In order to satisfy the detailed balance condition, the acceptance rate $\alpha(i, j)$ is introduced

$$\begin{cases} \alpha(i, j) = p(j)q(j, i) \\ \alpha(j, i) = p(i)q(i, j) \end{cases} \Rightarrow \underbrace{p(i)q(i, j)\alpha(i, j)}_{q'(i, j)} = \underbrace{p(j)q(j, i)\alpha(j, i)}_{q'(j, i)}$$

Therefore, the Markov chain of the transfer matrix Q' satisfies the detailed balance condition, that is, the stationarity distribution of the Markov chain Q' is $p(x)$.

Metropolis et al., J. Chem. Phys. 21, 1087 (1953); Hastings, Biometrika. 57, 97 (1970)

Markov Chain Monte Carlo

? If the acceptance rate $\alpha(i, j)$ is too small, the sampling process is easy to stay in place due to the high rejection rate, which makes the Markov chain traverse all the spaces too long and the sampling efficiency is too low.

For example: $\alpha(i, j) = 0.1$, $\alpha(j, i) = 0.2$, then

$$p(i)q(i, j) \times 0.1 = p(j)q(j, i) \times 0.2$$

Enlarged both sides of the above equation by a factor of 5

$$p(i)q(i, j) \times 0.5 = p(j)q(j, i) \times 1$$

The detailed balance condition is still satisfied now, but the acceptance rate is increased. Therefore, in order to improve the acceptance rate, $\alpha(i, j)$ and $\alpha(j, i)$ can be scaled up, so that the larger of the two is enlarged to 1, i.e.,

$$\alpha(i, j) = \min \left\{ \frac{p(j)q(j, i)}{p(i)q(i, j)}, 1 \right\} \text{ 或 } \min \left\{ \frac{p(j)q(i | j)}{p(i)q(j | i)}, 1 \right\}$$



$p(x)$ is no longer required to be a normalized probability distribution

Markov Chain Monte Carlo

● Metropolis-Hastings sampling steps:

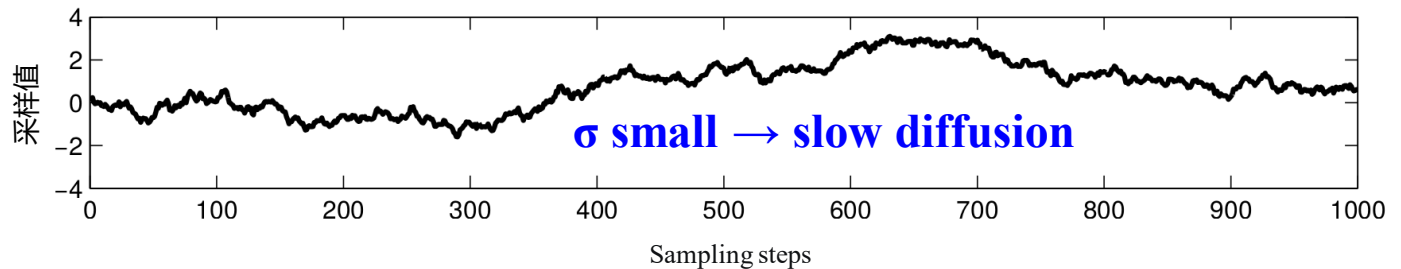
- ✓ Select the proposal distribution q
- ✓ Initialize the state of the Markov chain to $X_0 = x_0$
- ✓ For $t = 0, 1, 2, \dots$, loop the following process for sampling
 - The state of the Markov chain at the t th moment is $X_t = x_t$, sampling $x^* \sim q(x|x_t)$
 - Sample u from $[0, 1]$ uniform distribution
 - If $u < \alpha(x_t, x^*) = \min \left\{ \frac{p(x^*)q(x_t | x^*)}{p(x_t)q(x^* | x_t)}, 1 \right\}$, then accept the transfer, i.e. $X_{t+1} = x^*$; otherwise do not accept the transfer, i.e., $X_{t+1} = x_t$

Note: If the proposed distribution q is Gaussian, x_t is usually taken as the mean of the Gaussian distribution, and its standard deviation σ is the sampling step. If σ is too small, α will be larger, but the sampling will be slow due to the small step size. If σ is too large, although the step size is large, the sampling efficiency is also low because α will be too small.

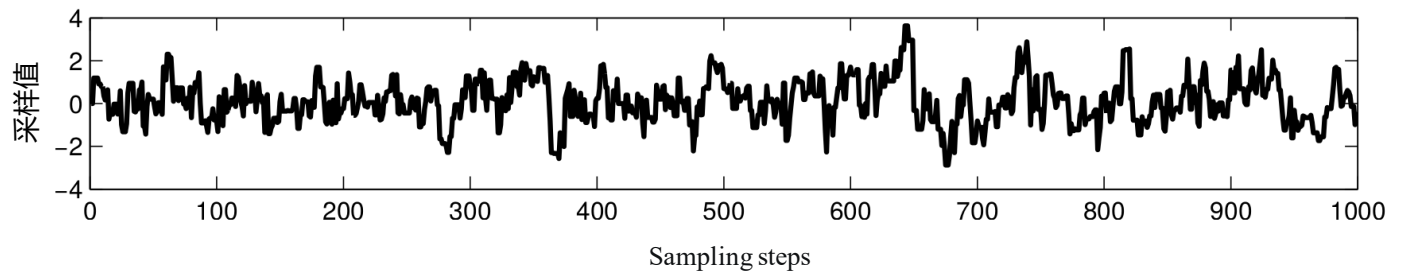
Markov Chain Monte Carlo

- Sampling the standard normal distribution using different step sizes σ

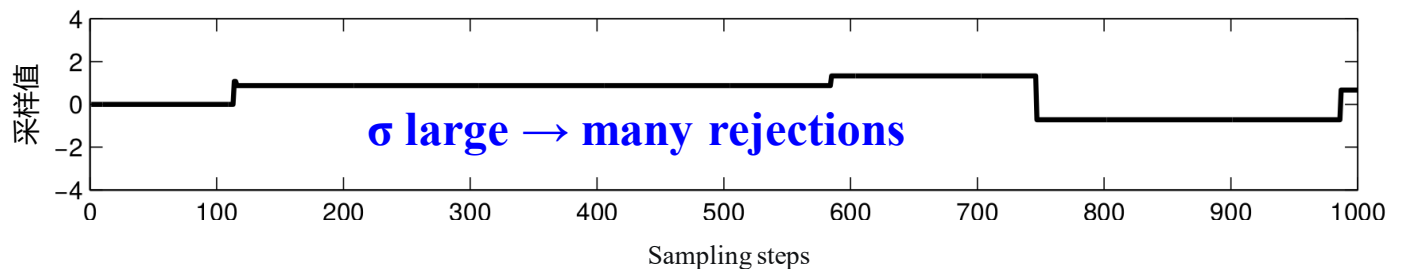
$\sigma = 0.1$
 $\alpha = 99.8\%$



$\sigma = 1$
 $\alpha = 68.4\%$



$\sigma = 100$
 $\alpha = 0.5\%$



Markov Chain Monte Carlo

➤ **0.234 rule** A. Gelman, G.O. Roberts, W.R. Gilks, *Bayesian Statistics* 5, 599 (1996)

For a high-dimensional unimodal probability density, e.g.

$$p = N(0, I_d) \text{ with } q(x^* | x_t) = N(x_t, \sigma_d^2 I_d)$$

$$p = N(\mu, \Sigma) \text{ with } q(x^* | x_t) = N(x_t, \sigma_d^2 \Sigma)$$

- ✓ optimal σ_d : $\sigma_{d \rightarrow \infty} = 2.38 / \sqrt{d}$ (it can hold well when $d \geq 1$)

Since the Σ of the actual problem is often unknown, this rule is difficult to use.

- ✓ optimal α_d : $\alpha_{d \rightarrow \infty} = 0.234$ (it can hold well when $d \geq 6$)

A reasonable sampling step size is usually chosen based on α . For practical problems, the sampling efficiency is generally higher when α around 0.15-0.40.

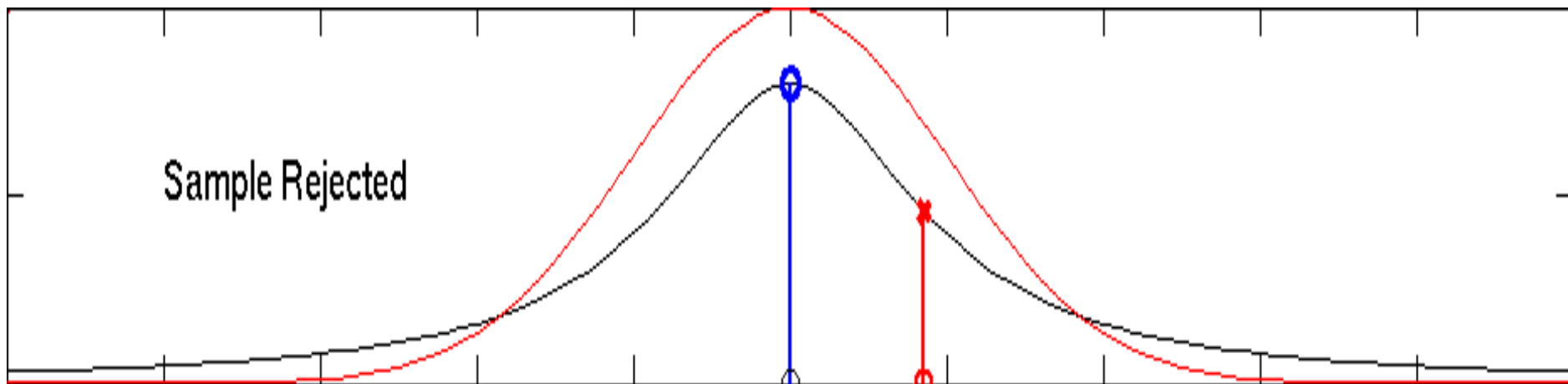
Markov Chain Monte Carlo

Sampling process:

$$x_0 \rightarrow \begin{cases} x_* \in N(x_0, \sigma) \\ \alpha(x_0, x_*) \\ u \in U(0,1) \end{cases} \rightarrow x_1 = \begin{cases} x_0, & u > \alpha(x_0, x_*) \\ x_*, & u \leq \alpha(x_0, x_*) \end{cases}$$

$$x_1 \rightarrow \begin{cases} x_* \in N(x_1, \sigma) \\ \alpha(x_1, x_*) \\ u \in U(0,1) \end{cases} \rightarrow x_2 = \begin{cases} x_1, & u > \alpha(x_1, x_*) \\ x_*, & u \leq \alpha(x_1, x_*) \end{cases}$$

.....



Flexible Bayesian Modeling

Software for Flexible Bayesian Modeling and Markov Chain Sampling

This software supports flexible Bayesian learning of regression, classification, density, and other models, based on multilayer perceptron neural networks, Gaussian processes, finite and countably infinite mixtures, and Dirichlet diffusion trees, as well as facilities for inferring sources of atmospheric contamination and for molecular simulation. These are implemented using Markov chain Monte Carlo methods. Facilities for Markov chain sampling from distributions specified by simple formulas for the density or for the prior and likelihood are also included.

Before trying to use the software, you may need to read various [references](#) that describe the models and the Markov chain methods used.

The software is written in C for Unix, Linux, and macOS systems.

Some of the programs in this package are designed to work well with my version of the 'graph' program, see github.com/radfordneal/plotutils, though the data can instead be plotted by whatever plot program you have available.

The source code is available at [gitlab](#).

Current release of 2022-04-21:

This release has major new features and performance improvements for neural network models, including support for convolutional networks and for computation on GPUs.

[Index to documentation in hypertext form](#)

[Tar archive of everything \(40MB\)](#)

[Gzipped tar archive of everything \(12MB\)](#)

The tar archives include the complete hypertext documentation, so that you can set it up to be read from your local machine. The documentation is also readable as simple text files. For directions on how to untar a tar archive, see the [installation instructions](#) within the above hypertext documentation.

Old release of 2020-01-24:

[Index to documentation in hypertext form](#)

[Tar archive of everything](#)

<https://glizen.com/radfordneal/fbm.software.html>

Flexible Bayesian Modeling

INSTALLING THE SOFTWARE:

1. Unpack the tar archive files 'fbm.YYYY-MM-DD.tar' by issuing the Unix command:

```
tar xf fbm.YYYY-MM-DD.tar
```

2. Compile the programs. Go to the directory 'fbm.YYYY-MM-DD' and issue the command

```
./make-all
```

3. Put the 'bin' directory in your search path by adding the following command to the '.bash_profile' file:

```
PATH=$PATH:/home/zmniu/fbm.2004-11-10/bin
```

```
export PATH
```

Flexible Bayesian Modeling

Construct a network

```
net-spec rlog.net 2 15 1 / - 0.05:0.5 0.05:0.5 - x0.05:0.5 - 0.05:0.5
```

```
model-spec rlog.net real 0.2:0.5
```

```
net-spec rlog.net
```

```
data-spec rlog.net 2 1 / rdata .
```

$$y(x, \omega) = a + \sum_{j=1}^H b_j \tanh \left(c_j + \sum_{i=1}^I d_{ji} x_i \right)$$

- ✓ "rlog.net" is the name of the log file, and the arguments "2", "15", and "1", specify the numbers of input, hidden, and output units. Following the "/", the priors for the various groups of network parameters are given, with a "-" indicating that a parameter group should be omitted. The groups in the above command that are not omitted are the **input-hidden weights (d_{ji})**, the **hidden biases (c_j)**, the **hidden-output weights (b_j)**, and the **output bias (a)**. The "x" in front of the prior for the hidden-to-output weights indicates that the prior should be automatically rescaled based on the number of hidden units, so as to produce an effect that is independent of the number of hidden units (in the limit of large numbers).
- ✓ Specify how the network outputs will be used to model the targets in the data set. "real 0.2:0.5" means the targets are real-valued, and are modeled as the network output plus Gaussian noise, with the noise standard deviation being a hyperparameter having the prior given by the last argument of the command.
- ✓ View the architecture and prior specifications stored in the log file.
- ✓ Specify the training data sets. The "2" and "1" arguments give the numbers of inputs and targets. After the "/", specifications for where to get the training are given, specification "rdata" means that the training inputs come from the file 'rdata'. "." means the target items are on the same lines as the inputs.

Flexible Bayesian Modeling

```
# Initialize the network and start the simulation for a few iterations so that the network parameters can take on moderate values
```

```
net-gen rlog.net fix 0.5
```

```
mc-spec rlog.net repeat 10 sample-noise heatbath hybrid 200:10 0.3
```

```
net-mc rlog.net 5
```

```
net-plt t r rlog.net
```

- ✓ The 'net-gen' command stores a network in the log file with index zero, in which the hyperparameters have values of 0.5, and the network parameters are zero. This is the initial state of the simulation run.
- ✓ The 'mc-spec' command specifies the Markov chain operations to be performed in the initial phase. Here, each iteration consists of ten repetitions of the following steps: Gibbs sampling for the noise level, a heatbath replacement of the momentum variables, and a hybrid Monte Carlo update with a trajectory 200 leapfrog steps long, using a window of 10, and a stepsize adjustment factor of 0.3. Note that the hyperparameters are not updated, and hence will remain fixed at values of 0.5.
- ✓ The 'mc-spec' command specifies 5 iterations starting from the initial state and the settings of 'net-gen' and 'mc-spec' commands.
- ✓ The 'net-plt' command is used to output the rejection rate of the sample in order to select the appropriate stepsize. If the rejection rate is too high, such as more than 0.3, then it is necessary to reduce the step size and re-run the last 3 steps until the rejection rate is less than 0.3.

Flexible Bayesian Modeling

start sampling

```
mc-spec rlog.net sample-sigmas heatbath hybrid 2500:20 0.3
net-mc rlog.net 2000 &
net-display rlog.net
```

Output results

```
net-tbl tw1@w2@w3@w4@ rlog.net > my.dat
```

$$y(x, \omega) = a + \sum_{j=1}^H b_j \tanh \left(c_j + \sum_{i=1}^I d_{ji} x_i \right)$$

- ✓ The 'mc-spec' command appends a new set of Markov chain operations to the log file, which will override the previous set. These operations are Gibbs sampling for both the hyperparameters and the noise level (the "sigmas"), a heatbath update for the momentum variables, and a hybrid Monte Carlo update with a trajectory 2500 leapfrog steps long, a window of 20, and a **stepsize adjustment factor of 0.3**. The stepsize may select again in order to find out how large this factor can be while keeping the rejection rate low.
- ✓ The 'net-mc' command will perform $2000-5=1995$ iterations of the sampling phase. '&' means one puts the command in the background.
- ✓ The 'net-display' command is used to **monitor progress**, with which you can look at the **parameters and hyperparameters** of the last network saved in the log file
- ✓ The 'net-tbl' command outputs the result to the file my.dat, where **t** represents an index of the iterations, i.e. 1, 2, 3,... etc., **w1@** represents the input hidden weight parameter group (d_{ji}), which is output in the order of first looping hidden unit and then looping input variable), **w2@** represents the hidden bias parameter group (c_j), **w3@** represents the hidden output weight parameter group (b_j), **w4@** represents the output bias parameter group (a).

Toy model

True : $y = 0.3 + 0.4x + 0.5\sin(2x)$

Data : $y = 0.3 + 0.4x + 0.5\sin(2x) + 0.2 \times \text{randn}$

➤ Number of training data: $N=61$, $x \in [-3, 3]$

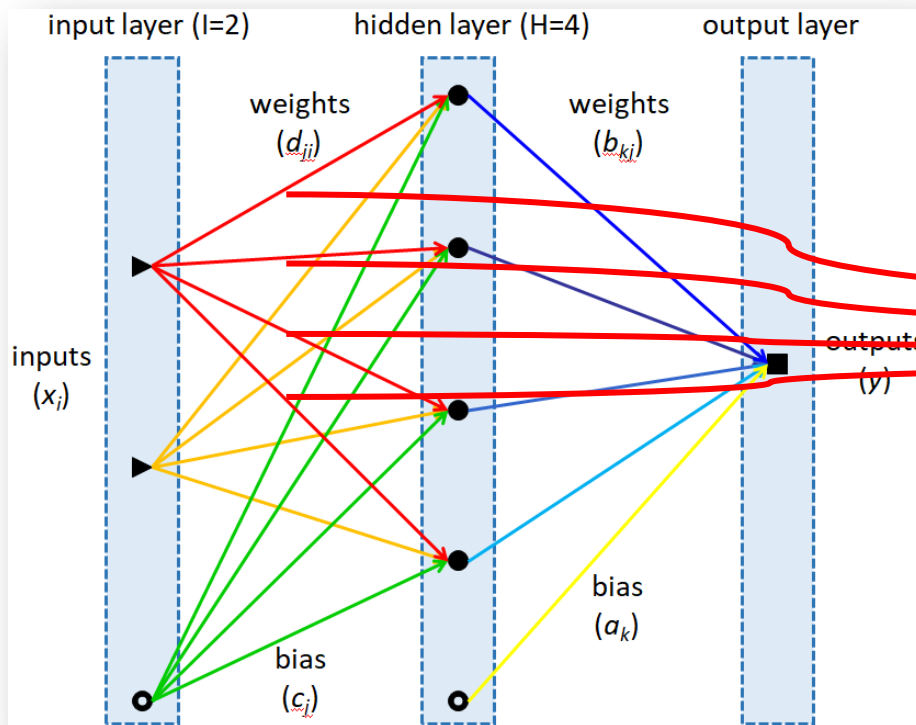
1 input : $y = f(x)$

2 inputs : $y = f[x_1 = x, x_2 = \sin(2x)]$

➤ Number of hidden unit:
 $H=20$ for $f(x)$; $H=15$ for $f(x_1, x_2)$

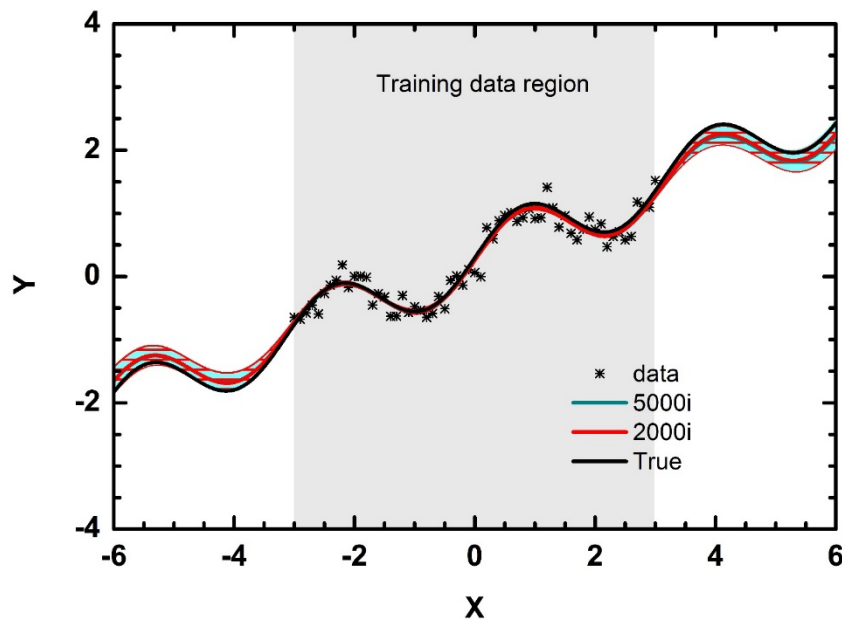
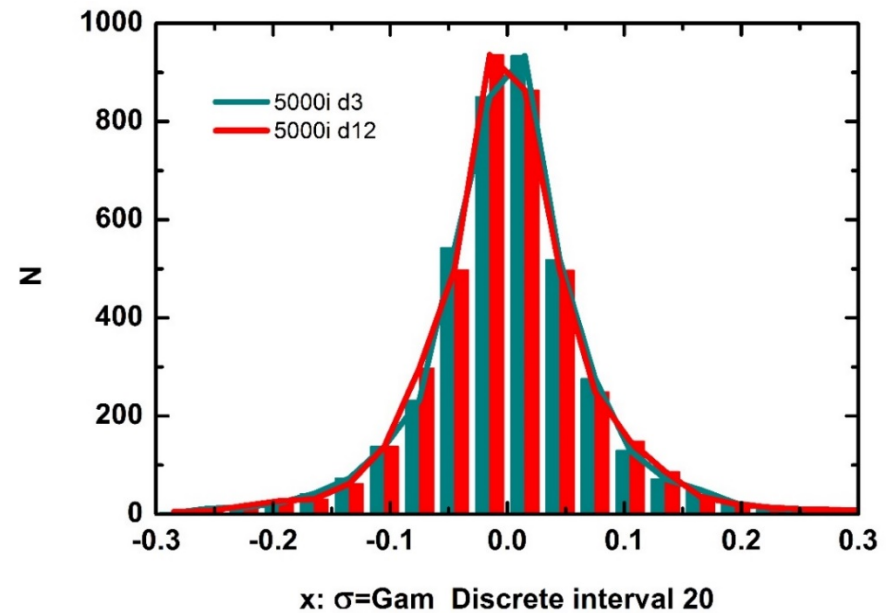
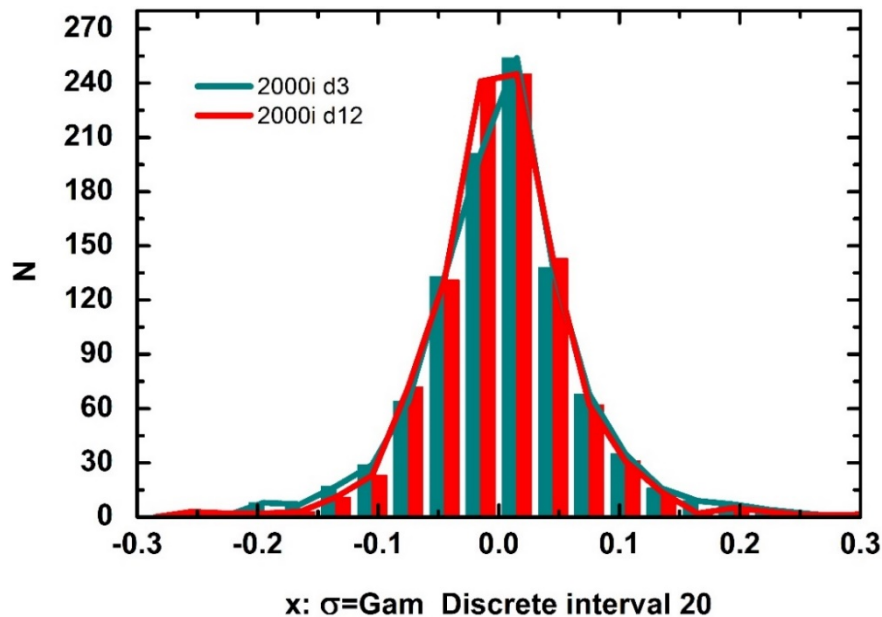
➤ Number of parameters: 61

Likelihood function: $p(x, y | \omega) = \exp(-\chi^2 / 2)$, $\chi^2 = \sum_{i=1}^N \left(\frac{y_i - f(x_i, \omega)}{\sigma} \right)^2$



$d_{11}, d_{21}, d_{31},$
 d_{41}, \dots should
have the same
probability
distribution!!!

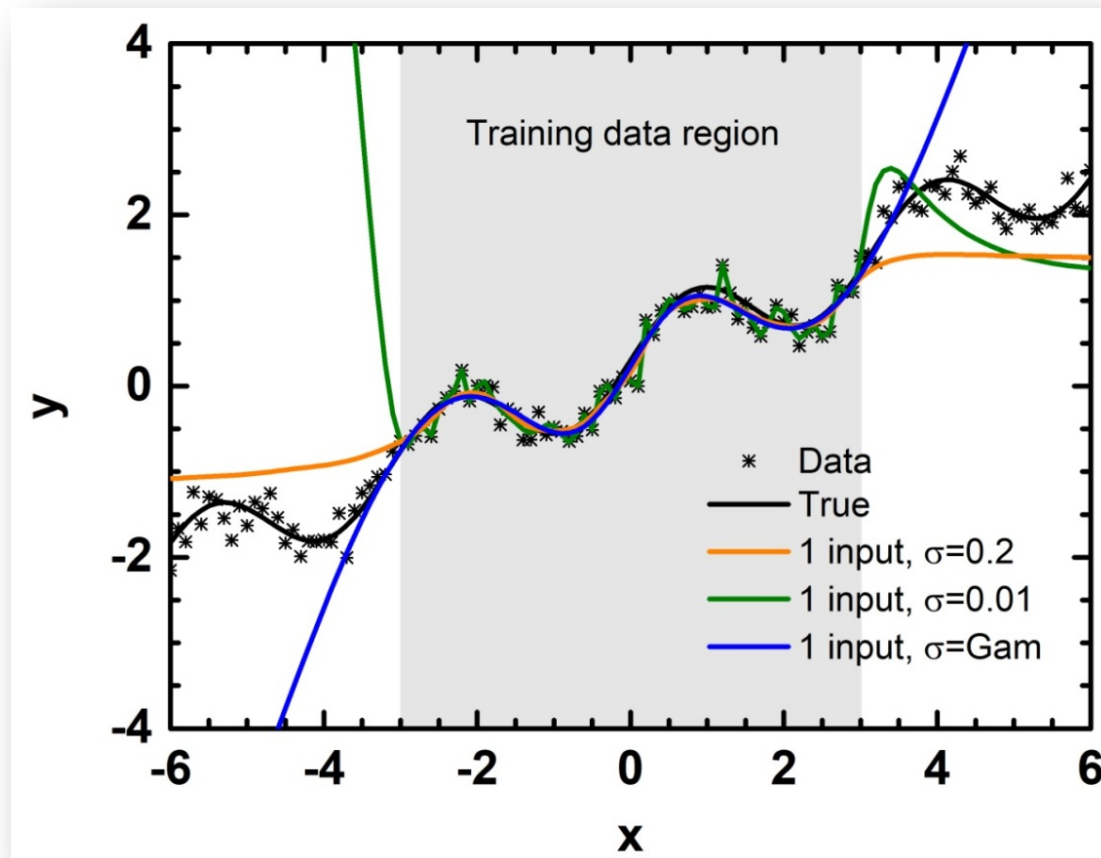
Toy model



✓ After 2000 steps of sampling, the distribution of different d parameters is almost the same.

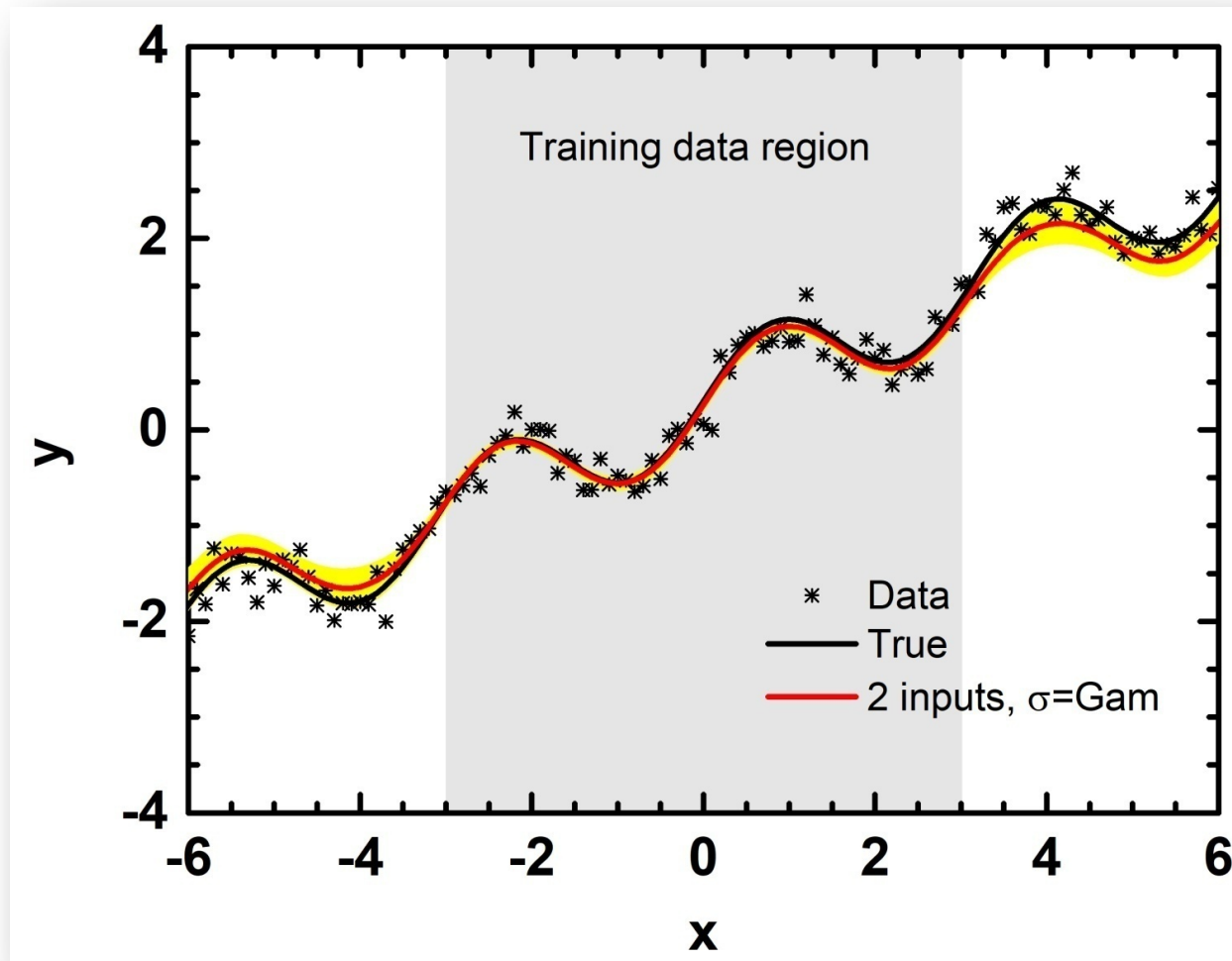
✓ The distribution and prediction results with 2000 and 5000 samples are almost the same.

Toy model



- ✓ BNN can avoid overfitting if a Gamma distribution is taken as the noise prior.
- ✓ Direct BNN fitting with x as the only input variable can only extrapolate around a few steps from known region, while the overfitting would make the extrapolation unacceptable.

Toy model



✓ Including reasonable variable is very effective for the extrapolation of neural network and the uncertainties of predictions are also reasonable.

Toy model

Homework: Please use Tensorflow Playground, 1stOpt, PyTorch, or flexible Bayesian modeling software to perform neural network calculations with some data you have or the following data for the toy model.

x_1	x_2	y
-3.00000	0.279415	-0.646512
-2.90000	0.464602	-0.678828
-2.80000	0.631267	-0.579860
-2.70000	0.772764	-0.452795
-2.60000	0.883455	-0.593300
-2.50000	0.958924	-0.267339
-2.40000	0.996165	-0.138229
-2.30000	0.993691	-0.060193
-2.20000	0.951602	0.184503
-2.10000	0.871576	-0.174407
-2.00000	0.756802	0.003048
-1.90000	0.611858	0.005739
-1.80000	0.442520	-0.010562
-1.70000	0.255541	-0.450648
-1.60000	0.058374	-0.268406
-1.50000	-0.141120	-0.322984
-1.40000	-0.334988	-0.629047
-1.30000	-0.515501	-0.626160
-1.20000	-0.675463	-0.301273
-1.10000	-0.808496	-0.570548
-1.00000	-0.909297	-0.476673
-0.90000	-0.973848	-0.529326
-0.80000	-0.999574	-0.646880
-0.70000	-0.985450	-0.584640
-0.60000	-0.932039	-0.317289
-0.50000	-0.841471	-0.510716
-0.40000	-0.717356	-0.062442
-0.30000	-0.564642	0.011471
-0.20000	-0.389418	-0.139052
-0.10000	-0.198669	0.107544
0.00000	0.000000	0.062445

x_1	x_2	y
0.10000	0.198669	-0.001129
0.20000	0.389418	0.771977
0.30000	0.564642	0.598594
0.40000	0.717356	0.884152
0.50000	0.841471	0.967547
0.60000	0.932039	1.010313
0.70000	0.985450	0.871936
0.80000	0.999574	0.930358
0.90000	0.973848	1.072038
1.00000	0.909297	0.917471
1.10000	0.808496	0.933068
1.20000	0.675463	1.412228
1.30000	0.515501	1.088899
1.40000	0.334988	0.784031
1.50000	0.141120	0.962315
1.60000	-0.058374	0.685144
1.70000	-0.255541	0.582374
1.80000	-0.442520	0.746519
1.90000	-0.611858	0.944764
2.00000	-0.756802	0.747328
2.10000	-0.871576	0.835506
2.20000	-0.951602	0.470635
2.30000	-0.993691	0.631033
2.40000	-0.996165	0.709430
2.50000	-0.958924	0.577907
2.60000	-0.883455	0.634385
2.70000	-0.772764	1.179861
2.80000	-0.631267	1.106616
2.90000	-0.464602	1.098670
3.00000	-0.279415	1.521438

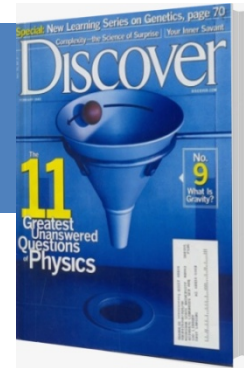
Outline

- ① Introduction
- ② Bayesian neural network approach
 - ★ Bayesian approach
 - ★ Neural network
 - ★ Bayesian neural network
- ③ Results and discussion
 - ★ Nuclear masses
 - ★ Nuclear β -decay half-lives
 - ★ Some other results
- ④ Summary and perspectives

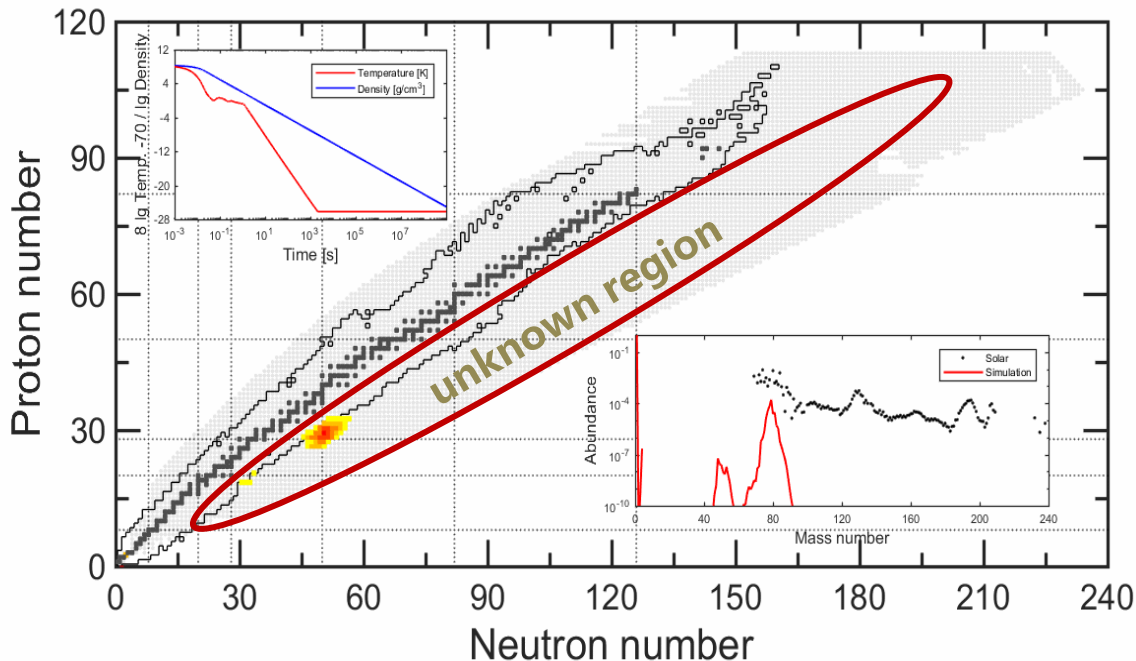
Origin of elements

How were the heavy elements from iron to uranium made?

DISCOVER: The 11 greatest unanswered questions of Physics



Rapid neutron-capture process (r-process)



Key nuclear physics inputs:

- ✓ Nuclear mass \rightarrow r-process path
- ✓ β -decay half-life \rightarrow r-process time scale

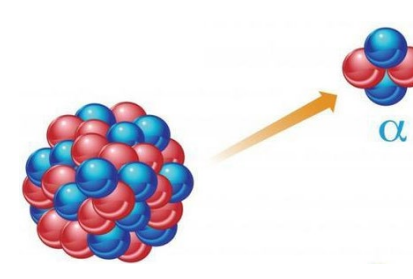
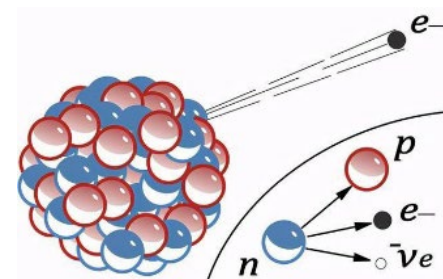
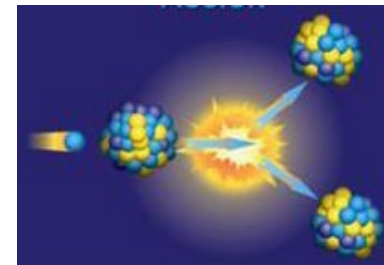
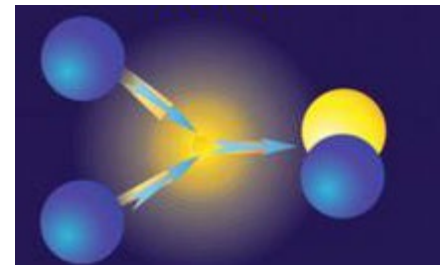
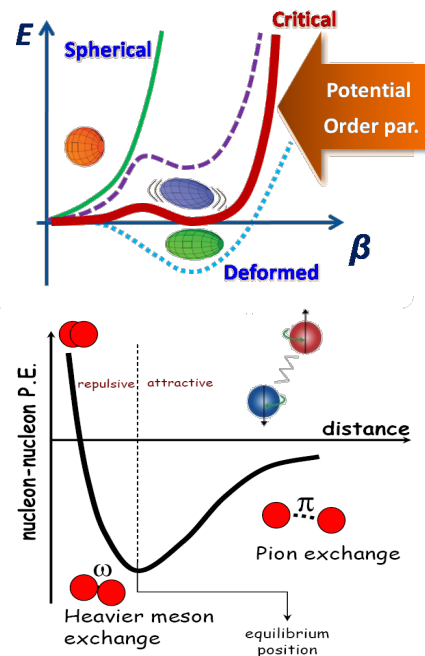
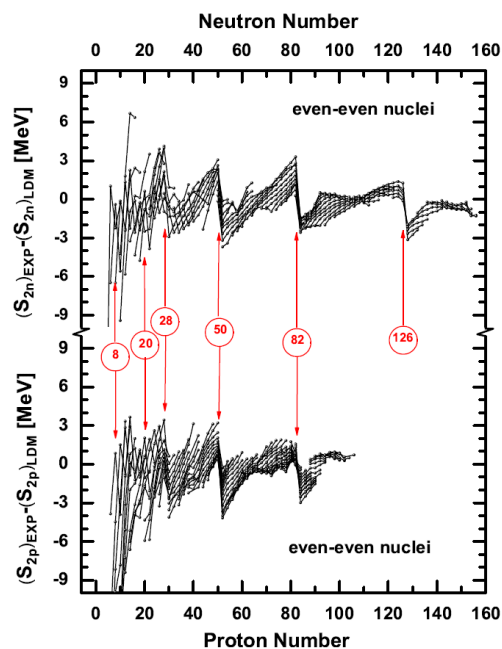
Accurate theoretical predictions of nuclear masses and β -decay half-lives are crucial for understanding the r-process.

Nuclear masses

● Nuclear mass is a fundamental quantity in nuclear physics. It plays important roles not only in various aspects of nuclear physics, but also in other branches of physics, such as astrophysics and nuclear engineering. [Lunney2003RMP, Burbidge1957RMP]

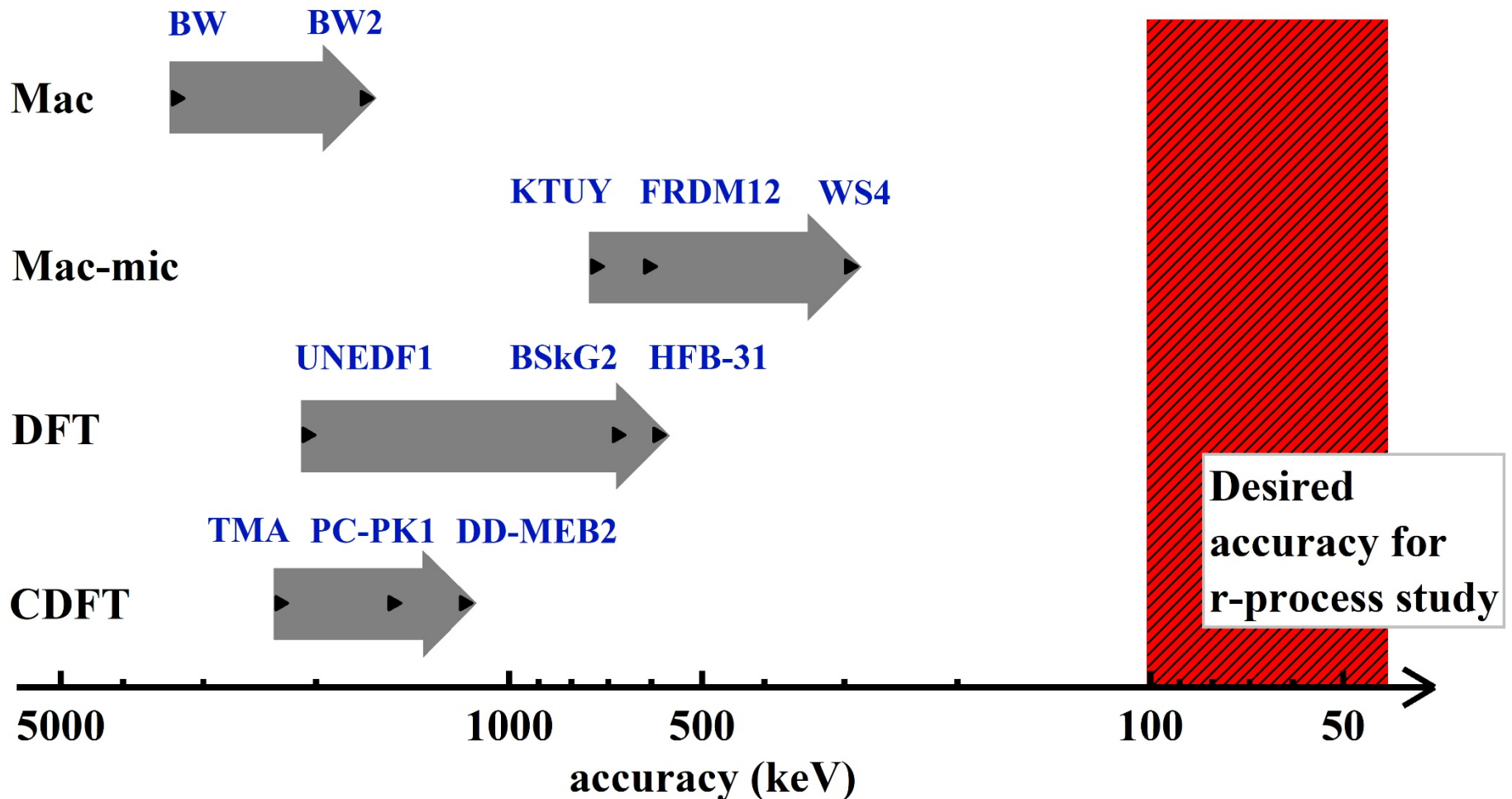
★ Nuclear physics: it contains wealth of nuclear structure information such as magic number and shape transition, and it is widely used to extract nuclear effective interactions.

★ Other branches: it is essential to determine nuclear reaction and decay energies, so it is important in astrophysics and nuclear engineering.



Nuclear mass models

- ★ Macroscopic mass models: BW [[Weizsäcker1935ZP](#), [Bethe1937RMP](#), [Kirson2008NPA](#)]
- ★ Macro-microscopic mass models: FRDM, WS4 [[Moller2012PRL](#), [Wang2014PLB](#)]
- ★ Microscopic mass models: Skyrme HFB, RMF [[Goriely2016PRC](#), [Geng2005PTP](#)]



Influence of masses

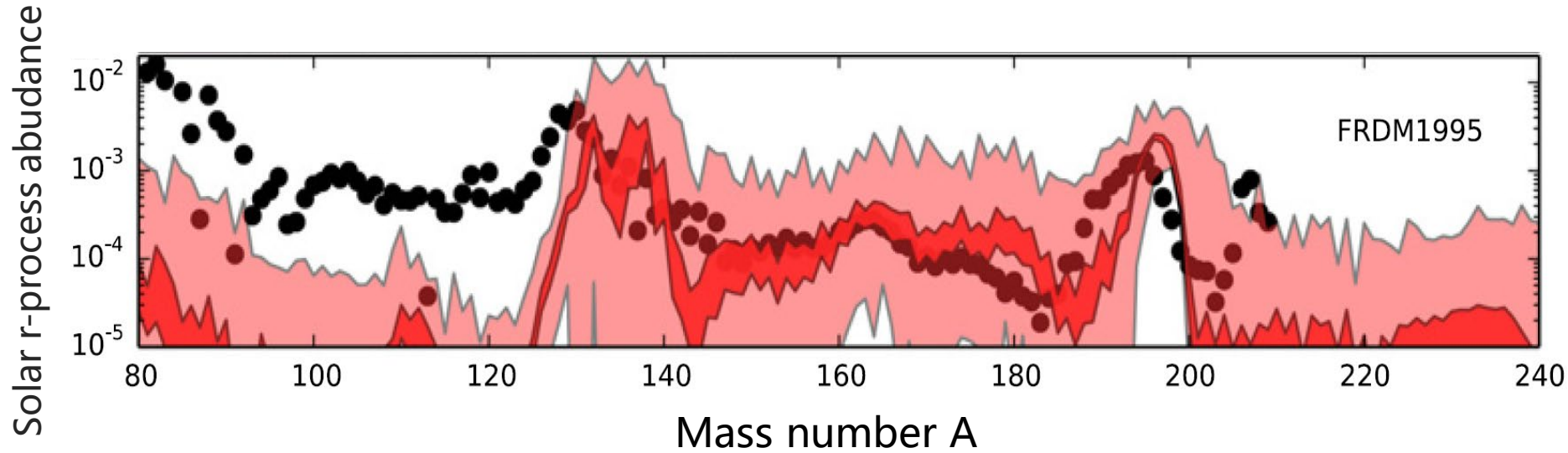


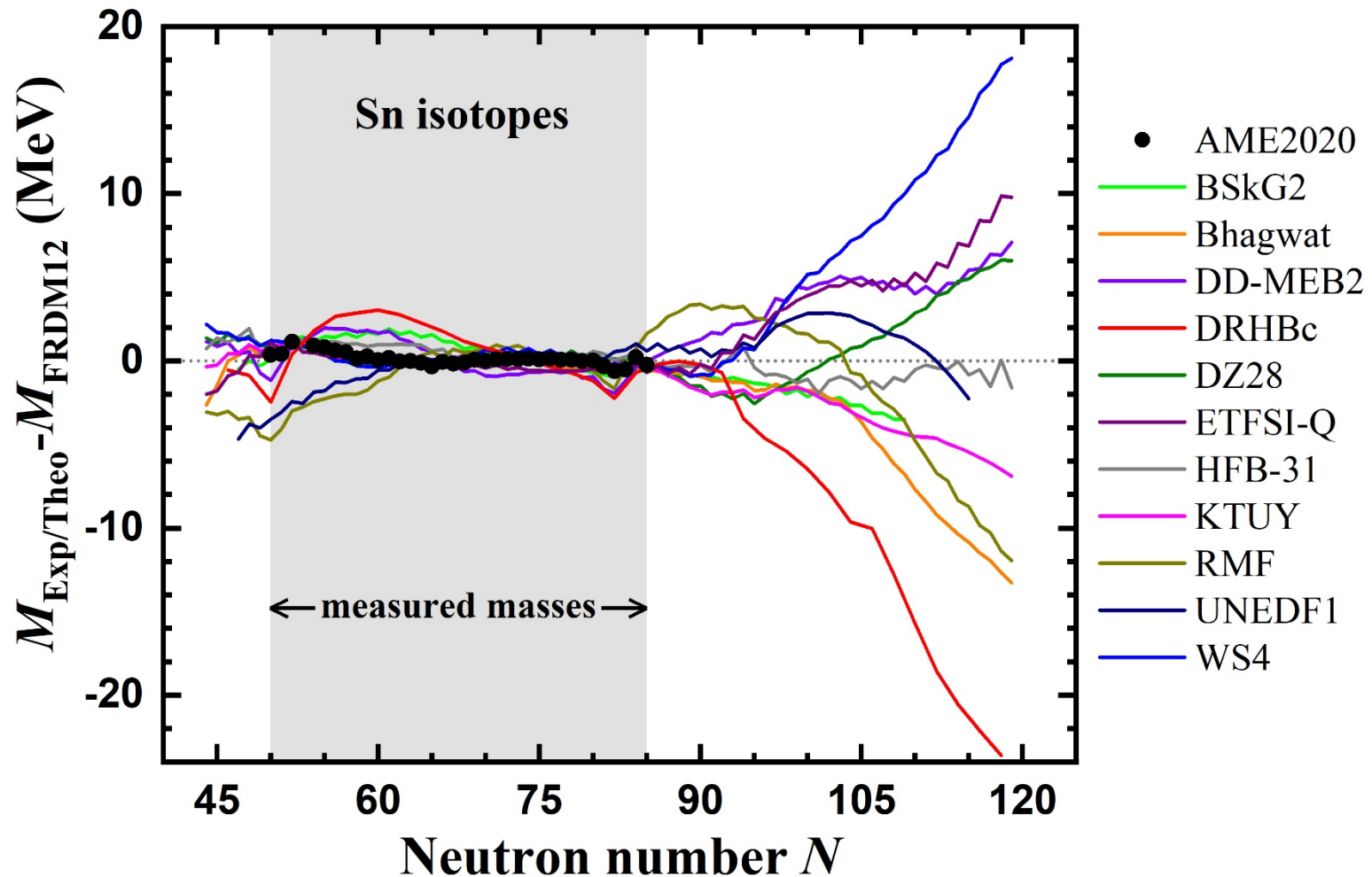
Figure: Variance in isotopic abundance patterns from FRDM1995 mass model predictions compared to the solar data (dots). [M.R. Mumpower *et al.*, PPNP 86, 86 \(2016\)](#)

Lighter and **darker** shaded bands represent Monte Carlo simulationa with mass model uncertainties of **500 keV** and **100 keV**

Accurate description of r-process abundance requires nuclear mass accuracy up to 100 keV!

Nuclear mass models

★ Different nuclear mass models giving comparable accuracy can extrapolate quite differently out to the neutron drip line.

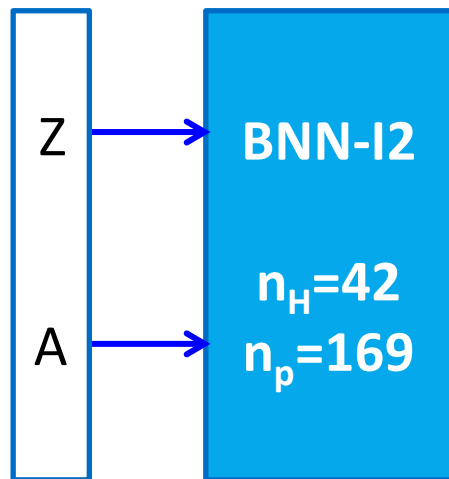


Numerical details

$$y_n(x, \omega) = a + \sum_{j=1}^H b_j \tanh \left(c_j + \sum_{i=1}^I d_{ji} x_{in} \right)$$

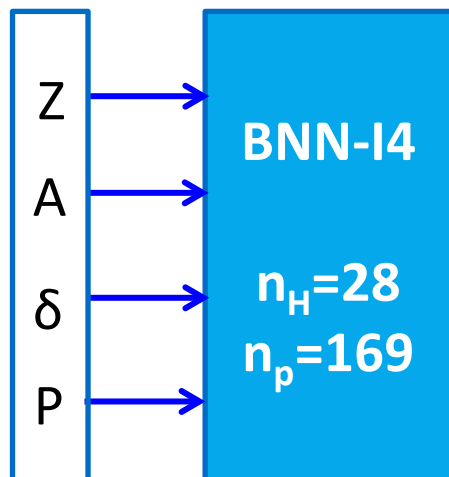
Data: 2272 nuclei from AME2016
($Z, N \geq 8$ and $\sigma^{\text{exp}} \leq 100$ keV)

M. Wang et al., CPC 41 030003



$$y_n + M_n^{\text{th}} : t_n = M_n^{\text{exp}} \Leftrightarrow y_n : t_n = M_n^{\text{exp}} - M_n^{\text{th}}$$

$$\Rightarrow M_n^{\text{rth}} = y_n + M_n^{\text{th}}$$



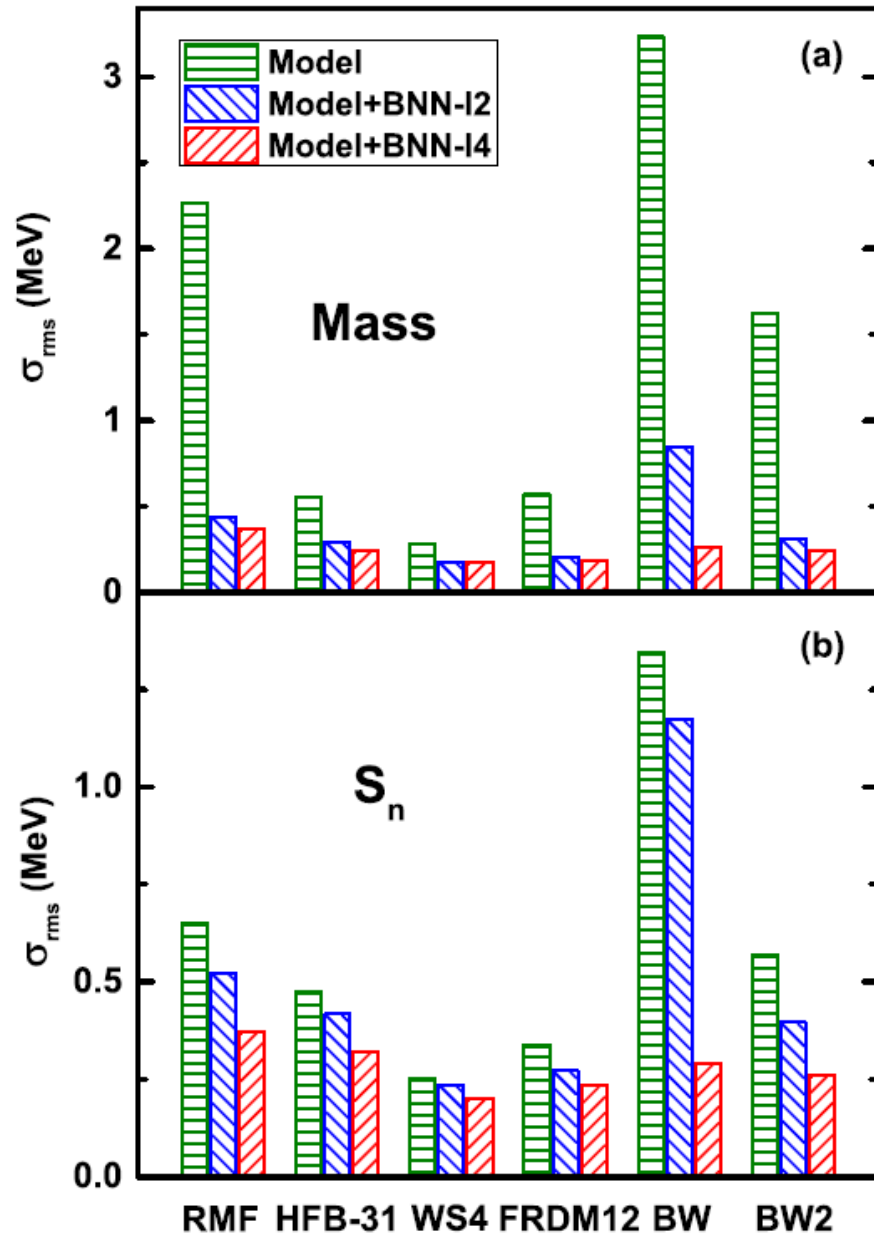
$$y_n + M_n^{\text{th}} : t_n = M_n^{\text{exp}} \Leftrightarrow y_n : t_n = M_n^{\text{exp}} - M_n^{\text{th}}$$

$$\Rightarrow M_n^{\text{rth}} = y_n + M_n^{\text{th}}$$

$$\delta = [(-1)^Z + (-1)^N] / 2, P = v_n v_p / (v_p + v_n)$$

$$v_p = \min(|Z - Z_0|), v_n = \min(|N - N_0|)$$

Rms deviations of mass and S_n

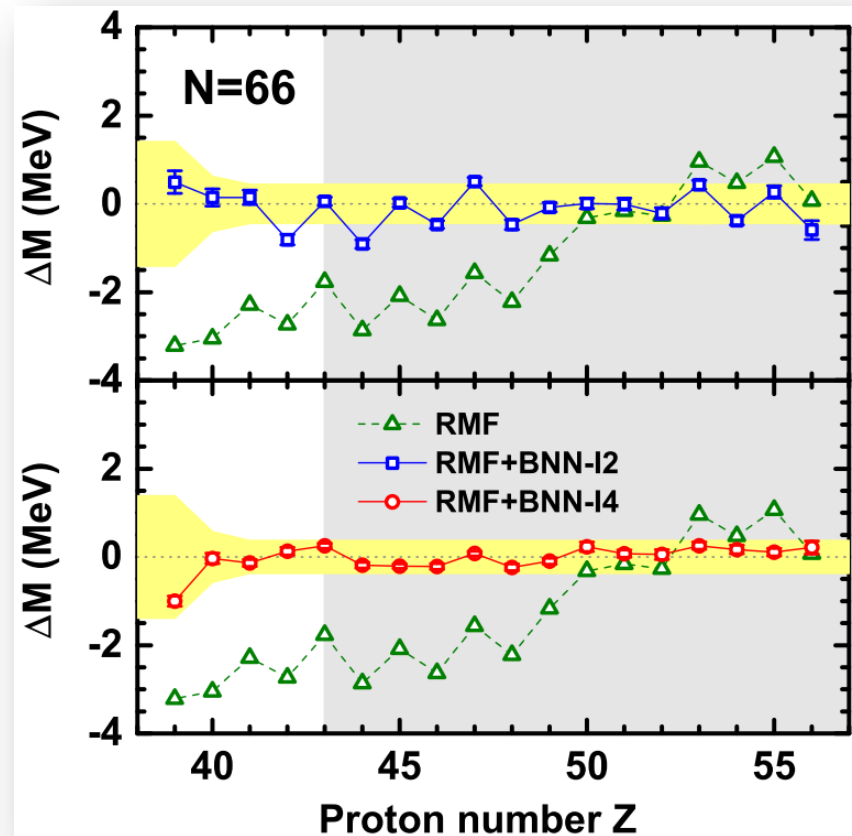
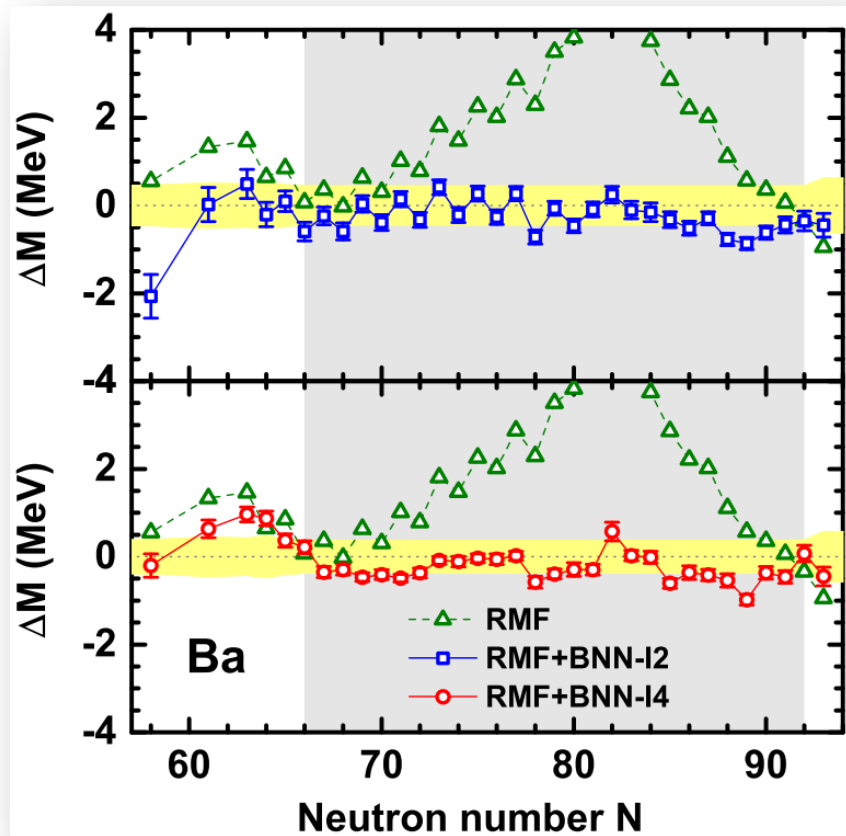


► The predictions of nuclear mass and neutron-separation energy are significantly improved with the BNN approach.

► After the improvement using the BNN approach with four inputs, the rms deviations are generally around 200 keV.

► The BNN with four inputs is more powerful than the BNN with two inputs, especially for the neutron separation energy.

Mass extrapolation



► The smooth deviations can be improved with both BNN approaches, while **the odd-even staggering can only remarkably reduced with BNN-I4 approach.**

► The BNN corrections are still reasonable if the extrapolation is not far away from the training region.

Z.M. Niu and H.Z. Liang, PLB 778, 48 (2018)

Mass predictions of RMF+BNN model

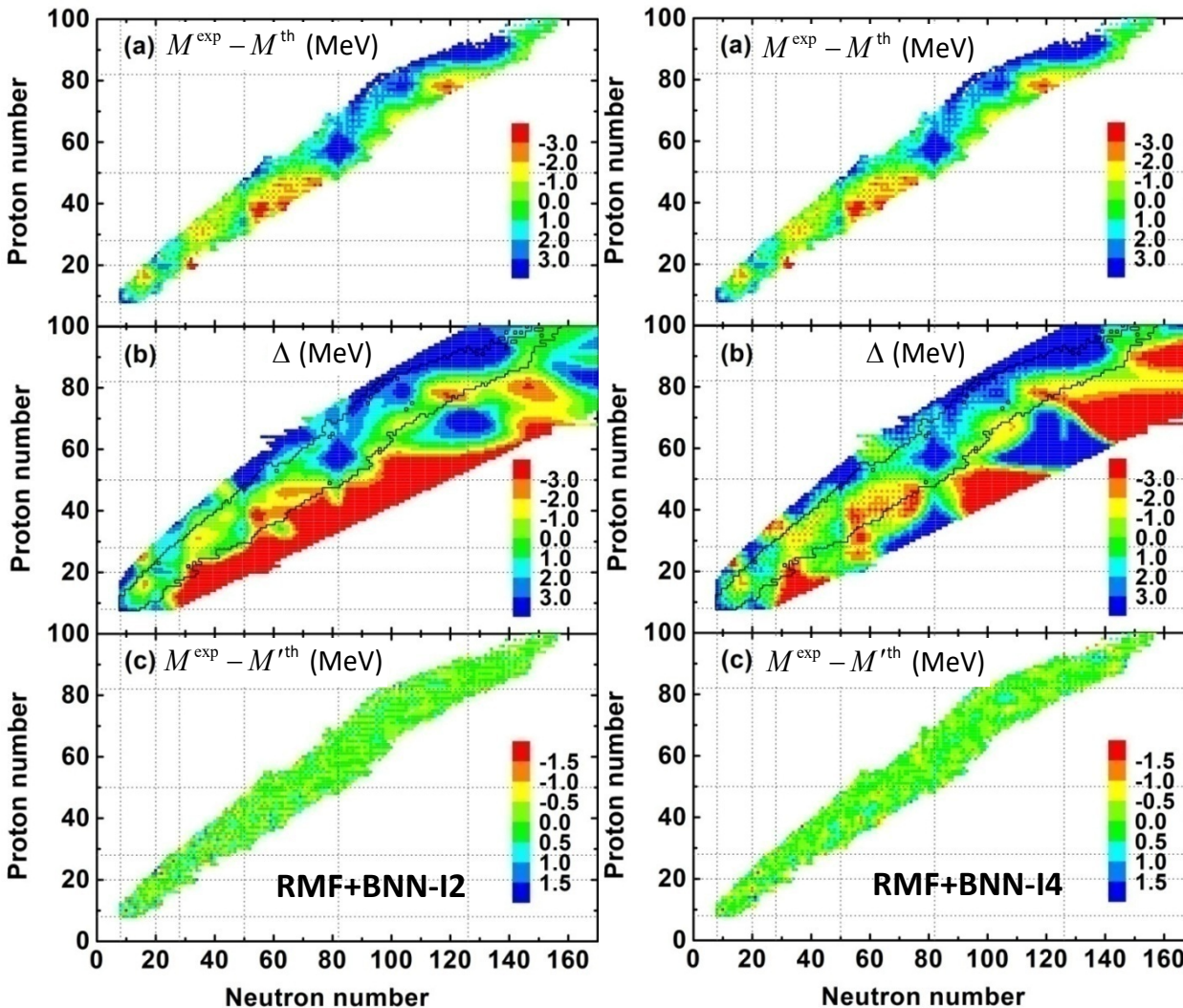


Figure: (a) Mass differences between the experimental data in AME16 and the predictions of the RMF model. (b) BNN corrections. (c) Mass differences after BNN improvement. [Niu and Liang, PLB 778, 48 \(2018\)](#)

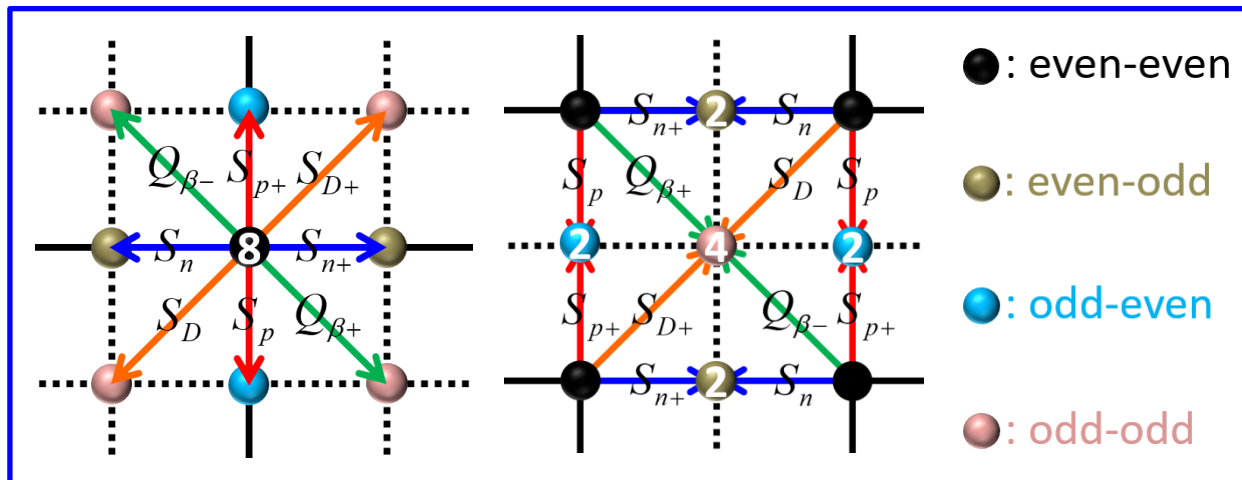
► Smooth mass deviations can be easily removed by both BNN approaches, while the odd-even staggering can be well reproduced only using BNN-I4 approach.

► The extrapolation of BNN correction show more structure information for the BNN-I4 approach, especially the shell effects around $(Z,N)=(28, 82)$ and $(50, 126)$.

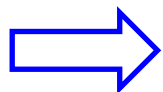
Idea of BML

$$\bullet \delta = [(-1)^Z + (-1)^N] / 2$$

$$= \begin{cases} 1, & \text{even-even} \\ 0, & \text{old-A} \\ -1, & \text{old-old} \end{cases}$$



$$\bar{M}(Z, N) = M(Z, N); \bar{M}(Z+1, N+1) = \sum_{i=1}^4 M^i(Z+1, N+1) / 4$$



$$\bar{M}(Z, N+1) = \sum_{i=1}^2 M^i(Z, N+1) / 2; \bar{M}(Z+1, N) = \sum_{i=1}^2 M^i(Z+1, N) / 2$$

$$\bullet P = v_n v_p / (v_p + v_n), \quad v_p = \min(|Z - Z_0|), \quad v_n = \min(|N - N_0|)$$

$$E_{\text{mic}}^{\text{model}} = M^{\text{model}} - E_{\text{mac}}^{\text{FRDM12}} \quad \text{or} \quad E_{\text{mic}}^{\text{model}} = M^{\text{model}} - E_{\text{mac}}^{\text{LDM}}$$

Extrapolation of BML

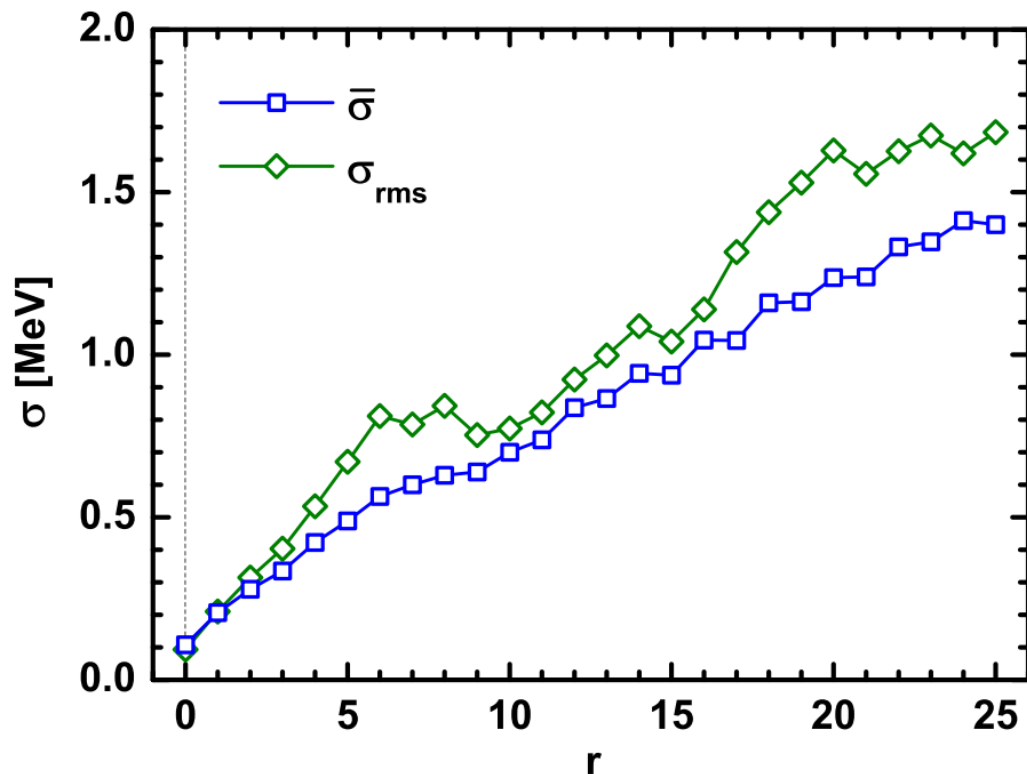
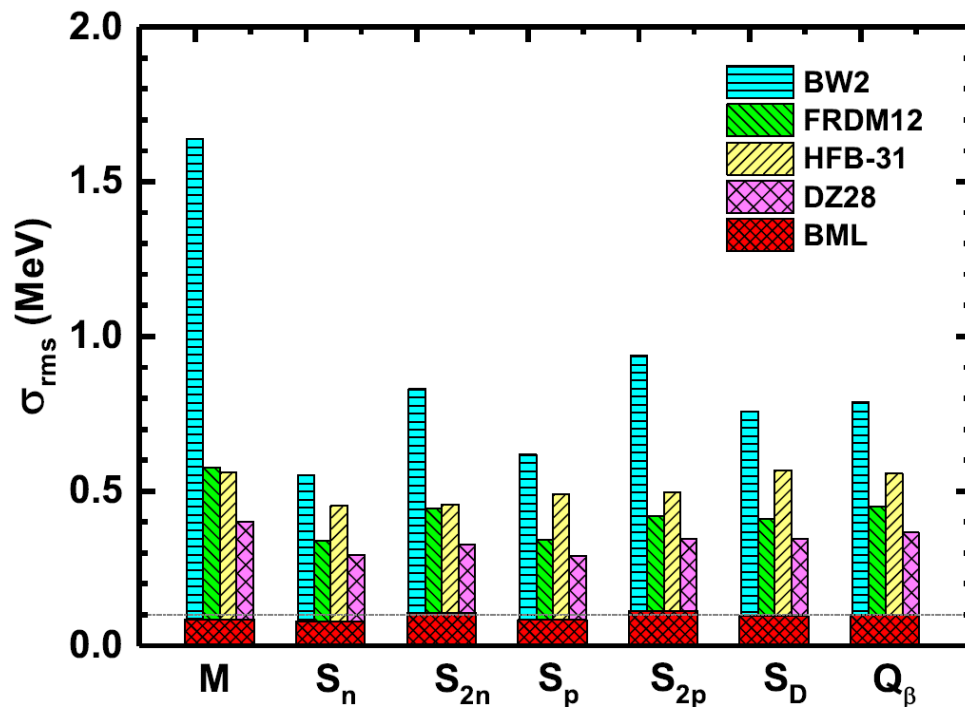


Fig: Average theoretical uncertainties $\bar{\sigma}$ (squares) of mass predictions by BML as a function of minimum distance r to the isotopes in the learning set. The rms deviations σ_{rms} of BML mass predictions with respect to the corresponding FRDM12 values are shown with diamonds.

Taking FRDM12 mass predictions as the Pseudoexperimental data:

- ★ The BML model can well reproduce the pseudoexperimental data within 100 keV for nuclei in the known region.
- ★ The rms deviation between the BML predictions and the pseudoexperimental data increases as the increase of the distance r . It is very similar to the average error of BML, which indicates **the BML model could give reasonable evaluations of the theoretical uncertainties.**

BML predictions



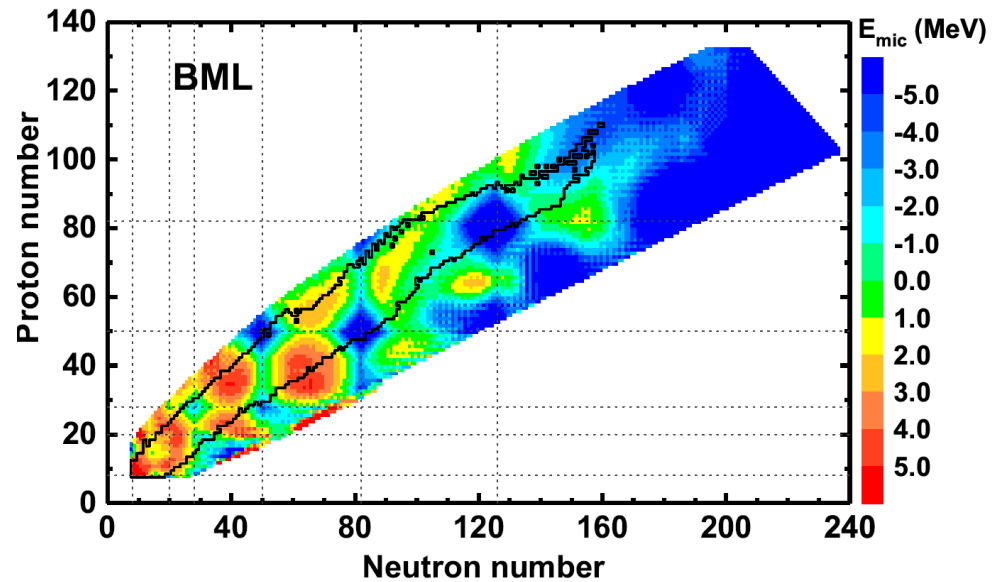
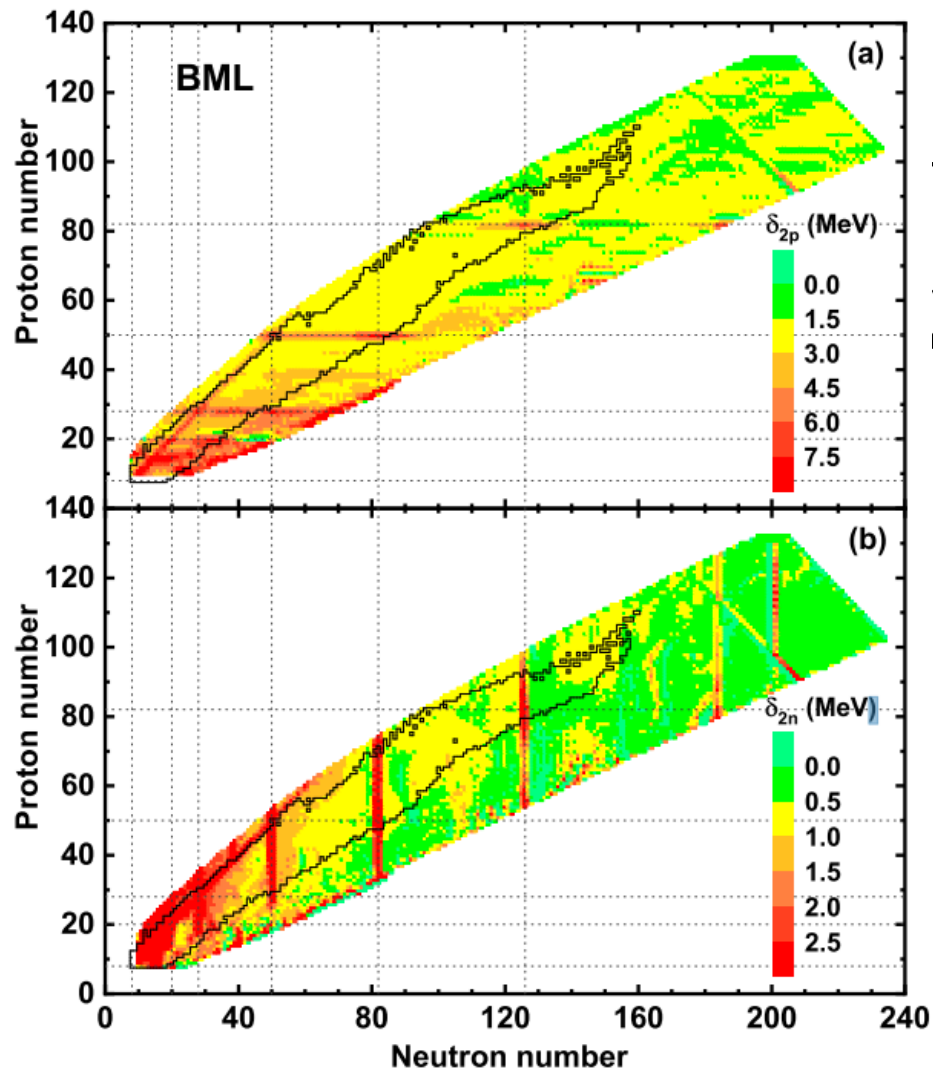
Z.M. Niu and H.Z. Liang, PRC 106, L021303 (2022)

- ★ A nuclear mass model with accuracy smaller than 100 keV is constructed.
- ★ Its accuracies to S_x and Q_x are at least about 3 times higher than other mass models.

Fig: σ_{rms} of M , S_n , S_{2n} , S_p , S_{2p} , S_D , and Q_β with respect to the experimental data for BML and other models.

Model	M	S_n	S_{2n}	S_p	S_{2p}	S_D	Q_β
FRDM12	0.576	0.340	0.442	0.341	0.420	0.411	0.450
HFB-31	0.559	0.451	0.456	0.489	0.496	0.566	0.557
WS4	0.285	0.254	0.261	0.261	0.300	0.324	0.327
BML	0.084	0.078	0.105	0.083	0.111	0.096	0.099

BML predictions



- ★ The shell structure in the known region is well reproduced.
- ★ Several important features in the unknown region are predicted, such as the magic numbers around **N=40** and **N=184**, the **robustness of N=82 shell**, the **quenching of N = 126 shell**.

Fig: δ_{2p} , δ_{2n} , and E_{mic} of BML.

Z.M. Niu and H.Z. Liang, PRC 106, L021303 (2022)

Comparison of mass predictions

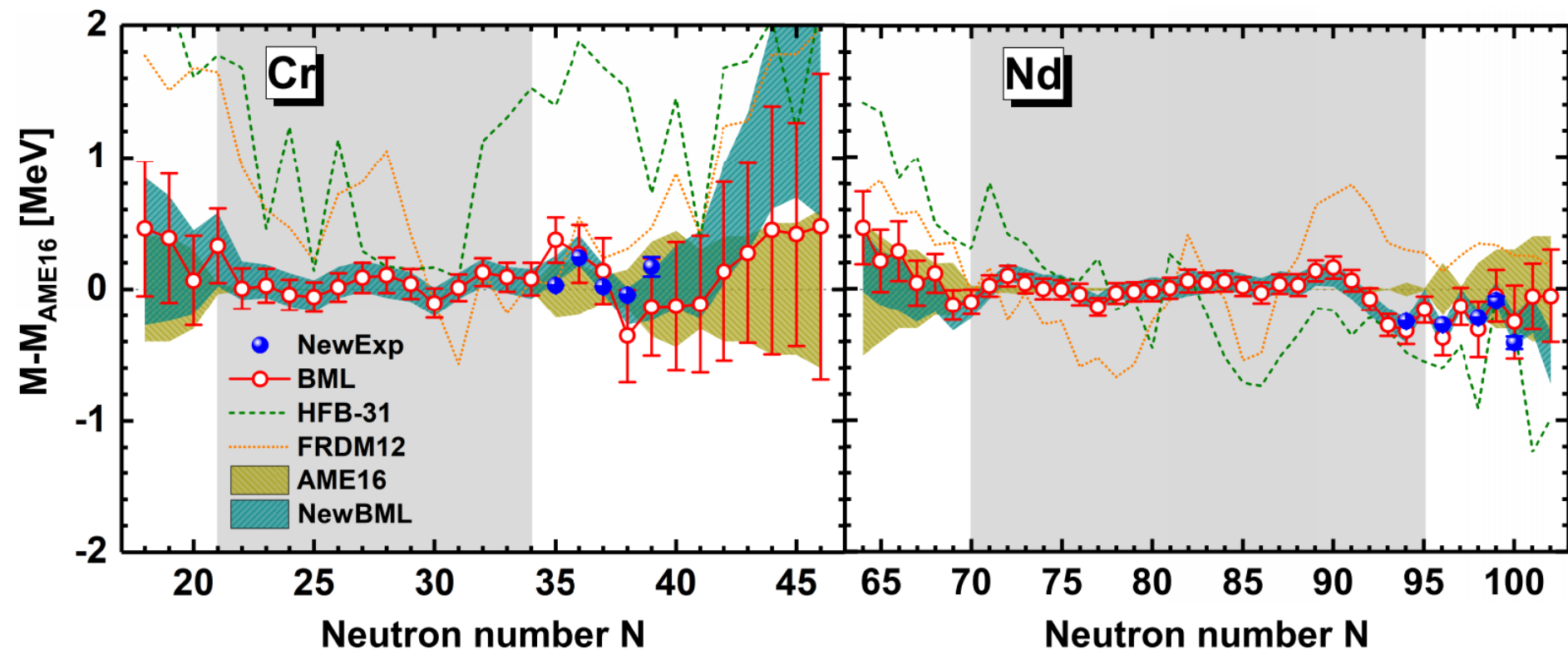


Fig: Mass differences between M_{exp} in AME16 and M_{th} from BML, NewBML, HFB-31, and FRDM12 models. The new experimental data from [Mougeot2018PRL; Orford2018PRL] are denoted by spheres.

★ The BML model well reproduces new experimental masses within errors. If new data are included in Lset, the errors near new data reduce to about half of original values.

FNN VS BNN

模型	Mass rms (MeV)		
	Adam	RMSprop	SGD
BW	3.068		
FNN-I2a ($Z, N \rightarrow M_{\text{exp}}$)	39.745	33.192	41.226
FNN-I2b ($Z, N \rightarrow M_{\text{exp}} - M_{\text{BW}}$)	1.186	2.247	2.812
FNN-I4 ($Z, N, \delta, P \rightarrow M_{\text{exp}} - M_{\text{BW}}$)	0.685	1.066	2.080
BNN -I4 ($Z, N, \delta, P \rightarrow M_{\text{exp}} - M_{\text{BW}}$)	0.333		

4 input: $n_H=28$

2 output: $n_H=42$

Number of parameter: 169

Reasonable design of the input and output layer can significantly improve the prediction performance of neural networks!

D.C. Tian, S.W. Chen, and Z.M. Niu*, SSPMA 52, 252007 (2022)

FNN VS BNN

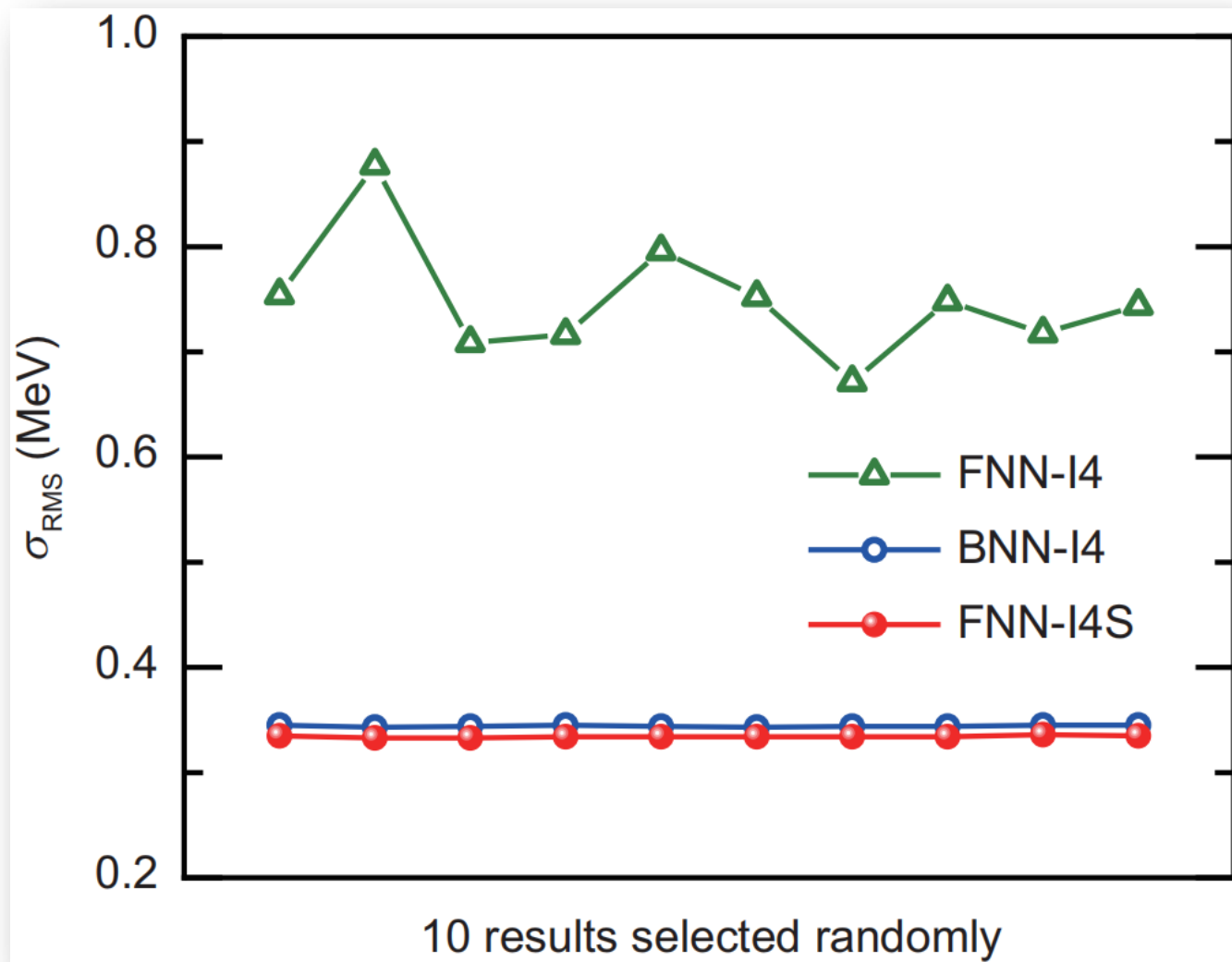


Fig: RMS deviations of nuclear mass between experimental data and FNN-I4, BNN-I4, FNN-I4S results.

D.C. Tian, S.W. Chen, and Z.M. Niu*, SSPMA 52, 252007 (2022)

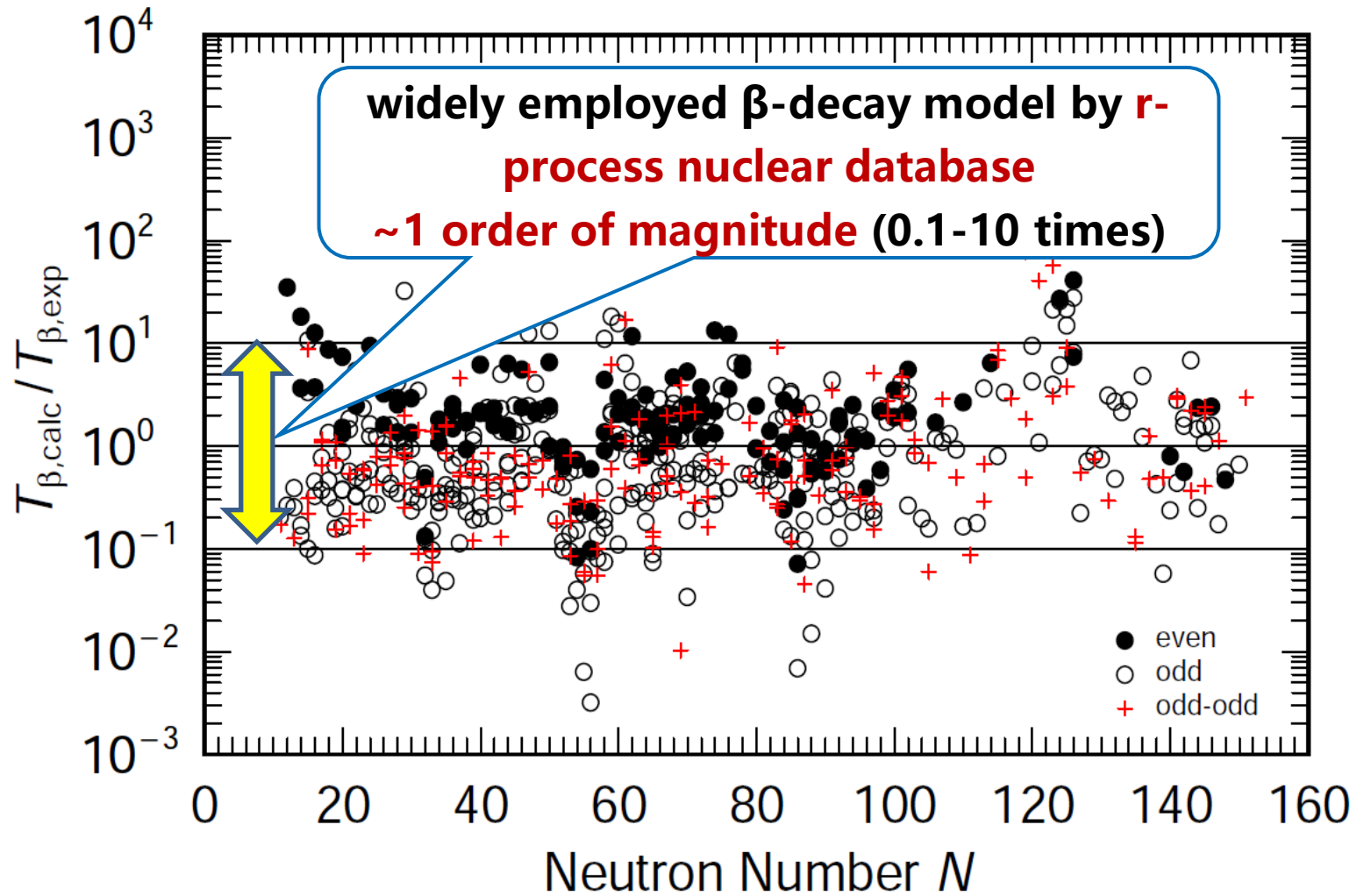
Outline

- ① Introduction
- ② Bayesian neural network approach
 - ★ Bayesian approach
 - ★ Neural network
 - ★ Bayesian neural network
- ③ Results and discussion
 - ★ Nuclear masses
 - ★ Nuclear β -decay half-lives
 - ★ Some other results
- ④ Summary and perspectives

Nuclear models for β -decay half-lives

- Phenomenological formula [Pfeiffer2000Report](#), [Zhang2006PRC](#), [2007JPG](#), [Zhou2017SCPMA](#), [Xia2024APS](#)
- Gross theory [Takahashi1969PTP](#), [Tachibana1990PTP](#), [Nakata1997NPA](#), [Koura2017PRC](#), [Fang2022PRC](#)
- Shell model [Pinedo1999PRL](#), [Caurier2002PRC](#), [Langanke2003RMP](#), [Zhi2013PRC](#)
- Quasiparticle random phase approximation (QRPA)
 - Nilsson BCS+QRPA: [Staudt1990ADNDT](#), [Hirsch1993ADNDT](#), [Nabi1999ADNDT](#)
 - FRDM+QRPA: [Möller1997,2018ADNDT](#), [Möller2003PRC](#)
 - Woods-Saxon+QRPA: [Ni2012JPG](#)
 - SHF BCS+QRPA: [Sarriguren2005](#), [2010](#), [2011PRC](#)
 - DF(Fayans)+CQRPA: [Borzov1996ZPA](#), [Borzov2003,2005PRC](#), [Borzov2008NPA](#)
 - ETFSI(Skyrme)+CQRPA: [Borzov1997NPA](#), [Borzov2000PRC](#)
 - SHF(BCS)+(Q)RPA: [Bai2010PRL](#), [Minato2013PRL](#), [Minato2022PRC](#)
 - SHFB+QRPA/FAM/QPVC: [Engel1999PRC](#), [NiuYF2015PRL](#), [2018PLB](#), FAM: [Ney2020PRC](#)
 - RHB+QRPA: [Nikšić2005PRC](#), [Marketin2007,2016PRC](#), [Wang2016JPG](#), [NiuZM2013PRC\(R\)](#)
 - RHFB+QRPA: [NiuZM2013PLB](#)
- Machine learning [Costiris2009PRC](#), [Li2022SSPMA](#), [Niu2019PRC](#), [Li2024JPG](#)

Nuclear models for β -decay half-lives



Möller *et al.*, ADNDT 125, 1 (2019)

Influence of half-lives

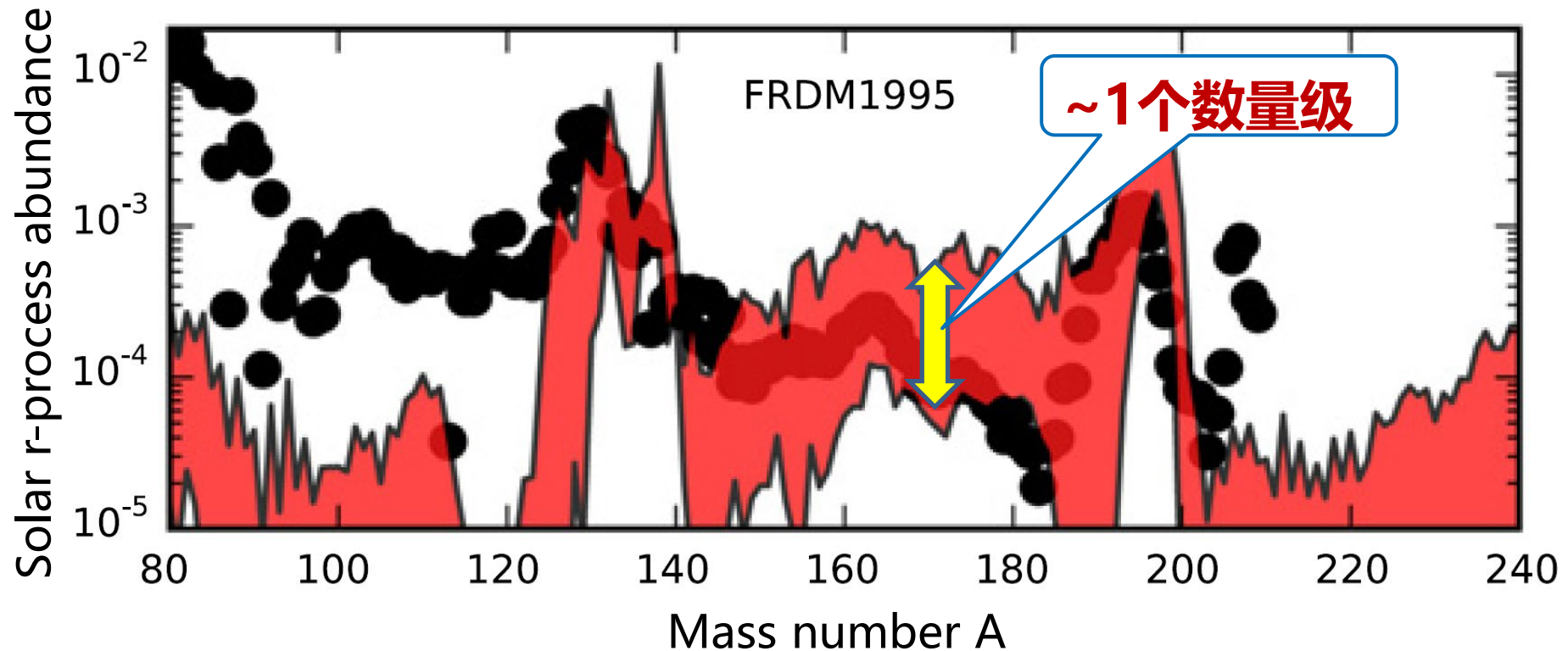


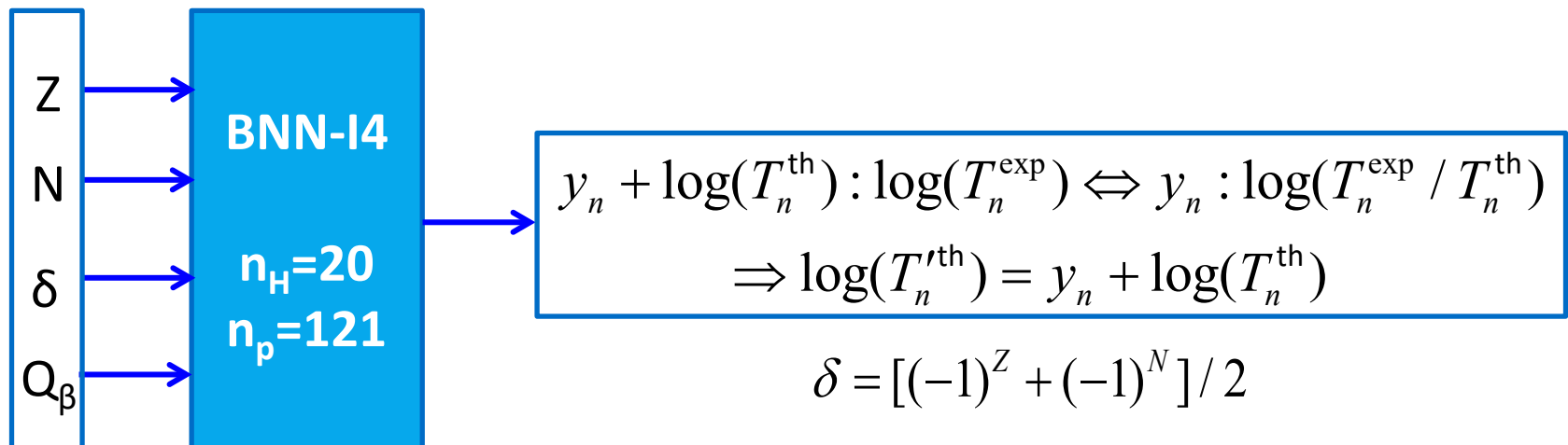
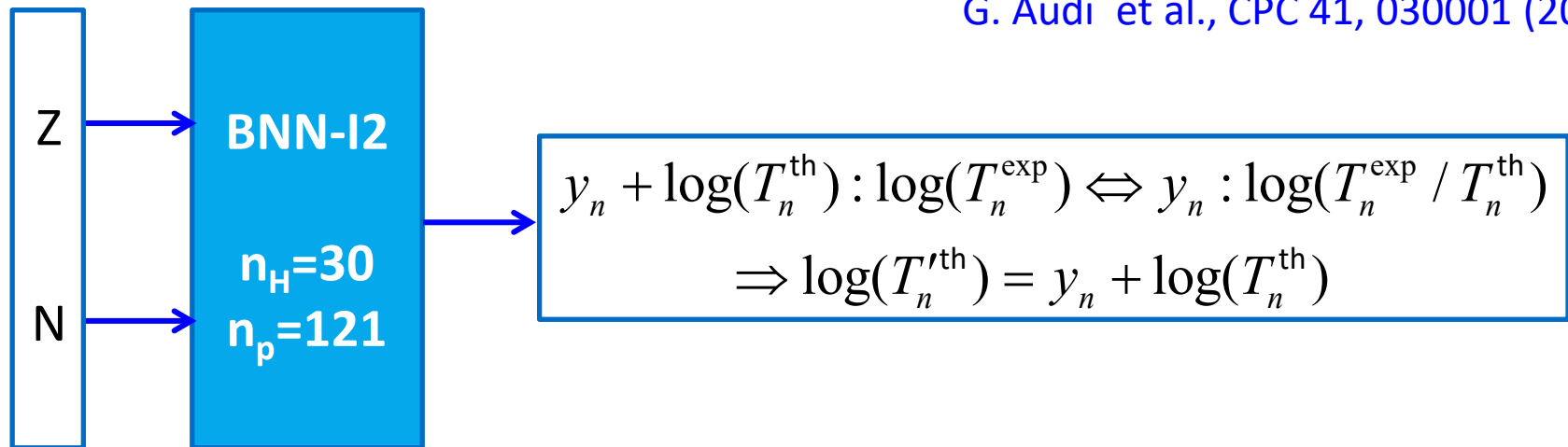
Figure: Variance in isotopic abundance patterns from FRDM1995 mass model predictions compared to the solar data (dots). [M.R. Mumpower *et al.*, PNP 86, 86 \(2016\)](#)

Numerical details

$$y_n(x, \omega) = a + \sum_{j=1}^H b_j \tanh \left(c_j + \sum_{i=1}^I d_{ji} x_{in} \right)$$

Data: 1009 nuclei in NUBASE2016 ($Z, N \geq 8$ and β^- -decay fraction=100%)

G. Audi et al., CPC 41, 030001 (2017)



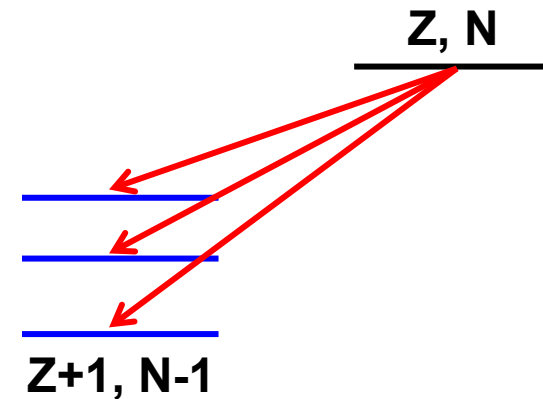
Nuclear β -decay half-lives

- The nuclear β -decay half-life in allowed GT approximation reads as follows:

$$T_{1/2} = \frac{\ln 2}{\lambda_\beta} = \frac{D}{g_A^2 \sum_m B_{GT}(E_m) f(Z, A, E_m)}$$

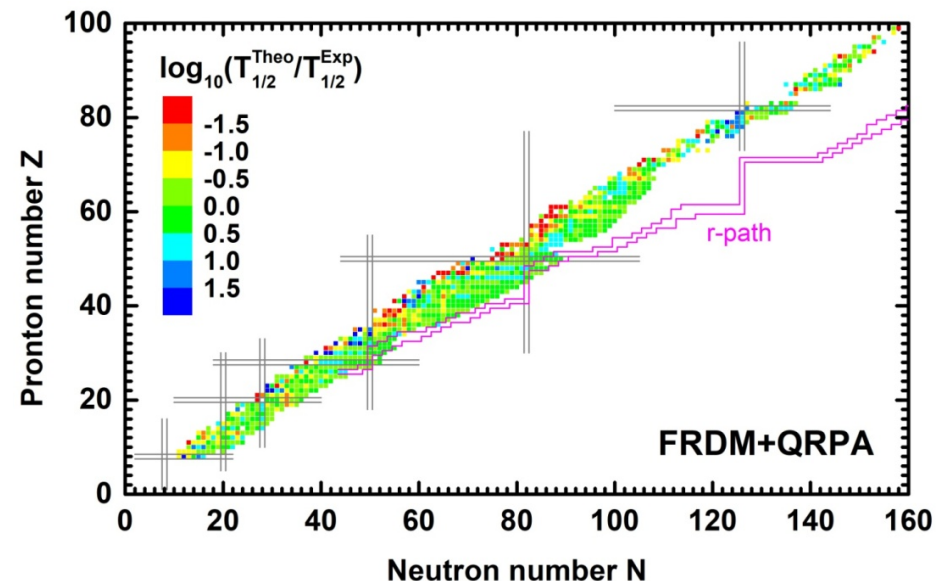
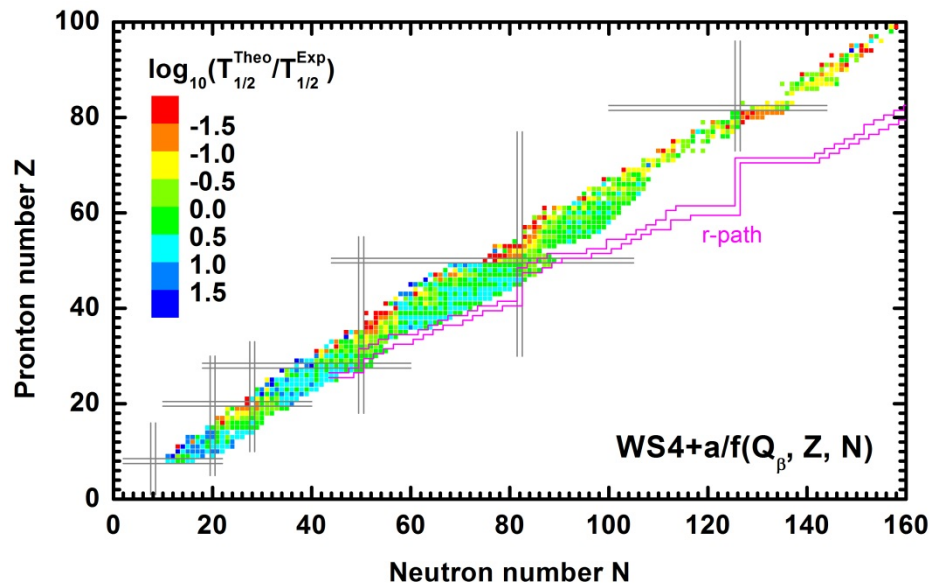
$$\rightarrow T_{1/2} = a / f(Z, A, E_m = Q_\beta - c(\delta - 1) / \sqrt{A})$$

where $D = \frac{\hbar^7 2\pi^3 \ln 2}{g^2 m_e^5 c^4} = 6163.4 \text{ s}$, $g_A = 1$, $B_{GT}(E_m)$ is the transition probability, and E_m is the maximum value of β -decay energy.

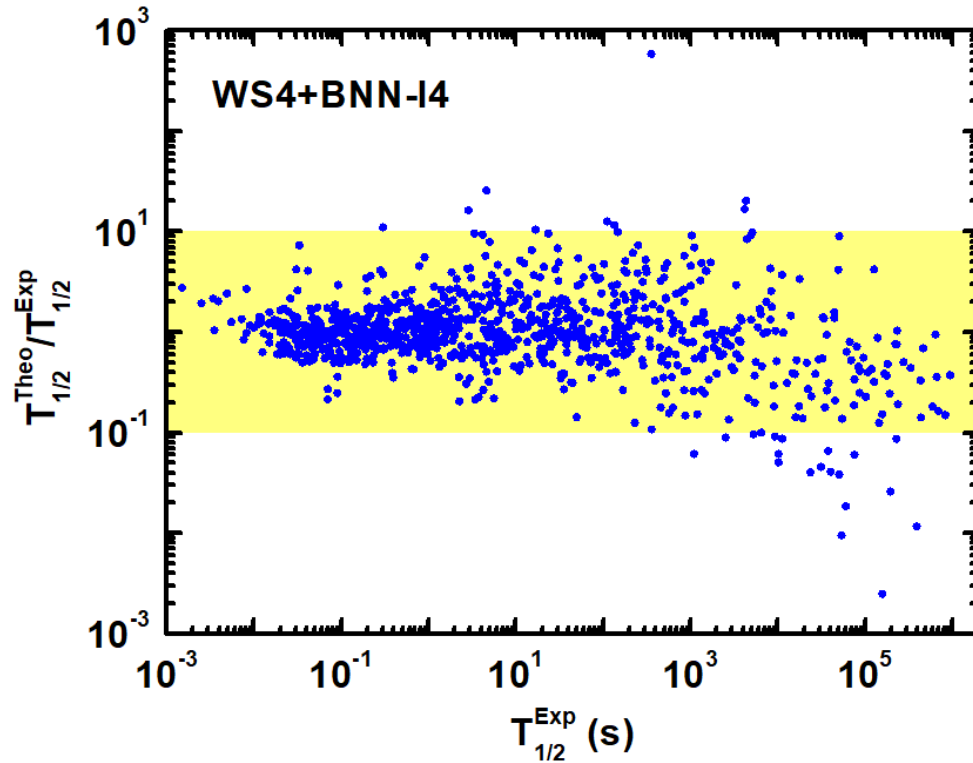


The phase volume is

$$f(Z, A, E_m) = \frac{1}{m_e^5} \int_{m_e}^{E_m} p_e E_e (E_m - E_e)^2 F(Z, A, E_m) dE_e,$$



Half-lives of BNN approaches

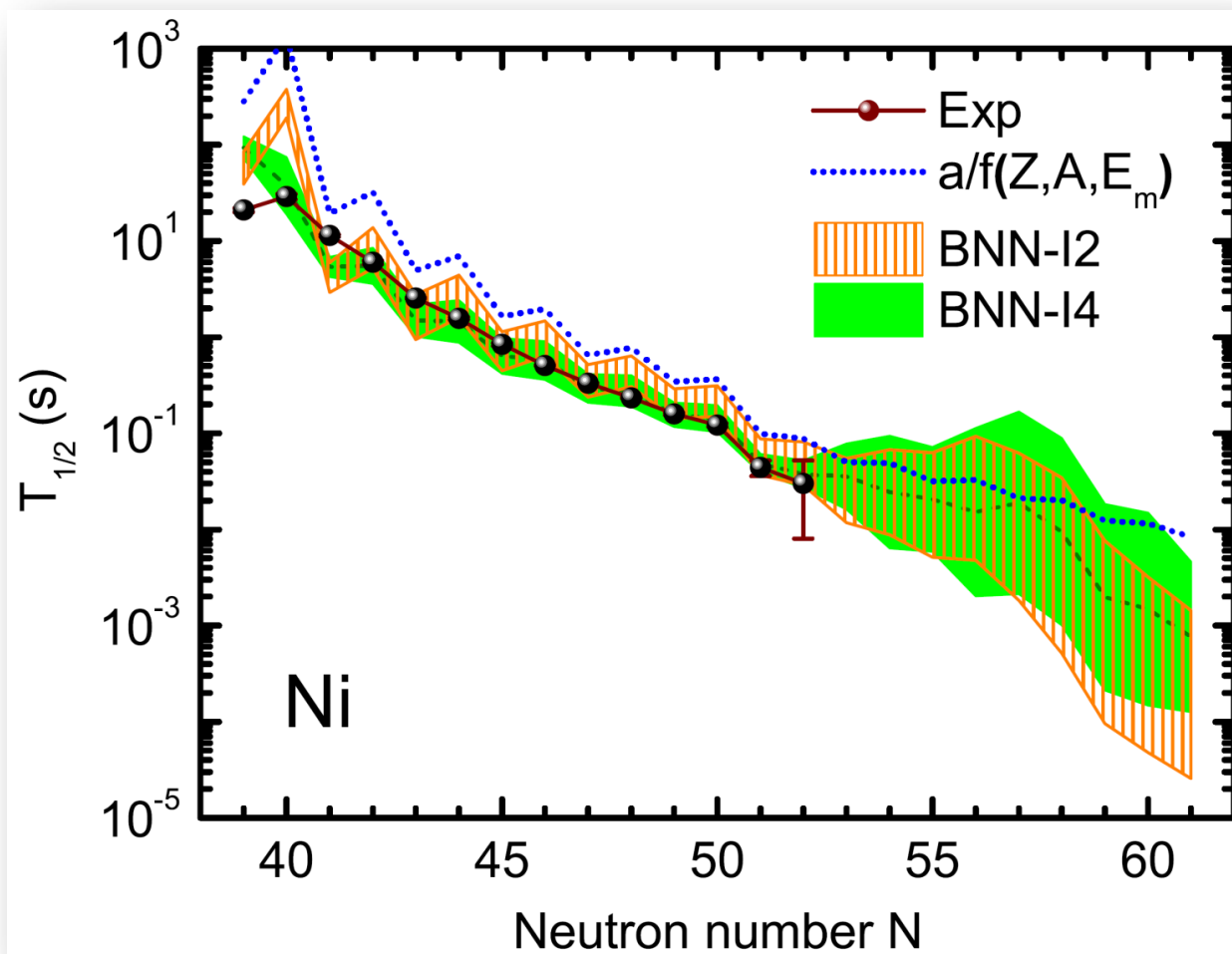


- The WS4+BNN-I4 approach usually better reproduce the half-lives of short-lived nuclei.
- The WS+BNN-I4 approach gives the best results, which can describe nuclear half-lives around $10^{0.2}=1.6$ times of experimental data for nuclei with half-lives shorter than 1 s.

	$T_{1/2} < 10^6$ s	$T_{1/2} < 10^3$ s	$T_{1/2} < 1$ s
WS4+a/f	0.8060	0.6302	0.5631
WS4+BNN-I2	0.4766	0.3542	0.2383
WS4+BNN-I4	0.3999	0.3146	0.2036
FRDM+QRPA	0.8190	0.5969	0.3906
RHB+QRPA	1.8844	1.6196	0.4631

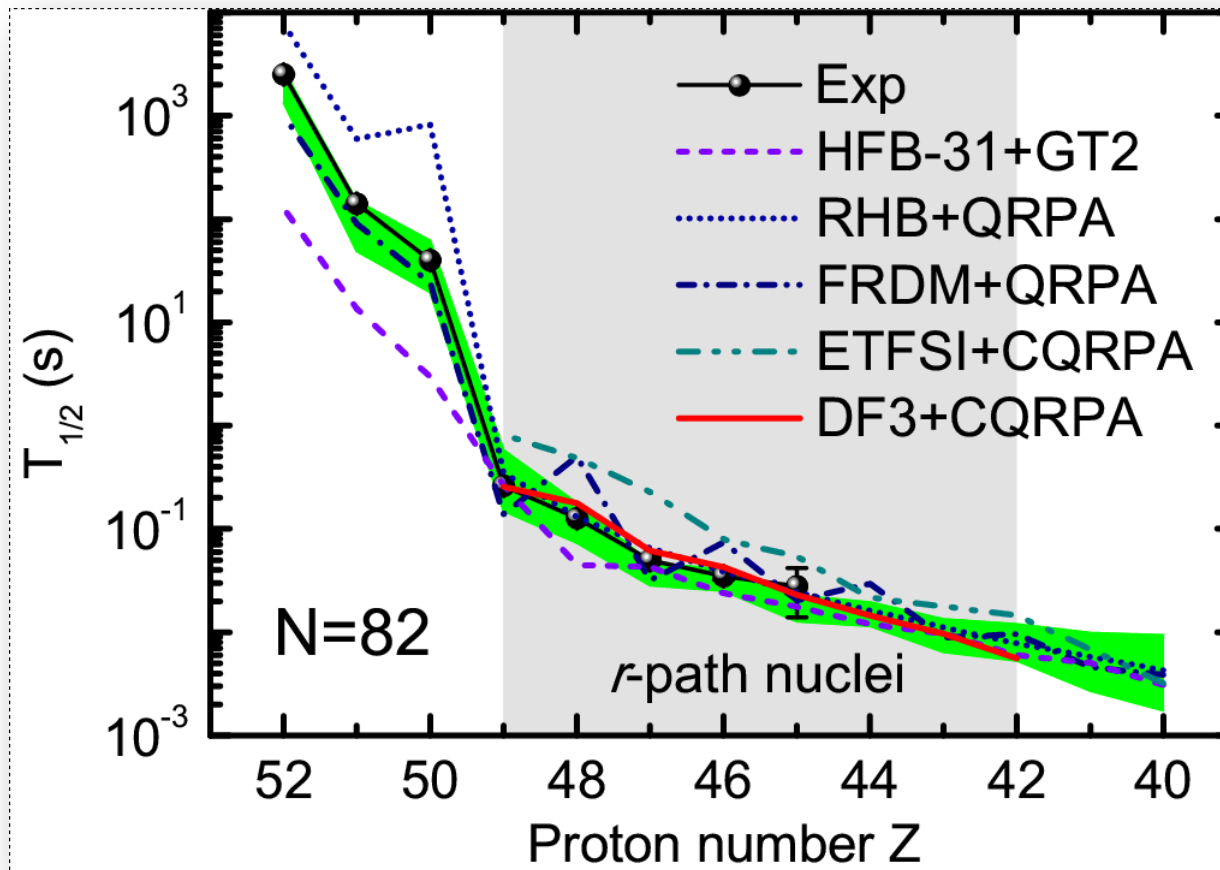
$$\sigma_{\text{rms}} = \sqrt{\frac{\sum_{i=1}^n \left[\log_{10} \left(T_{1/2}^{\text{Exp}} / T_{1/2}^{\text{Theo}} \right) \right]^2}{n}}$$

Half-lives with BNN approaches



- $T_{1/2}=a/f(Q_\beta, Z, N)$ generally overestimates the odd-even staggering in half-lives.
- BNN-I2 approach cannot easily remove odd-even staggering in half-lives, while BNN-I4 approach well reproduce the experimental data. [Z. M. Niu et al., PRC 99, 064307 \(2019\)](#)

Predictions of nuclear half-lives



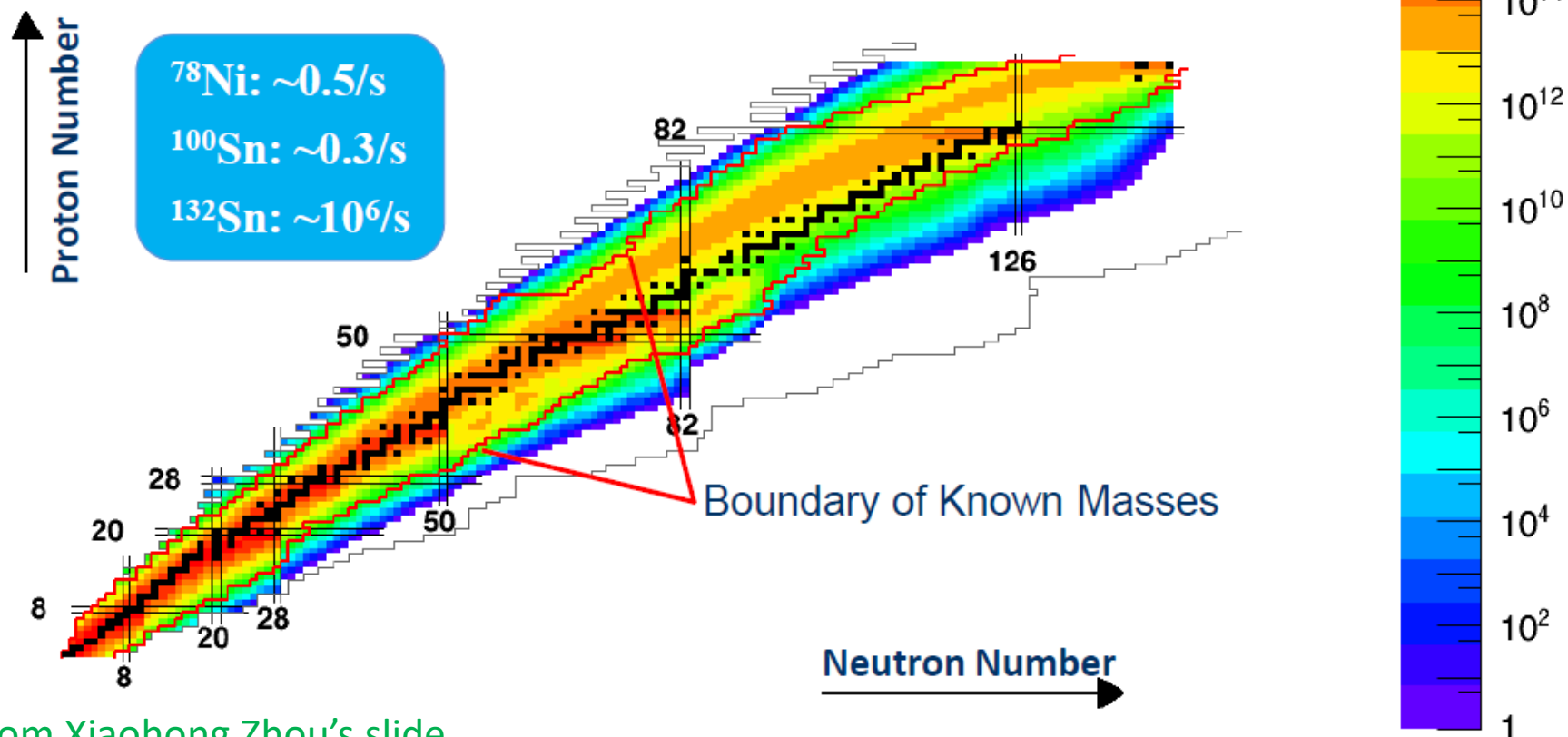
- The results of WS4+BNN-I4 approach are in good agreement with the experimental data, even completely agree with the experimental data within uncertainties for short-lived nuclei.
 - When extrapolate from known region, the results of other models generally agree with WS4+BNN-I4 predictions within uncertainties.
- Z. M. Niu et al., PRC 99, 064307 (2019)



Capability of Producing Nuclides

Nuclides Available (Production Yield) at HIAF

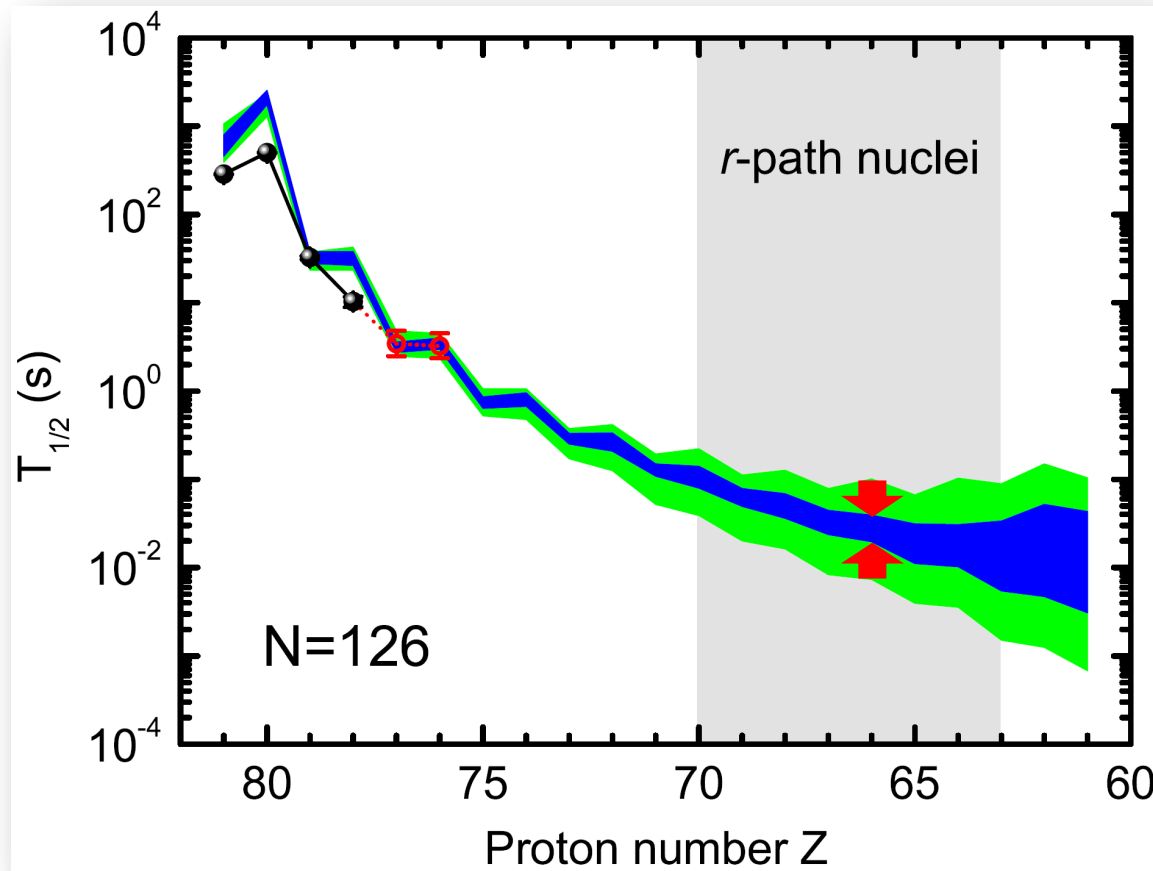
One of the world's most powerful facilities to explore the nuclear chart



From Xiaohong Zhou's slide

Prolific sources of nuclides far away from the stability line will be provided using projectile fragmentation, in-flight fission, multi-nucleon transfer, and fusion reactions. The limits shown are the production rate of one nuclide per day, which enable the “discovery experiments”

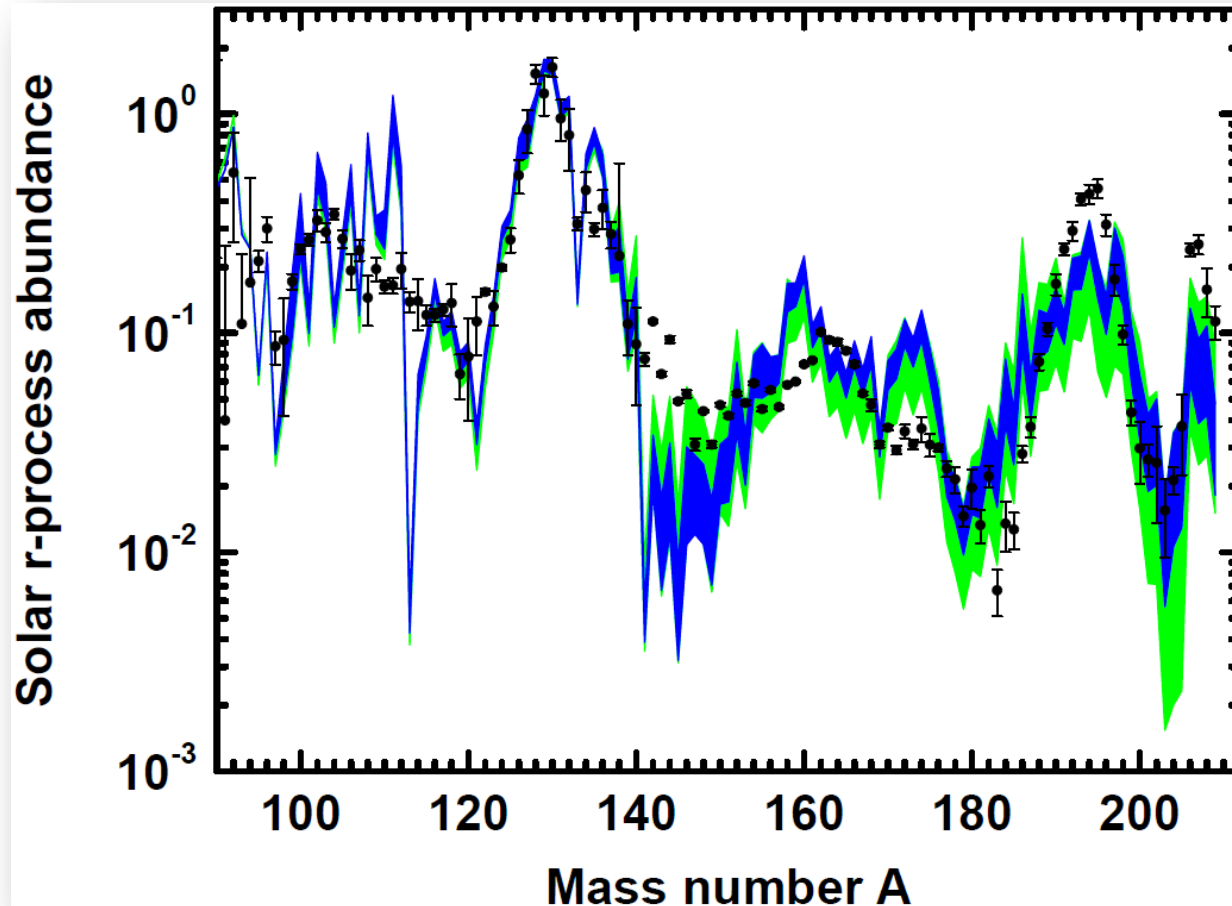
Predictions of nuclear half-lives



- If we can further measure three more β -decay half-lives for each isotopes
 - ✓ uncertainties of BNN predictions are similar in the training region
 - ✓ they will be decreased about 3 times when extrapolate to the region far from known region.

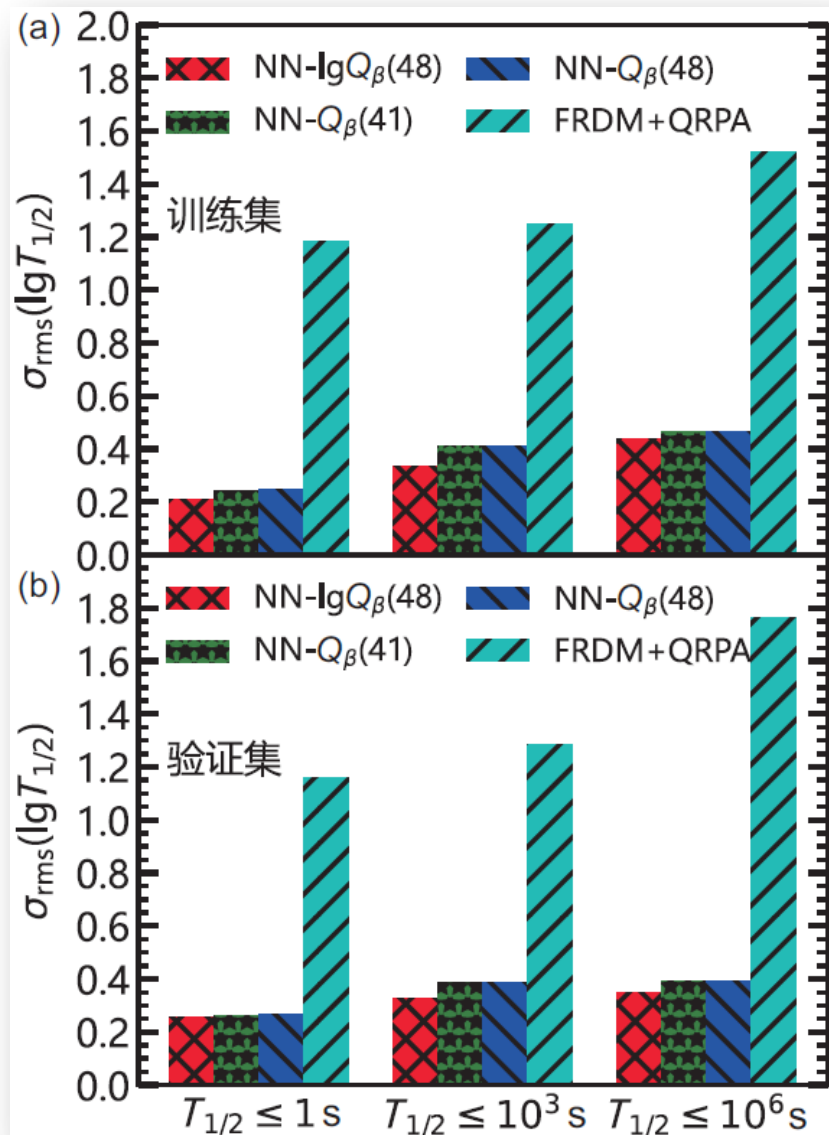
Z. M. Niu et al., PRC 99, 064307 (2019)

Predictions of r-process abundances



- Uncertainties from β -decay half-lives lead to large uncertainties for the r -process abundances of elements with $A > \sim 140$, which can be remarkably reduced if we can further measure three more β -decay half-lives. [Z. M. Niu et al., PRC 99, 064307 \(2019\)](#)

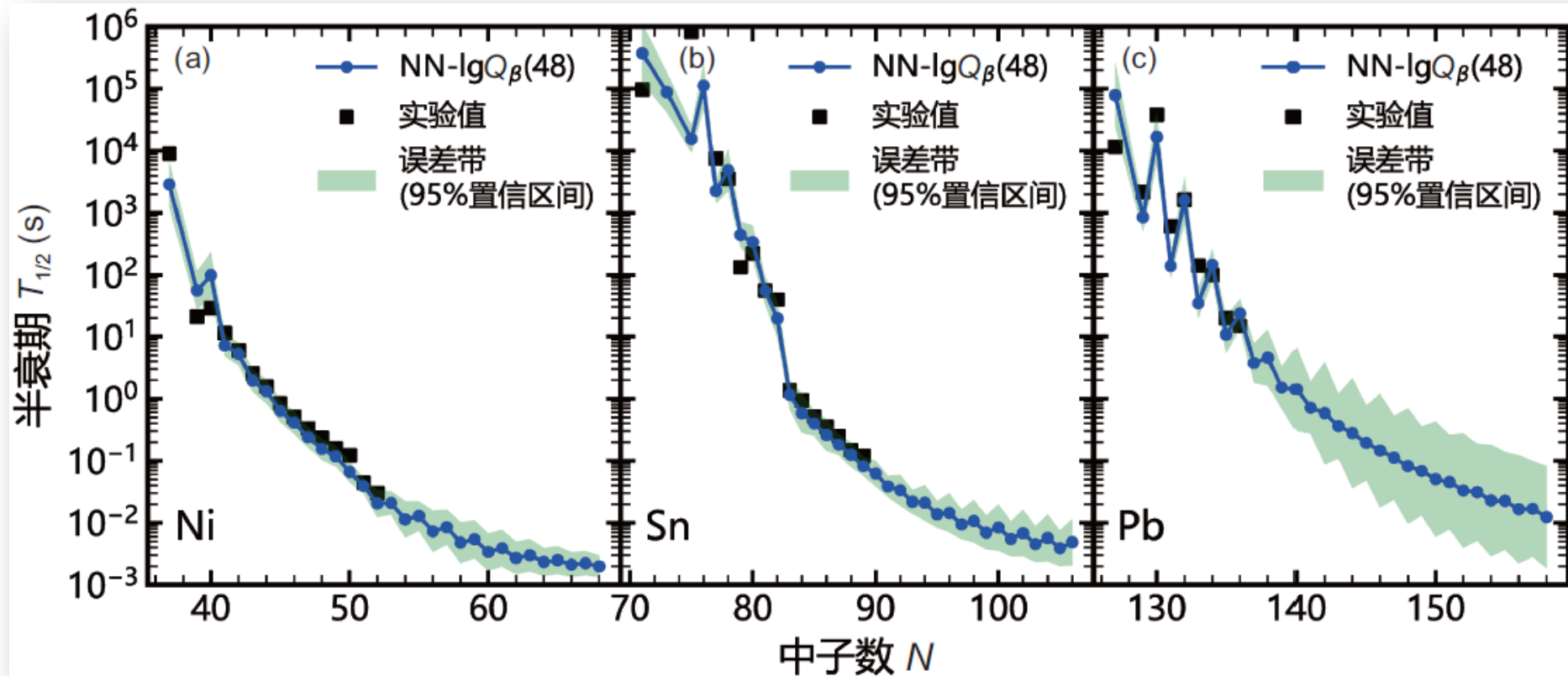
Predictions of nuclear half-lives



- Compared with FRDM+QRPA, the accuracy of the description of half-lives is improved by about 2.6 times.

- For the nuclei with half-lives of $< 1 \text{ s}$, the rms value reaches $10^{0.22} = 1.66$.

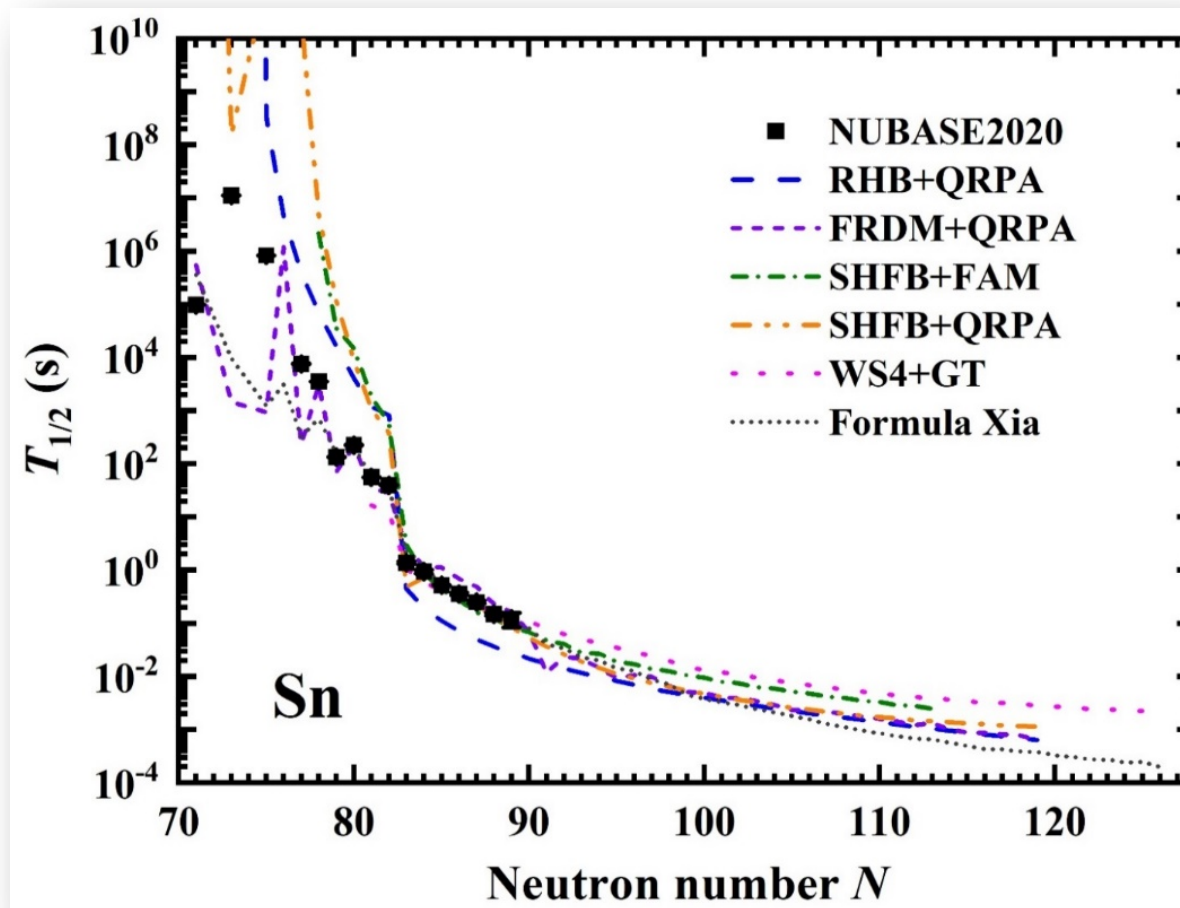
Predictions of nuclear half-lives



- The NN can reproduce the experimental values well, including the jumps at the magic numbers for Ni and Sn isotopes and the odd-even oscillations of Pb isotopes.
- The error bands becomes larger and larger as move away from the known region.

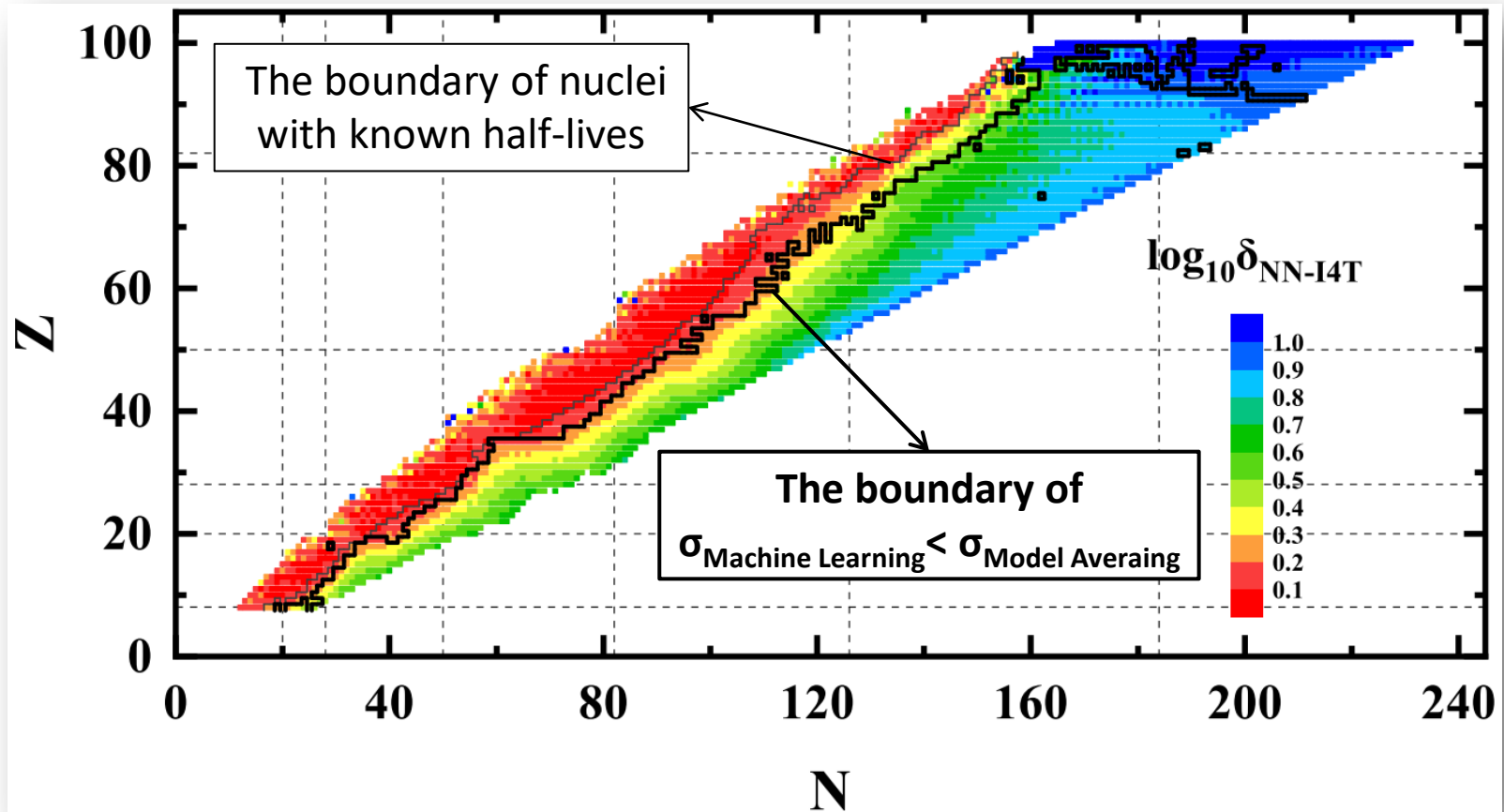
P. Li, J.H. Bai, Z.M. Niu, and Y.F. Niu*, SSPMA 52, 252006 (2022)

Nuclear models for β -decay half-lives



- Different models generally better reproduce β -decay half-lives of short-lived nuclei. The deviations between different predictions slowly increase to an order of magnitude even out to the drip line.

β -decay half-lives from machine learning



- The half-life uncertainties of the neural network are still smaller than those of the model averaging method within about 5–10 steps for nuclei with $35 \lesssim Z \lesssim 90$.

W. F. Li *et al.*, *J. Phys. G* 51, 015103 (2024)

Outline

- ① Introduction
- ② Bayesian neural network approach
 - ★ Bayesian approach
 - ★ Neural network
 - ★ Bayesian neural network
- ③ Results and discussion
 - ★ Nuclear masses
 - ★ Nuclear β -decay half-lives
 - ★ Some other results
- ④ Summary and perspectives

Outline

① Introduction

② Bayesian neural network approach

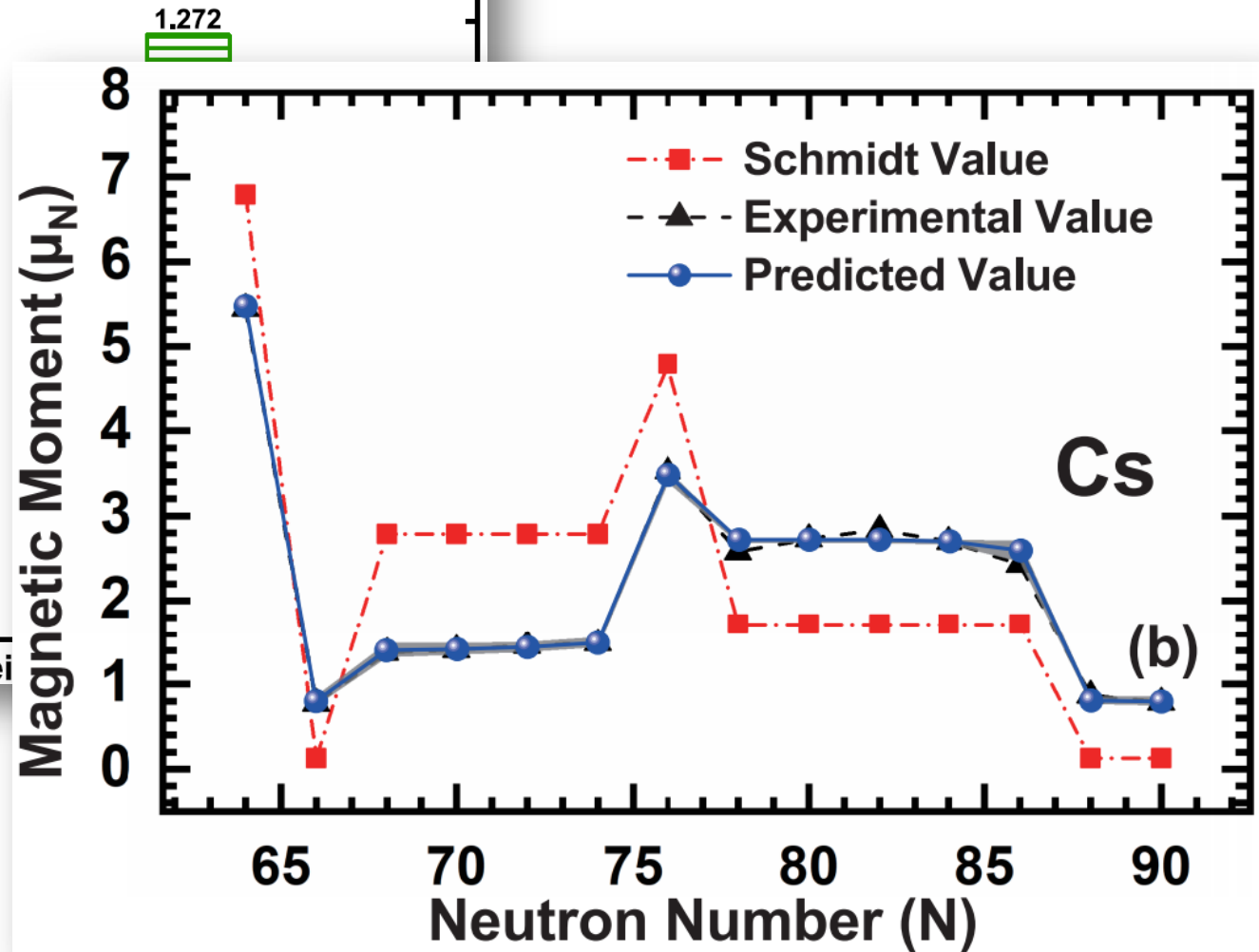
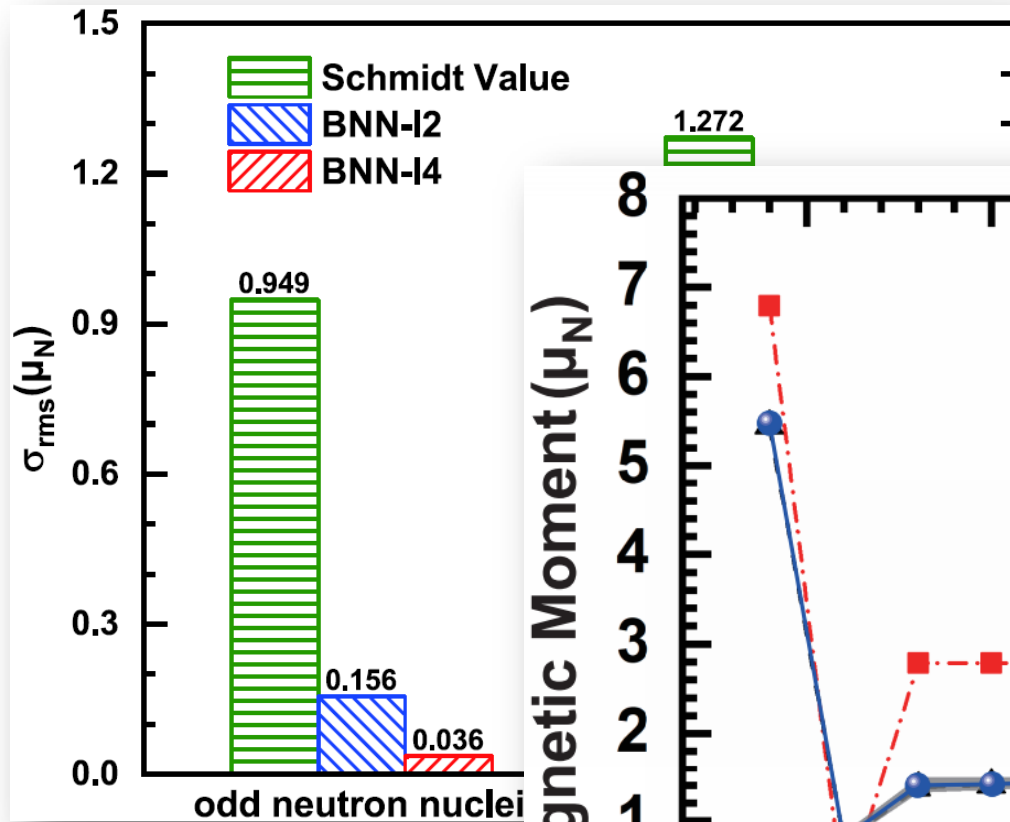
③ Results and discussion

★ Some other results

- ✓ Other properties with enough data or theoretical results
- ✓ Less controlled model parameters
- ✓ Construct theory guaranteed by physics, e.g. DFT
- ✓ New applications, e.g. data selection or evaluation

④ Summary and perspectives

Magnetic moments: BNN



Z.L. Yuan, D.C. Tian, J. Li*, and Z.M. Niu*, CPC 45, 124107 (2021)

Low-lying excitation spectra: BNN

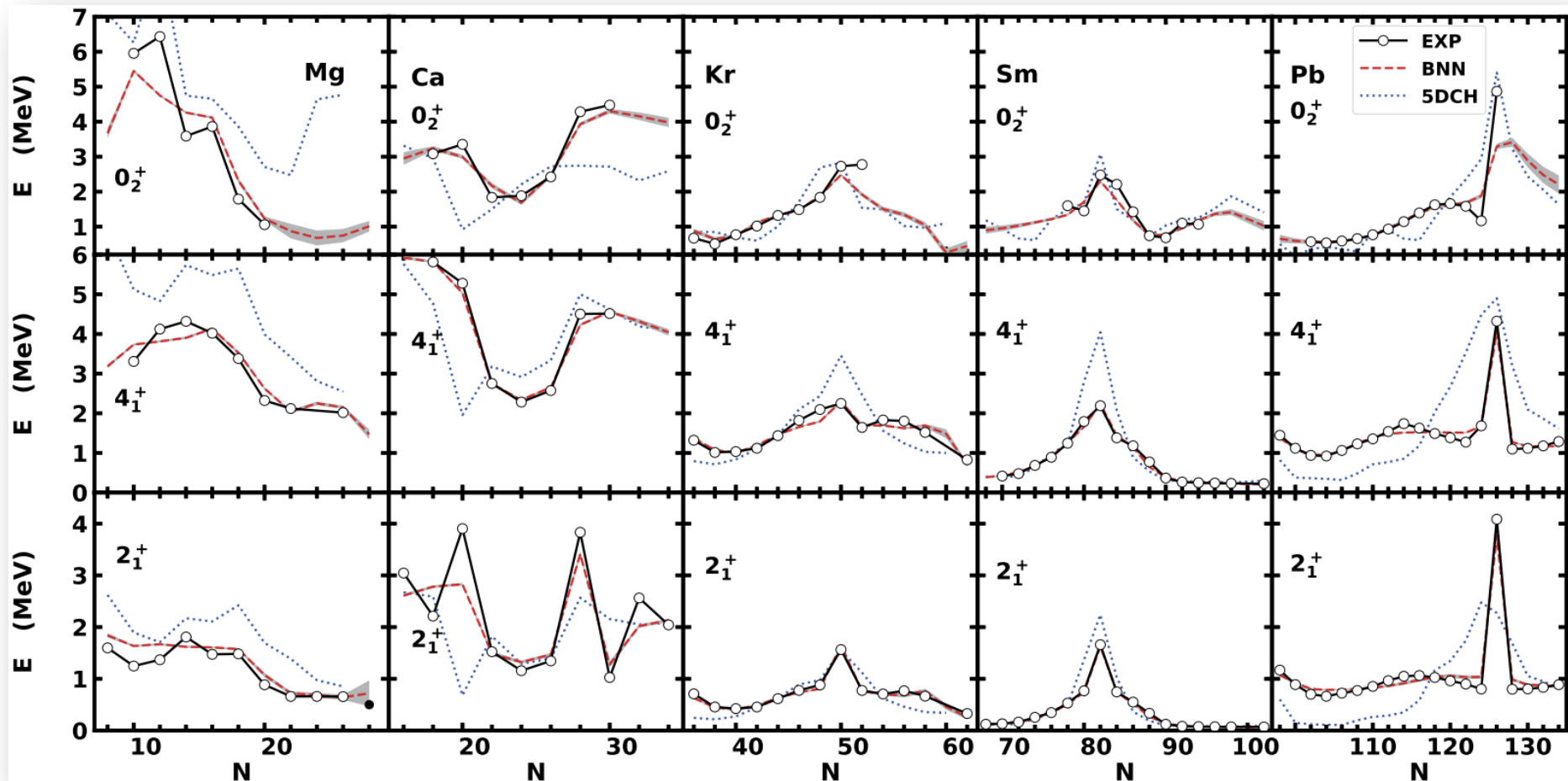
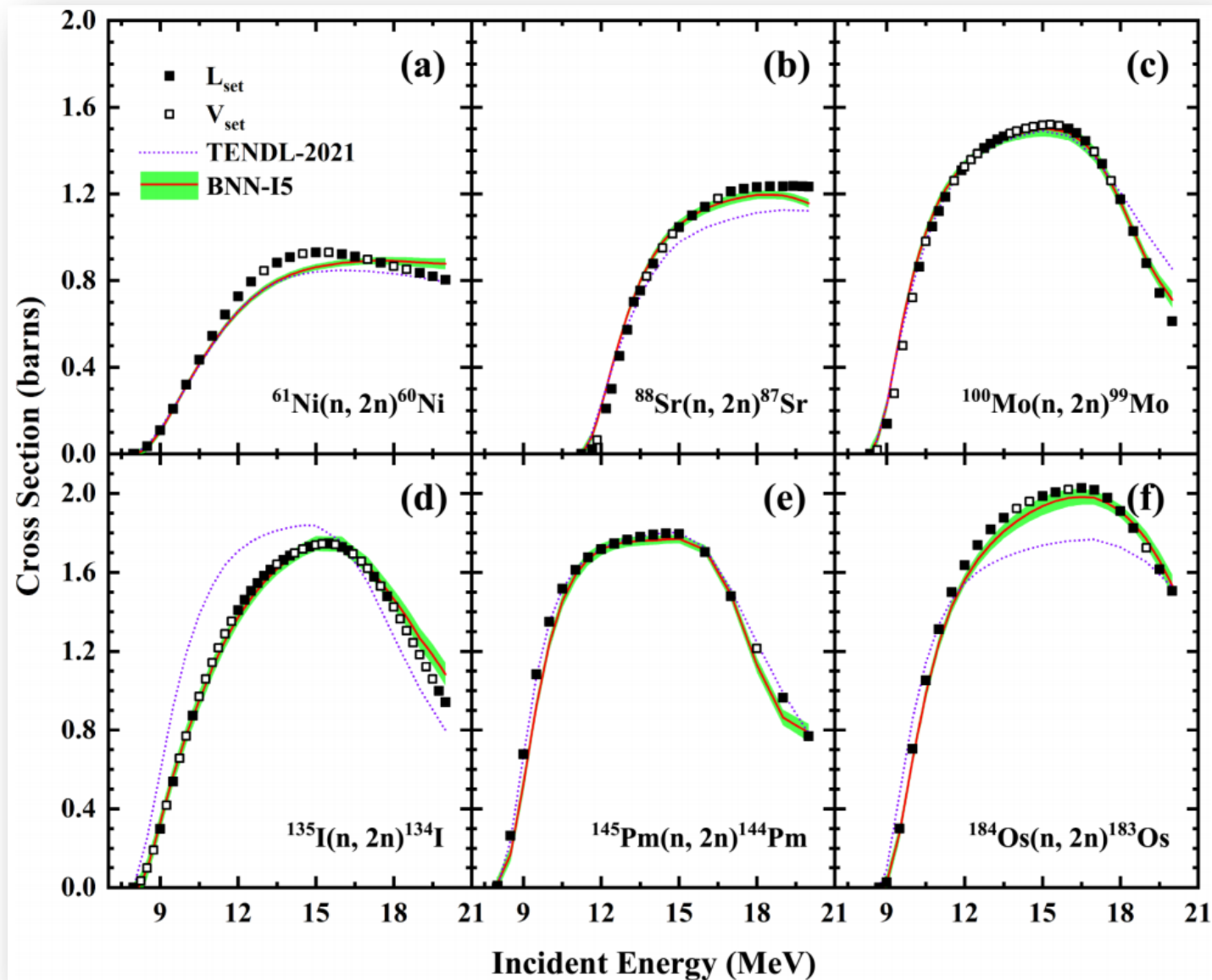


Fig: $E(0_2^+)$, $E(2_1^+)$, and $E(4_1^+)$ of Mg, Ca, Kr, Sm, and Pb isotopes predicted by BNN and 5DCH.

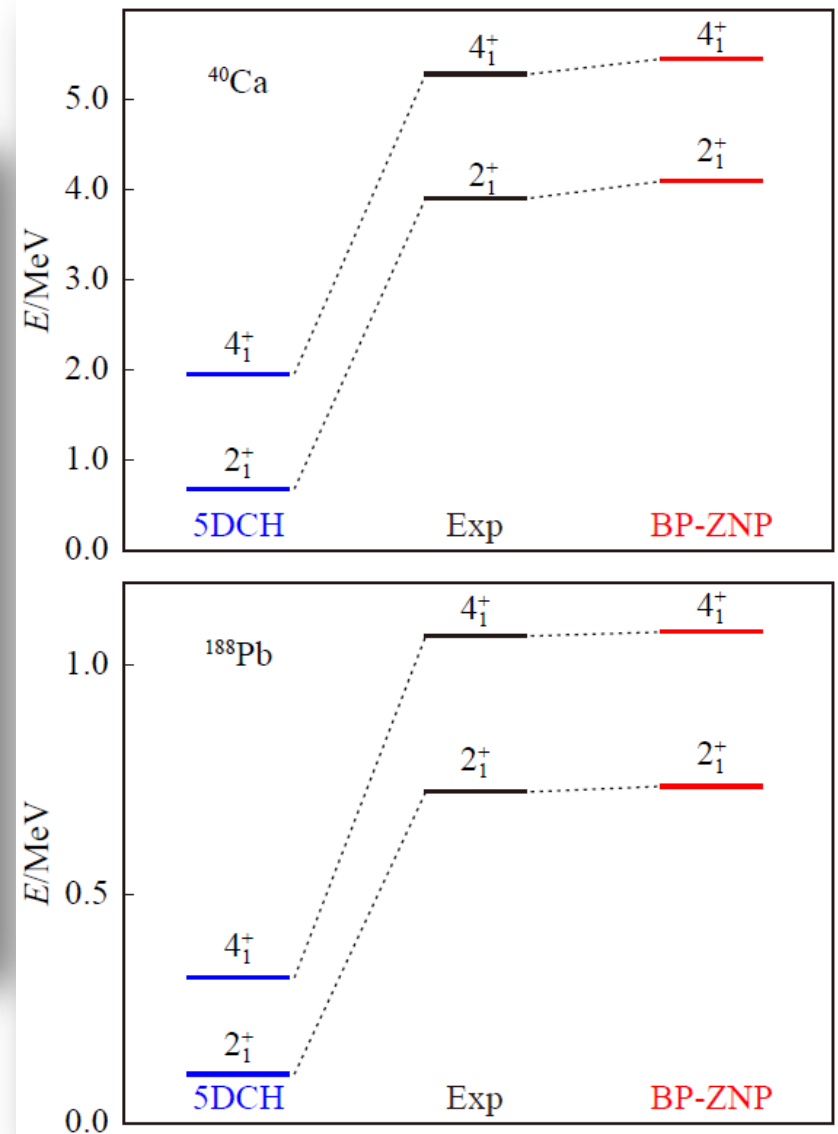
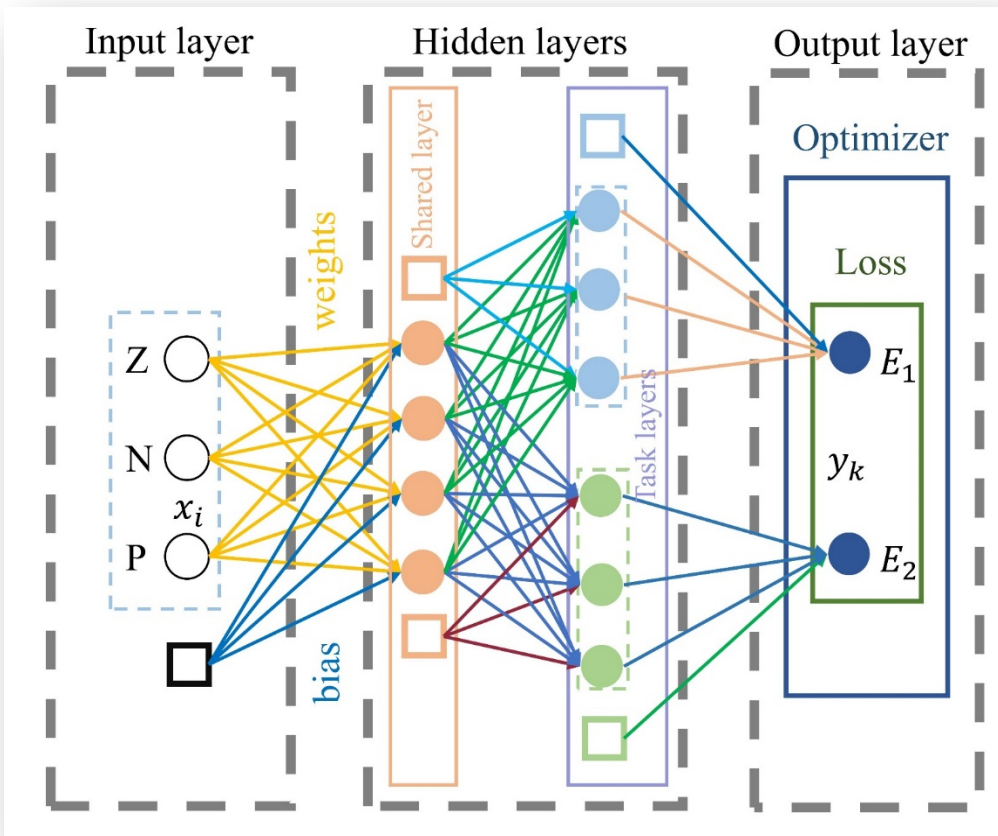
Y.F. Wang, X.Y. Zhang, Z.M. Niu*, and Z.P. Li, PLB 830, 137154 (2022)

Cross section: BNN



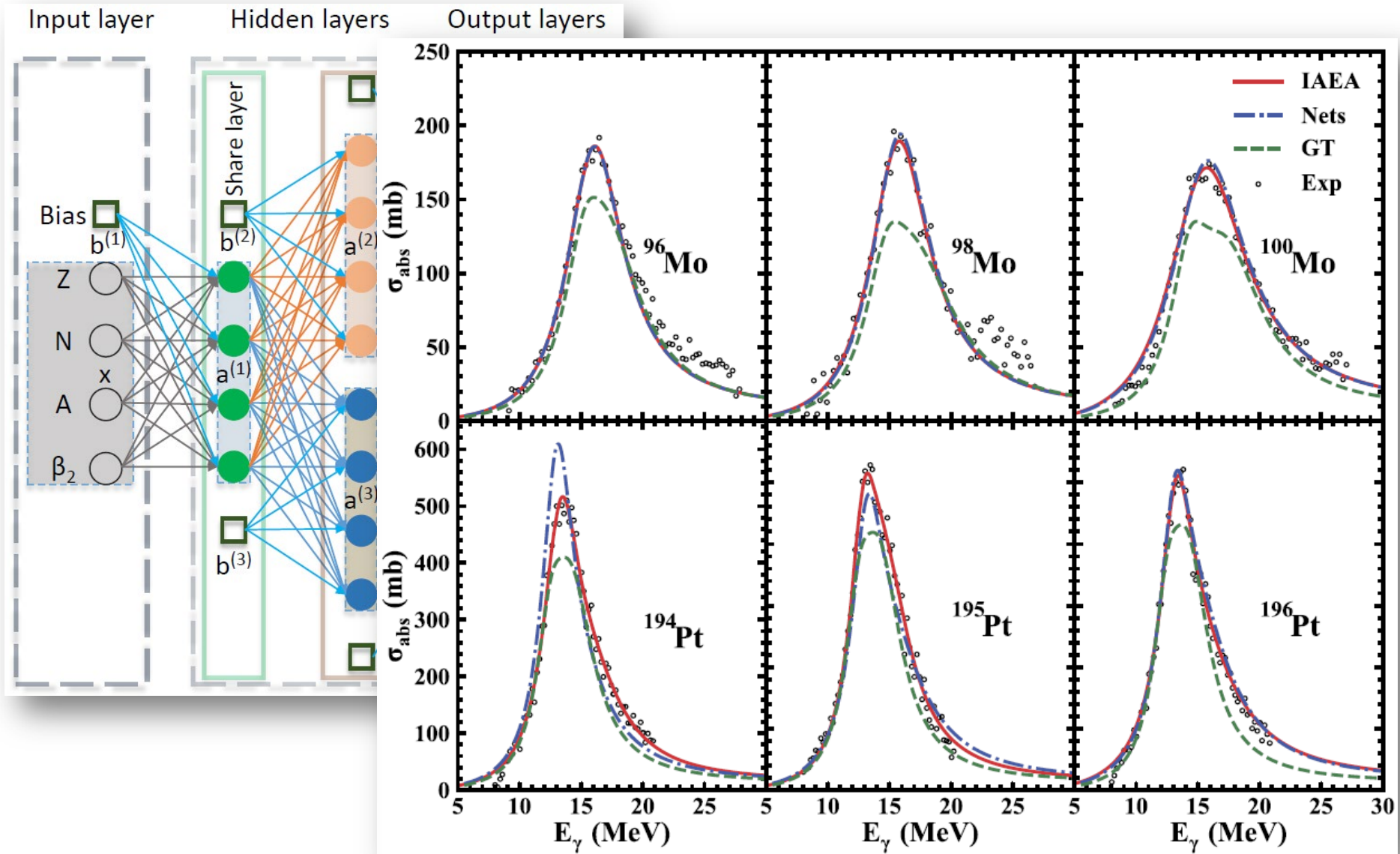
W. F. Li, L. L. Liu, Z. M. Niu*, Y. F. Niu, and X. L. Huang, PRC 109, 044616 (2024)

Low-lying excitation spectra: MTNN



Y.F. Wang and Z.M. Niu*, NPR 39, 273 (2022)

Giant dipole resonance key parameters: MTNN



J.H. Bai, Z.M. Niu, B.Y. Sun, and Y.F. Niu*, PLB 815, 136147 (2021)

Outline

① Introduction

② Bayesian neural network approach

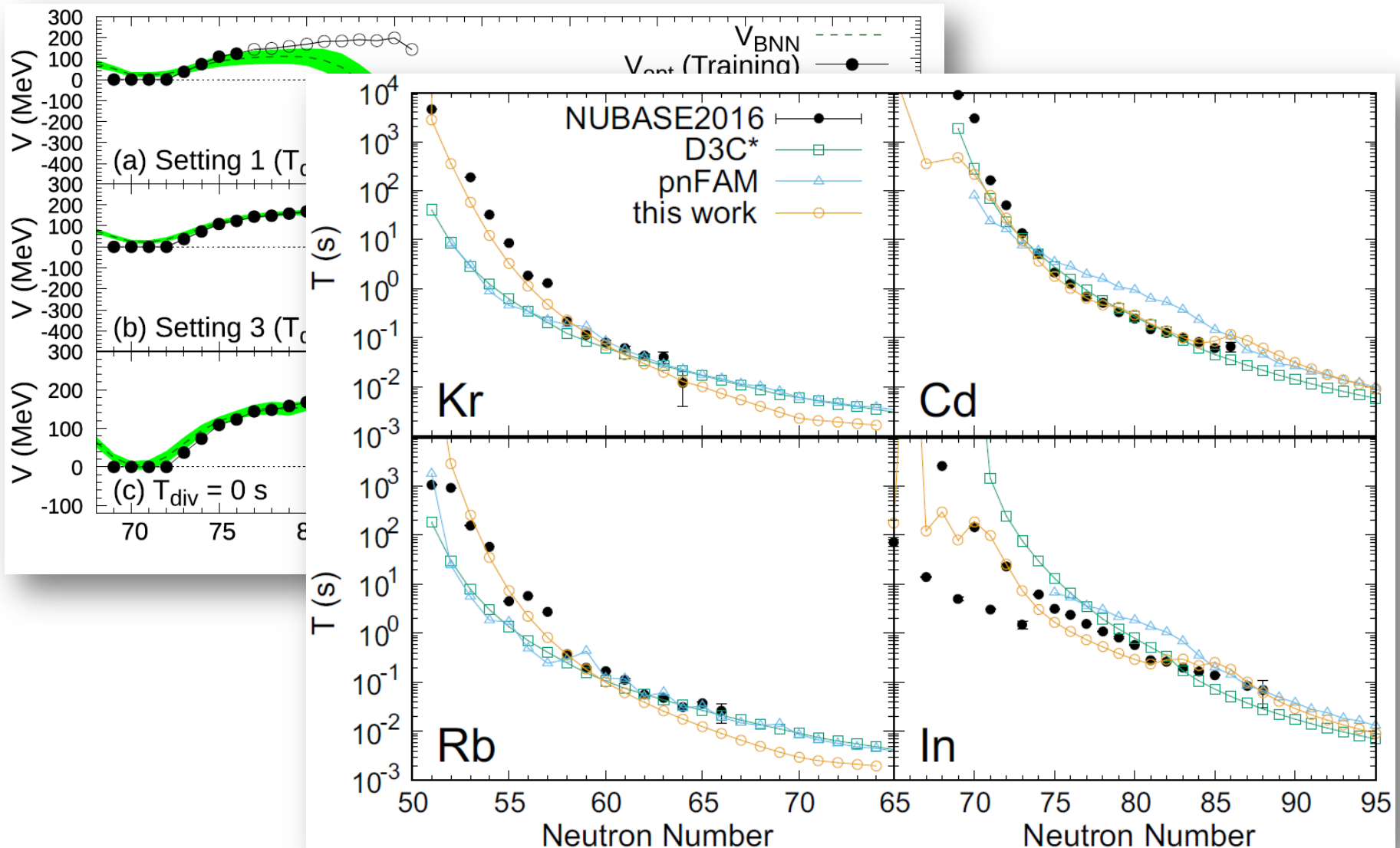
③ Results and discussion

★ Some other results

- ✓ Other properties with enough data or theoretical results
- ✓ Less controlled model parameters
- ✓ Construct theory guaranteed by physics, e.g. DFT
- ✓ New applications, e.g. data selection or evaluation

④ Summary and perspectives

Isoscalar pairing strengths: BNN



F. Minato*, Z. M. Niu*, and H. Z. Liang*, PRC 106, 024306 (2022)

Outline

① Introduction

② Bayesian neural network approach

③ Results and discussion

★ Some other results

- ✓ Other properties with enough data or theoretical results
- ✓ Less controlled model parameters
- ✓ Construct theory guaranteed by physics, e.g. DFT
- ✓ New applications, e.g. data selection or evaluation

④ Summary and perspectives

Density Functional Theory

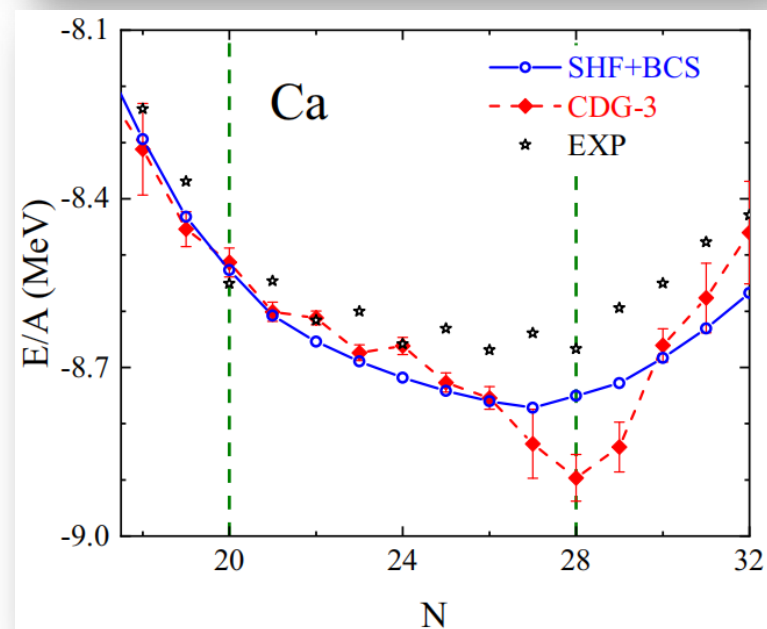
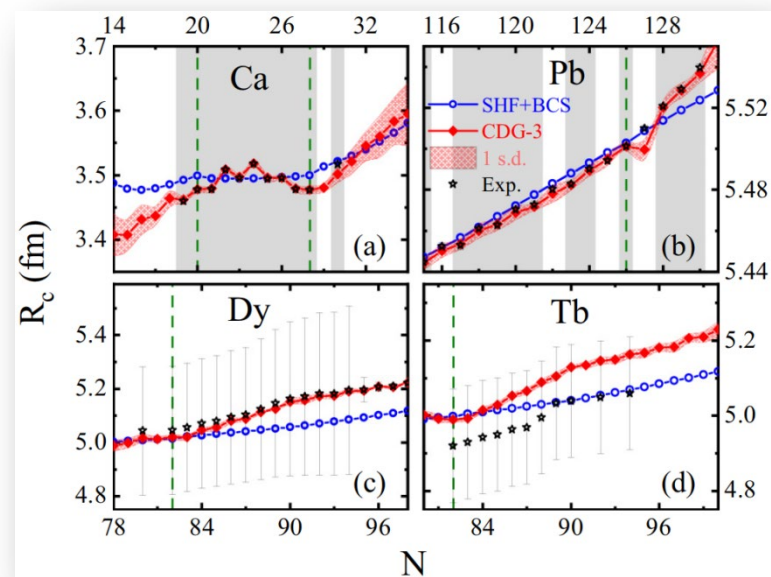
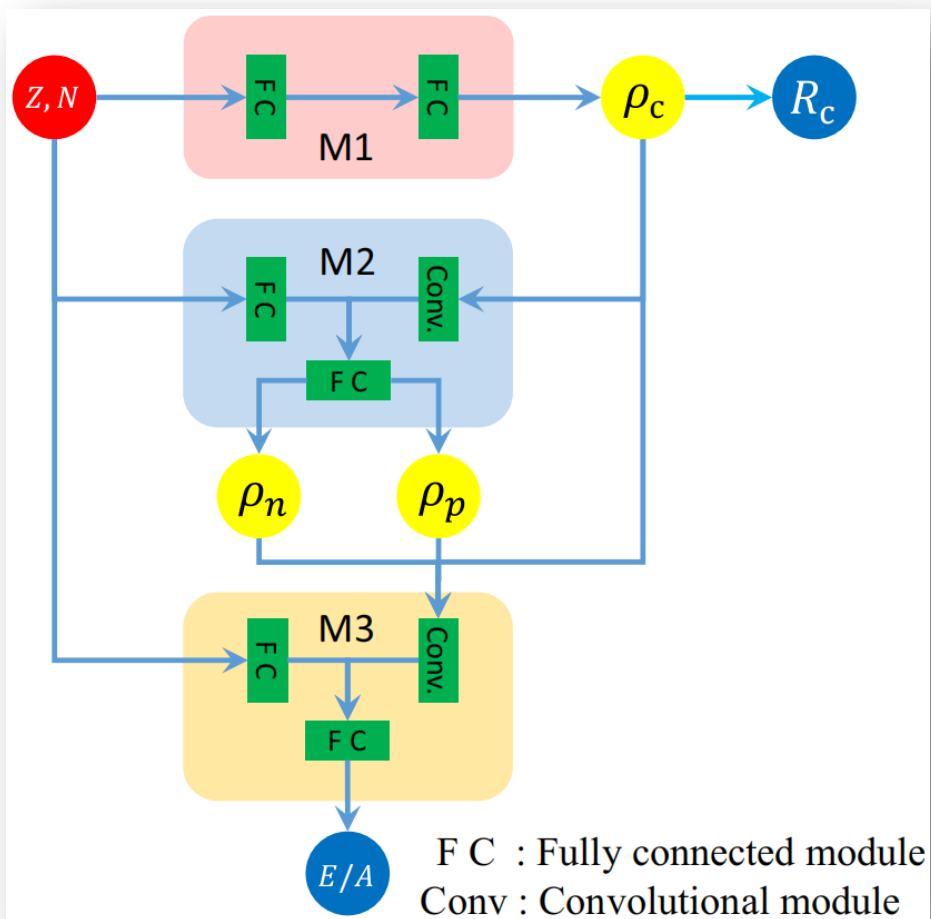
- The aim of density functional theory (DFT) is
 - to reduce the many-body quantum mechanical problem formulated in terms of N -particle wave functions Ψ to the one-particle level with the local density distribution $\rho(\mathbf{x})$.
- Hohenberg-Kohn theorem [Phys. Rev. 136, B864 (1964)]
 - ✓ There exist **a universal density functional** $F_{\text{HK}}[\rho(\mathbf{x})]$.
 - ✓ The ground-state energy $E_{\text{g.s.}}$ attains its minimum value when the density $\rho(\mathbf{x})$ has its correct ground-state value.
- HK variational principle

$$E_U = \inf_{\rho} \left\{ F_{\text{HK}}[\rho(\mathbf{x})] + \int d^d x U(\mathbf{x}) \rho(\mathbf{x}) \right\}$$

where $F_{\text{HK}}[\rho(\mathbf{x})] = \min_{\Psi_{\rho}} \langle \Psi_{\rho} | \hat{T} + \hat{V} | \Psi_{\rho} \rangle$ is **a universal functional**, which is valid for any number of particles N and for any external field $U(\mathbf{x})$.

Goal: $F_{\text{HK}}[\rho(\mathbf{x})]$

Density functional: FCNN+CNN



Outline

① Introduction

② Bayesian neural network approach

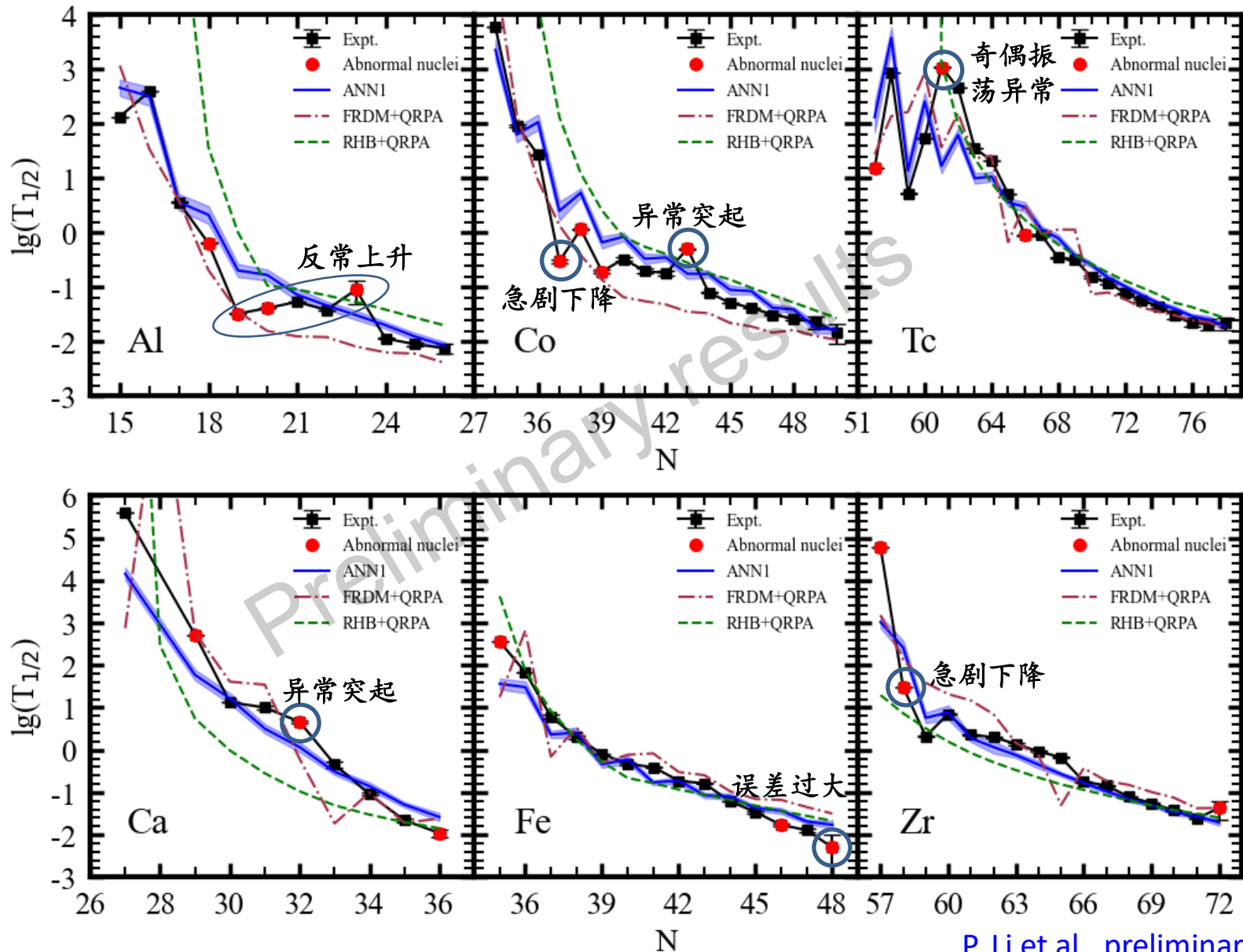
③ Results and discussion

★ Some other results

- ✓ Other properties with enough data or theoretical results
- ✓ Less controlled model parameters
- ✓ Construct theory guaranteed by physics, e.g. DFT
- ✓ New applications, e.g. data selection or evaluation

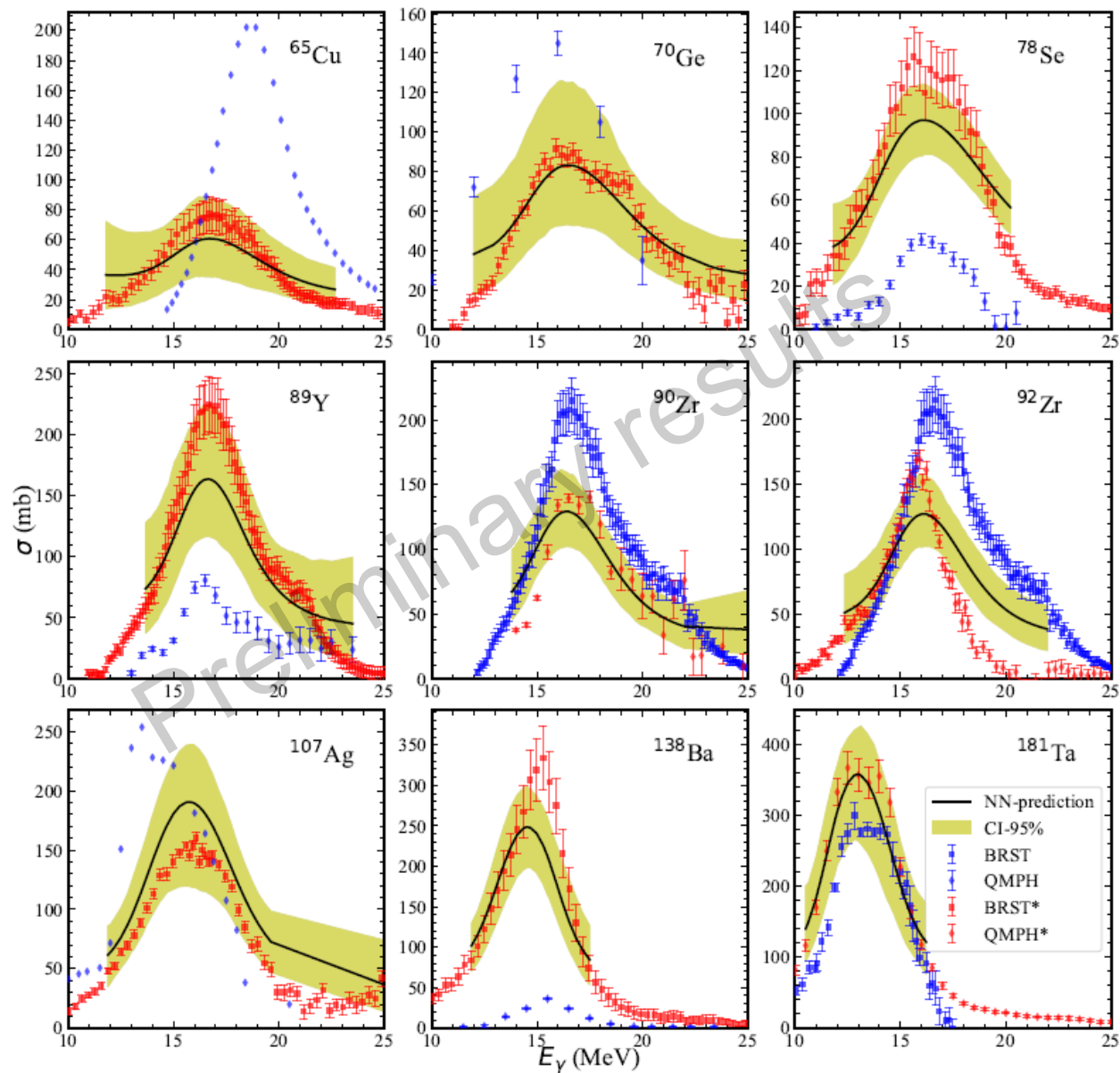
④ Summary and perspectives

Selection of abnormal β -decay half-lives: ANN



P. Li et al., preliminary results

Evaluation of photonuclear data



J. H. Bai et al., preliminary results

Outline

- ① Introduction
- ② Bayesian neural network approach
- ③ Results and discussion
 - ★ Toy model
 - ★ Nuclear masses
 - ★ Nuclear β -decay half-lives
 - ★ Some other results
- ④ Summary and perspectives

Summary and perspectives

Summary: BP neural network and BNN approach are introduced in detail and they are applied to predict various nuclear properties:

- ★ Neural network approach can significantly improve the prediction accuracies of various nuclear properties, e.g. nuclear masses, half-lives, nuclear properties, and low-lying excitation spectra.
- ★ It is found that the inclusion of more physics is very important to achieve better predictive performance.

Perspectives:

- ★ Other nuclear properties with enough data or theoretical results
- ★ Less controlled model parameters
- ★ Construct the theory guaranteed by physics, e.g. DFT
- ★ New applications, e.g. data selection or evaluation

Acknowledgements

Collaborators:

Anhui University: J. Y. Guo, Y. F. Wang, W. F. Li

Beihang University: B. H. Sun

Jilin University: J. Li, Z. L. Yuan, D. C. Tian, T. S. Shang

Lanzhou University: W. H. Long, Y. F. Niu, B. Y. Sun, J.H. Bai, P. Li

Japan Atomic Energy Agency: F. Minato

The University of Tokyo: H. Z. Liang, Z. X. Yang

Thank you!

Influence of noise variance

Model	Fixed	Gamma
RMF	0.732	0.443
HFB-31	0.354	0.296
WS4	0.203	0.178
FRDM2012	0.282	0.208
BW	1.035	0.850
BW2	0.497	0.313

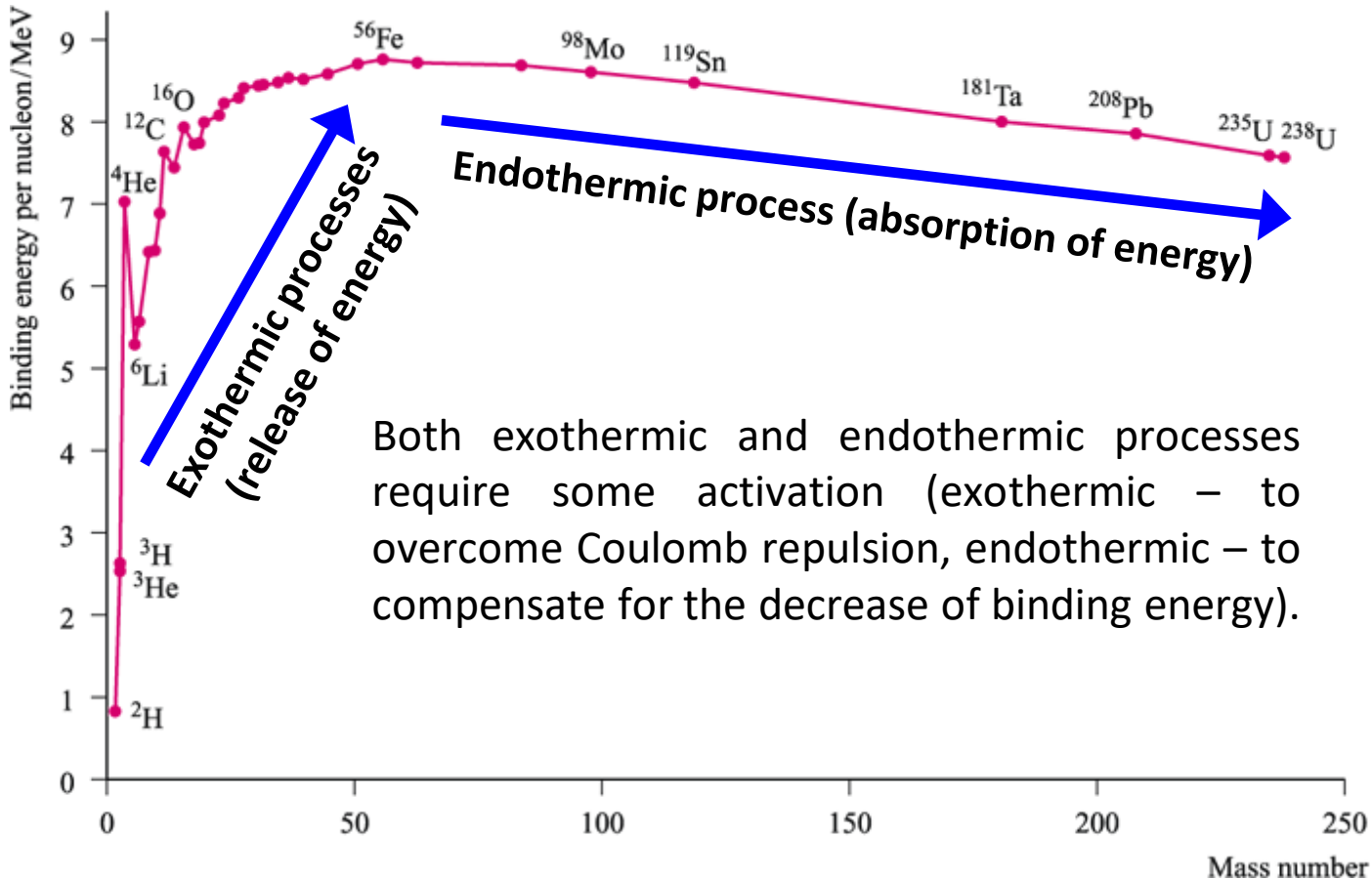
Figure: rms deviations between experimental data and mass predictions for various models improved by BNN approach. The 2nd and 3rd columns denote the results with fixed value and gamma distribution for noise variance, respectively.

$$\chi^2 = \sum_{n=1}^N \left[\frac{t_n - y_n(x, \omega)}{\sigma_n} \right]^2$$

► The BNN approach can automatically find the optimal value for the noise variance, which can reduce the rms deviations by about 20%.

Stellar nucleosynthesis

➤ Stellar nucleosynthesis



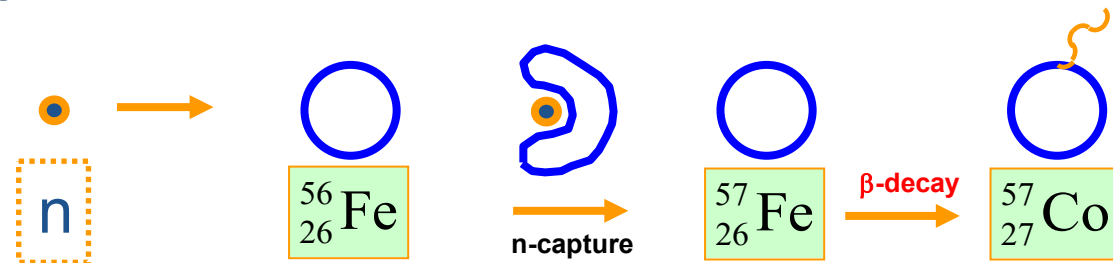
- Exothermic processes: H burning, He burning, and C, Ne, O, Si burning (source of stellar energy)
- Endothermic process: r-process, s-process, rp-process, p-process (γ -process)

Neutron capture process

Neutron capture process

$$\tau_n \gg \tau_\beta$$

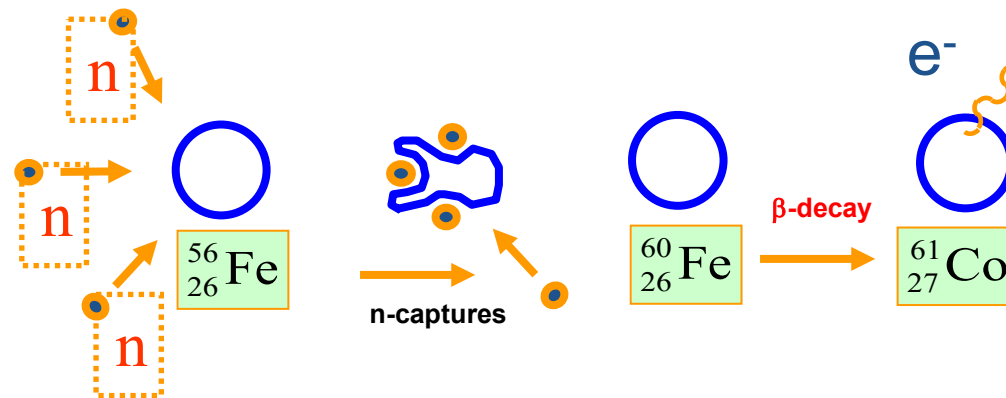
s-process
(slow neutron
capture):



unstable nucleus decays before capturing another neutron
progress up the valley of stability.

$$\tau_n \ll \tau_\beta$$

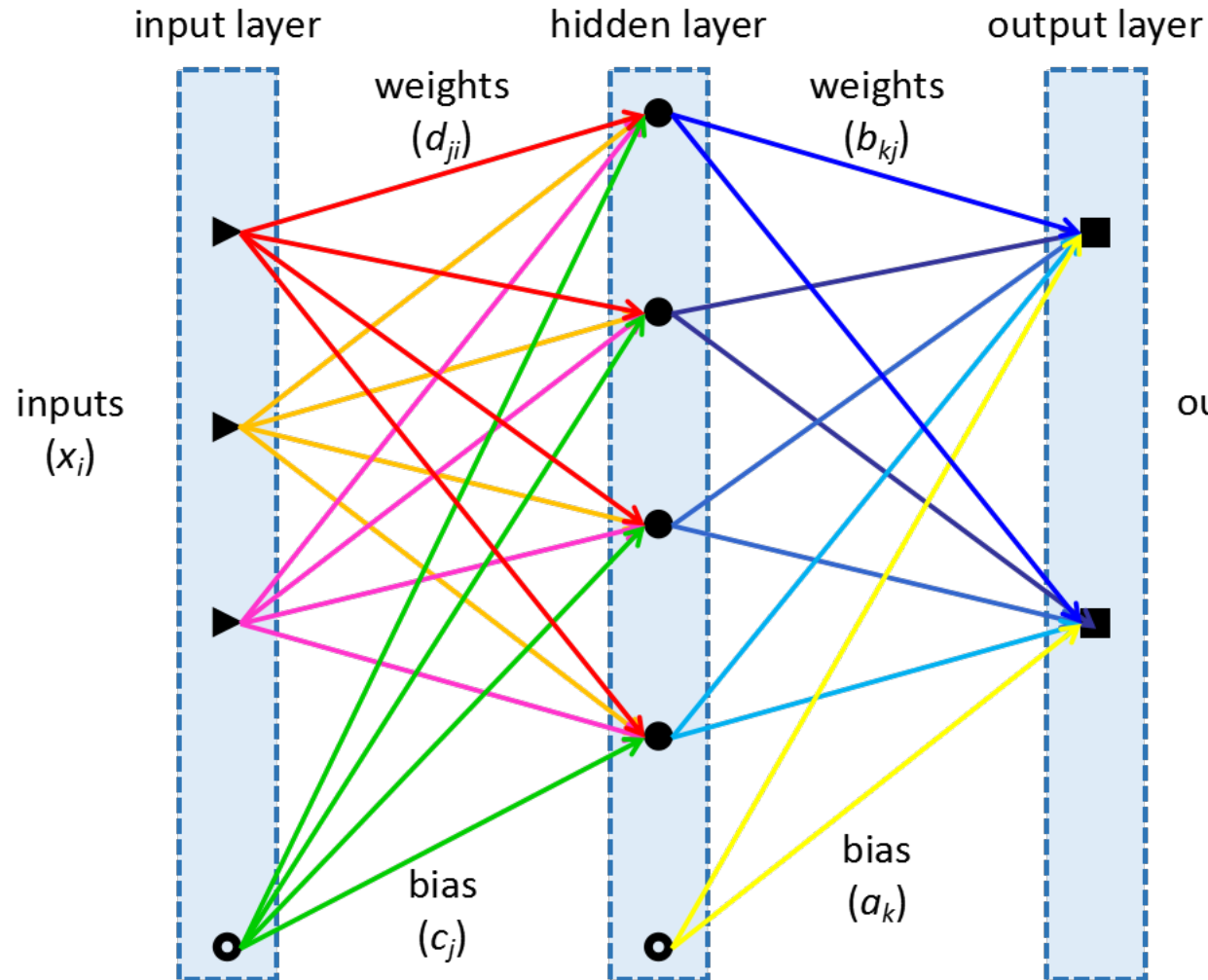
r-process
(rapid neutron
captures):



fast neutron capture until the nuclear force is unable to bind an extra neutron. Then, a beta decay occurs, and in the new chain the neutron capture continues.

These processes require energy, occur only at high densities & temperatures (e.g., *r*-processes occur in core-collapse supernovae).

Details: Neural network

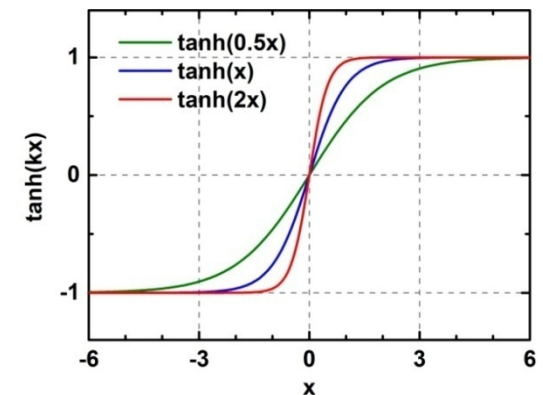


$$a_{k1} : n_O; b_{kj} : n_O n_H;$$

$$c_{j1} : n_H; d_{ji} : n_I n_H$$

$$n_O + n_O n_H + n_H + n_I n_H$$

$$= (n_I + n_O + 1)n_H + n_O$$



$$y_{k1} = a_{k1} + \sum_{j=1}^H b_{kj} \tanh \left(c_{j1} + \sum_{i=1}^I d_{ji} x_{i1} \right)$$

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Influence of prior

Differences between Bayesians and frequentists



机器说假话的概率：

$$p = \frac{1}{6} \times \frac{1}{6} = \frac{1}{36} \approx 0.028 < 5\%$$

- 频率学派：因 $p < 5\%$ ，位于可置信区间内，所以机器可信，即认为太阳爆炸了
- 贝叶斯学派：
 - ✓ 因为太阳不容易爆炸，假设太阳爆炸的先验概率为0.0000000001
 - ✓ 即使 $p < 5\%$ ，但后验概率仍然很小，所以认为太阳没有爆炸。

频率学派统计学家：

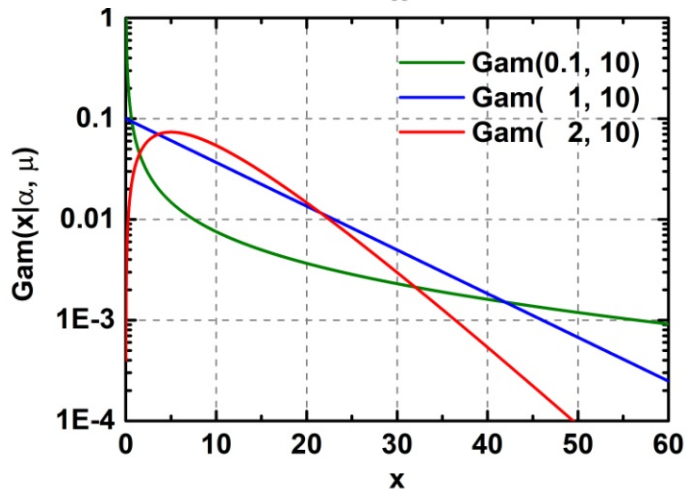
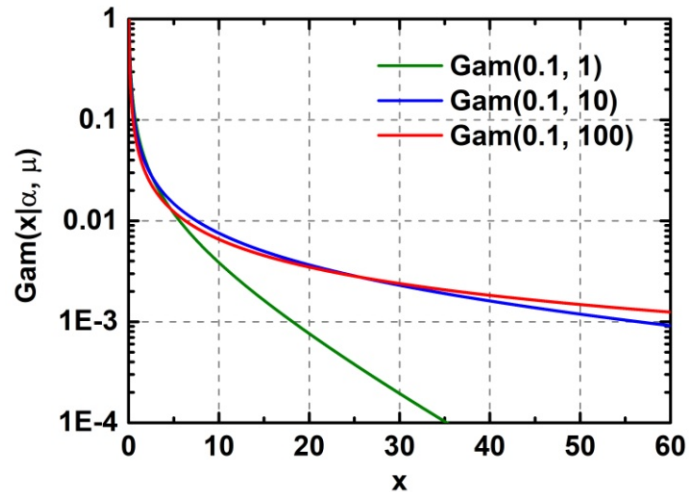
贝叶斯学派统计学家：



Details: Gamma distribution

$$\text{Gam}(x | \alpha, \mu) = \frac{(\alpha / \mu)^\alpha}{\Gamma(\alpha)} x^{\alpha-1} \exp\left(-\frac{\alpha x}{\mu}\right) \xrightarrow[\mu \rightarrow 1/w^2]{\alpha \rightarrow \alpha'/2} \text{Gam}(x | \alpha', w) = \frac{(\alpha' w^2 / 2)^{\alpha'/2}}{\Gamma(\alpha' / 2)} x^{\alpha'/2-1} \exp\left(-\frac{\alpha' w^2 x}{2}\right)$$

- Smaller α and larger μ (smaller width w) will lead to more vague distribution, i.e. the probabilities of very small and very large x are relatively larger.



$w:\alpha'$		μ, α
0.05:0.5	→	400, 0.25 (for parameters)
0.2:0.5	→	25, 0.25 (for noise)
1:0.2	→	1, 0.1 (upper figure)
0.316227766:0.2	→	10, 0.1 (upper figure)
0.1:0.2	→	100, 0.1 (upper figure)
0.316227766:0.2	→	10, 0.1 (lower figure)
0.316227766:2	→	10, 1 (lower figure)
0.316227766:4	→	10, 2 (lower figure)

随机变量的抽样

● 单位均匀分布的随机数

最简单且最基本的连续型随机变量的分布是单位均匀分布，即 $[0, 1]$ 上的均匀分布，其分布密度函数为：

$$f(x) = \begin{cases} 1, & 0 \leq x \leq 1 \\ 0, & \text{其他} \end{cases}$$

- ✓ **随机数表**：随机数表由等概率出现的0, ..., 9十个数字组成，依次取出表中 n 个相邻的数字合并在一起作为随机数。
- ✓ **物理方法**：利用某些随机物理现象，如计算机固有噪声，通过某些特殊设备，可以在计算机上直接产生随机数。
- ✓ **伪随机数**：同余法

$$x_{n+1} = (ax_n + c) \bmod m$$

$$\xi_{n+1} = x_{n+1} / (m - 1)$$

a 、 c 和 m 分别为乘子、增量和模， x_0 为初值。

随机变量的抽样

● 直接抽样法（反函数法）

设随机变量 x 的分布密度函数为 $f(x)$

$$0 \leq F(x) = \int_{-\infty}^x f(t) dt \leq 1$$

$f(x)$ 必须归一!

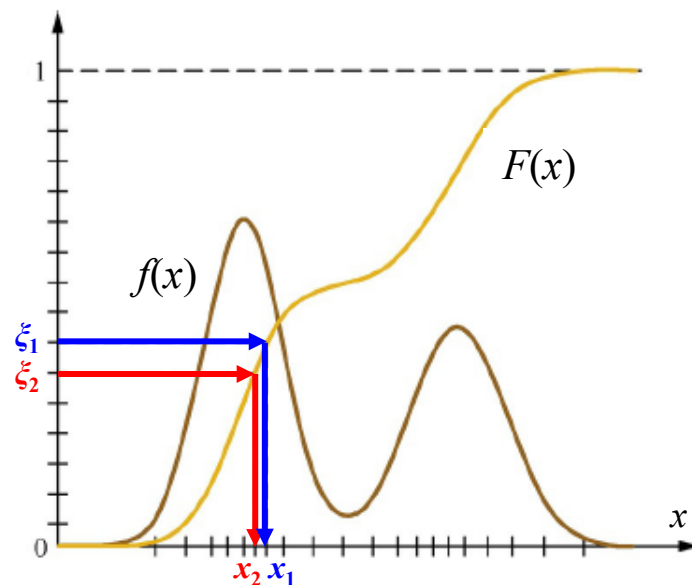
产生 $[0, 1]$ 均匀分布的随机数 ξ , 令 $\xi = F(\eta)$ 或 $\eta = F^{-1}(\xi)$

$$1d\xi = dF(\eta) = F'(\eta)d\eta = f(\eta)d\eta$$

即按反函数计算出的 η 就是服从分布密度函数 $f(x)$ 的随机变量。

➤ 抽样步骤:

- (1) 计算 $f(x)$ 的分布函数 $F(x)$
- (2) 产生 $[0, 1]$ 均匀分布随机数 ξ
- (3) 计算 $F^{-1}(\xi)$, 令 $\eta = F^{-1}(\xi)$
- (4) 重复(2)和(3)



随机变量的抽样

➤ 例：对指数分布的直接抽样

分布密度函数为：

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x > 0, \lambda > 0 \\ 0, & x \leq 0 \end{cases}$$

积分得到分布函数：

$$F(\eta) = \int_{-\infty}^{\eta} f(x) dx = \int_0^{\eta} \lambda e^{-\lambda x} dx = 1 - e^{-\lambda \eta}$$

令 $\xi = F(\eta) = 1 - e^{-\lambda \eta}$ ，则指数分布随机变量的抽样为：

$$\eta = -\frac{1}{\lambda} \ln(1 - \xi) = -\frac{1}{\lambda} \ln \xi$$

(1- ξ)和 ξ 同样服从[0, 1]均匀分布

随机变量的抽样

● 变换抽样法

若对 $f(x)$ 的抽样比较复杂，而对 $g(y)$ 的抽样已知且简单。设法找到变换关系： $y=y(x)$ ，使得

$$f(x)dx = g(y)dy$$

按变换 $x=x(y)$ 计算出的 x 就是服从分布密度函数 $f(x)$ 的随机变量。

➤ 抽样步骤：

- (1) 找到变换关系 $y=y(x)$
- (2) 对 $g(y)$ 进行抽样得到 y
- (3) 计算 $x=x(y)$
- (4) 重复(2)和(3)

若 $g(y)$ 为 $[0, 1]$ 的均匀分布，则

$$f(x)dx = dy$$

$$\Rightarrow dy / dx = y' = f(x)$$

$$\Rightarrow y(x) = \int_{-\infty}^x f(x)dx = F(x)$$

$$\Rightarrow x = F^{-1}(y)$$

即直接抽样法。

随机变量的抽样

● 重要抽样法

$$J = \int_a^b g(x) f(x) dx = \frac{1}{n} \sum_{i=1}^n g(x_i)$$

$$\sigma^2 = \int_a^b [g(x) - J]^2 f(x) dx = \frac{1}{n} \sum_{i=1}^n [g(x_i) - J]^2$$

当 $g(x)$ 在定义域内有显著起伏变化时，方差 σ^2 较大，进而蒙卡积分 J 的误差 σ/\sqrt{N} 较大。

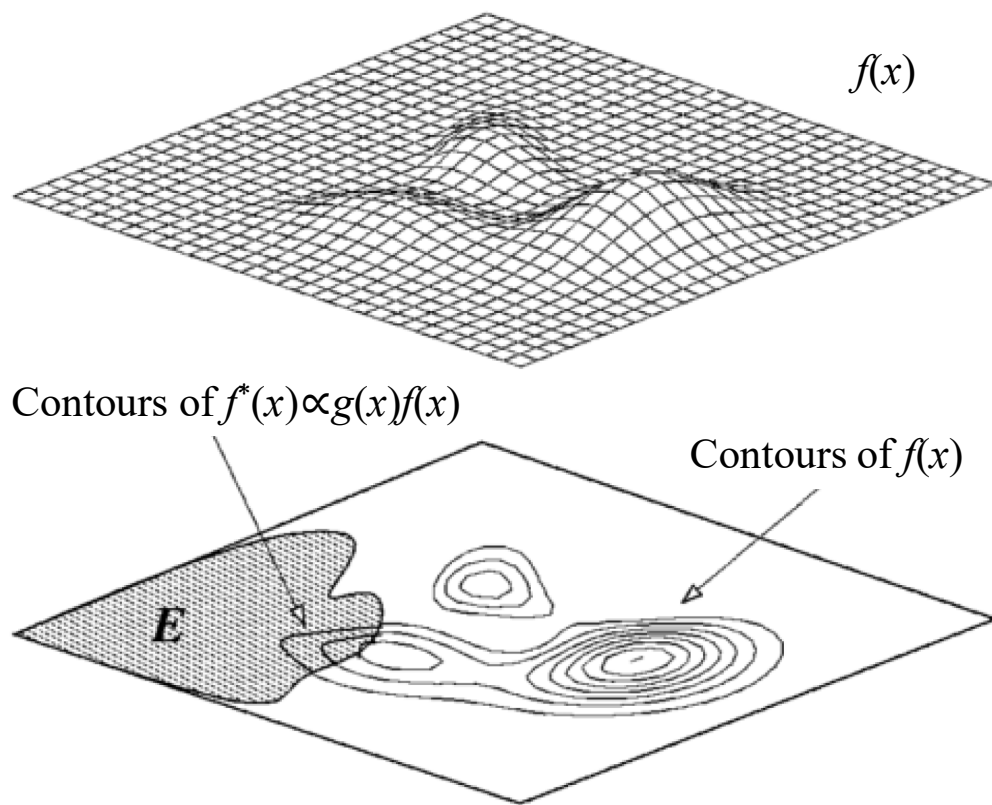
✓ 适当选取偏倚分布密度函数 $f^*(x)$ ，使得 $g^*(x)$ 在定义域内变化比较平坦或者 $g(x)f(x)$ 大的地方 $f^*(x)$ 值也较大。然后产生分布密度函数为 $f^*(x)$ 的随机变量 x_i ，则有

$$J = \int_a^b g(x) f(x) dx = \int_a^b \frac{g(x) f(x)}{f^*(x)} \underbrace{f^*(x)} dx = \int_a^b g^*(x) f^*(x) dx = \frac{1}{n} \sum_{i=1}^n g^*(x_i)$$

偏倚分布密度函数

随机变量的抽样

$$J = \int_a^b g(x) f(x) dx = \int_a^b \frac{g(x) f(x)}{f^*(x)} f^*(x) dx = \int_a^b g^*(x) f^*(x) dx = \frac{1}{n} \sum_{i=1}^n g^*(x_i)$$



例 $J = \int_0^{\infty} e^{-x/\lambda} f(x) dx$

取 $f^*(x)$ 为指数分布 $e^{-x/\lambda} / \lambda$

$$J = \int_0^{\infty} \frac{e^{-x/\lambda} f(x)}{f^*(x)} f^*(x) dx$$

$$= \lambda \int_0^{\infty} f(x) f^*(x) dx$$

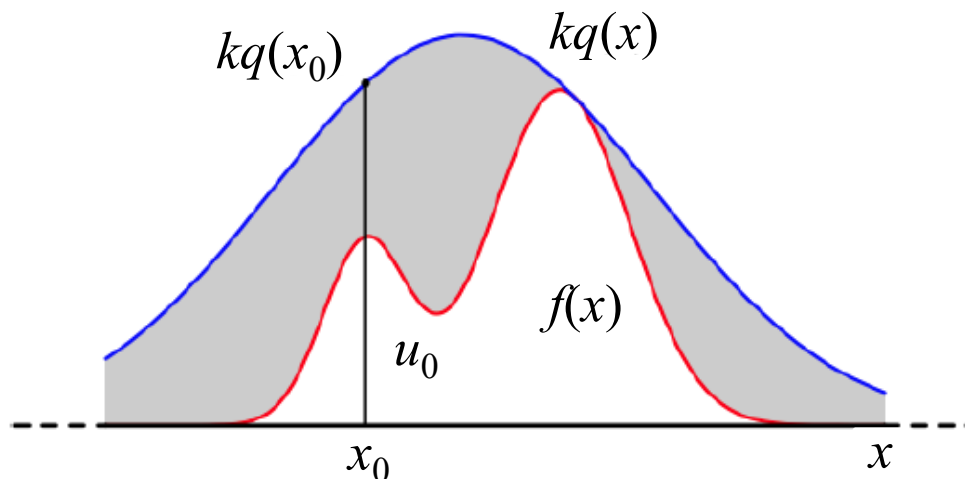
按照 $f^*(x)$ 抽样 x_i

$$J = \frac{\lambda}{n} \sum_{i=1}^n f(x_i)$$

随机变量的抽样

● 拒绝抽样法

若 $f(x)$ 太复杂无法直接抽样，可以设定一个可抽样的分布 $q(x)$ ，如高斯分布，然后按照一定规则拒绝某些样本，实现对 $f(x)$ 的抽样



➤ 抽样步骤:

- (1) 确定 $q(x)$ 和常量 k ，使得 $f(x)$ 总在 $kq(x)$ 的下方
- (2) x 轴的方向：从 $q(x)$ 分布抽样取得 x_0
- (3) y 轴的方向：从 $[0, kq(x_0)]$ 均匀分布中抽样得到 u_0
若 $u_0 > f(x_0)$ ，拒绝抽样 x_0 ；否则接受 x_0
- (4) 重复(2)和(3)

Markov chain

● **马尔科夫链**: 设 $\{X_n, n=1, 2, \dots\}$ 是一个随机序列, 状态空间 E 为有限或可列集, 对于任意的正整数 n , 若 $j, i_k \in E (k=1, \dots, n-1, n)$, 有

$$P\{X_{n+1} = j | X_n = i_n, X_{n-1} = i_{n-1}, \dots, X_1 = i_1\} = P\{X_{n+1} = j | X_n = i_n\}$$

则称 $\{X_n, n=1, 2, \dots\}$ 为马尔科夫链, $P\{X_{n+1}=j | X_n=i_n\}$ 称为转移概率。

- ? 放回的袋中取球问题
- ? 不放回的袋中取球问题
- ? 布朗运动
- ? 多次复习的记忆曲线

● **齐次马氏链**: 设 $\{X_n, n=1, 2, \dots\}$ 是一个马氏链, 若 $P\{X_{n+1}=j | X_n=i_n\}$ 与 n 无关, 即

$$P\{X_{n+1} = j | X_n = i_n\} = p_{ij}$$

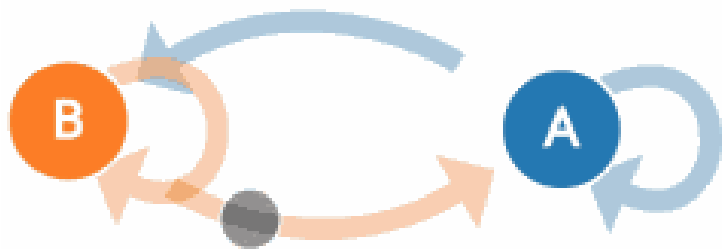
则称 $\{X_n, n=1, 2, \dots\}$ 为时齐的马氏链 (也称时齐马氏链), p_{ij} 为系统由状态 i 转移到状态 j 的转移概率。

Markov chain

● **转移矩阵:** 由转移概率 p_{ij} 组成的矩阵 P 称为转移矩阵, 其具有性质

(1) 对一切 $i, j \in E, 0 \leq p_{ij} \leq 1$

(2) 对一切 $i \in E, \sum_{j \in E} p_{ij} = 1$



	A	B
A	$P(A A): 0.50$ <input type="range"/>	$P(B A): 0.50$ <input type="range"/>
B	$P(A B): 0.50$ <input type="range"/>	$P(B B): 0.50$ <input type="range"/>

Markov chain

● 齐次马氏链的遍历性：设齐次马氏链 $\{X_n, n=1, 2, \dots\}$ 的状态空间为 $E=\{1, 2, \dots\}$ ，若对于一切状态 $i, j \in E$ ，存在不依赖于 i 的常数 π_j ，为其转移概率的极限，即

$$\lim_{n \rightarrow \infty} p_{ij}^{(n)} = \pi_j, \quad i, j \in E$$

$$P^{(n)} = P^n = \begin{pmatrix} p_{11} & p_{12} & \dots & p_{1j} & \dots \\ p_{21} & p_{22} & \dots & p_{2j} & \dots \\ \dots & \dots & \dots & \dots & \dots \\ p_{i1} & p_{i2} & \dots & p_{ij} & \dots \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix} \xrightarrow{n \rightarrow \infty} \begin{pmatrix} \pi_1 & \pi_2 & \dots & \pi_j & \dots \\ \pi_1 & \pi_2 & \dots & \pi_j & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \pi_1 & \pi_2 & \dots & \pi_j & \dots \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix}$$

则称此齐次马氏链具有遍历性。若 $\pi_j, j \in E$ 满足

$$\pi_j \geq 0, \quad \sum_j \pi_j = 1$$

则称其为转移概率的极限分布。备注：有限状态的遍历马氏链必存在极限分布，无限（即可列）状态的遍历马氏链不一定存在极限分布，只有其极限概率构成概率分布时才存在极限分布。

Markov chain

● 齐次马氏链的平稳分布：设 $\{X_n, n=1, 2, \dots\}$ 为一齐次马氏链，若存在实数集合 $\{\pi_j, j \in E\}$ ，满足

$$\begin{cases} \pi_j \geq 0, j \in E \\ \sum_{j \in E} \pi_j = 1 \end{cases}$$

则称 $\{\pi_j, j \in E\}$ 为概率分布。如果此概率分布满足

$$\pi_j = \sum_{i \in E} \pi_i p_{ij}, j \in E$$

则 $\{X_n, n \geq 0\}$ 是一平稳齐次马氏链，并称 $\{\pi_j, j \in E\}$ 为该过程的一个平稳分布。备注：有限马氏链转移概率的极限分布一定是平稳分布，因此有限状态遍历马氏链的极限分布就是平稳分布；但无限马氏链转移概率的极限分布不一定是平稳分布。

Markov chain

例：设齐次马氏链的状态空间为 $E=\{1, 2\}$ ，其转移概率矩阵为

$$P = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

可见 $P^n=P$ ，但 P^n 某一系列的所有元素并不是同一值，所以此马氏链不是遍历的。设有 π_1, π_2 ，满足 $\pi_1+\pi_2=1$ 且 $0<\pi_1, \pi_2<1$ ，有

$$(\pi_1 \quad \pi_2) = (\pi_1 \quad \pi_2) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

可见，平稳分布是存在的，而且有无穷多个。

备注：虽然遍历的马氏链具有平稳性，但是平稳的马氏链不一定具有遍历性，即不遍历的马氏链也可以具有平稳性。

Markov chain

● **遍历定理1:** 对于有限状态的齐次马氏链 $\{X_n, n=1, 2, \dots\}$, 设状态空间为 $E=\{1, 2, \dots, k\}$, 若存在正整数 m , 对任意状态 $i, j \in E$, 其 m 步的转移概率均大于0, 即

$$p_{ij}^{(m)} > 0$$

则此马氏链具有遍历性, 且 $\{\pi_j\}=\{\pi_1, \pi_2, \dots, \pi_k\}$ 是方程组

$$\pi_j = \sum_i \pi_i p_{ij}$$

满足条件 $\sum_j \pi_j = 1, \pi_j > 0$ 的唯一解。

备注: 利用该定理, 可以判断有限状态齐次马氏链的遍历性以及求出稳态概率 π_j , 此时稳态概率即为平稳分布。

Markov chain

例：设齐次马氏链的状态空间为 $E=\{1, 2, 3\}$ ，其转移概率矩阵为

$$P = \begin{pmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1/2 \\ 0 & 1/2 & 1/2 \end{pmatrix}$$

试问此链是否具有遍历性？若有，试求其稳态概率。

$$\text{解： } P^{(2)} = P^2 = \begin{pmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1/2 \\ 0 & 1/2 & 1/2 \end{pmatrix} \begin{pmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1/2 \\ 0 & 1/2 & 1/2 \end{pmatrix} = \begin{pmatrix} 1/2 & 1/4 & 1/4 \\ 1/4 & 1/2 & 1/4 \\ 1/4 & 1/4 & 1/2 \end{pmatrix}$$

可见所有的二步转移概率均大于0，由上面定理可知，此链具有遍历性。再由转移概率与稳态概率满足的方程组得

$$\begin{cases} \pi_1 = \pi_1 \frac{1}{2} + \pi_2 \frac{1}{2} + \pi_3 \cdot 0 \\ \pi_2 = \pi_1 \frac{1}{2} + \pi_2 \cdot 0 + \pi_3 \frac{1}{2} \\ \pi_3 = \pi_1 \cdot 0 + \pi_2 \frac{1}{2} + \pi_3 \frac{1}{2} \end{cases} \quad \text{且} \quad \begin{cases} \pi_1, \pi_2, \pi_3 > 0 \\ \pi_1 + \pi_2 + \pi_3 = 1 \end{cases} \quad \text{解之可得稳态概率 } \pi_1 = \pi_2 = \pi_3 = \frac{1}{3}$$

Markov chain

遍历定理2: 对于不可约且非周期的可列状态的齐次马氏链 $\{X_n, n=1, 2, \dots\}$, 设状态空间为 $E=\{1, 2, \dots\}$, 其存在平稳分布的充要条件是, 这个链所有状态都是正常返的, 并且此时极限分布

$$\pi_j = \lim_{n \rightarrow \infty} p_{ij}^{(n)} > 0$$

是唯一的平稳分布。注意: 对于有限状态的齐次马氏链 $\{X_n, n=1, 2, \dots\}$, 设状态空间为 $E=\{1, 2, \dots, k\}$, 若其是不可约的, 则其所有的状态都是正常返的, 因此, 其存在平稳分布的充要条件仅为其是不可约且非周期的马氏链。

