Introduction to Machine Learning Methods in Condensed Matter Physics

LECTURE 2, FALL 2021

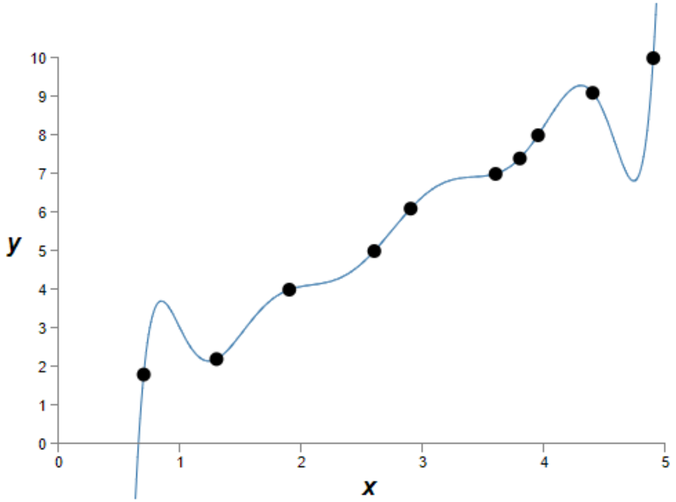
Yi Zhang (张亿)

International Center for Quantum Materials, School of Physics
Peking University, Beijing, 100871, China

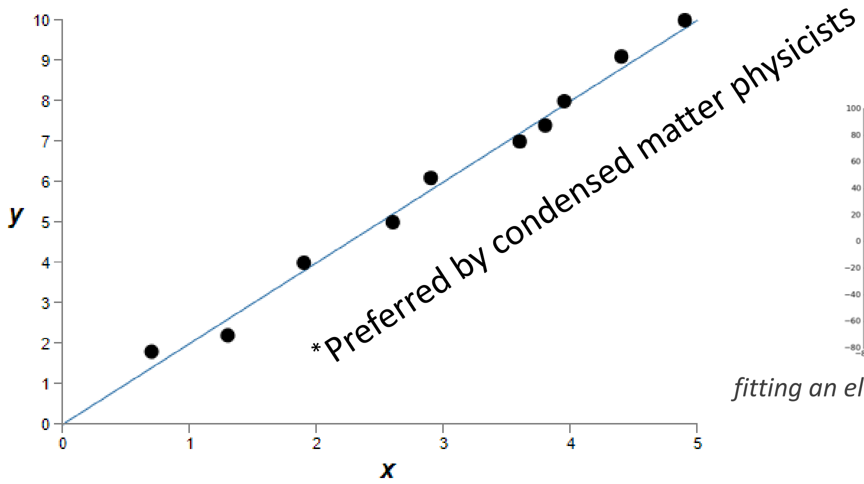
Email: frankzhangyi@pku.edu.cn

Overfitting

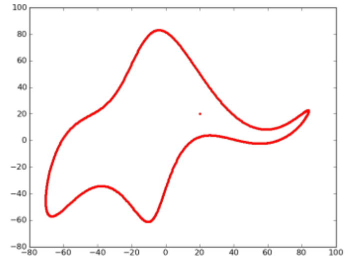
- We have a large number of fitting parameters, running the risk of **overfitting**.



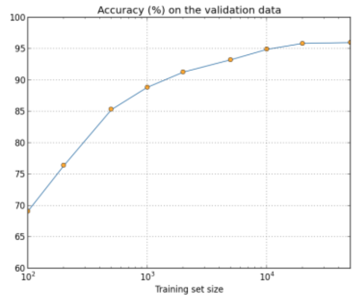
Global minimum of cost function
Sensitive to small input changes
Complex interpretation
Memorizing → poor generalizability



Local low plateau of cost function
Insensitive to small input changes
Simpler interpretation
Learning → good generalizability



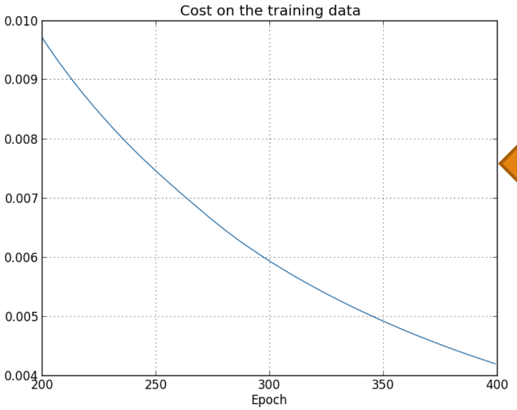
fitting an elephant with four parameters



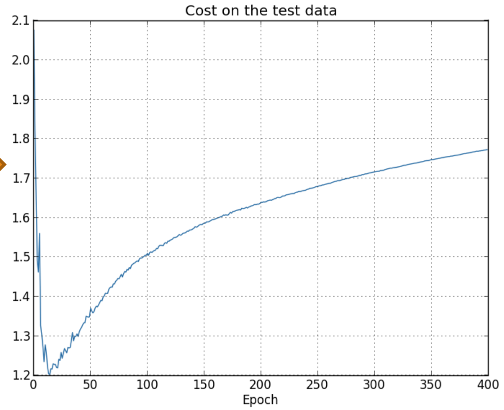
- Direct solution: reduce ANN complexity and increase sample number. (Noise can be useful sometimes.)

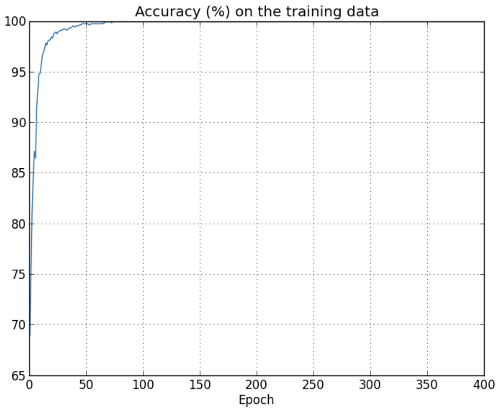
Overfitting

- Example: machine learning with only 1000 MNIST samples:

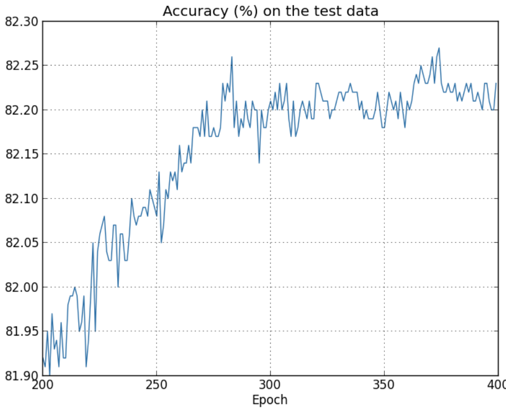


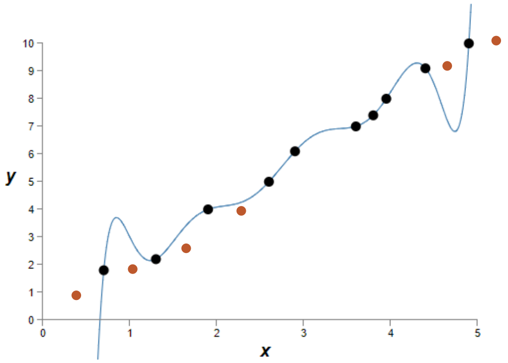
Deceptively good training on and only on the training data samples





Unsuccessful "problem sea tactics”





Global minimum of cost function
Sensitive to small input changes
Complex interpretation
Memorizing → poor generalizability
NOT reliable for unseen samples

For comparison, machine learning with 50,000 MNIST samples:

L2 Regularization

- Introduce the **regularization constant** (**weight decay**) λ:

$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2 = \frac{1}{2n}\sum_x \|y - a^L\|^2 + \frac{\lambda}{2n}\sum_w w^2$$

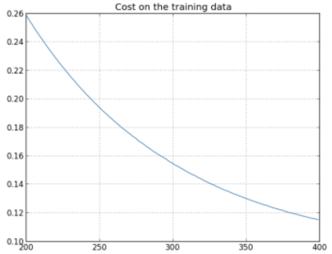
$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w$$

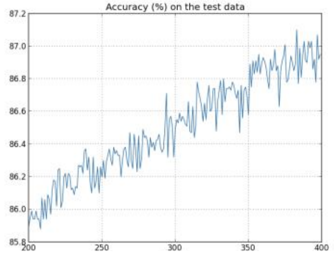
$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b}$$

$$w \rightarrow \left(1 - \frac{\eta\lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}$$

$$b \rightarrow b - \eta \frac{\partial C_0}{\partial b}$$

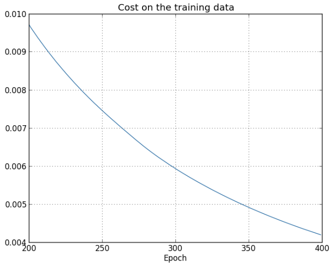
With L2 regularization:

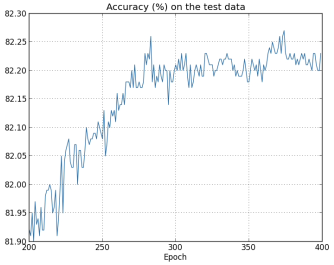




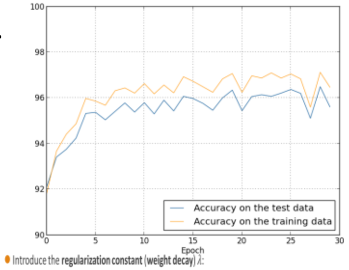
←1000 MNIST samples

Without L2 regularization:

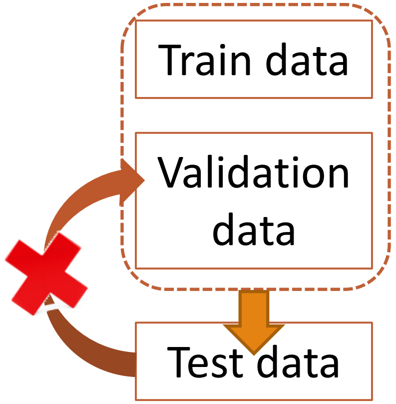




50,000 MNIST samples→



In practice, we separate the original training set into training data and **validation data**.



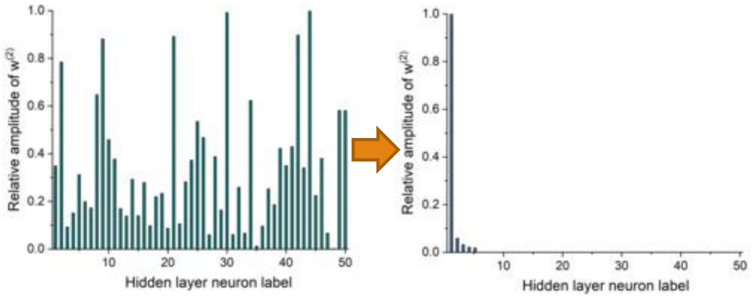
L1 Regularization and dropout

- Introduce the L1 regularization constant λ:

$$C = C_0 + \frac{\lambda}{n}\sum |w|$$

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n}\,\mathrm{sgn}(w)$$

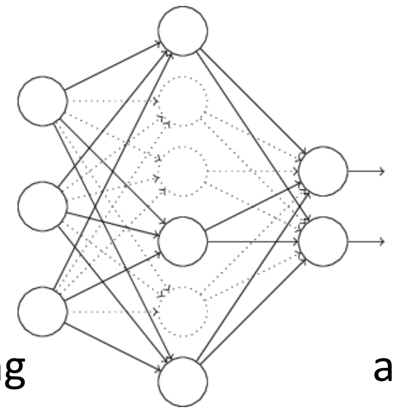
$$w \rightarrow w - \frac{\eta\lambda}{n}\mathrm{sgn}(w) - \eta\frac{\partial C_0}{\partial w}$$



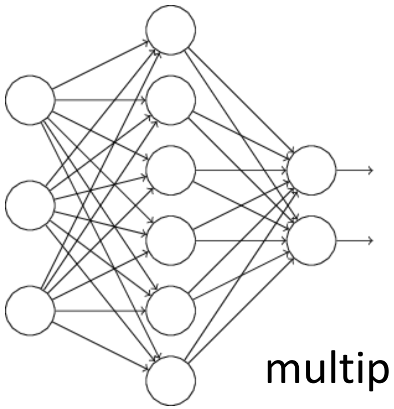
Eliminate less important connections

- **Dropout** is especially helpful for deep neural networks, together with L2 regularization:

Training: randomly ignore a fraction (dropout rate p) of nodes at training



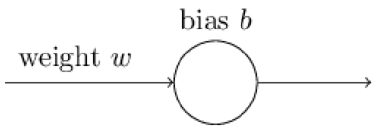
Testing: include all nodes but with an extra factor p to the weights



It is similar to training and then averaging over multiple neural networks.

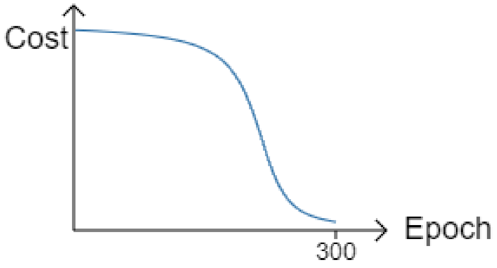
Cross-entropy cost function

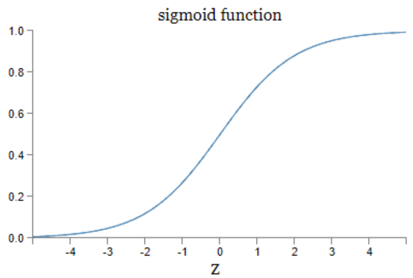
- The gradient vanishes if the output is badly wrong:



$$a = \sigma(z)$$
$$z = wx + b$$

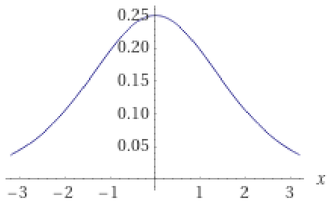
$$C = \frac{(y-a)^2}{2}$$
$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x$$
$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z)$$





$\sigma'(z)$ flats out at both ends:

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$



- Instead, we may use the **cross-entropy** cost function:

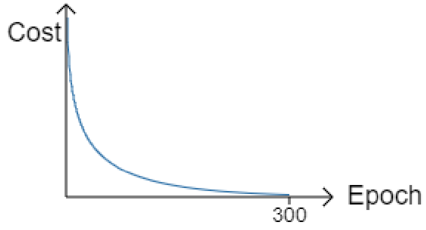
This heavily depends on the activation function of the output neurons.

$$C = -\frac{1}{n}\sum_x \left[y \ln a + (1-y)\ln(1-a)\right]$$
$$\frac{\partial C}{\partial w_j} = -\left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)}\right)\sigma'(z)x_j = x_j(\sigma(z)-y)$$
$$\frac{\partial C}{\partial b} = \sigma(z) - y$$

In terms of error:

$$\delta^L = a^L - y$$

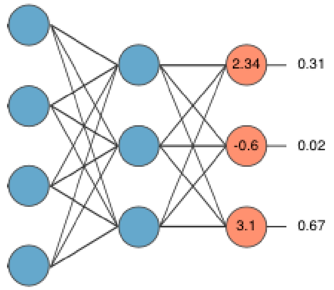
No more **learning slowdown**!



- More generally: $$C = -\frac{1}{n}\sum_x \sum_j \left[y_j \ln a_j^L + (1-y_j)\ln(1-a_j^L)\right]$$ for multiple outputs and samples

Softmax layer

- Normalize over multiple output neurons (good as probability distributions)



$$z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L$$

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}} \qquad \sum_j a_j^L = \frac{\sum_j e^{z_j^L}}{\sum_k e^{z_k^L}} = 1$$

Monotonicity: increasing z_j^L is guaranteed to increase a_j^L.

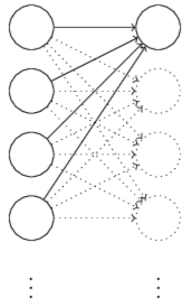
- Efficient cost function similar to cross entropy: (returns to cross entropy on binary Yes/No outputs)

$$C \equiv -\ln a_y^L \quad \Rightarrow \quad \frac{\partial C}{\partial b_j^L} = a_j^L - y_j \qquad \frac{\partial C}{\partial w_{jk}^L} = a_k^{L-1}(a_j^L - y_j)$$

- Q: what about optimizing the ANN for a regression problem (target label y is continuous) instead of a classification problem (target label $y = 0,1$)?

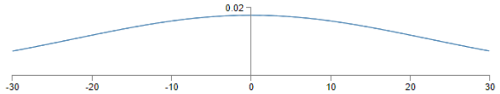
Initialization and gradient methods

- Initialization: make hidden neurons closer to neutrality initially (to avoid slowdown):

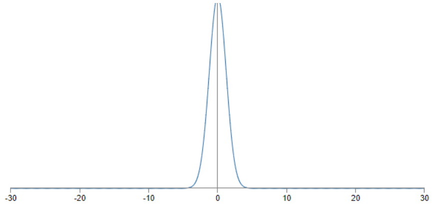


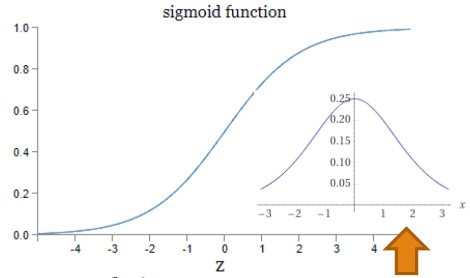
probability distribution of input into the neuron $z = \sum_j w_j x_j + b$

We don' want:



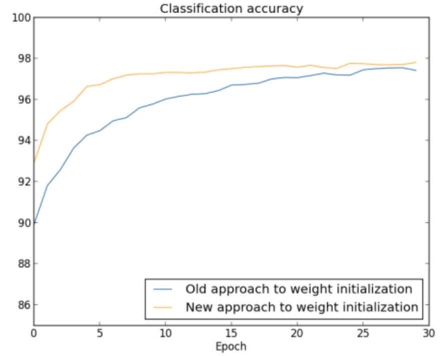
We want:





$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} w_{kj}^{l+1} \sigma'(z_j^l)$$

Therefore, we may initialize weights (biases) with standard deviation $\sim n_{in}^{-1/2}$ (~ 1).



- Momentum-dependent gradient descent:

friction

$$w \to w' = w - \eta \nabla C$$

$$v \to v' = \mu v - \eta \nabla C$$
$$w \to w' = w + v'$$

- Learning rate scheduler: reduce the learning rate when validation accuracy starts to stall.

The First ML code revisited

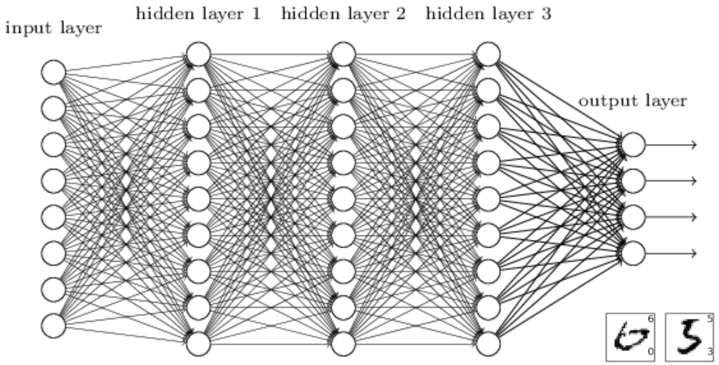
```python
def update_mini_batch(self, mini_batch, eta, lmbda, n):
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    for x, y in mini_batch:
        delta_nabla_b, delta_nabla_w = self.backprop(x, y)
        nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
        nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
    self.weights = [(1-eta*(lmbda/n))*w-(eta/len(mini_batch))*nw
                    for w, nw in zip(self.weights, nabla_w)]
    self.biases = [b-(eta/len(mini_batch))*nb
                   for b, nb in zip(self.biases, nabla_b)]
```

```python
class CrossEntropyCost(object):

    def fn(a, y):
        return np.sum(np.nan_to_num(-y*np.log(a)-(1-y)*np.log(1-a)))

    def delta(z, a, y):
        return (a-y)
```

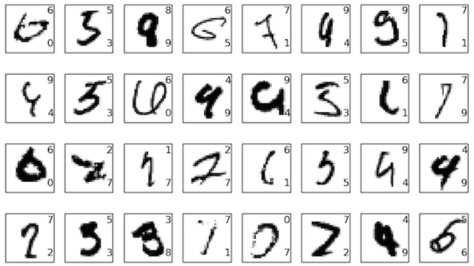
```python
def default_weight_initializer(self):
    self.biases = [np.random.randn(y, 1) for y in self.sizes[1:]]
    self.weights = [np.random.randn(y, x)/np.sqrt(x)
                    for x, y in zip(self.sizes[:-1], self.sizes[1:])]
```

```python
net = network2.Network([784, 100, 10], cost=network2.CrossEntropyCost)
net.large_weight_initializer()
net.SGD(training_data, 30, 10, 0.5, lmbda=5.0, evaluation_data=validation_data)
```

- With a bit of fine-tuning of the hyperparameters (100 hidden neurons, learning rate $\eta = 0.1$, L2 regularization $\lambda = 5.0$, 60 epochs), we can achieve ~98% accuracy on the MNIST test data.
- To push our accuracy over 99%, we need deep learning.



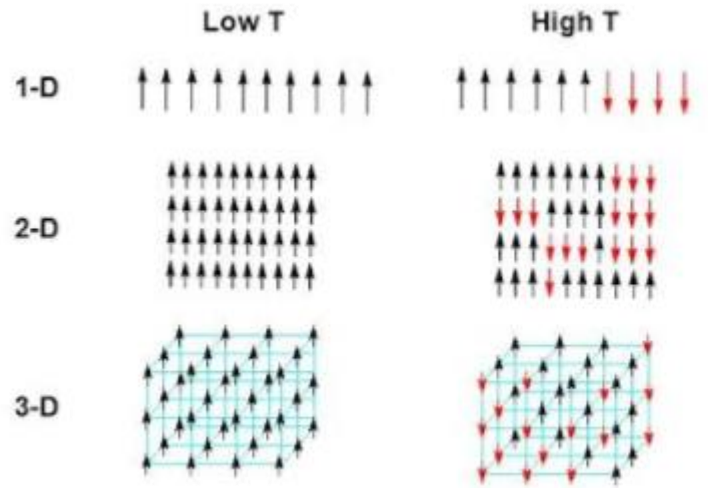
- Examples of mistakes made by ANN:



Ising model

- General Ising model:

$$H(\sigma) = -\sum_{i\,j} J_{ij}\sigma_i\sigma_j - \mu\sum_j h_j\sigma_j$$



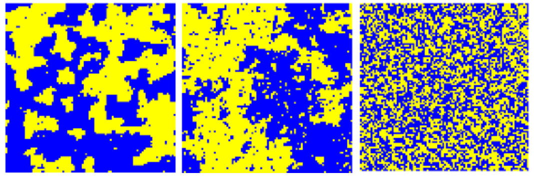
- Ising model with nearest-neighbor ferromagnetic interactions:

$$H(\sigma) = -J\sum_{\langle i\,j\rangle}\sigma_i\sigma_j$$

- Configuration probability:

$$P_\beta(\sigma) = \frac{e^{-\beta H(\sigma)}}{Z_\beta} \qquad Z_\beta = \sum_\sigma e^{-\beta H(\sigma)} \qquad \beta = (k_B T)^{-1}$$

- Phases: (described by spontaneous broken symmetry and order parameter)



ferromagnetic phase at low temperature (small T)

paramagnetic phase at high temperature (high T)

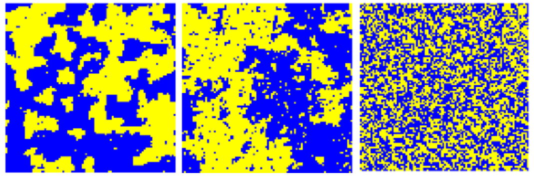
Play it yourself: https://www-np.acs.i.kyoto-u.ac.jp/~harada/education/demos/mc_ising/index-en.html

Markov Chain Monte Carlo method

- 1. We start from / are currently at a spin state μ with energy H_μ. ⇦ Markov chain
- 2. Shuffle the spin configuration for a new state ν, calculate its energy H_ν.
- 3. If the new state energy is less, keep the new state ν as the current state.
- 4. If the new state energy is more, accept the new state ν with probability $e^{-\beta(H_\nu - H_\mu)}$.
- 5. Repeat and evaluate observables at intervals larger than the auto-correlation length.

Detailed balance:

$$A(\mu,\nu) = \begin{cases} e^{-\beta(H_\nu - H_\mu)}, & \text{if } H_\nu - H_\mu > 0, \\ 1 & \text{otherwise.} \end{cases}$$



Play it yourself: https://www-np.acs.i.kyoto-u.ac.jp/~harada/education/demos/mc_ising/index-en.html