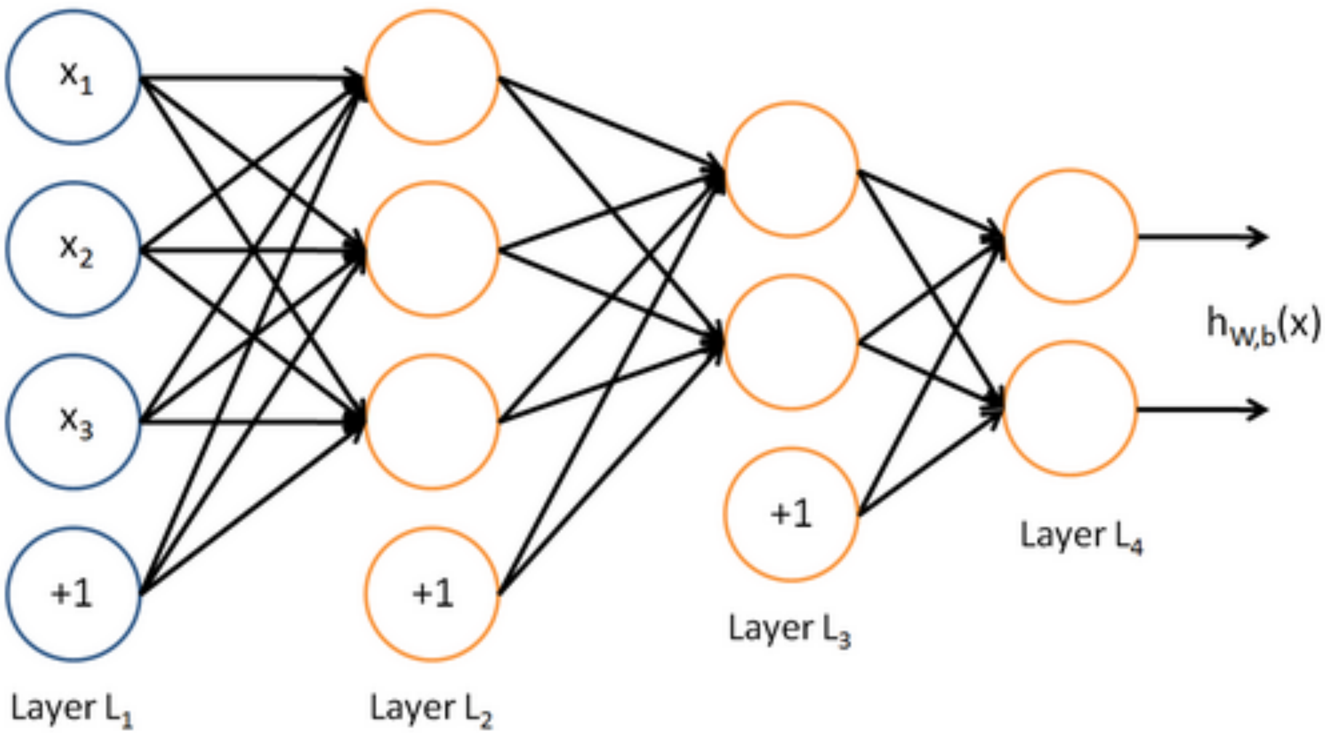
Feedforward Neural Networks Training

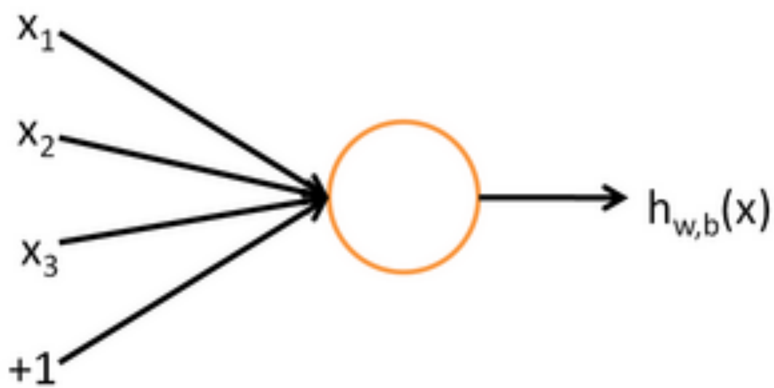
Introduction

Back propagation is main algorithm used to training feed forward NN.GA PSO ACO



Introduction

Neuron Structure



$$h_{W,b}(x) = f(W^T x) = f(\textstyle\sum_{i=1}^{3} W_i x_i + b)$$

Activation Function

$$f(z) = \frac{1}{1+\exp(-z)}.$$

Sigmoid Function

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

Tanh Function

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$$

Leaky ReLU Function

Back Propagation

Back Propagation

Gradient Decent

More details ———> Xidian University *Engineering Optimization Methods*

Chain Rule

Error Back Propagation

Back Propagation

Cost(Error) Function

Squared-error

$$J(W,b;x,y) = \frac{1}{2}\left\| h_{W,b}(x) - y \right\|^2.$$

$$\begin{aligned} J(W,b) &= \left[\frac{1}{m} \sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)}) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \\ &= \left[\frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \left\| h_{W,b}(x^{(i)}) - y^{(i)} \right\|^2 \right) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \end{aligned}$$

Cross entropy//use with softmax together

$$C = -\sum_j t_j \log y_j$$

target value

$$y_i = \frac{e^{z_i}}{\sum e^{z_j}}$$

Back Propagation

Gradient

Squared error

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \; \frac{1}{2} \left\|y - h_{W,b}(x)\right\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

$$\begin{aligned}
\delta_i^{(n_l)} &= \frac{\partial}{\partial z_i^{n_l}} J(W,b;x,y) = \frac{\partial}{\partial z_i^{n_l}} \frac{1}{2} \left\|y - h_{W,b}(x)\right\|^2 \\
&= \frac{\partial}{\partial z_i^{n_l}} \frac{1}{2} \sum_{j=1}^{S_{n_l}} (y_j - a_j^{(n_l)})^2 = \frac{\partial}{\partial z_i^{n_l}} \frac{1}{2} \sum_{j=1}^{S_{n_l}} (y_j - f(z_j^{(n_l)}))^2 \\
&= -(y_i - f(z_i^{(n_l)})) \cdot f'(z_i^{(n_l)}) = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})
\end{aligned}$$

Cross entropy

$$\frac{\partial C}{\partial z_i} = \sum_j \frac{\partial C}{\partial y_j} \frac{\partial y_j}{\partial z_i} = y_i - t_i$$

Back Propagation

Back Propagation

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

$$\begin{aligned}
\delta_i^{(n_l-1)} &= \frac{\partial}{\partial z_i^{n_l-1}} J(W,b;x,y) = \frac{\partial}{\partial z_i^{n_l-1}} \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 = \frac{\partial}{\partial z_i^{n_l-1}} \frac{1}{2} \sum_{j=1}^{S_{n_l}} (y_j - a_j^{(n_l)})^2 \\
&= \frac{1}{2} \sum_{j=1}^{S_{n_l}} \frac{\partial}{\partial z_i^{n_l-1}} (y_j - a_j^{(n_l)})^2 = \frac{1}{2} \sum_{j=1}^{S_{n_l}} \frac{\partial}{\partial z_i^{n_l-1}} (y_j - f(z_j^{(n_l)}))^2 \\
&= \sum_{j=1}^{S_{n_l}} -(y_j - f(z_j^{(n_l)})) \cdot \frac{\partial}{\partial z_i^{(n_l-1)}} f(z_j^{(n_l)}) = \sum_{j=1}^{S_{n_l}} -(y_j - f(z_j^{(n_l)})) \cdot f'(z_j^{(n_l)}) \cdot \frac{\partial z_j^{(n_l)}}{\partial z_i^{(n_l-1)}} \\
&= \sum_{j=1}^{S_{n_l}} \delta_j^{(n_l)} \cdot \frac{\partial z_j^{(n_l)}}{\partial z_i^{n_l-1}} = \sum_{j=1}^{S_{n_l}} \left(\delta_j^{(n_l)} \cdot \frac{\partial}{\partial z_i^{n_l-1}} \sum_{k=1}^{S_{n_l-1}} f(z_k^{n_l-1}) \cdot W_{jk}^{n_l-1} \right) \\
&= \sum_{j=1}^{S_{n_l}} \delta_j^{(n_l)} \cdot W_{ji}^{n_l-1} \cdot f'(z_i^{n_l-1}) = \left(\sum_{j=1}^{S_{n_l}} W_{ji}^{n_l-1} \delta_j^{(n_l)} \right) f'(z_i^{n_l-1})
\end{aligned}$$

Back Propagation

Partial Derivative

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$
$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}.$$

Update Parameters

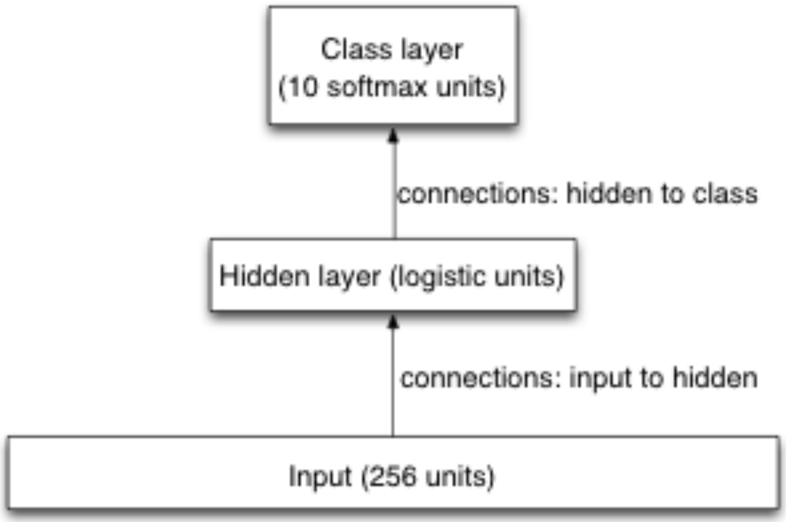
$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$
$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)$$

$$W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right]$$
$$b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} \Delta b^{(l)} \right]$$

Back Propagation

Programing Practice

Coursera, Hinton, Neural Networks for Machine Learning, Week 9
Program Assignment 3: Optimization and Generalization



"In this assignment, you're going to train a simple Neural Network, for recognizing handwritten digits."

Back Propagation

Full Batch(Global Minimum)

Accumulated error back propagation, performance badly when the dataset is highly redundant.

Mini-Batch

Less Computation is used updating the weights.

Computing the gradient for many cases simultaneously uses matrix multiplies which are very efficient, especially on GPUs

Online(Local Minimum,Stochastic Gradient Descent)

One sample, one time.

Back Propagation

Ways to reduce overfitting

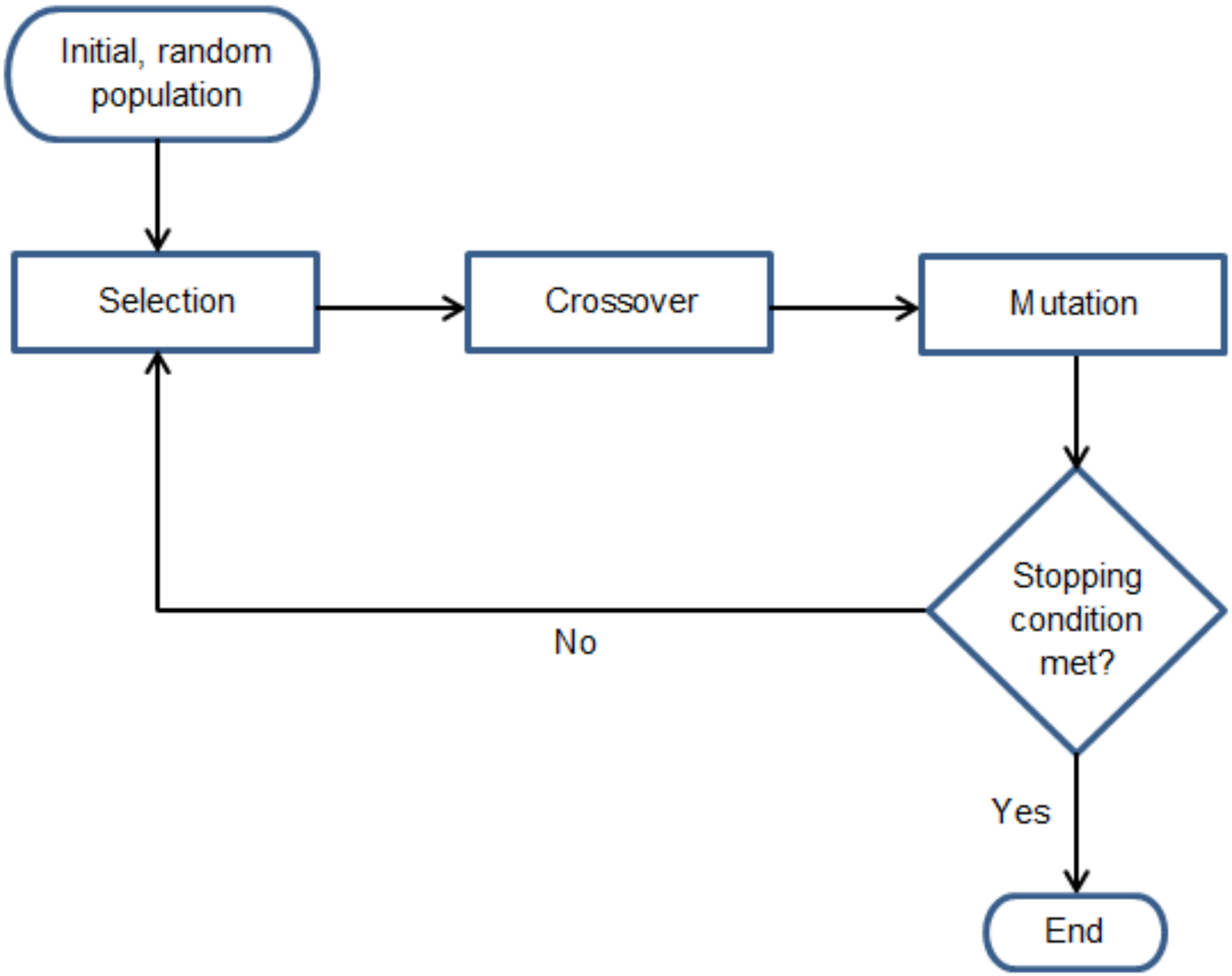
Get more data
Use a model that has right capacity.
Average many different models.
Use a single neural network architecture, but average the predictions

Limit the capacity of neural net

Weight-decay
Weight-sharing
Early stopping
Model averaging
Bayesian fitting of neural nets
Dropout
Generative pre-training

Genetic Algorithm

Genetic Algorithm



Genetic Algorithm

Crossover

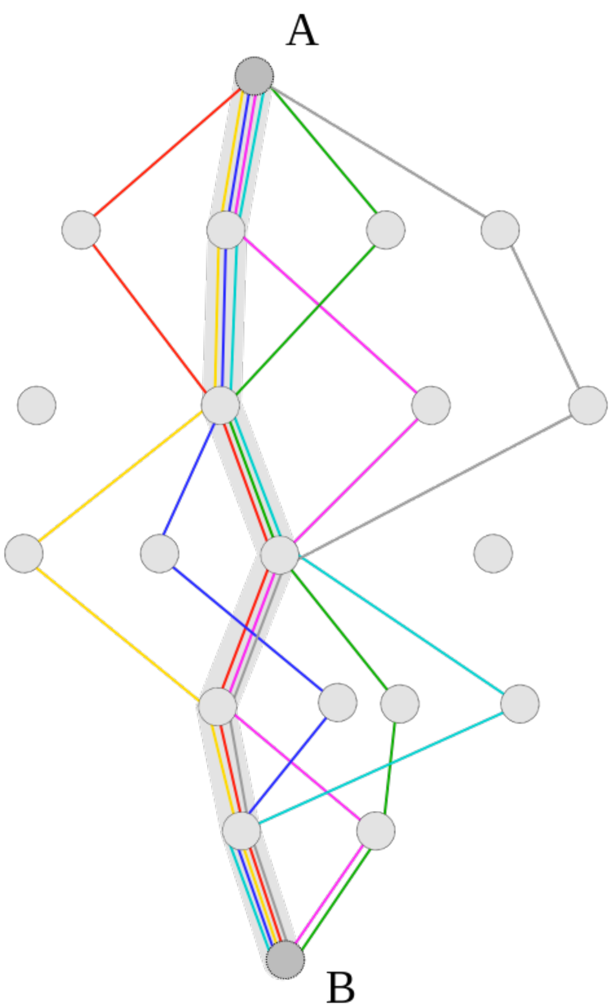
			*			*		
Patent1	3	5	7	2	1	6	4	8
Patent2	2	5	7	6	8	1	3	4
Child	5	8	7	2	1	6	3	4

Mutation

			*			*		
Patent	3	5	7	2	1	6	4	8
Child	3	5	6	2	1	7	4	8

Ant Colony Optimization Algorithm

Ant Colony Optimization Algorithm



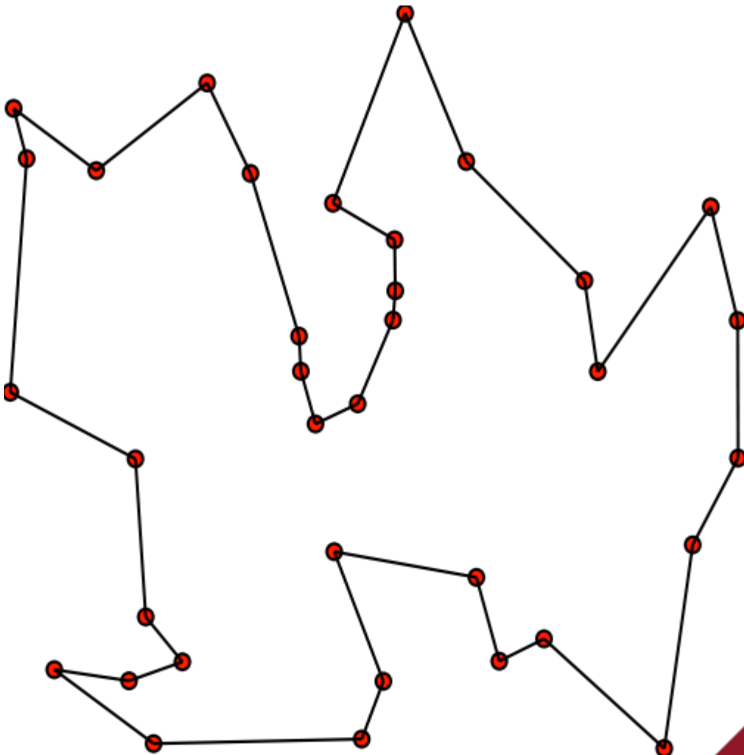
$$\rho_{ij}^{k}(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}\cdot[\eta_{ik}(t)]^{\beta}}{\sum\limits_{s\subset allowed_{k}}[\tau_{is}(t)]^{\alpha}\cdot[\eta_{is}(t)]^{\beta}}, \text{若} j\in allowed \\ 0, \text{否则}\end{cases}$$

$$\tau_{ij}(t+1)=(1-\rho_1)\tau_{ij}(t)+\rho_1\Delta\tau_{ij}(t)$$

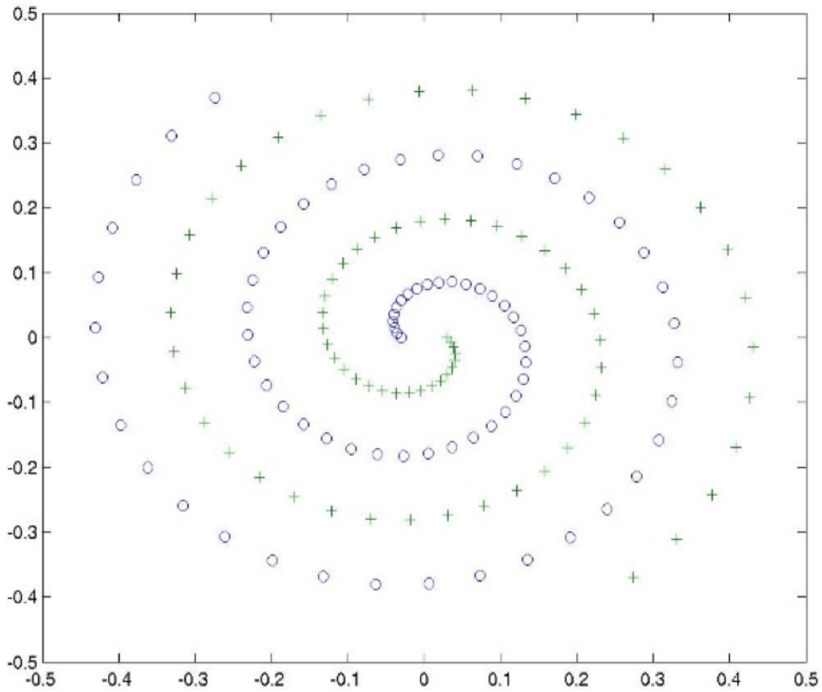
Traveling Salesman Problem

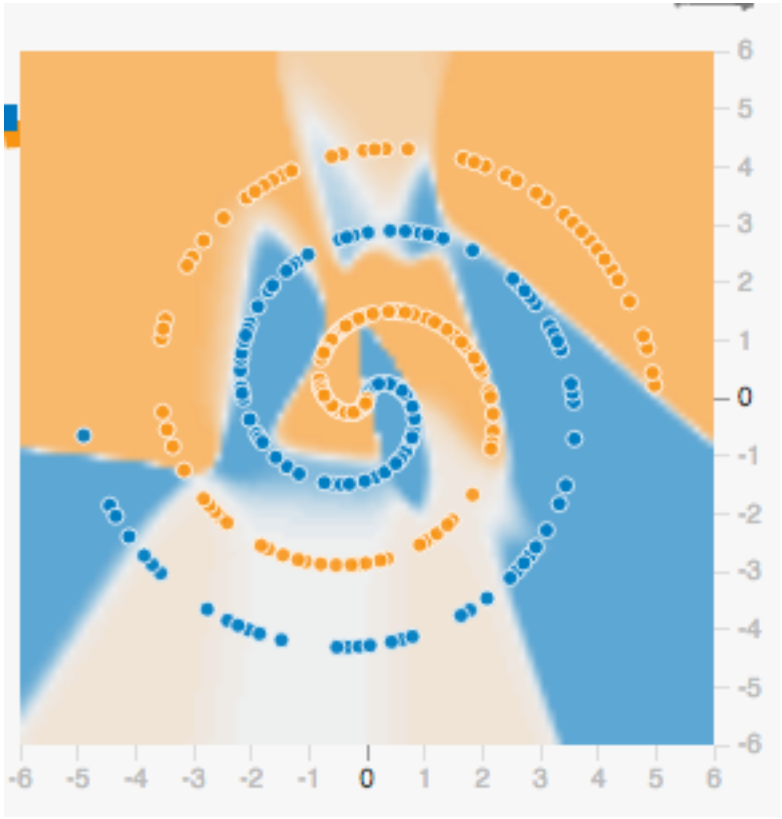
```
NAME: berlin52
TYPE: TSP
COMMENT: 52 locations in Berlin (Groetschel)
DIMENSION: 52
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 565.0 575.0
2 25.0 185.0
3 345.0 750.0
4 945.0 685.0
5 845.0 655.0
6 880.0 660.0
7 25.0 230.0
8 525.0 1000.0
9 580.0 1175.0
10 650.0 1130.0
11 1605.0 620.0
12 1220.0 580.0
13 1465.0 200.0
14 1530.0 5.0
15 845.0 680.0
16 725.0 370.0
17 145.0 665.0
18 415.0 635.0
19 510.0 875.0
20 560.0 365.0
21 300.0 465.0
22 520.0 585.0
23 480.0 415.0
24 835.0 625.0
25 975.0 580.0
```

Berlin 52



Two Spirals Problem





Genetic Algorithm

GA tutorial in chinese

http://blog.csdn.net/v_JULY_v/article/details/6132775?123

ACO tutorial in chinese

http://www.nocow.cn/index.php/%E8%9A%81%E7%BE%A4%E4%BC%98%E5%8C%96%E7%AE%97%E6%B3%95

TSP Berlin52 and Two Spirals dataset

https://github.com/KuangRD/Computational-Intelligence

TensorFlow Playground

http://playground.tensorflow.org/

Other Algorithm

Simulated Annealing(SA)

Particle Swarm Optimization(PSO)

Differential Evolution(DE)

Reference

1, UFLDL
2, Hinton, Neural Networks for Machine Learning
3, Furuzuki, Computational Intelligence
4, Zhou Zhihua, Machine Learning
5, Wikipedia

Thank you for listening!