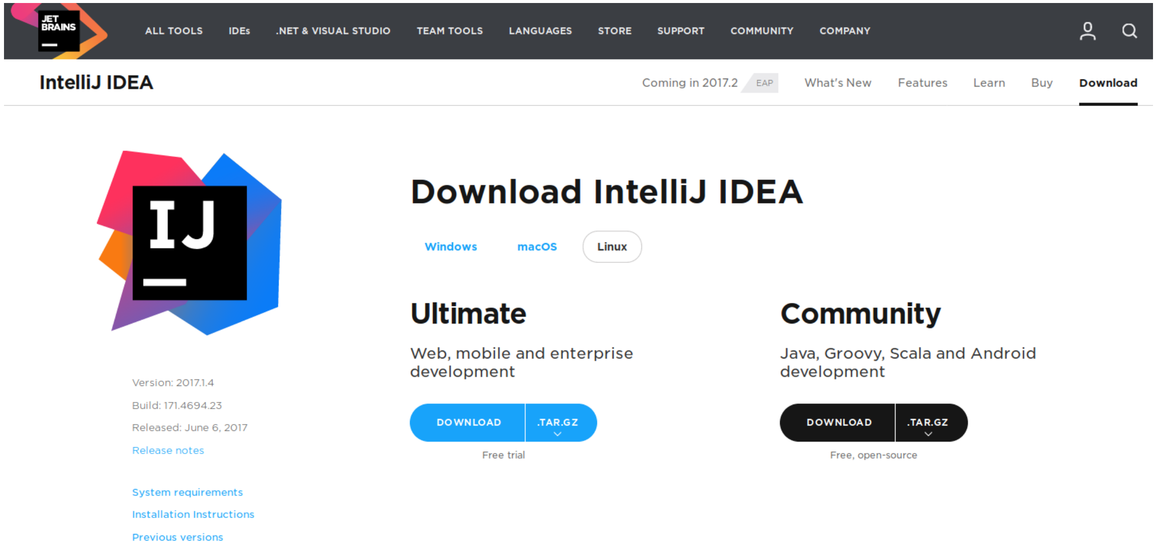
Algorithms for Machine Learning

Outline

- Getting Started
- The Need for Training in Machine Learning
- Supervised/Unsupervised Learning
- Machine Learning Application Flow
- Theories and Algorithms of Neural Networks
- Summary

Getting Started

- JDK 1.8 (will cover lambda functions)
- IntelliJ IDEA >= 14.1 (download a new one!)
- DL4J and Maven (will be used in deep learning chapters)

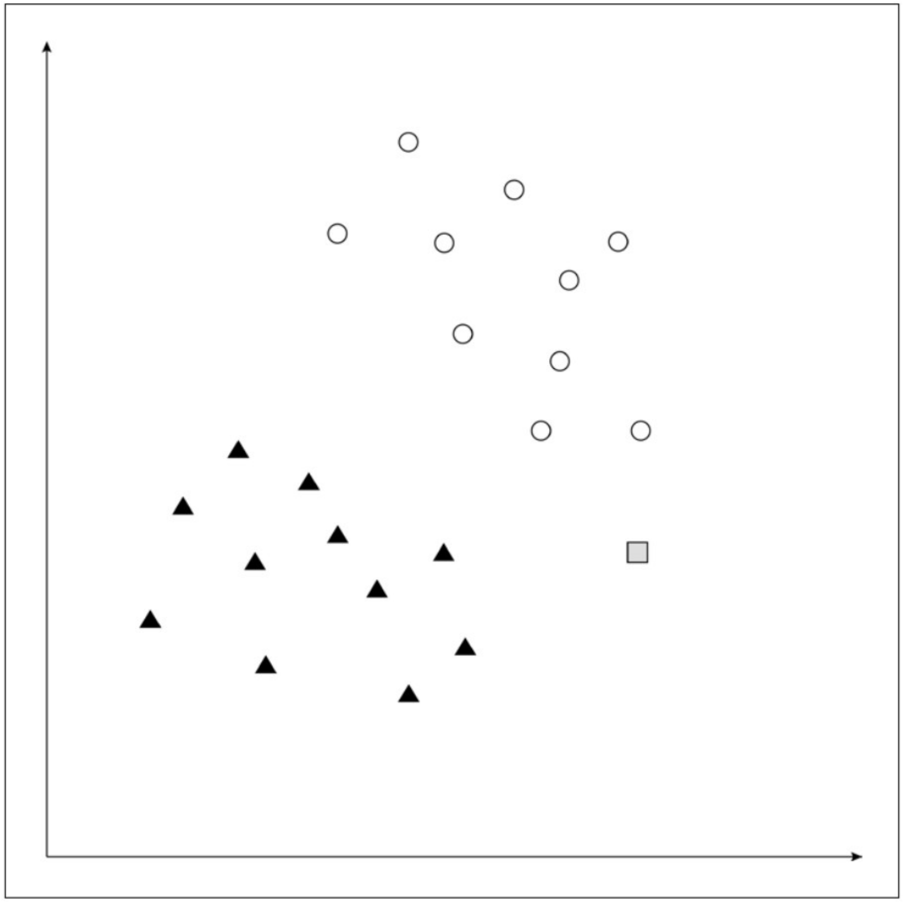


Outline

- Getting Started
- The Need for Training in Machine Learning
- Supervised/Unsupervised Learning
- Machine Learning Application Flow
- Theories and Algorithms of Neural Networks
- Summary

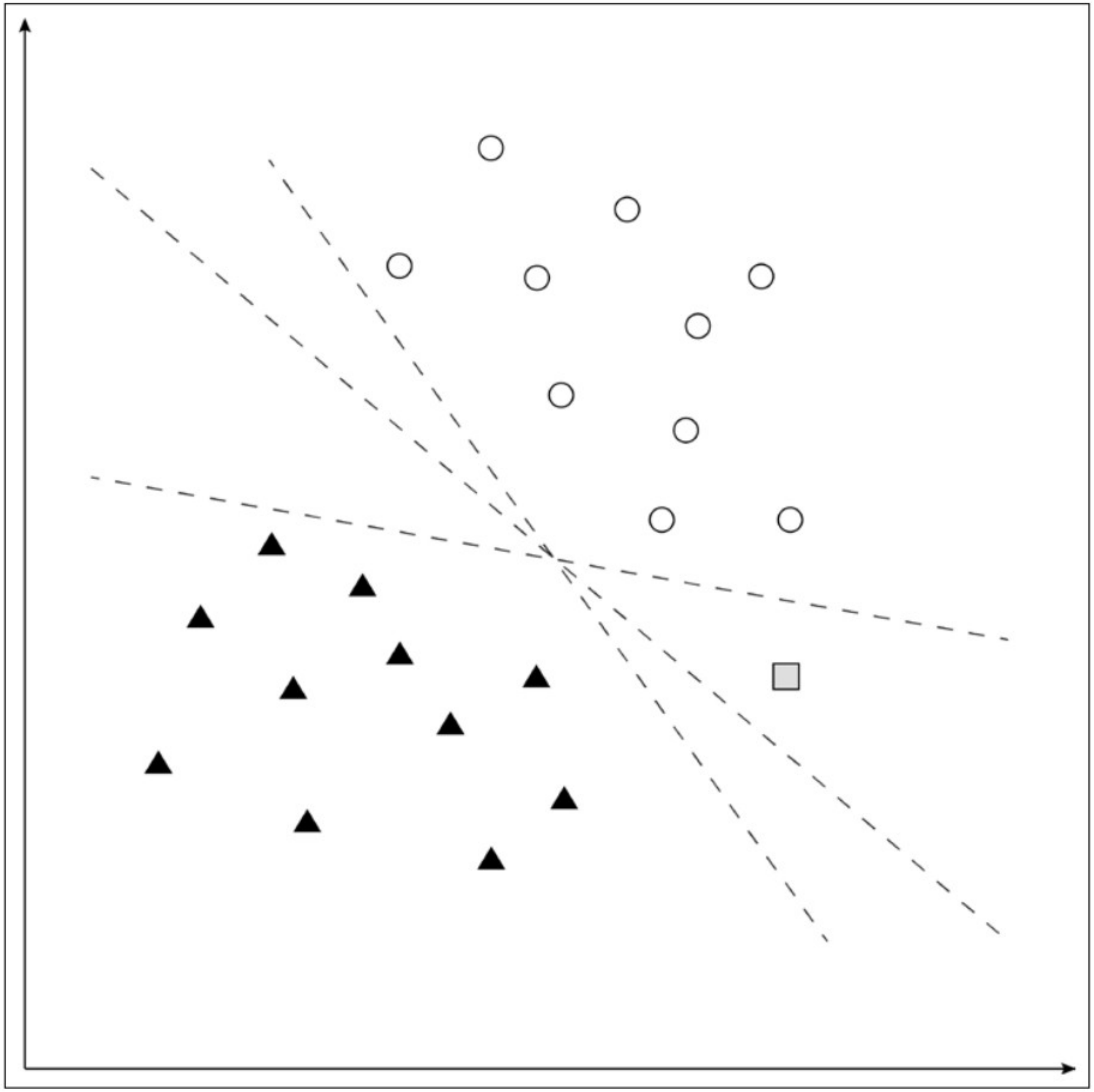
Training in Machine Learning

- Machine learning reaches an answer by recognizing and sorting out patterns from the given learning data
  - Difficult to sort out data appropriately



Training in Machine Learning

- Not so easy to define the boundary
- Nonlinear boundary?



Decision Boundary

- In machine learning, what a machine does in training is choose the most likely boundary from these possible patterns
- Decision boundary is not necessarily a linear or nonlinear boundary
  - Think about multi-dimensional classification problem
  - Can also be a hyperplane

Outline

- Getting Started
- The Need for Training in Machine Learning
- Supervised/Unsupervised Learning
- Machine Learning Application Flow
- Theories and Algorithms of Neural Networks
- Summary

Hard Problem

- Millions of boundaries even for a simple classification problem
- If we could properly sort out patterns in the known data, it doesn't mean that unknown data can also be classified in the same pattern
  - Increase the percentage of correct pattern categorization, how?
    - Machine learning algorithms

Supervised and Unsupervised

- Difference: labeled data or unlabeled data
- Supervised learning: give past correct answer
- Unsupervised learning
  - Learn patterns and rules from dataset
  - Grasp the structure of the data

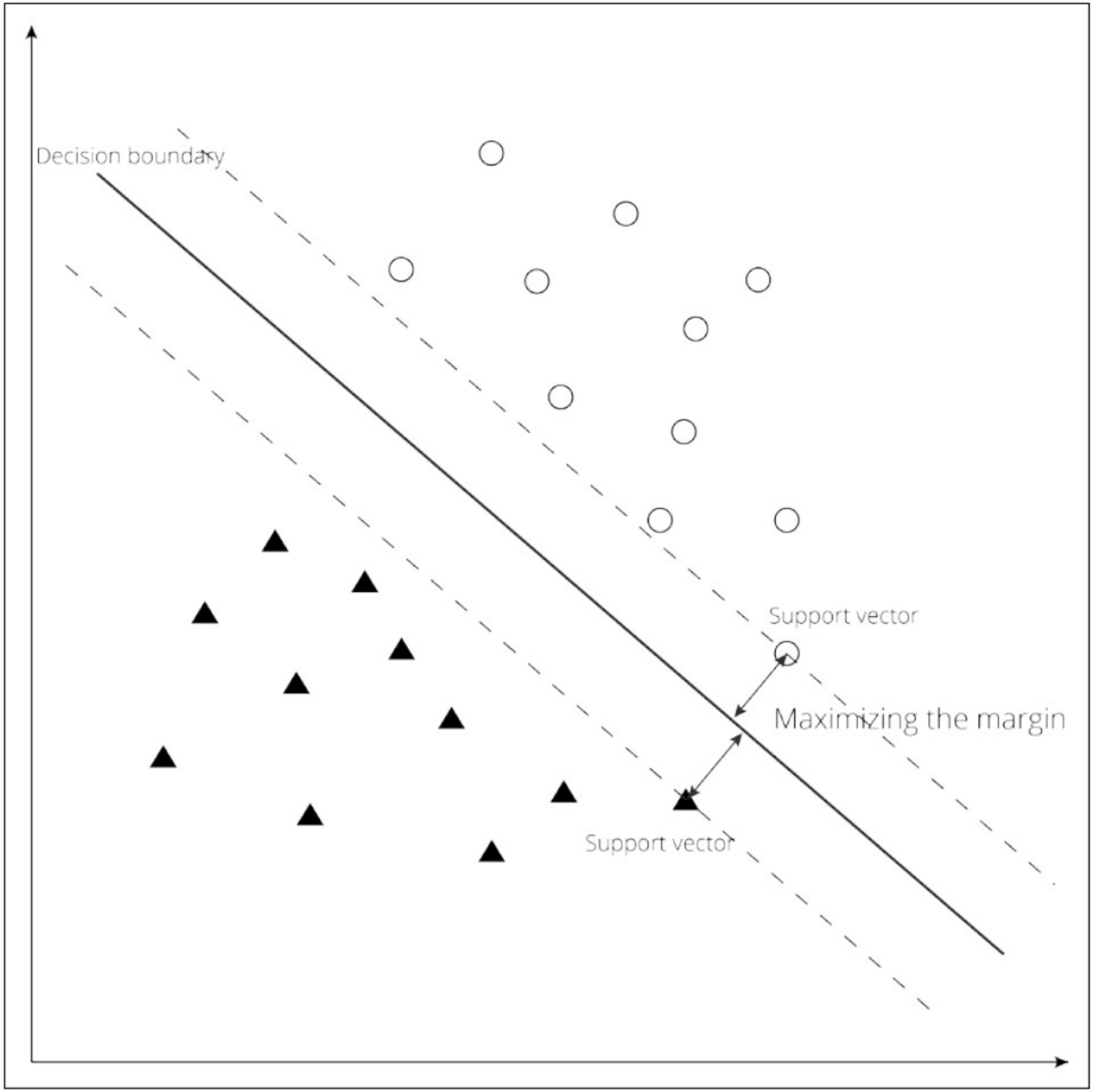
Some Representative Algorithms

- Support Vector Machine (SVM)
- Hidden Markov Model (HMM)
- Neural Networks
- Logistic Regression
- Reinforcement Learning

Support Vector Machine (SVM)

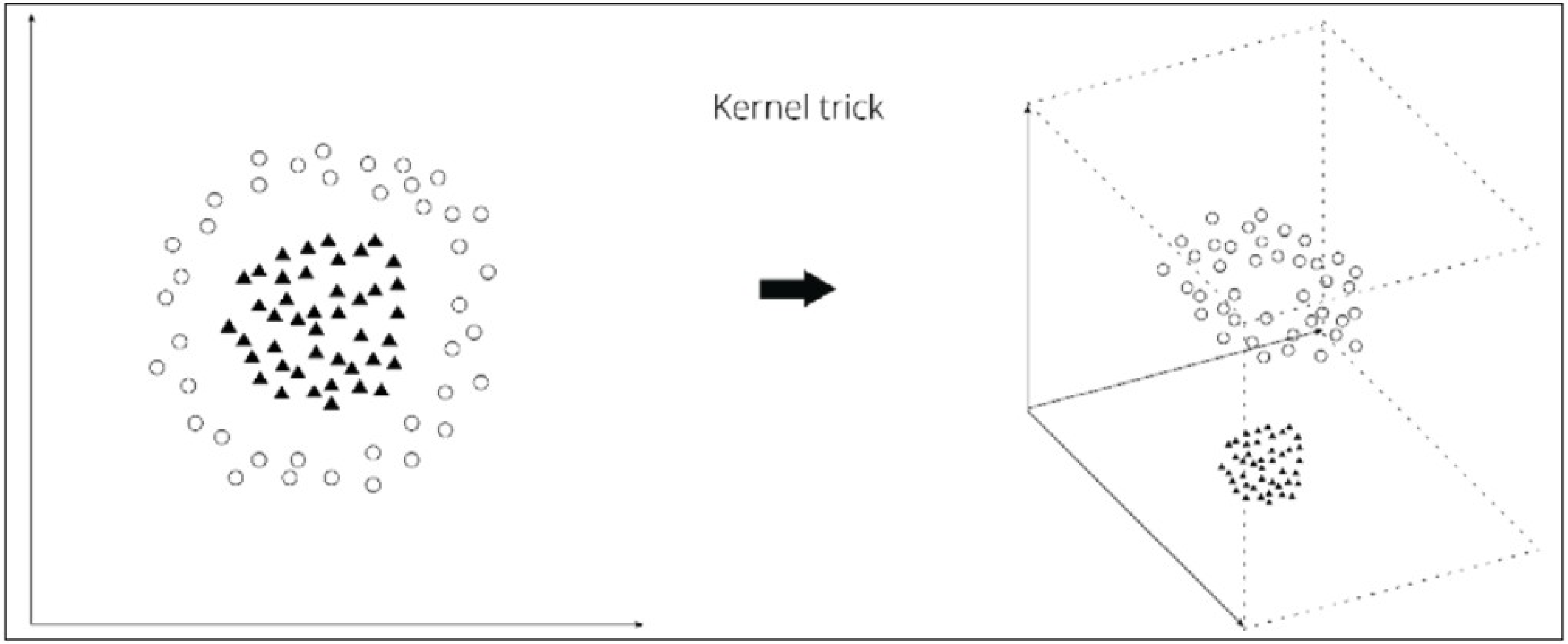
- Data from each category located the closest to other categories is marked as support vectors
- Decision boundary is determined using the vectors so that the sum of the Euclidean distance from each marked data and the boundary is maximized
  - Maximizing the margin

Support Vector Machine (SVM)



Kernel Trick

- Maps data to a higher dimensional space so that it can be classified linearly (the number of calculations often increases)

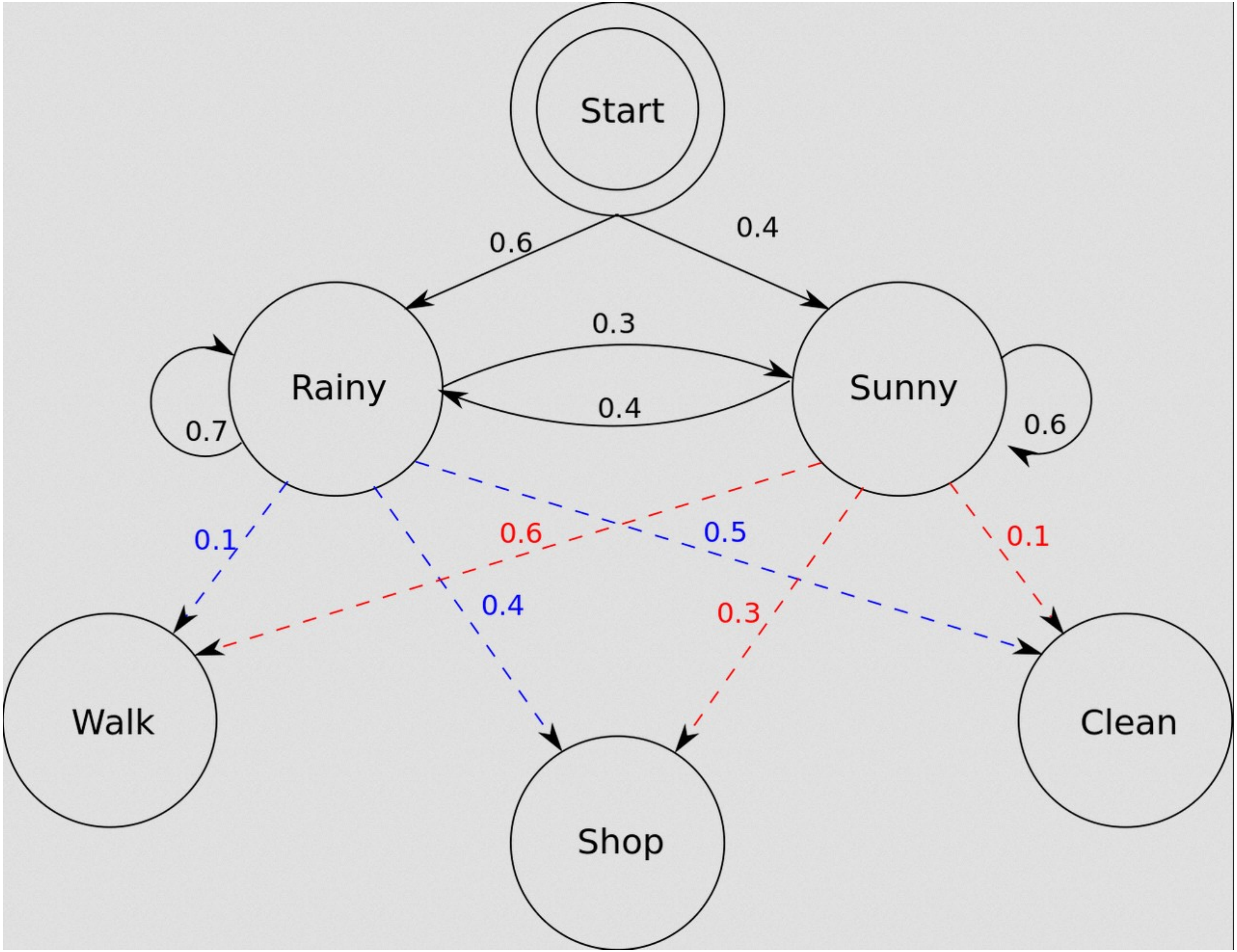


Kernel Trick

- Make SVM popular
- Only useful in SVM? No!!

Hidden Markov Model (HMM)

- Unsupervised training method that assumes data follows the Markov process
- The Markov process is a stochastic process in which a future condition is decided solely on the present value and is not related to the past condition

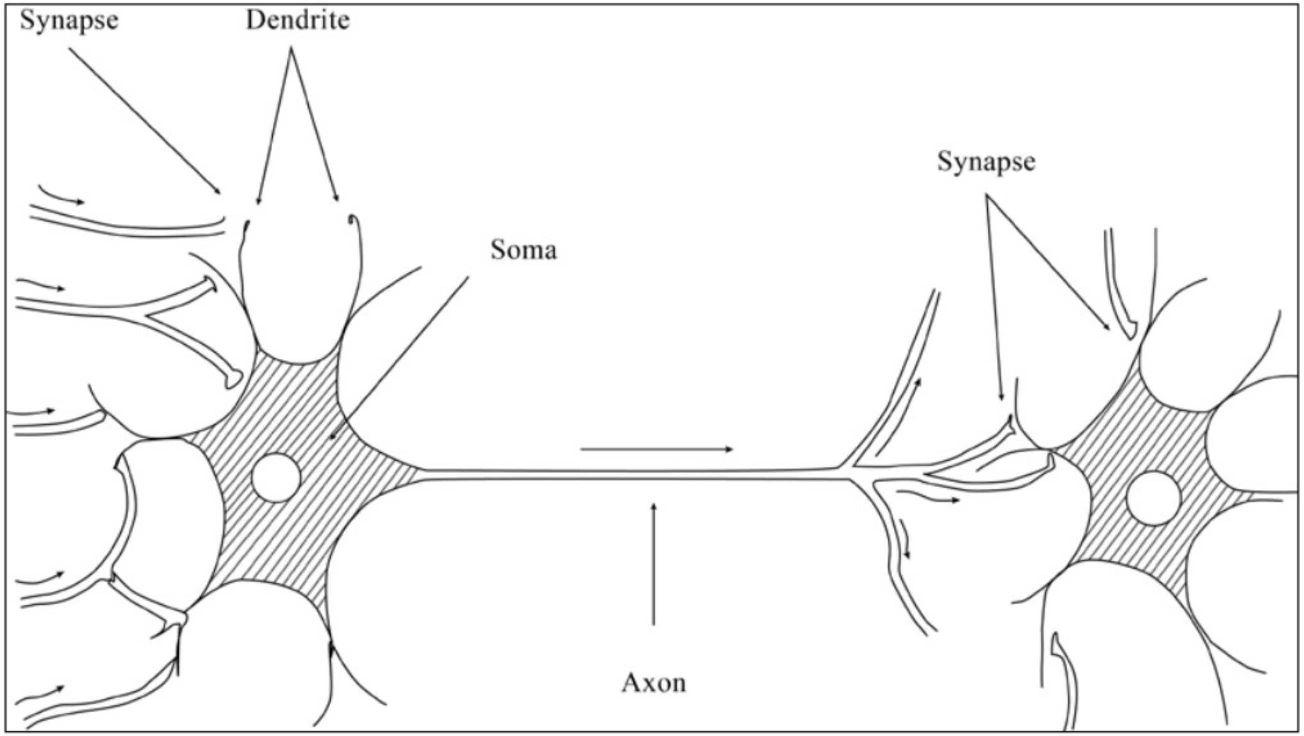

Start
0.6
0.4
Rainy
Sunny
0.3
0.4
0.7
0.6
0.1
0.6
0.5
0.1
0.4
0.3
Walk
Shop
Clean

Hidden Markov Model (HMM)

- HMM is often used to analyze a base sequence
  - AGCT
- Also used in time sequence patterns
  - Syntax analysis of natural language processing (NLP)
  - Sound signal processing

Neural Networks

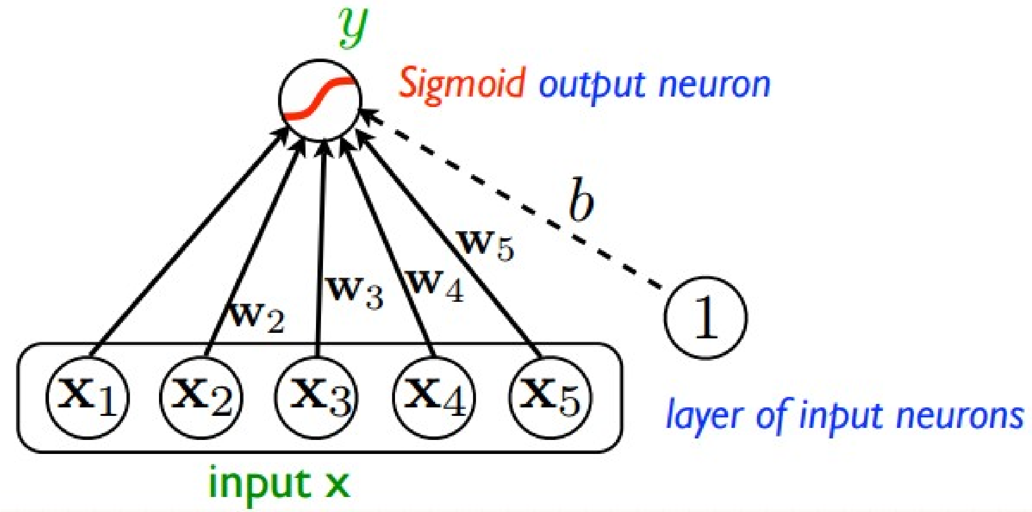
- Imitate the structure of a human brain

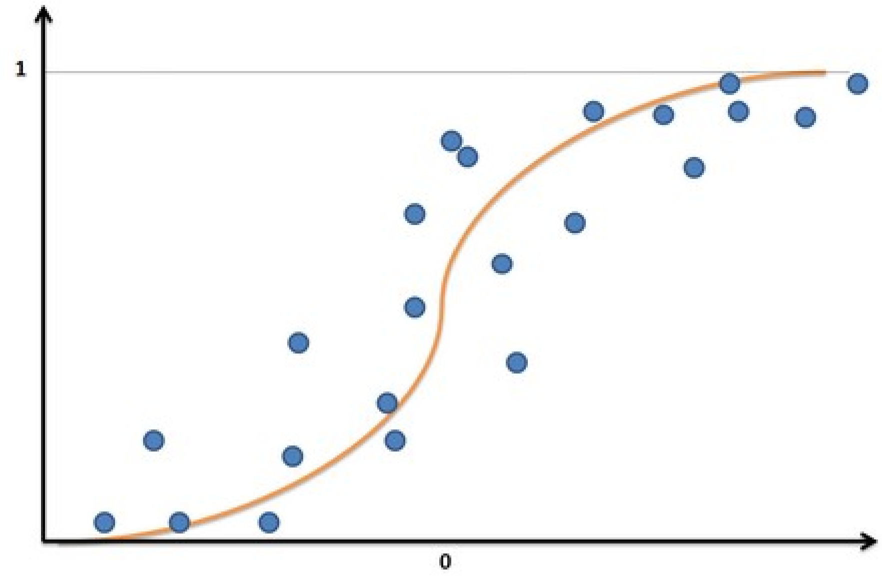


- Distinguish things based on how electrical stimulations are transmitted

Logistic Regression

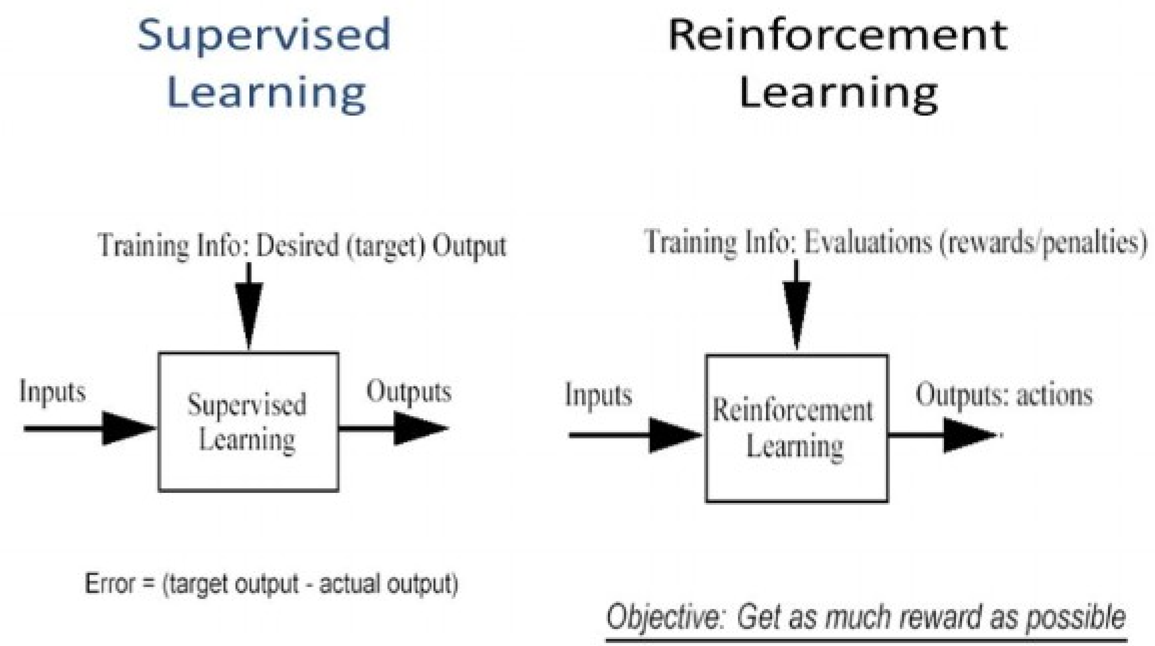
- Statistical regression models of variables with the Bernoulli distribution
- Can be thought of as one of the neural networks when you look at its formula



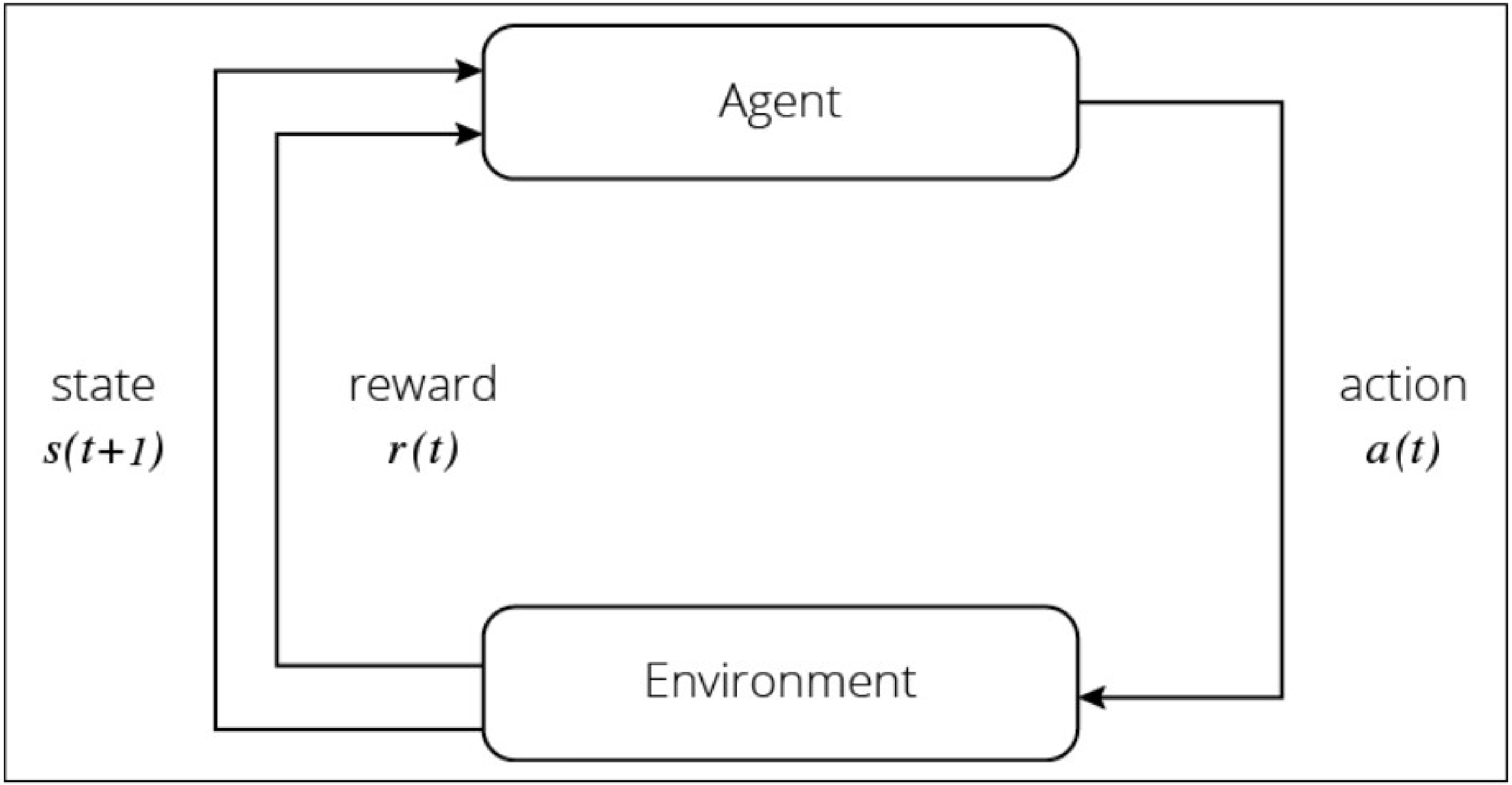


Reinforcement Learning

- Some categorize reinforcement learning as unsupervised learning
- Some declare that all three learning algorithms, supervised learning, unsupervised learning, and reinforcement learning



Reinforcement Learning

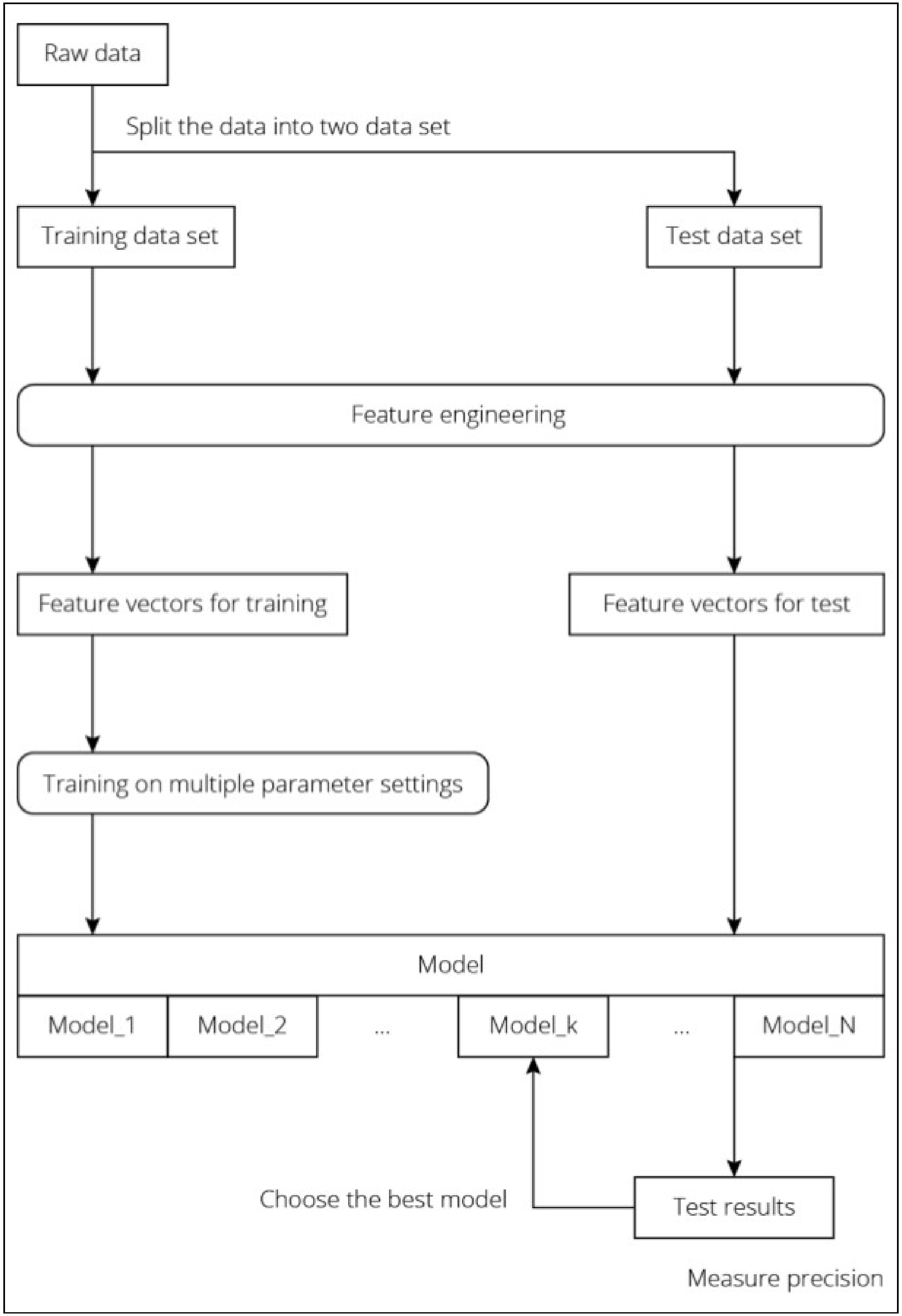


Outline

- Getting Started
- The Need for Training in Machine Learning
- Supervised/Unsupervised Learning
- Machine Learning Application Flow
- Theories and Algorithms of Neural Networks
- Summary

Application Flow

- Weakest point of machine learning: feature engineering
  - Deciding which features are to be created from raw data
- Tasks for preprocessing to build an appropriate classifier
  - Deciding the machine learning method
  - Deciding the features
  - Deciding the model parameters setting

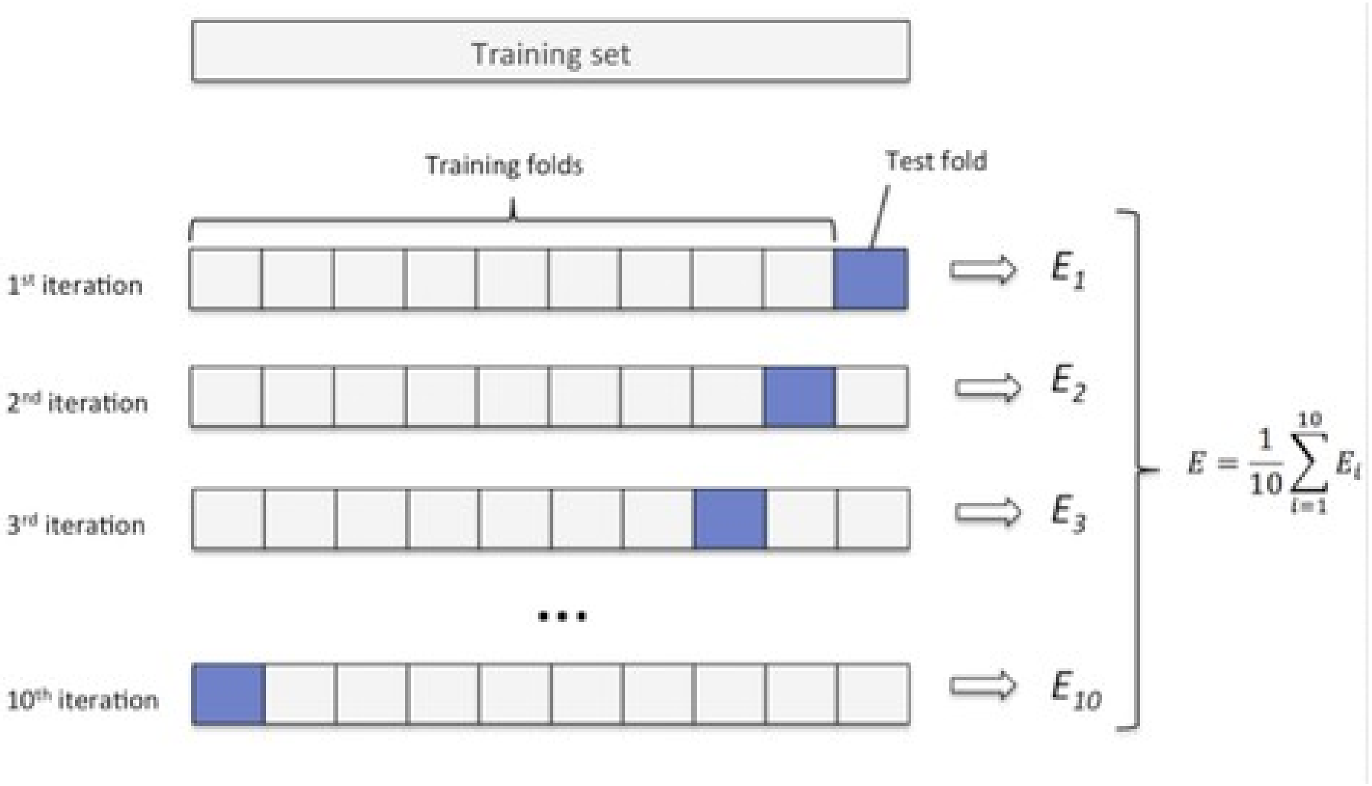

Raw data
Split the data into two data set
Training data set
Test data set
Feature engineering
Feature vectors for training
Feature vectors for test
Training on multiple parameter settings
Model
Model_1
Model_2
...
Model_k
...
Model_N
Choose the best model
Test results
Measure precision

Overfitting Problem

- Incorrect optimization by classifying noisy data blended into a training dataset
  - Most for a data in the real world also contains noises, making it difficult to classify data into proper patterns
- Incorrect optimizing by classifying data that is characteristic only in a training dataset
- K-fold cross-validation to help

Smart Computing
SAIC
智慧計算實驗室

K-fold Cross-validation



Outline

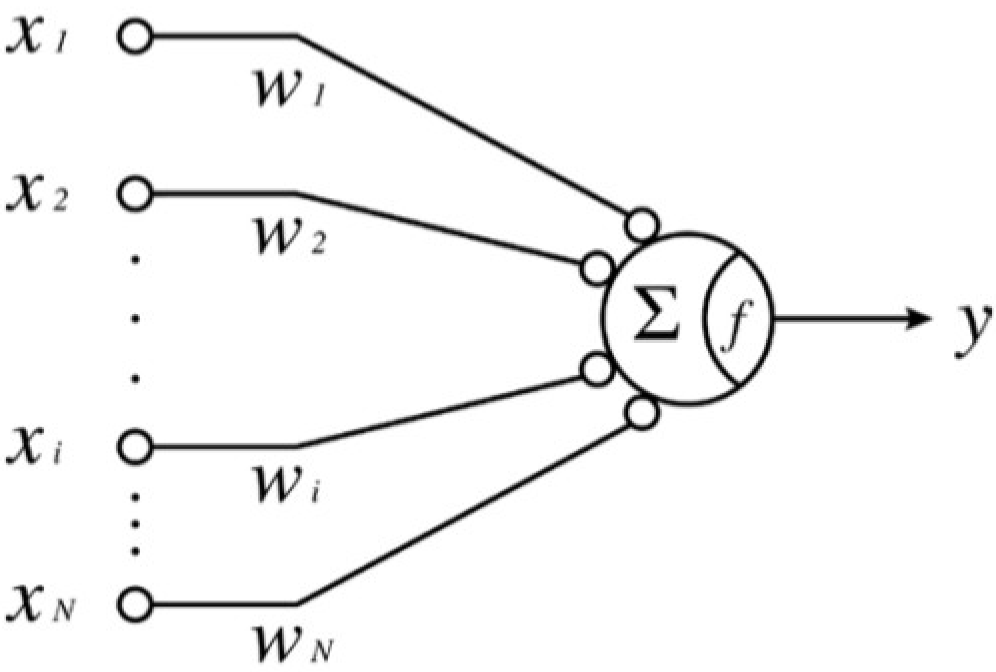
- Getting Started
- The Need for Training in Machine Learning
- Supervised/Unsupervised Learning
- Machine Learning Application Flow
- Theories and Algorithms of Neural Networks
- Summary

Theories and Algorithms of NNs

- Perceptrons
- Logistic regression
- Multi-class logistic regression
- Multi-layer perceptrons

Perceptrons

- Single-layer neural networks



$$y(x) = f\left(w^T x\right)$$

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$

f (∗) is called the step function

Let t be the value of the labeled data. Then, the formula can be represented as follows:

$$t \in \{-1, 1\}$$

If some labeled data belongs to class 1, C_1, we have $t = 1$. If it belongs to class 2, C_2, we have $t = -1$. Also, if the input data is classified correctly, we get:

$$\begin{cases} w^T x_n > 0 \ \ where\, x_n \in C_1 \\ w^T x_n < 0 \ \ where\, x_n \in C_2 \end{cases}$$

So, putting these equations together, we have the following equation of properly classified data:

$$w^T x_n t_n > 0$$

Therefore, you can increase the predictability of Perceptron by minimizing the following function:

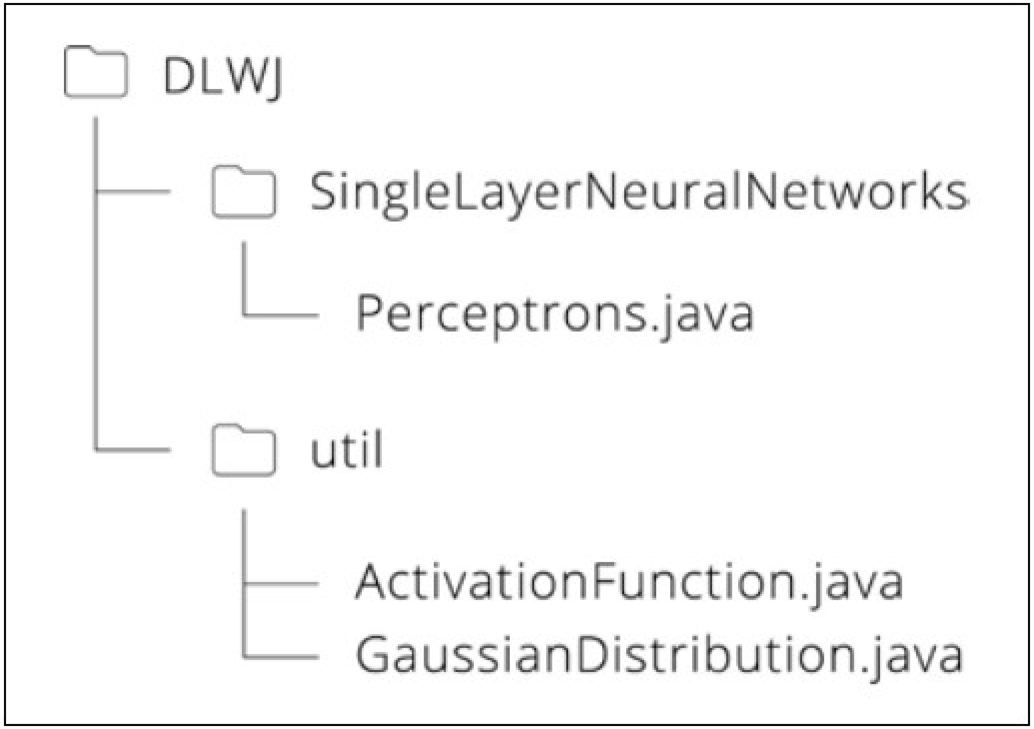
$$E(w) = -\sum_{n \in M} w^T x_n t_n$$

Here, E is called the error function. M shows the set of misclassification. To minimize the error function, gradient descent, or steepest descent, an optimization algorithm is used to find a local minimum of a function using gradient descent. The equation can be described as follows:

$$w^{(k+1)} = w^{(k)} - n\nabla E(w) = w^{(k)} + \eta x_n t_n$$

Here, η is the learning rate, a common parameter of the optimization algorithm that adjusts the learning speed, and k shows the number of steps of the algorithm.

The Code


DLWJ
SingleLayerNeuralNetworks
Perceptrons.java
util
ActivationFunction.java
GaussianDistribution.java

Performance Measurement

	p_predicted	**n_predicted**
p_actual	True positive (TP)	False negative (FN)
n_actual	False positive (FP)	True negative (TN)

The three indicators are shown below:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

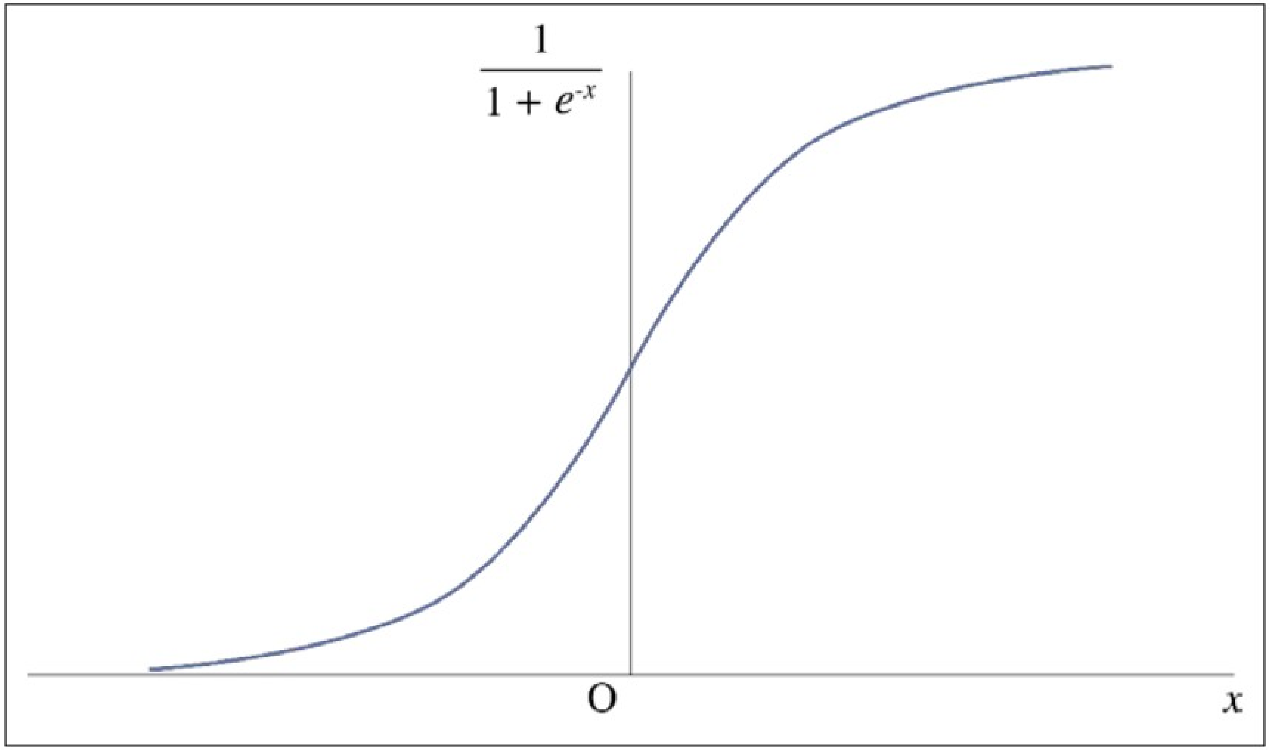
$$Recall = \frac{TP}{TP + FN}$$

Logistic Regression

- Sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

The graph of this function can be illustrated as follows:



Output of Logistic Regression

- Can be regarded as the posterior probability for each class

$$p(C=1 \mid x) = y(x) = \sigma\left(w^T x + b\right)$$

$$p(C=0 \mid x) = 1 - p(C=1 \mid x)$$

These equations can be combined to make:

$$p(C=t \mid x) = y^t (1-y)^{1-t}$$

Smart Computing 智慧計算實驗室 SAIC

Likelihood Function

- Estimates the maximum likelihood of the model parameters

$$L(w,b)=\prod_{n=1}^{N} y_n^{t_n}\left(1-y_n\right)^{1-t_n}$$

Where:

$$y_n = p\left(C=1 \mid x_n\right)$$

Math Product Problem

- Take the logarithm (log) of the likelihood function
- Negative log likelihood function
  - cross-entropy error function

$$E(w,b) = -In\,L(w,b) = -\sum_{n=1}^{N}\left\{t_n In\,y_n + (1-t_n)\,In(1-y_n)\right\}$$

Optimize the Model

- By computing the gradients of the model parameters, w and b
- Gradient Descent

$$\frac{\partial E(w,b)}{\partial w} = -\sum_{n=1}^{N}(t_n - y_n)x_n$$

$$\frac{\partial E(w,b)}{\partial b} = -\sum_{n=1}^{N}(t_n - y_n)$$

With these equations, we can update the model parameters as follows:

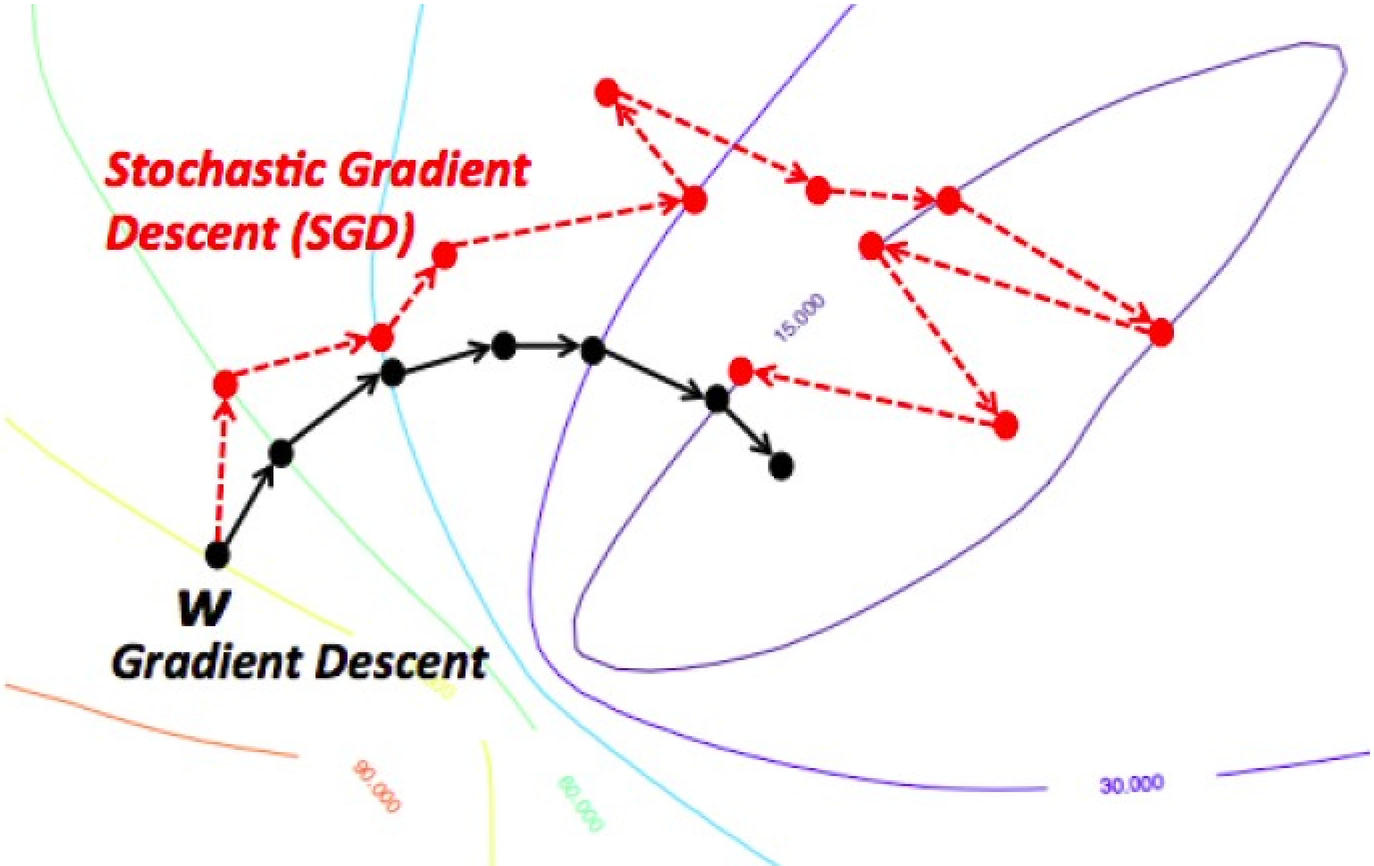
$$w^{(k+1)} = w^{(k)} - \eta \frac{\partial E(w,b)}{\partial w} = w^{(k)} + \eta \sum_{n=1}^{N}(t_n - y_n)x_n$$

$$b^{(k+1)} = b^{(k)} - \eta \frac{\partial E(w,b)}{\partial b} = b^{(k)} + \eta \sum_{n=1}^{N}(t_n - y_n)$$

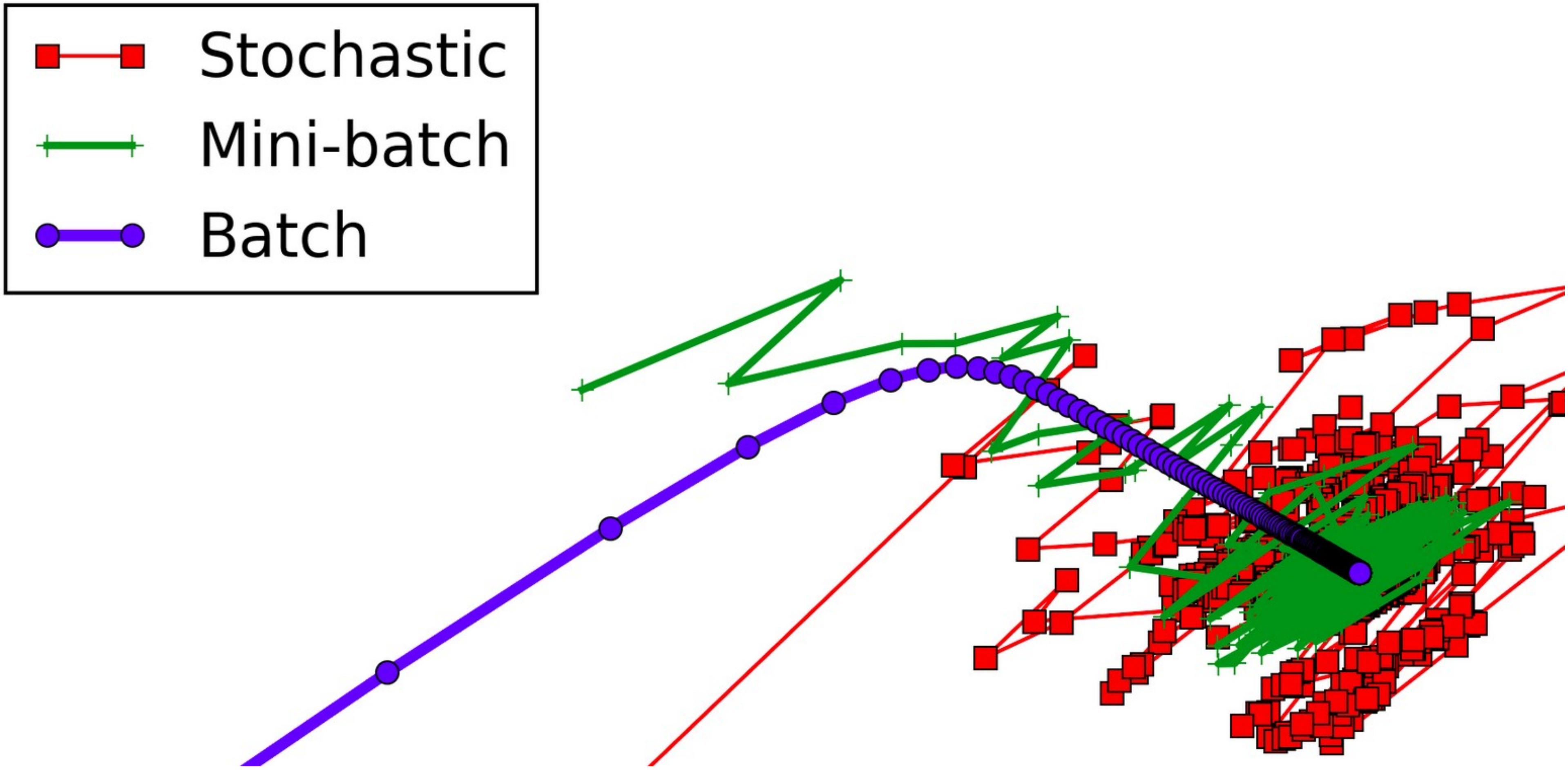
Problem of Gradient Descent

- Calculate the sum of all the data to compute the gradients
- Stochastic gradient descent (SGD)
  - Partially picks up some data from the dataset,
  - Computes the gradients by calculating the sum only with picked data
  - Renews the parameters
- SGD using a mini-batch is sometimes called mini-batch stochastic gradient descent (MSGD)

Effect of SGD



Batch, Mini-batch, Single



Multi-class Logistic Regression

- Posterior probability of each class using softmax function

$$p\left(C=k\mid x\right)=y_k\left(x\right)=\frac{\exp\left(w_k^T x+b_k\right)}{\sum_{j=1}^{K}\exp\left(w_j^T x+b_j\right)}$$

With this, the same as two-class cases, you can get the likelihood function and the negative log likelihood function as follows:

$$L\left(W,b\right)=\prod_{n=1}^{N}\prod_{k=1}^{K}y_{nk}^{t_{nk}}$$

$$E\left(W,b\right)=-In\,L\left(W,b\right)=-\sum_{n=1}^{N}\sum_{k=1}^{K}t_{nk}In\,y_{nk}$$

Smart Computing 智慧計算實驗室 SAIC

The Gradients

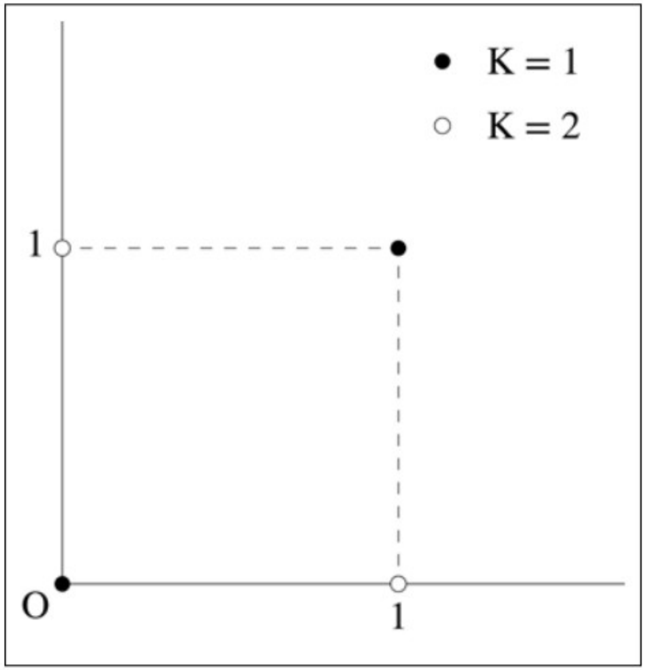
$$\frac{\partial E}{\partial w_j} = -\sum_{n=1}^{N} \left(t_{nj} - y_{nj} \right) x_n$$

$$\frac{\partial E}{\partial b_j} = -\sum_{n=1}^{N} \left(t_{nj} - y_{nj} \right)$$

The Code

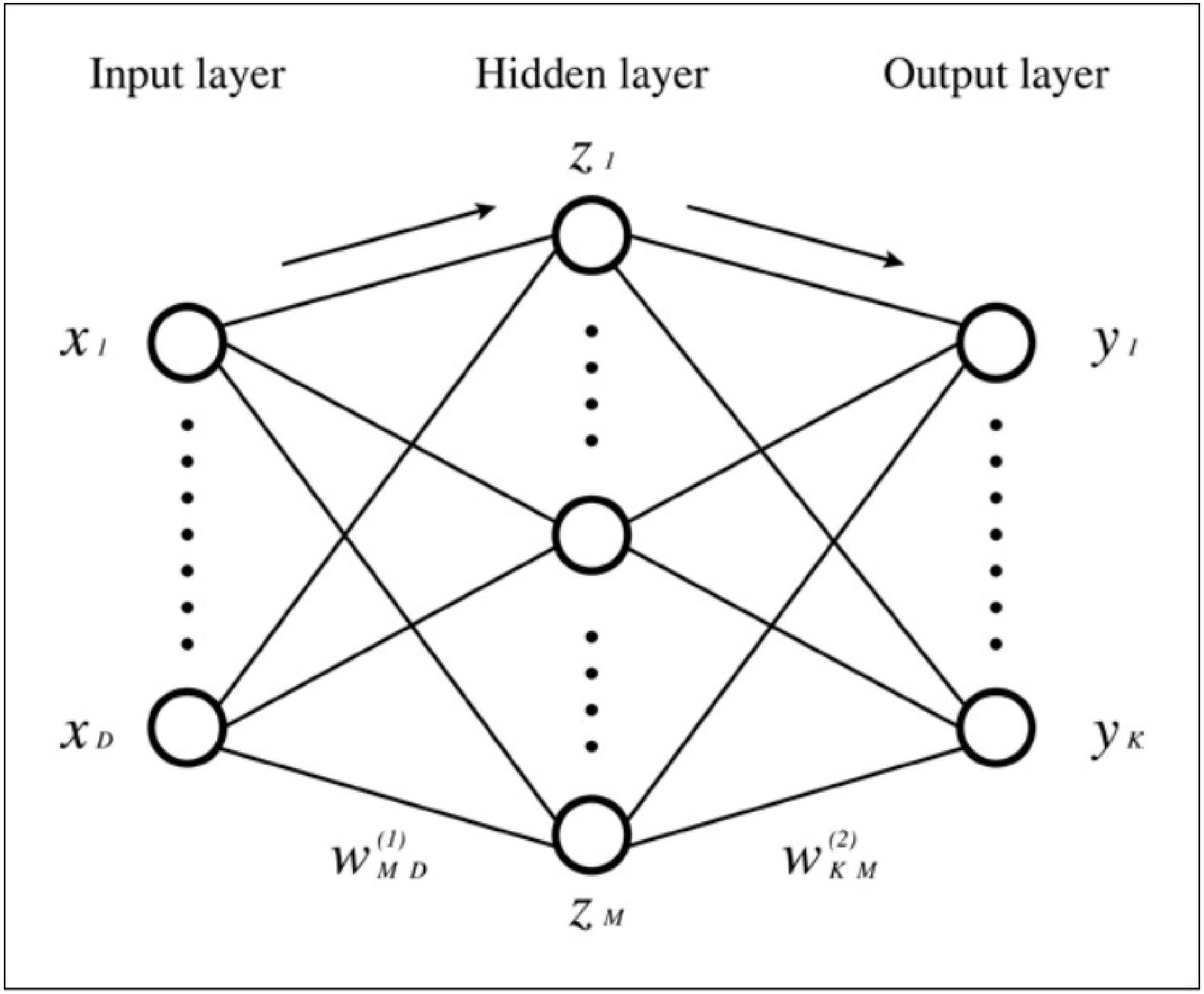
Multi-layer Perceptrons

- Single-layer neural networks have a huge problem
  - Perceptrons or logistic regressions are efficient for problems that can be linearly classified but they can't solve nonlinear problems at all
  - For example, XOR problem



Multi-layer Perceptrons

- Or multi-layer neural networks, MLPs



The Output

$$E(W,b) = -In\,L(W,b) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\,In\,Y_{nk}$$

Here, h is the activation function of the hidden layer and g is the output layer.

As has already been introduced, in the case of multi-class classification, the activation function of the output layer can be calculated efficiently by using the `softmax` function, and the error function is given as follows:

$$\begin{aligned} y_k &= g\left(\sum_{j=1}^{M} w_{kj}^{(2)} z_j + b_k^{(2)}\right) \\ &= g\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}\right) + b_k^{(2)}\right) \end{aligned}$$

Error Propagation

- As for a single layer, it's fine just to reflect this error in the input layer
- For the multi-layer, neural networks cannot learn as a whole unless you reflect the error in both the hidden layer and input layer

$$E(W,b)=\sum_{n=1}^{N}E_n(W,b)$$

$$E(W,b)=-In\,L(W,b)=-\sum_{n=1}^{N}\sum_{k=1}^{K}t_{nk}\,In\,Y_{nk}$$

Smart Computing 智慧計算實驗室 SAIC

Each unit in the feed-forward network is shown as the sum of the weight of the network connected to the unit, hence the generalized term can be represented as follows:

$$a_j = \sum_i w_{ji} x_i + b_j$$

$$z_j = h(a_j)$$

Be careful, as x_i here is not only the value of the input layer (of course, this can be the value of the input layer). Also, h is the nonlinear activation function. The gradient of weights and the gradient of the bias can be shown as follows:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} x_i$$

$$\frac{\partial E_n}{\partial b_j} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial b_j} = \frac{\partial E_n}{\partial a_j}$$

$$\delta_j := \frac{\partial E_n}{\partial a_j}$$

Then, we get:

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i$$

$$\frac{\partial E_n}{\partial b_j} = \delta_j$$

Backpropagation Formula

- delta: backpropagated error

Therefore, when we compare the equations, the output unit can be described as follows:

$$\delta_k = y_k - t_k$$

Also, each unit of the hidden layer is:

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

$$\delta_j = h'\left(a_j\right) \sum_k w_{kj} \delta_k$$

The Code

```
DLWJ
├── MultiLayerNeuralNetworks
│   ├── MultiLayerPerceptrons.java
│   └── HiddenLayer.java
├── SingleLayerNeuralNetworks
│   └── LogisticRegression.java
└── util
    ├── ActivationFunction.java
    └── RandomGenerator.java
```

Outline

- Getting Started
- The Need for Training in Machine Learning
- Supervised/Unsupervised Learning
- Machine Learning Application Flow
- Theories and Algorithms of Neural Networks
- Summary

Summary

- Three representative algorithms of single-layer neural networks: perceptrons, logistic regression, and multi-class logistic regression
- MLPs can solve nonlinear problems says that the networks can learn more complicated logical operations by adding layers and increasing the number of units
- By backpropagating the error of the output to the whole network, the model is updated and adjusted to fit in the training data