Automaton and its Application

Lecture notes for special course in 2012

Akira Imada
Brest State Technical University, Belarus

(last modified on)

May 9, 2012

1 Introduction

The most important keyword throughout this lecture is *"state."* Our behavior when we see or something is dependent on our state of mind. For example, our behavior in meeting our friend when we are in a happy state of mind would be different than when we are in a unhappy state of mind. In computational scenario, ...

2 Conway's Game of Life

In 1970 John Conway devised a zero-player game called the Game of Life[1]. The universe is toroidal 2-D grid. Each cell has either one of two states. We may call the two states, say, dead and alive. The state is determined by the following rules.

Rule 1 (Classification only with prior probability)

1. *Any live cell will die if its live neighbors are fewer than two (too lonely to live).*
2. *Any live cell will remain live if its live neighbors are two or three.*
3. *Any live cell will die if its live neighbors are more than three (too crowdy to live).*
4. *Any dead cell will become a live cell only if its live neighbors are exactly three, otherwise remain die.*

Note that the number of live neighbors is based on the cells before the rule was applied.

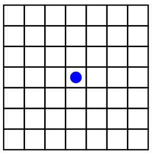


Figure 1: What do you think the fate of this center point?

[1] The first successfully mass marketed personal computer was the Commodore PET introduced in January 1977.

Examples of pattern to start with.

The game is zero-player in which we create an initial configuration and observe how the initial configuration evolves. Now let's see typical initial configuration.

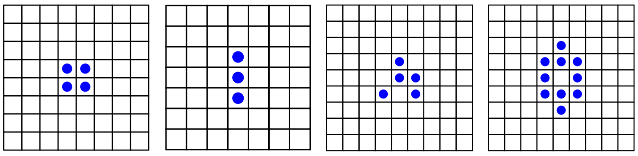


Figure 2: Examples of simple patterns to start with.

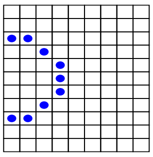


Figure 3: Initial configuration of Queenbee.

3 Cell Automaton (CA)

Universe is any dimension of grid. Each cell in the grid take one state out of k different state. We represent those states by number $1, 2, 3, \cdots, k$, k different clour like *white, black, blue, red,* $\cdots$, or by using a daily language, like *happy, unhappy, angry,* $\cdots$. These distinct states are determined being affected by states of neighbor cell of each cell.

3.1 The Simplest Nontrivial 1-D CA

Each cell has a state either zero or one, or equivalently. *white* and *black*. The state of a cell a determined by the state of the two nearest neighbor cell of right and left and itself. For example:

> *The cell of state '1' whose right cell is state '0' and left cell is state '1' at time t becomes '1' at time $t + 1$.*

Let's describe this by

$$(0\ 1\ 1) \to 1.$$

Note that we need $2^3 = 8$ such lines in order to specify all cases. Hence, there are 256 such rules. Each of the 256 rules is indexed by a binary number whose decimal representation is given to the rule as its name.

3.1.1 Rule 30

Exercise 1 *Apply the Rule 30 starting with a single black cell. Then display those evolved cells from $t = 0$ to $t = 20$.*

3.1.2 Rule 110

Exercise 2 *What about Rule 110?*

3.1.3 Application

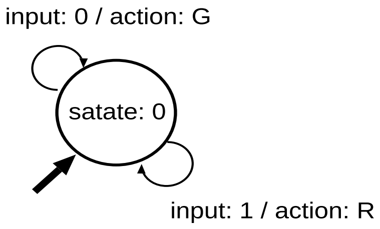
It's easy to imagine we can apply, say Rule 30, to cryptography

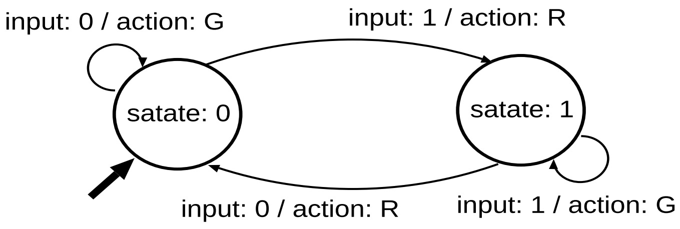
3.2 Two dimensional cell automaton

For the sake of simplicity we assume only two possible states here, say black and white. Also assume each cell concerns only the states of eight its nearest cells and the state of itself. There are $2^9 = 512$ possible configuration. For each of these 512 patterns at time t, The rule specifies the state of the center cell at time $t + 1$ The game of life by Conway is one of them.

4 Finite State Automaton (FSA)

Finite State Automaton (FSA) is made up of a number of states such as cell automaton. But states are not affected by other states but affected by input. Then the state output an action and transfer to the other state (including itself).





The behavior of a FSA is defined by a transition table such as the below. In this example, the FSA has 8 states (A, B, C, D, E, F, G, and H) of which A is the initial state. The number of inputs is two (0 and 1) and the number of actions is four (a, b, c, and d).

current state	input	action	next state
A	0	a	B
	1	c	H
B	0	d	C
	1	b	G
C	0	c	D
	1	a	B
D	0	d	E
	1	a	D
E	0	c	F
	1	a	C
F	0	d	F
	1	a	D
G	0	a	H
	1	b	F
H	0	c	F
	1	a	B

Table 4: An example of transition table of FSA with 8 states 2 inputs, 4 actions

The schematic diagram of this automaton is shown in Figure 5.

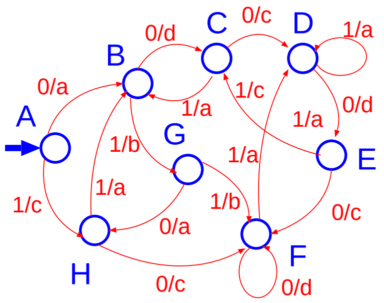


Figure 5: A schematic diagram of the FSA of the above transition table.

4.1 How to design FSA?

We have to determine, (i) initial state to start with, (ii) how many states it has, (iii) which inputs will be given, (iv) how many actions it will make. Then for each of the states it should be specified, which action it will take on which input and which state it will transfer. That is to say, the 3rd column and the 4th column above should be specified by the designer. Note that the 1st and the 2nd is automatic depending on how many states and how many inputs.

4.2 A toy application of FSA

Now let's make an artificial ant explore an gridworld. A strange assumption but there is one grain of sugar, say 1mg, in the dark cell while nothing in the white cells. Now an ant explores the grid starting from the entrance with the aim of eating all those grains by following the shortest path to the last grain of sugar, that is to say, with 98 steps. Try to specify the transition table for such an FSA.

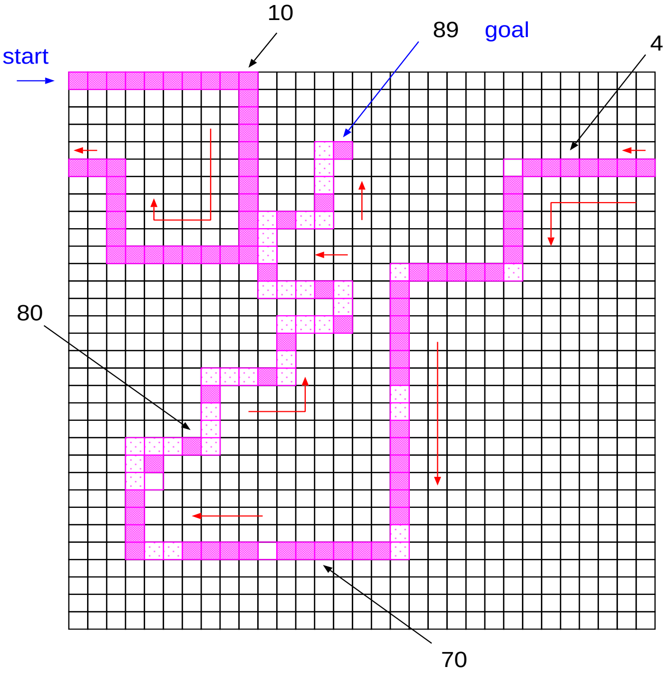


Figure 6: John-muir-trail. An agent should follow the path shown with shaded cell as effectively as possible. Note that the grid has a toroidal structure (the left neighbor cell of the leftmost cell is the rightmost cell).

Generally speaking, in order for an FSA to make a meaningful behavior, it will not easy to specify a transition table. In this example, it would be interesting to make it using Genetic Algorithm. For the purpose, let's represent the transition table by all binary numbers. For the above example, the binary version of the transition table mentioned above is shown in Table 2.

Now we express this specific FSA with a chromosome

00001101101101001110100110000111100000111010100010111010001100110011011010100001

current state	input	action	next state
000	0	00	001
	1	10	110
001	0	11	010
	1	01	110
010	0	10	011
	1	00	001
011	0	11	100
	1	00	011
100	0	10	101
	1	00	010
101	0	11	101
	1	00	011
110	0	00	110
	1	01	101
111	0	10	101
	1	00	001

Table 7: A binary version of the above mentioned transition table.

Now you may notice the number of states and the number of action is better to be in the form of 2^n.

We are now ready to apply a Genetic Algorithm to evolve random FSA to be cleaver enough to solve this John-muir-trail by the following procedure.

1. Create a population of, say 100, random binary chromosome whose length is 80.
2. Make it try the trail one by one and evaluate how many sugar it can collect (fitness).
3. Select two chromosome as parents (by, e.g., Roulette wheel selection) and create on child by crossover and mutation.
4. repeat 3. (say, 100 times) untill we create a new generation.
5. repeat 3. and 4. untill we find a perfect chromosome.

4.3 Different version of Finite state automaton: Accept input series or reject?

Now we are proceeding to yet another version of FSA. We don't specify action for each input, but instead, we assign each state final one or not final one. If the state is one of those final states when all inputs are given, then FSA say the series of input is *accepted*, otherwise *rejected*.

In other words, we give a specific status to some states if a series of state transition by a series of inputs ends at one of these states, the series of inputs is accepted as legal. We don't need to specify *actions* any more when a state shifts to another state. This is an FSA to check a series of inputs is accept or not, not for produce a series of actions.

4.3.1 A toy examples - Vending machine for Coke

Now as a toy example, let's design an FSA for vending machine for coke. (i) As a first scenario, we assume we have only 10 yen coins, while one coke costs 110 yen; (ii) the second one is a little more complicated. Assume we have 2 distinct coins of 10, 100 yen while cole is till 110 yen.

We may think a more complicated one by adding Santa whose price is 80 yen, and/or we have yet another coins of 500 yen. Further, how could we design an USA which give back a return. For example, some one put a 500 yen coin to buy cola, expecting a return of coins for 390 yen. But let me skip these examples.

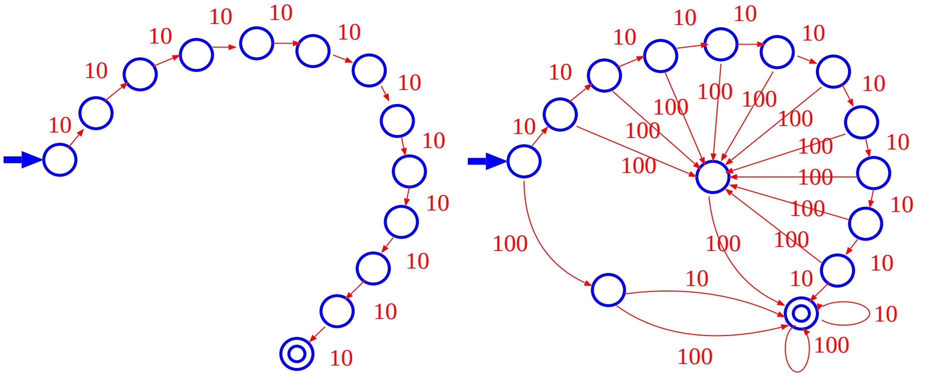


Figure 8: Left: FSA accept only 10 yen coins. Right: FSA accept both 10 yen coins and 100 yen coins. Both for 110 yen cola.

4.3.2 A language recognition by FSA

Again directed graph where nodes denote states, arc from the source state to the target state labeled with a symbol from an alphabet. As before, in the diagram the initial state is shown with a short arrow that points to it from nowhere. The final states are represented as two concentric circles.

We now take a look at Figure 9. It contains 5 states (A, B, C, D, and E) of which A is the initial state, and C and D are the final states, and the alphabet is made up 6 symbols (a, b, c, d, e, and f). The FSA processes a sequence of input symbols. In each state, it checks if the next input symbol matches any of the labels on the arc that goes out from the state. If so, the state move to this target state. In case of no such transition the FSA stops and rejects the input. The input is accepted when all inputs are read[2] and the FSA is in a final State.

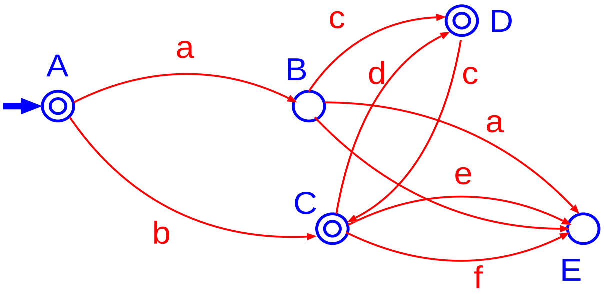


Figure 9: An example FSA to accept a language.

A set of all the possible sequences which can be accepted by the FSA is called the *language* accepted by the FSA. For example, $b, ac, acc, accd, accdc, ..., accef, ...$ is a language accepted by the FSA shown in Figure 9.

A state X is said to be reachable from another state Y if and only if there is a sequence of transitions leading the state X from the state Y. An FSA is said to be start-useful if and only if every state is reachable from the initial state. An FSA is useful in and only if it is start-useful and every state has at least one reachable final state. See Figure 10.

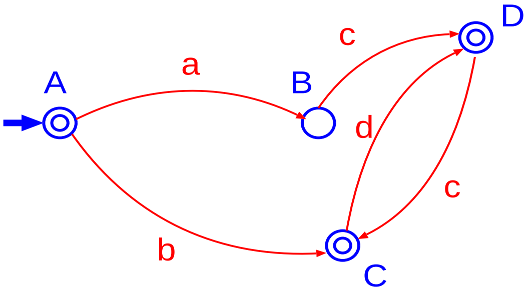


Figure 10: An example useful FSA.

A state may have more than one transition with the same label. In this case the target state is chosen arbitrary at random.

[2] one method to give the end of sequence might be a carriage-return.

We can add a special symbol ϵ in the alphabet. If ϵ is among the labels of the arc going out from the state the transition occurs to the state the arc with label ϵ leads without reading the input. See Figure 17.

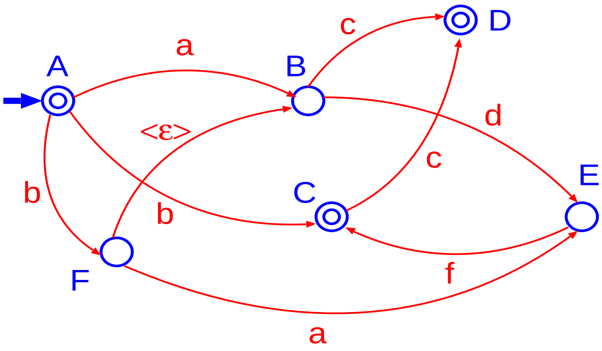


Figure 11: An example of more general FSA.

Exercise 3 *Try to show the language accepted by the FSA shown in Figure 10 and Figure 17.*

5 Formal Language Theory

The concept of the finite state machine comes from linguistics, or to be more specific, Formal Language Theory.

In formal language theory, a language is a set of strings. A string is a sequence of symbols chosen from preset set of specific symbols called the vocabulary. All legal symbols are made up of letter from alphabet.

A rule that generate a string is called *grammar* and those strings are called *language.* A machine such as Finite state automaton can recognize whether the language is legitimate or not.
Let's make a toy example of formal language.

- *Strings over {a,b,c} starting with a.*
  - Example is $\{a, ab, ac, abc, abccc, acbbbbbbc, ...\}$ and $\{bc, cab\}$ is not example.
- *Strings over {a,b,c,d} in alphabetical order.*
  - Example is $\{acd, bcd, c, abcd, ...\}$ and $\{ba, dca, ddabc\}$ is not example.
- *Strings that construct a palindrome over all the Roman alphabet with punctuation, capitalization, and spacing being ignored.*
  - Examples are "Never odd or even." "Madam, I'm Adam." or Madam in Eden, I'm Adam." and "I am a cat." is not.

5.1 Grammar to generate language vs. Machine to recognize language

Let's now compare a grammar to generate regular language and FSA to recognize it. See Figure [**?**].

Again take

string over $\{a, b, c\}$ with alphabetical order.

as an example.

This language can be generated by a grammar and recognized by the FSA shown in Figure[**?**].
For example, we can generate *abc* and *bc* from *S* as follows:

$$S \Rightarrow aS_1 \Rightarrow a(bS_2) = abS_2 \Rightarrow\Rightarrow abc$$

$$S \Rightarrow bS_2 \Rightarrow bc$$

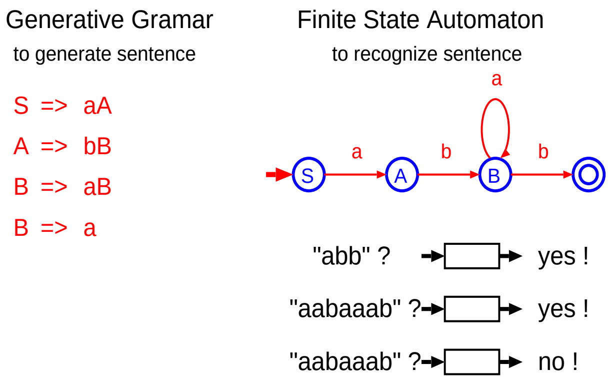


Figure 12: A tiny example of grammar to generate regular language and FSA to recognize it.

S	$\Rightarrow$	aS_1
S	$\Rightarrow$	bS_2
S	$\Rightarrow$	c
S_1	$\Rightarrow$	bS_2
S_1	$\Rightarrow$	c
S_2	$\Rightarrow$	c

Here we call a, b, c terminal, and S, S_1, S_2 non-terminal.

Then can sequences of a's followed by an equal number of b's $a^n b^n$, or palindrome over $\{a, b\}$ generate in this way? The answer is no. it's not possible.

But, instead palindrome over {a,b} is simply generated by the grammar

S	$\Rightarrow ab$
S	$\Rightarrow aSb$

But this is no more Finite State Grammar in which the rule should be of the form

5.2 Generative Grammar

Generative grammar is a grammar to define a set of languages somehow resembles a natural language.

5.2.1 Finite-state grammar

Let's take the above mentioned grammar as an example, that is, the grammar that generates:

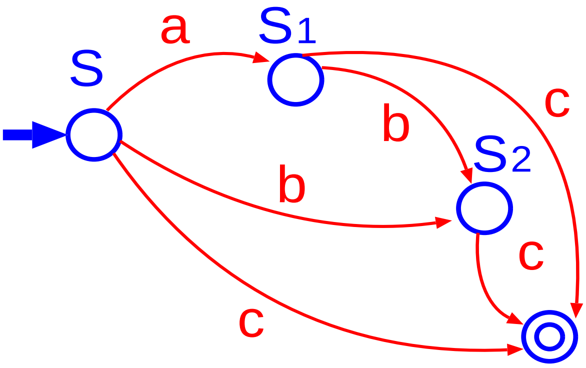


Figure 13: A FSA that can recognize a string over $\{a, b, c\}$ in alphabetical order.

S	$\Rightarrow$	aT

· *Strings over* $\{a,b,c\}$ *in alphabetical order.*

Exercise 4 *Design an FSM which tests strings of the grammar is legal or not.*

We now take a different aspects. We are not testing if a string is accepted or rejected, but *how we generate a string that will be accepted as legitimate string of the grammar?"* Let's S be the start symbol. ...

Where S, S_1, S_2, and S_3, are called non-terminal and a, b, and c are called terminal.

This grammar corresponds to an FSA that recognizes all and only the sentences it generates. Thus,

Rule 2 (Finite State Grammar) *Any grammar that is made up of only rules of the form $A \to bC$ where A, B are non-terminals and b is a terminal has a corresponding FSA. This is called Finite State Grammars.*

Let's summarize here. Assuming we have one grammar, we can design two distinct algorithms. One is to generate strings that follow the grammar, and the other is to see if a string follows the grammar. So what we have studied so far is Finite state grammar is an example of the former and finite state automaton is an example of the latter.

Grammar	Machine to test
Finite State Grammar	Finite State Automaton

Exercise 5 *Can we design a grammar which creates palindrome?*

The answer is no. We need an another grammar.

5.2.2 Yet another way of represent FSA

For the next three subsections, we see yet another way of represent a schematic diagram of Finite State Automaton. See Figure [?]

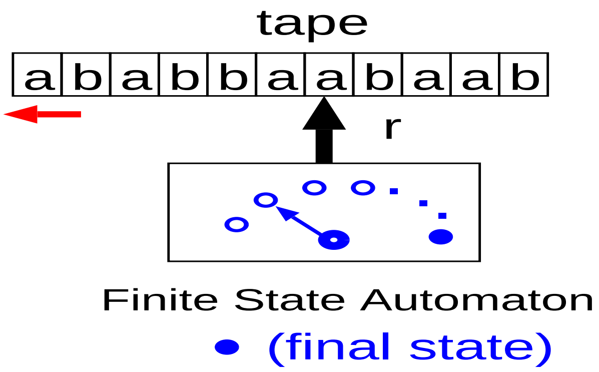


Figure 14: A FSA that can recognize a string over $\{a, b, c\}$ in alphabetical order.

5.2.3 Context-free Grammar & Pushdown Automaton

- **Context-free Grammar**

We now look at a more flexible rule:

S	$\Rightarrow$	α

where α is a non-empty sequence either terminal or non-terminal, such as ab or aSb.

- **Pushdown Automaton**

A PDA is defined by (i) State to start, (ii) alphabet (iii) coins for push or pop on stack. If stack is empty when all inputs are read then the series of inputs will be accepted. Otherwise, like not possible to proceed or still stack is not empty when input was finished, the input was rejected.

We can design PDA in a similar way when we design FSA by a transition table. To generate $a^n b^n$ over $\{a, b\}$, for example, the transition table is as follows:
Or to generate palindrome over $\{a, b\}$:
Notice that "#" is a symbol to let the machine know this is the center, such as $aabbb\#bbbaa$.

See a schematic diagram of pushdown automaton below.
As PDA may use a set of coins with different colors to push or pop in the stack.

current state	input	action	next state
A	a	push	A
	b	push	A
B	a	pop	A
	b	pop	A

current state	input	action	next state
A	a	push red	A
	b	push blue	A
	#	nothing	B
B	a	pop red	B
	b	pop blue	B

- **$a^n b^n$ type of sentence in English**

The above examples are not a meaningless toy example. Actually in English we come across a same kind of structure. See

The dog the cat chased died.

This is modified from

The dog died.

Similarly:

The dog that the cat chased died.

Or

The dog the cat the mouse bite chased died.

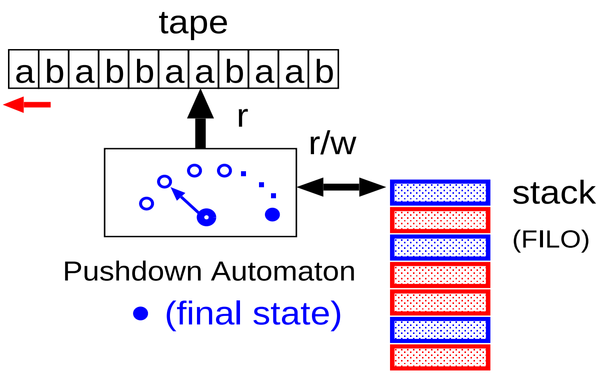


Figure 15: A schematic diagram of pushdown automaton.

Further

> *The dog the cat the mouse the monkey loved bite chase died.*

All of these above sentences are grammatically correct.

Yet the other example in English is

> *Anyone who feels that if so-many more students whom we havent actually admitted are sitting in on the course than ones we have that the room had to be changed, then probably auditors will have to be excluded, is likely to agree that the curriculum needs revision.*

The main sentence is:

> Anyone is likely to agree that the curriculum needs revision.

But many clauses (sub-sentences) or phrases are nested in it. That is,

- We haven't actually admitted.
- Student are sitting in.
- So many more students are sitting that the room had to be changed.
- more students than ones we have

Namely, the structure is:

> Anyone who feels that if so many more ... whom he ... than ... that ... then ... must ...

The same holds in a sentence including "either ... or ..."

5.2.4 Context-sensitive Grammar & Linear-bounded Automaton

$$\alpha A \beta \quad \Rightarrow \quad \alpha \gamma \beta$$

where A is a single non-terminal, α and β are strings of non-terminals and terminals, and γ is a non-empty string of non-terminals and terminals.

- **Example 1:** $a^n b^n c^n$
  The generation of *"aaabbbccc"* is:

$$S \Rightarrow aSS_1S_2 \Rightarrow aaSH_1H_2H_1H_2 \Rightarrow \cdots \Rightarrow aaabbbccc$$

- **Example 2:** $\{xx | x \in a, b^*\}$

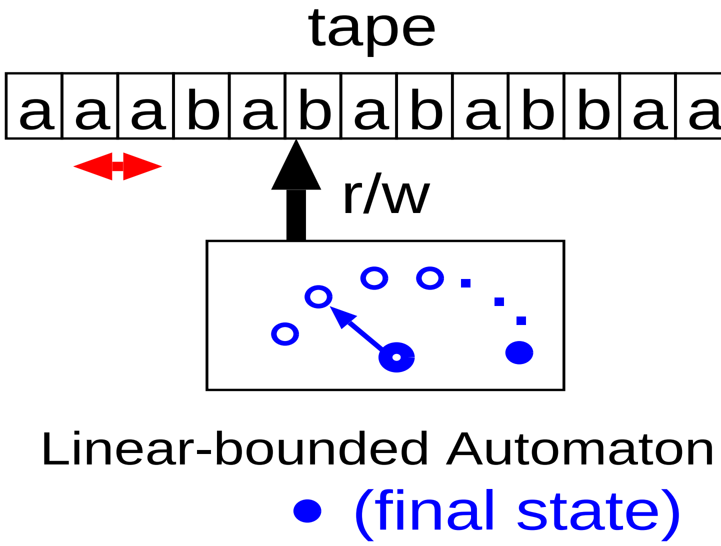


Figure 16: A schematic diagram of Linear-bounded Automaton.

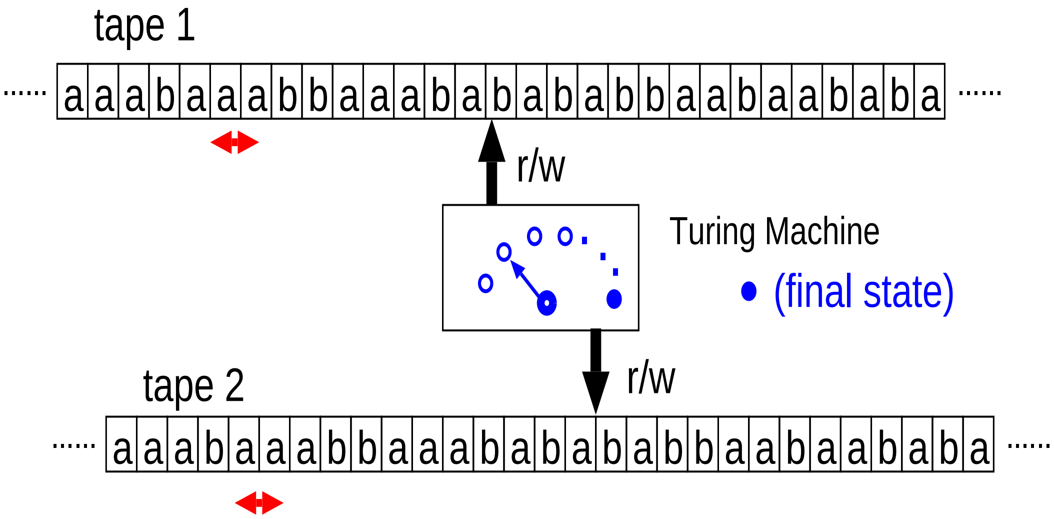

tape 1
...... a a a b a a a b b a a a b a b a b a b b a a b a a b a b a ......
r/w
Turing Machine
● (final state)
r/w
tape 2
...... a a a b a a a b b a a a b a b a b a b b a a b a a b a b a ......

Figure 17: A schematic diagram of Turing machine.

S	$\Rightarrow$	$aSS_1S_2S_3$
S	$\Rightarrow$	$aS_1S_2S_3$
S_2S_1	$\Rightarrow$	S_4S_1
S_4S_1	$\Rightarrow$	S_4S_2
S_4S_2	$\Rightarrow$	S_1S_2
aS_1	$\Rightarrow$	ab
bS_1	$\Rightarrow$	bb
bS_2	$\Rightarrow$	bc
cS_2	$\Rightarrow$	cc

S	$\Rightarrow$	aS_1S
S	$\Rightarrow$	bS_2S
S	$\Rightarrow$	S_3
S_1a	$\Rightarrow$	aS_1
S_2a	$\Rightarrow$	aS_2
S_1a	$\Rightarrow$	aS_1
S_2b	$\Rightarrow$	bS_2
S_1S_3	$\Rightarrow$	S_1a
S_2S_3	$\Rightarrow$	S_3b
S_3	$\Rightarrow$	ϵ

5.3 Un-restricted Natural language & Turing Machine

5.3.1 Natural Language Grammar & Turing machine

- **Any Grammar**
- **Turing Machine**

5.4 Summary of Grammar and machine to test

Grammar	Rule	Machine to test
Finite State Grammar	$A \Rightarrow aB$	Finite State Automaton
Context-free Grammar	$A \Rightarrow \alpha$	Pushdown Automaton
Context-sensitive Grammar	$\beta A\gamma \Rightarrow \alpha$	Linear-bounded Automaton
Unrestricted Grammar	*no restricted*	Turing Machine

6 Stochastic FSA

(Let me skip this sub-topic.)

7 Hidden Markov Model

This subsection describes what is Hidden Markov Model[3], taking 3 examples from the wonderful tutorial on Hidden Markov Model by Ravinar[4]. Nowadays almost all speech recognition systems use this model. Or many areas in computational molecular biology such as grouping amino-acid sequences into protein families, or fault detection.

7.1 Markov Process

Before we study Hidden Markov Model, let's see Markov process in general. Image a system with N states, each of which can transfers to another state, or to itself, with a probability. To be more specific, state i transfer to the state j with a probability a_{ij}.

$$a_{ij} = p(q_t = S_i | q_{t-1} = S_j)$$

This is called N-state Markov Model. For example, assume now $N = 3$, we have nine such transition probability and we can represent them as the matrix A.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

7.1.1 A 3-state Markov model of the weather – Not yet "hidden" though

We now take a daily weather transition as an example. Imagine the weather at noon everyday. One day it's rain, the next day it's cloudy, the third day it's cloudy again, the fourth day it's fine, the fifth day it's rain again, the sixth day it's fine, and so on.

Let's assume the transition probability is:

$$A = \left(\begin{array}{l|ccc} & \mathit{Fine} & \mathit{Cloudy} & \mathit{Rain} \\ \hline \mathit{Fine} & 0.7 & 0.1 & 0.2 \\ \mathit{Cloudy} & 0.5 & 0.4 & 0.1 \\ \mathit{Rain} & 0.6 & 0.3 & 0.1 \end{array} \right).$$

Or, schematically,

[3] Named after a mathematician Andrei Andreyevich Markov who were born in Ryazan, Russia in 1856 and died in Petrograd, Russia in 1922.

[4] L. R. Ravinar (1989) "a tutorial on Hidden Markov Models and selected applications in speech recognition." Proceedings of the IEEE Vol. 77 No. 22.

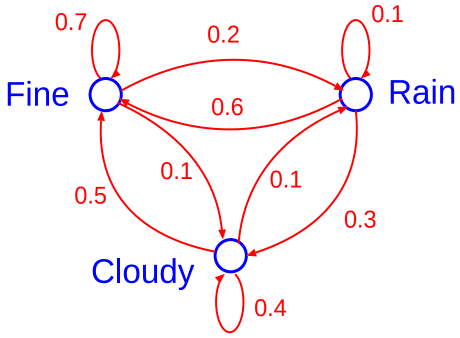


Figure 18: A toy example of State transitions between three distinct weathers.

Exercise 6 ***Calculate the probability of the observation during a 7 day in a row.***

$$sunny - sunny - sunny - rainy - rainy - sunny - cloudy - sunny$$

7.2 Hidden Markov Model

Then what is Hidden Markov Model? Assume now we have N states. The probability of the first state to be started with is i-th state is denoted.

$$\pi_i, \quad i = 1, 2, \cdots, N.$$

Then the probability of transition from state i to state j is a_{ij}. Let's denote this with the matrix A:

$$A = \{a_{ij}\}, \quad i = 1, 2, \cdots, N.$$

We further assume each state results in one of M distinct observations:

$$o_1, o_2, o_3, \cdots, o_M.$$

The probability that state S_i results in the observation o_j is denoted as b_{ij} also denoted with the matrix B:

$$B = \{b_{ij}\}, \quad i = 1, 2, \cdots, M.$$

The important thing is, all we can know is only a series of observations. we cannot know the state that each of those observations has been resulted from. The state transition is hidden to us. This is the reason of the we call it Hidden Markov Model.

Let's take an example when $N = 2$ and $M = 4$.

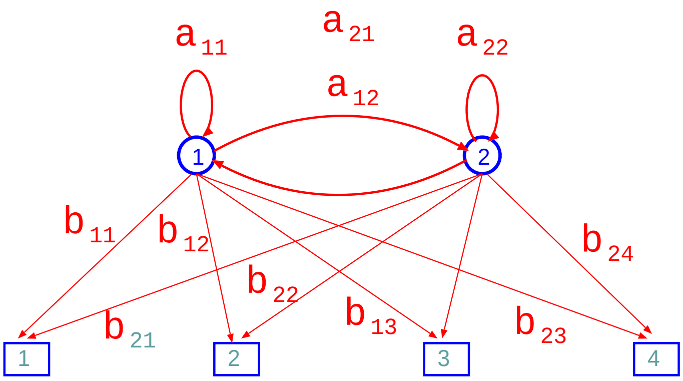


Figure 19: A schematic diagram of Hidden Markov model with 2 states and 4 observations.

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \end{pmatrix}.$$

Now assume a series of observations was:

$$O = \{o_1, o_2, o_3, \cdots, o_T\}$$

Let us now denote the state at time t as q_t. Generally speaking, the transition probability depends on all those states it has shifted from the start. That is,

$$a_{ij} = p(S_j).$$

Then let's think of just a special case where the state at time t is only affected by the state at time $t-1$. Then Notation

$$\lambda = \{\pi, A, B\}$$

We express the state at time t as q_t, that is, $q_t = S_i$ is the state at time t is S_i. o_t is observation at time t.

Then the probability of this observation given the model λ is

$$p(O|\lambda) = \sum_q \pi_{q_0} \prod_{t=1}^{T} a_{q_{t-1}} a_{q_t} b_{q_t} o_t$$

7.2.1 An example: Fair dice and biased dice

Now let's see an example. Assume we have two dices. One is fair dice and the other is biased one. We have two states. The first state is the fair dice, and the second state is the biased one. Each gives one of 6 observations with a probability distribution. For example, fair coin (state 1), of course, results

in either of 1, 2, 3, 4, 5, or 6 with the equal probability, namely 1/6, while the biased coin (state 2) results in 1, 2, 3, 4, and 5 with the probability of 1/10 and 1/2 for 6. Assume transition probability between two states is

$$A = \begin{pmatrix} 0.90 & 0.05 \\ 0.10 & 0.95 \end{pmatrix}.$$

See the Figure below.

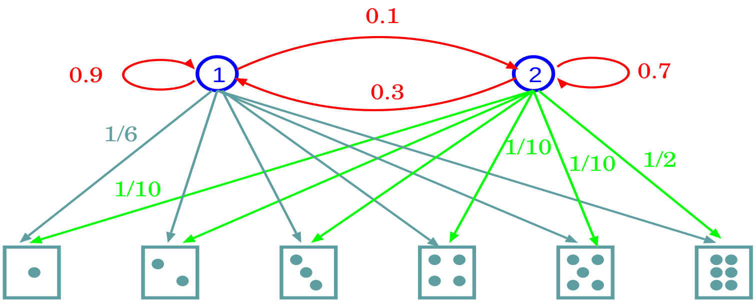
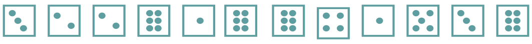


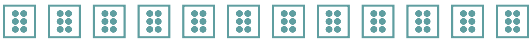
Figure 20: A toy example of a fair dice and biased one.

Assume we cannot know which dice is used. Behind the curtain a hidden man changes the dice from time to time according to the probability distribution A, with the first dice is chosen according to the probability π. All we can observe is the result of the dice, such as:

Then, a question is, e.g., "How likely the following result will occur?"



Or



Or, we can imagine the situation at a casino, where a roulette dealer tries to switch his roulette machine a fair mode and a biased mode.

7.2.2 Another example: Coin toss models

Now behind the curtain a dealer tosses coins several times. And each time the dealer tells us the result is *"Head"* **or** *"Tail."*

The dealer has a multiple coins each has a different probability to result in Head or Bottom. The dealer changes from one coin to another using a prefixed transition probability table. Then assume the result was

Head, Head, Head, Tail, Head, Tail, Head, Head, Head, Tail, Tail, Head

We might guess that the occurrences of *Head* are too many for the coins are fair. Then the question is, "what happens behind the curtain?"

Two coin model

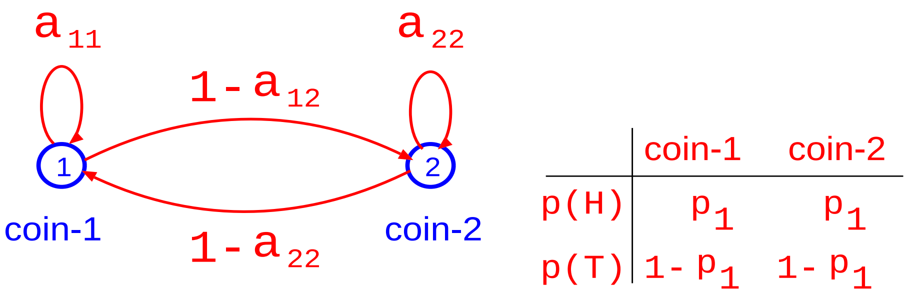
We suppose the dealer has two unfair coins and the dealer follows the transition rule

$$A = \left(\begin{array}{c|cc} & Coin-1 & Coin-2 \\ \hline Coin-1 & a_{11} & a_{12} \\ Coin-2 & a_{21} & a_{22} \end{array}\right).$$

And each coin result in *Head* or *Tail* is from the probability table:

$$A = \left(\begin{array}{c|cc} & Coin-1 & Coin-2 \\ \hline Head & b_{11} & b_{12} \\ Tail & b_{21} & b_{22} \end{array}\right).$$

Further, the dealer start the tossing with i-th coin ($i = 1, 2$) with the probability π_i. See the Figure below.



	coin-1	coin-2
p(H)	p_1	p_1
p(T)	$1-p_1$	$1-p_1$

Figure 21: A coin model assuming two coins.

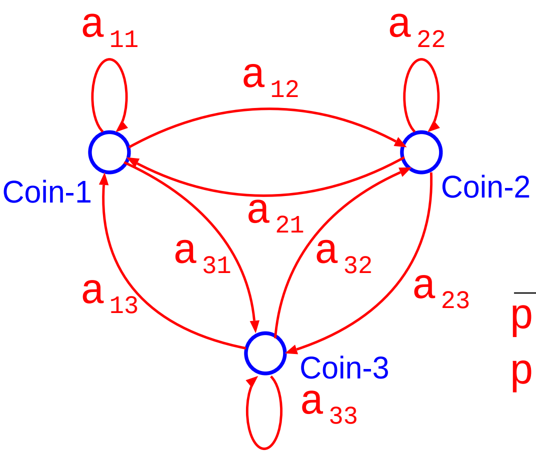
Now our question might be to guess these 15 parameters, that is, $\lambda = \{\pi_i, A, B\}$ so that the observation is most likely, or equivalently, to maximize the probability of the observation:

$$p(Head, Head, Head, Tail, Head, Tail, Head, Head, Head, Tail, Tail, Head).$$

Or, the other question might be, "Which series of hidden states are most likely?"

Three coin model

Yet another possibility is, the dealer has 3 coins, not two. Then the state transition would be:



	Coin-1	Coin-2	Coin-3
p(H)	p_1	p_2	p_3
p(T)	$1-p_1$	$1-p_2$	$1-p_3$

Figure 22: A coin model assuming three coins.

7.2.3 Further example: two urns with colored balls

The final toy example is with two urns each contains a number of balls with red, blue, and purple color. The. The state-1 is the first urn and the state-2 is the second urn. Assume now, the first urn contains 5 red, 3 blue, and 1 purple balls while the second contains 2 red, 4 blue, and 2 purple balls. The dealer, behind the curtain, pick up one ball from either of the urns and tell us the color of the ball. Then the dealer return the ball to the urn he picked up the ball. This is repeated by changing urn to pick up the ball from according to the preset probability table as previous two examples. See the Figure.

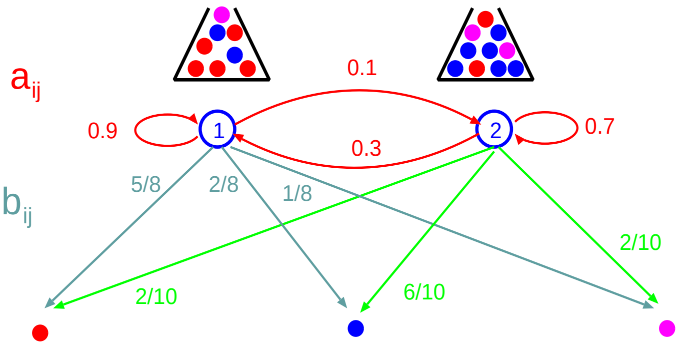


Figure 23: A toy example of thought experiment with two urns each of which contains balls with three different colors.

The question might be, ”What is the probability of a series of observations?” like in the Figure below.

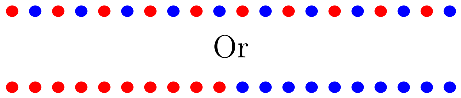


Figure 24: A toy example of thought experiment with two urns each of which contains balls with three different colors.

7.3 Three important problems

As we have seen in examples above, three important questions raise.

- Given $\lambda = \{\pi, A, B\}$, estimate ”How likely the observation is?” That is:

  $$p(o_1, o_2, o_3, \cdots, o_T) = ?$$

- Given $\lambda = \{\pi, A, B\}$, guess the most likely series of hidden states?

  $$(q_1, q_2, q_3, \cdots, q_T)$$

  so that we can predict the probability of the next state:

  $$p(q_{T+1}|q_1, q_2, q_3, \cdots, q_T)$$

- Given a series of observations $(o_1, o_2, o_3, \cdots, o_T)$ modify λ so that it will maximize the $p(o_1, o_2, o_3, \cdots, o_T)$.

  ⋆ This might be called *TRAINING*, or equivalently, *LEARNING* of the model.

To be more specific:

- The Evaluation Problem

  ⋆ Given the observation sequence O and a model $lambda$, how do we efficiently evaluate the probability of being produced by the source model $/lambda$. That is, what is $p(O|\lambda)$.

- The Decoding Problem

  ⋆ how to deduce from O the most likely state sequence q in a meaningful manner.

  ⋆ it is important in many applications to have the knowledge of the most likely state sequence for several reasons. As an example, if we use the states of a word model to represent the distinct sounds in the word, it may be desirable to know the correspondence between the speech segments and the sounds of the word, because the duration of the individual speech segments provides useful information for speech recognition.

- **The Estimation Problem**

  ⋆ **Given the observation O, how do we solve the inverse problem of estimating the parameters in λ?**

  ⋆ **Given an observation sequence (or a set of sequence) O, the estimation problem involves finding the "right" model parameter values that specify a model most likely to produce the given sequence. In speech recognition, this is often called "training,"**

7.4 Stock Market Forecasting

7.4.1 A simple model - Not hidden

Let's start with a very simple model in which states are not hidden. States are, for example, (1) good financial state also know as Bull State, (2) normal state, and (3) bad state a.k.a. Bear state. Assume state transition probabilities are know. When one state changes to another state, economy also change. For example, either of (i) large rise, (ii), small rise, (iii) no change, (iv) small fall, and (v) large fall. Then the phenomena will be shown as Figure 25 [5].

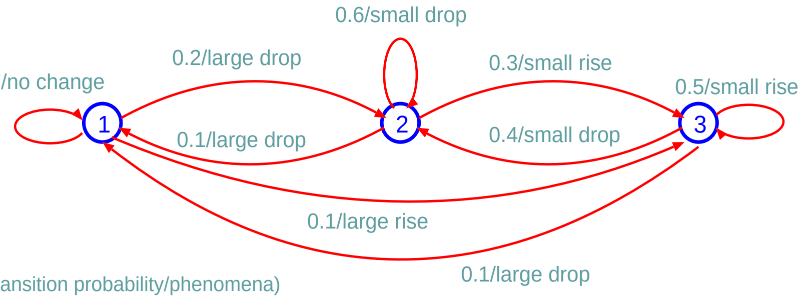


Figure 25: An example of three-state model for stock market forecasting.

Then we can calculate the probability of a series of state changes, like

$$1 \to 1 \to 3 \to 2 \to 3 \to 2 \to 1$$

7.4.2 Hidden Markov Model for stock market forecasting

States are financial states, as the example in the previous section. Or trends of investors, government's financial strategies, or something else. In any case each state reflects, or equivalently, results in, the stock price. Depending on what we want to forecast, the price is average of all stocks, or specific one stock, such as Microsoft Corporation, I.B.M, or Google Co, etc. Or instead of prices we can think of fluctuation of the price. Anyway those values are not discrete as we have studied so far, but a continuous value. We can digitize the value, but usually we analyse the continuous values. Therefore, the probability of one state to result in an observation should be also continuous. The most popular way is using a weighting sum of Gaussian probability distribution

[5] This example was taken from
Y. Zhang (2001) "Prediction of financial time series with Hidden Markov Models." Msc dissertation submitted to Simon Fraser University.

functions. This is called *Gaussian Mixture p.d.f.*

$$\sum_{i=1}^{k} N_i(\mu_i, \sigma_i)$$

where

$$N(x, \mu, \sigma) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h\sqrt{2\pi}} \exp(-\frac{1}{2}(\frac{x - x_i}{h})^2).$$

For example:

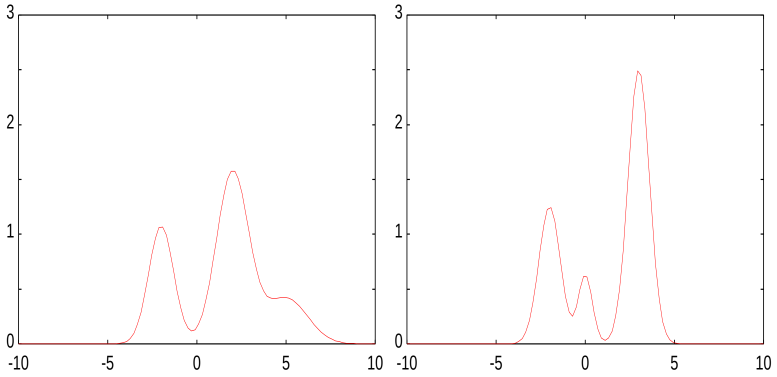


Figure 26: Two Gaussian-mixtures. Left $0.2 * N(5, 1.2) + 0.5 * N(2, 0.9) + 0.3 * N(-2, 0.7)$ and right $0.6 * N(3, 0.6) + 0.1 * N(0, 0.4) + 0.3 * N(-2, 0.6)$

Or, if observation is not one dimensional value such as price, but a high-dimensional vector, like

$$\{\textit{today's price, yesterday's price, price two day's ago}\}$$

then we use a high-dimensional Gaussian p.d.f.

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma_i|^{1/2}} \exp\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^t \Sigma_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i)\}$$

as we studied previously in Part I.

Then imagine two-dice-example we saw in Figure 20. We have two states each of which result in observation of a continuous value following the Gaussian mixture p.d.f., instead of 1, 2, 3, 4, 5, or 6 following a probability like 1/6 or 1/10.

Recall a training of a HMM from a dataset D **is obtaining the model** M **by maximizing**

$$p(M|D) = \frac{p(D|M)p(M)}{p(D)}.$$

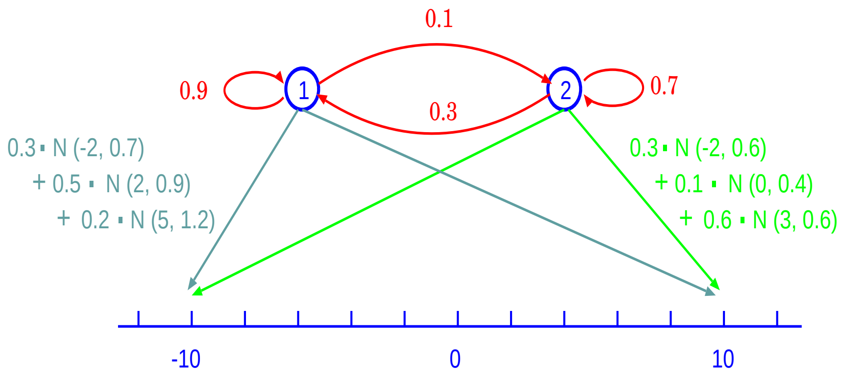


Figure 27: A schematic diagram of two states each of which yields an observation with its Gaussian mixture p.d.f.

When, however, the model M includes a set of continuous parameters θ the probability is

$$p(M|D) = \frac{\int p(D|\theta, M)p(\theta)d\theta p(M)}{p(D)}.$$

In stock market prediction, assume we have a series of observation y_t, that is

$$D = \{y_1, y_2, y_3, \cdots, y_t\}.$$

Then what we want to know is,

$$p(y_{t+1}|D) = \int p(y_{t+1}|\theta, M, D)p(\theta|M, D)p(M|D)d\theta dM.$$

This is called *predictive distribution*.

Now our hidden states are investment strategies or trend that are mixture of the weighted sum of 4 multi dimensional Gaussian pdf.

$$\sum_{i=1}^{4} w_i N_i(\Sigma_i, \mu_i)$$

Then we can calculate how likely the series of state transitions which yields the series of change of prices we have observed, or more importantly, we can calculate the probability of the next state and most likely observations from the predicted state, that is, the price of tomorrow, for example.

Probably most importantly, from a series of observations:

$$O_1, O_2, O_3, O_4, O_5, O_6, O_7, O_8, \cdots, O_T$$

you can optimize the parameters such that this series of observations is most likely, then with those parameters you may infer the most likely series of hidden states,

$$S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, \cdots, S_T$$

and S_{T+1} which means a most likely state of tomorrow. Finally, you can infer the most likely O_{T+1} that will most likely result in from S_{T+1}. This is what we might want to know, namely, tomorrow's price.

I wish your luck!

APPENDIX

Genetic Algorithm

8 How we select parents?

Hopefully, *the better the fitness the more likely to be selected.*

- **Three different versions of Selection.**
  - ⋆ **Truncation Selection (Simplest of the three):**
    - · **Select parents from the best some-percentage of the population.**
  - ⋆ **Roulette Wheel Selection (or Fitness Proportionate Selection):**
    - · **Select such that the probability to be selected is proportional to the fitness value.**

  For example,

  To be more specific, sort the individual from low to high and calculate cumulative value of fitness as follows:

individual after sort	cumulative value of fitness
#2	0.125
#3	0.250
#4	0.375
#5	0.625
#1	1.000

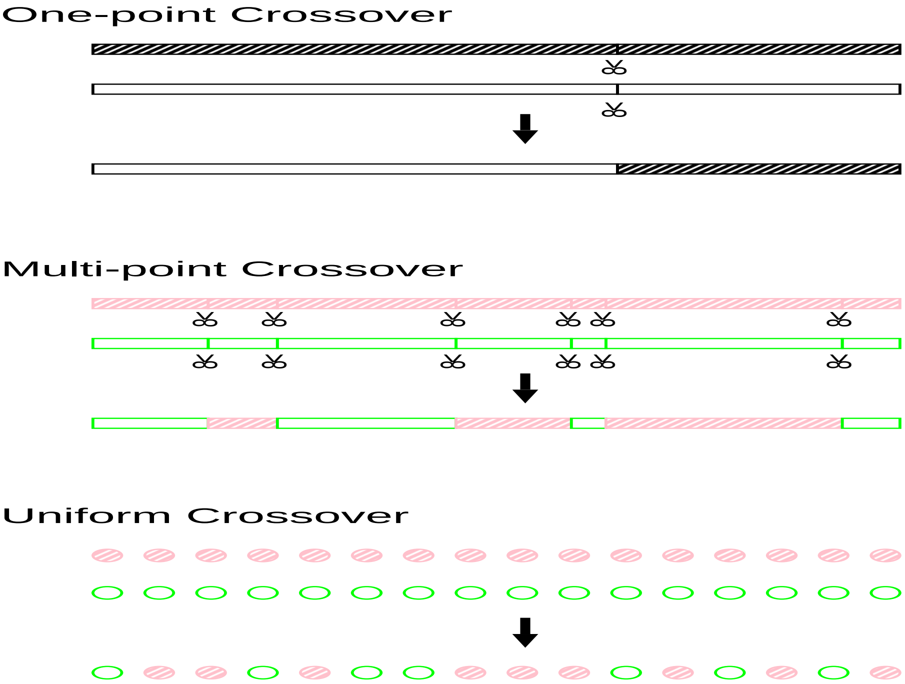
Then create a random number r from 0 to 1 if $r < 0.125$ then select #1, else if $r < 0.250$ then select #2 , else if $r < 0.375$ then select #3 and so on.

Note that even the worst fitness individual have a chance to be selected under Roulette Selection however few it might be, while under Truncate Selection worse individual have no chance to be selected. Tournament Selection could also select a worse individual except for the worst q individuals. We can control the probability of selecting worse individual by changing q.

9 How parents produce a child?

9.1 Cross-over

So-called *crossover* **is exploited for the reproduce children. Here, we see three different versions of crossover bellow.**



9.2 Mutation

We should give a mutation to introduce new genes. This is to avoid for individuals in the population to be trapped into a *local minimum*. The probability for mutation to occur is small — typically 1/number-of-genes. To be more specific,

> *If a randomly generated number from 0.0 to 1.0 is smaller than the mutation rate then mutate otherwise do nothing.*