CS 10:
Problem solving via Object Oriented Programming

Pattern Recognition

Main goals

- Implement **Hidden Markov Models (HMMs)** based on finite automata for **pattern recognition**
    - (More on HMMs in COSC 76 – Artificial Intelligence)

Agenda

1. Pattern matching vs. recognition
2. From Finite Automata to Hidden Markov Models
3. Decoding: Viterbi algorithm
4. Training

Sometimes our input is noisy and does not exactly match a pattern

Pattern matching vs. recognition

Is this a duck?

	Matching	**Recognition**
Looks like a duck	✓	✓
Quacks like a duck	✓	✓
Does not wear cool eyewear	✗	✗
Is it a duck?	✗	✓

Pattern recognition still accepts this as a duck, even though not all features match

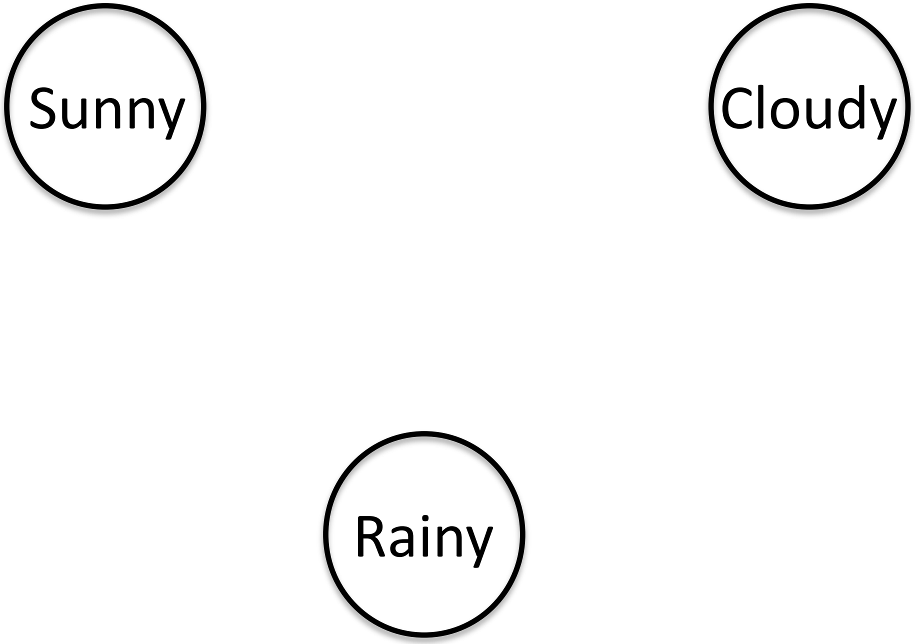
Image: duckrace.com

Agenda

1. Pattern matching vs. recognition
2. From Finite Automata to Hidden Markov Models
3. Decoding: Viterbi algorithm
4. Training

We can model systems using Finite Automata

Weather model: possible states



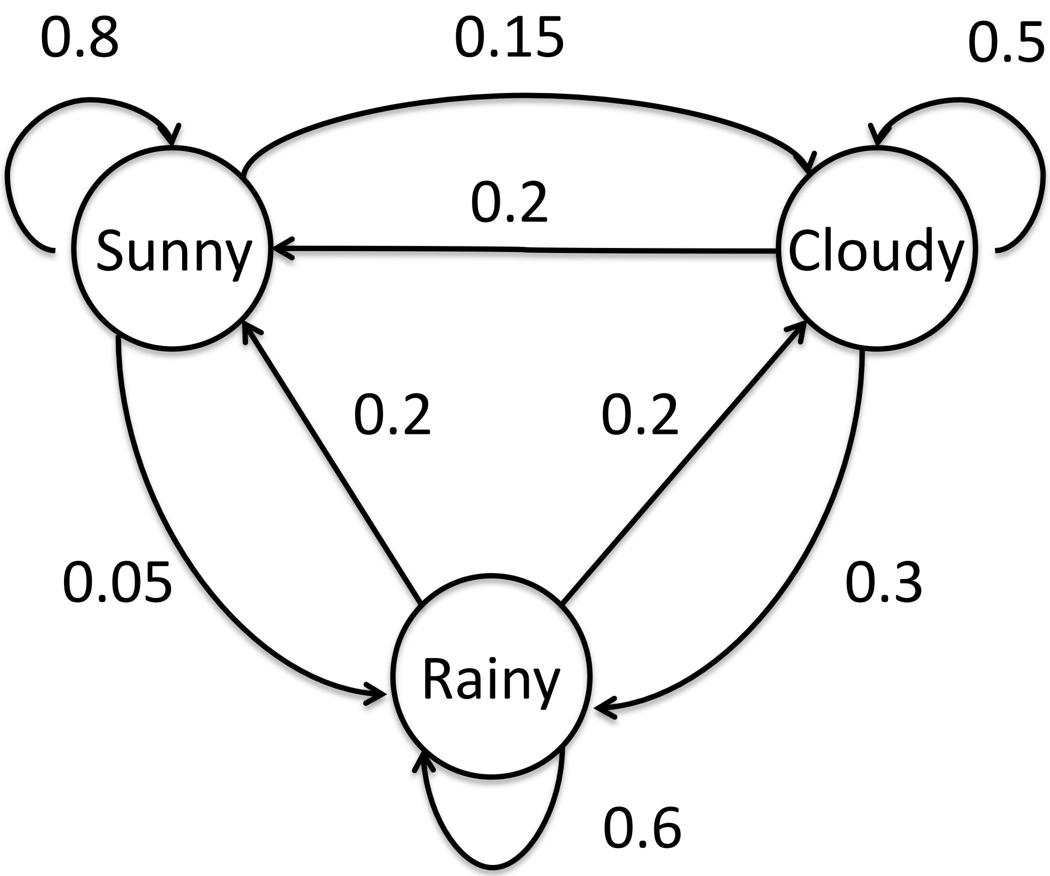
The **State** of the weather can be:

- Sunny
- Cloudy
- Rainy

Adapted from: https://pdfs.semanticscholar.org/b328/2eb0509442b80760fea5845e158168daee62.pdf

We can model systems using Finite Automata

Weather model: transitions

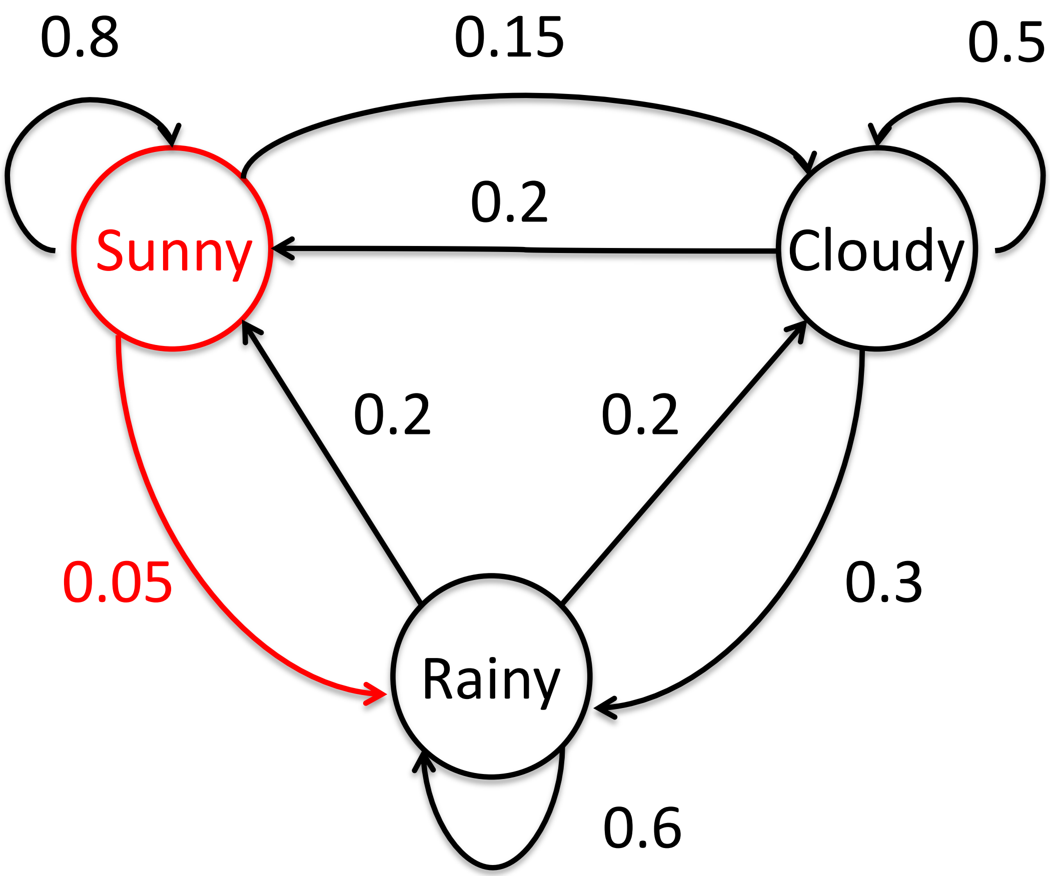


We can observe weather patterns and determine probability of ***transition*** between states

Adapted from: https://pdfs.semanticscholar.org/b328/2eb0509442b80760fea5845e158168daee62.pdf

We can model systems using Finite Automata

Weather model: Sunny day example



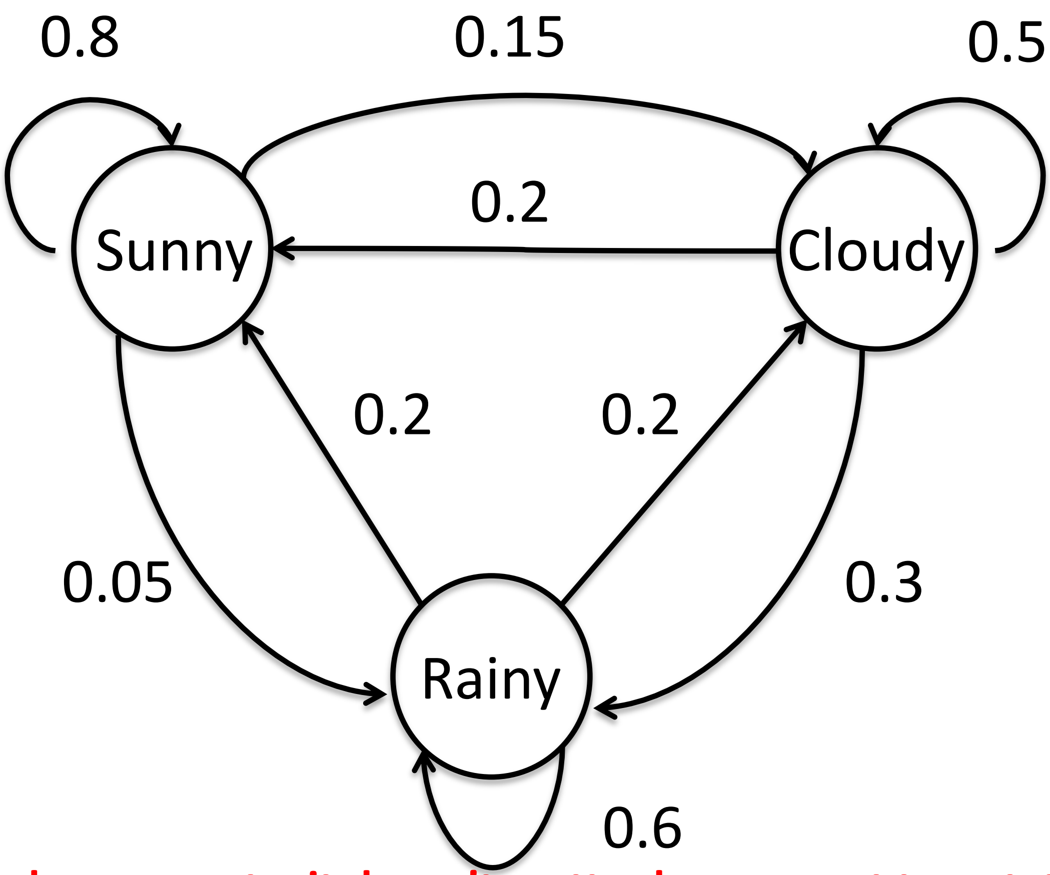
Probability a sunny day is followed by:

- Another sunny day 80%
- A cloudy day 15%
- A rainy day 5%

Adapted from: https://pdfs.semanticscholar.org/b328/2eb0509442b80760fea5845e158168daee62.pdf

Markov property suggests it doesn't really matter how we got into the current State

Given current State, can predict likelihood of future states



Given that we can observe the state we are in, it doesn't really matter how we got there:

- $P(w_n | w_{n-1}, w_{n-2}, w_{n-3}) \approx P(w_n | w_{n-1})$

Markov property: it doesn't matter how we got to a state, the current state is all we need to predict the next state

Adapted from: https://pdfs.semanticscholar.org/b328/2eb0509442b80760fea5845e158168daee62.pdf

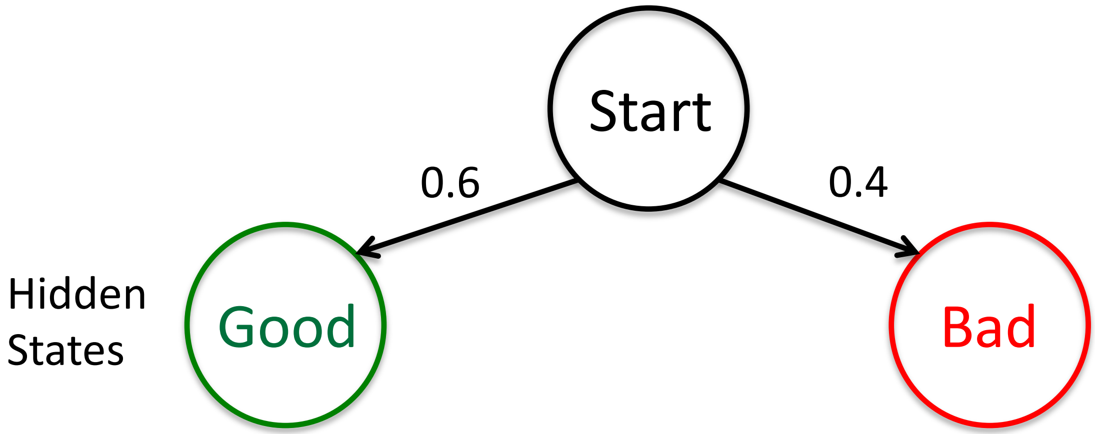
Model works well if we can directly observe the state, what if we cannot?

Sometimes we cannot directly observe the state

- You’re being held prisoner and want to know the weather outside. You can’t see outside, but you can observe if the guard brings an umbrella.
- You observe photos of your friends. You don’t know what city they were in, but do know something about the cities. Can you guess what cities they visited?
- You want to ask for a raise, but only if the boss is in a good mood. How can you tell if the boss is in a good mood if you can’t tell by looking?

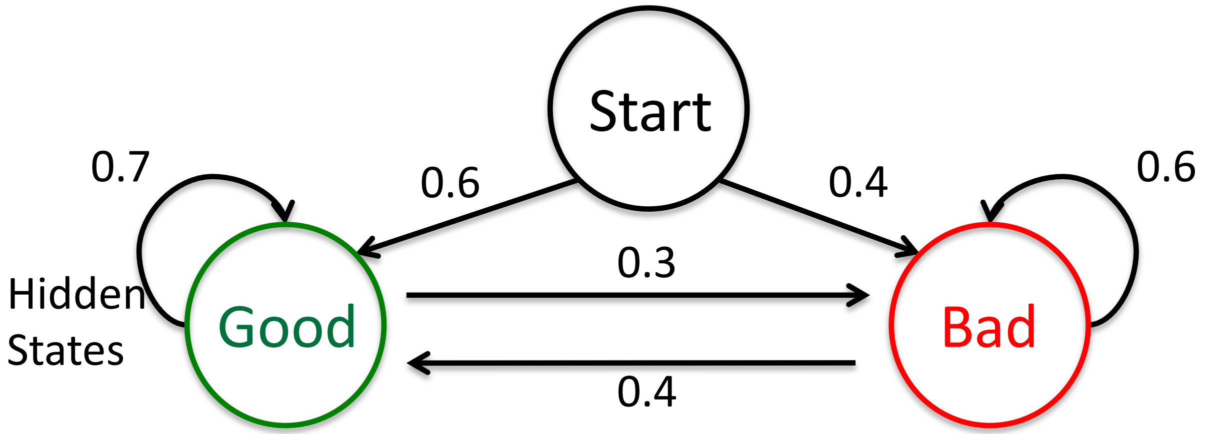
Want to ask the boss for raise when the boss's state is a Good mood

Gather stats about likelihood of states



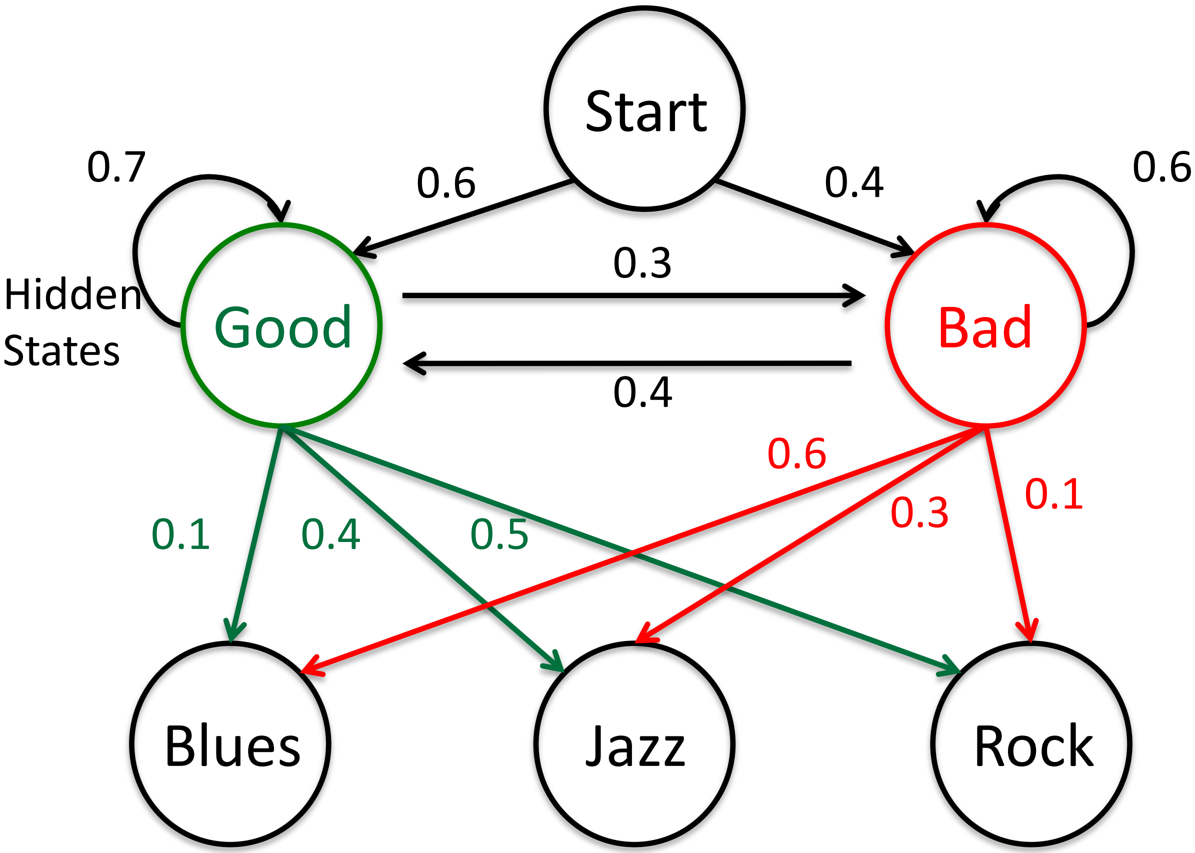
In addition to states, find likelihood of *transitioning* from one state to another

Gather stats about state transitions



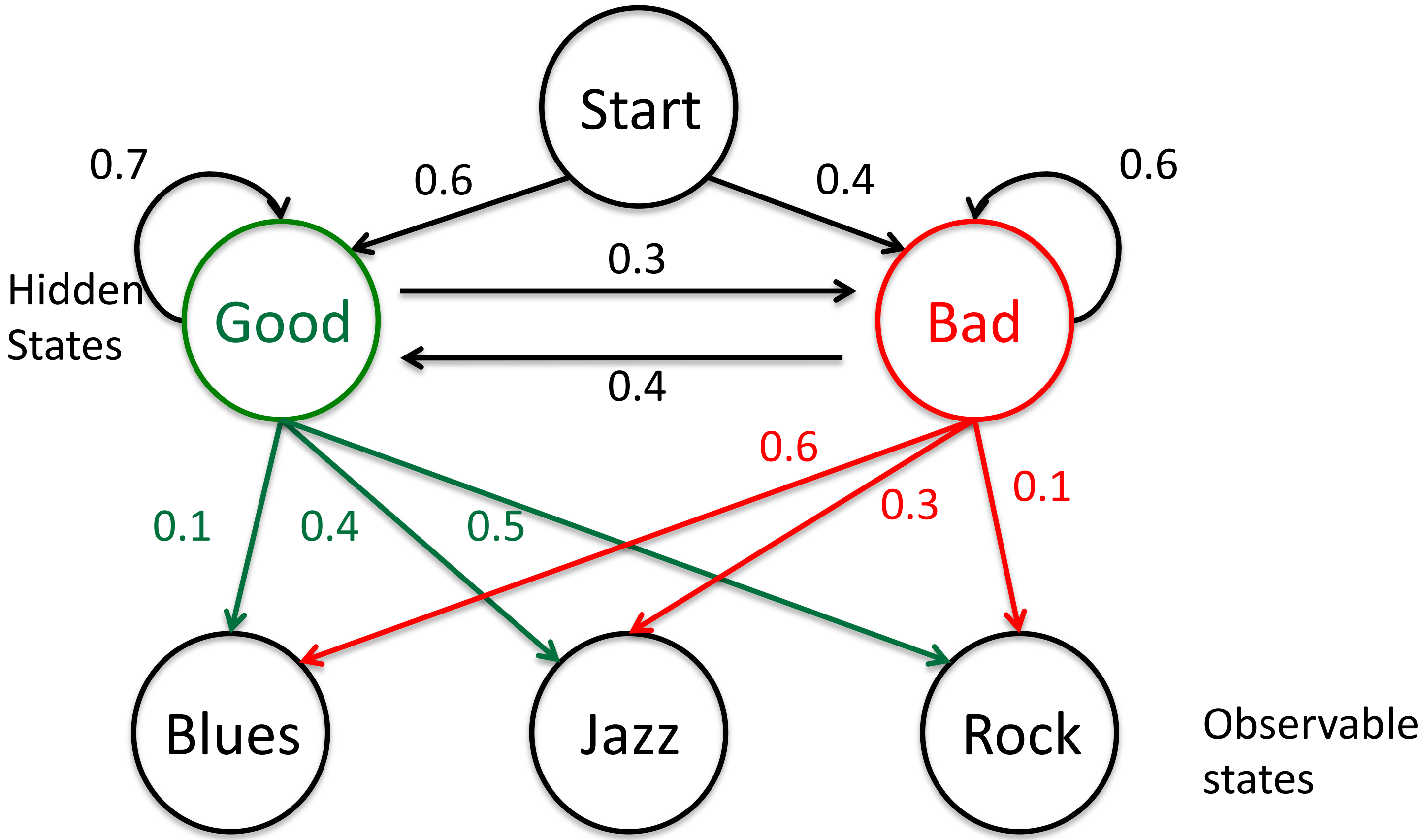
Once have states and transitions, might find something we *can* directly observe

Might be able to observe music playing



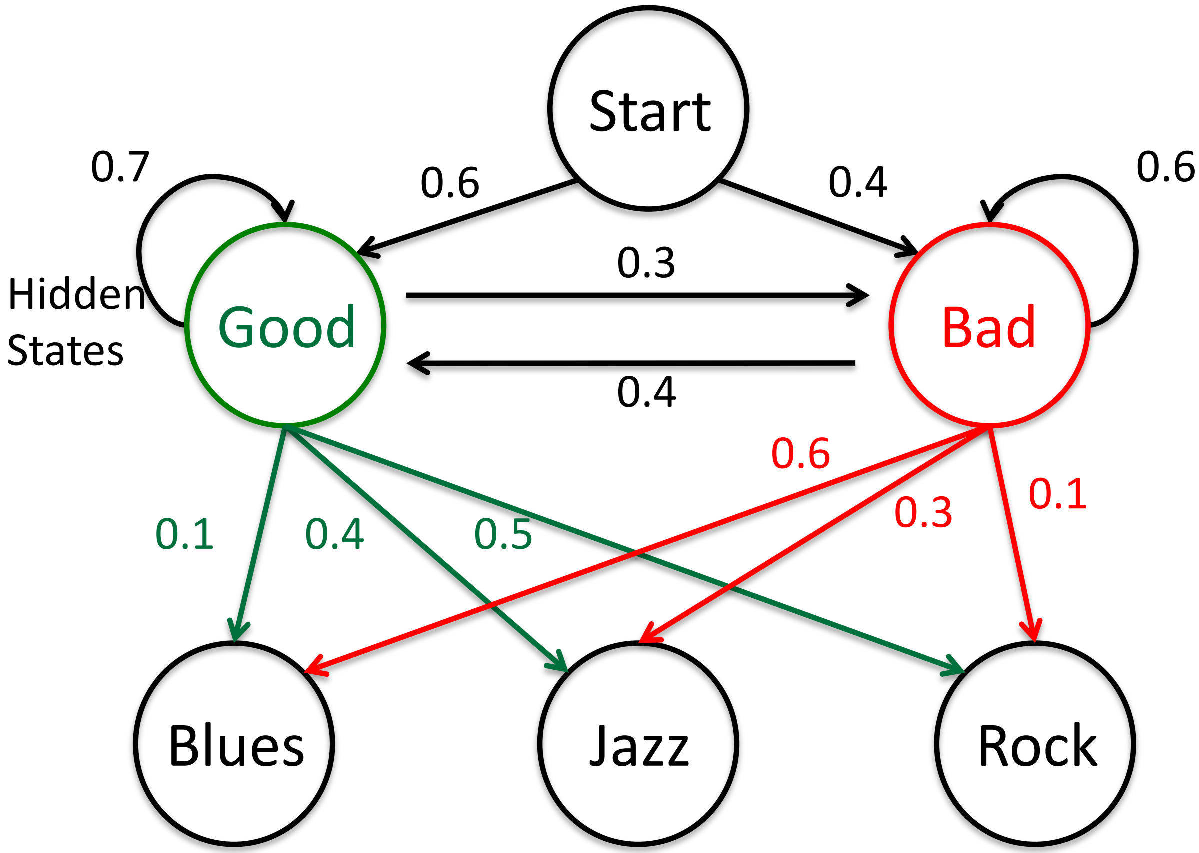
This is a Hidden Markov Model (HMM)

Hidden Markov Model



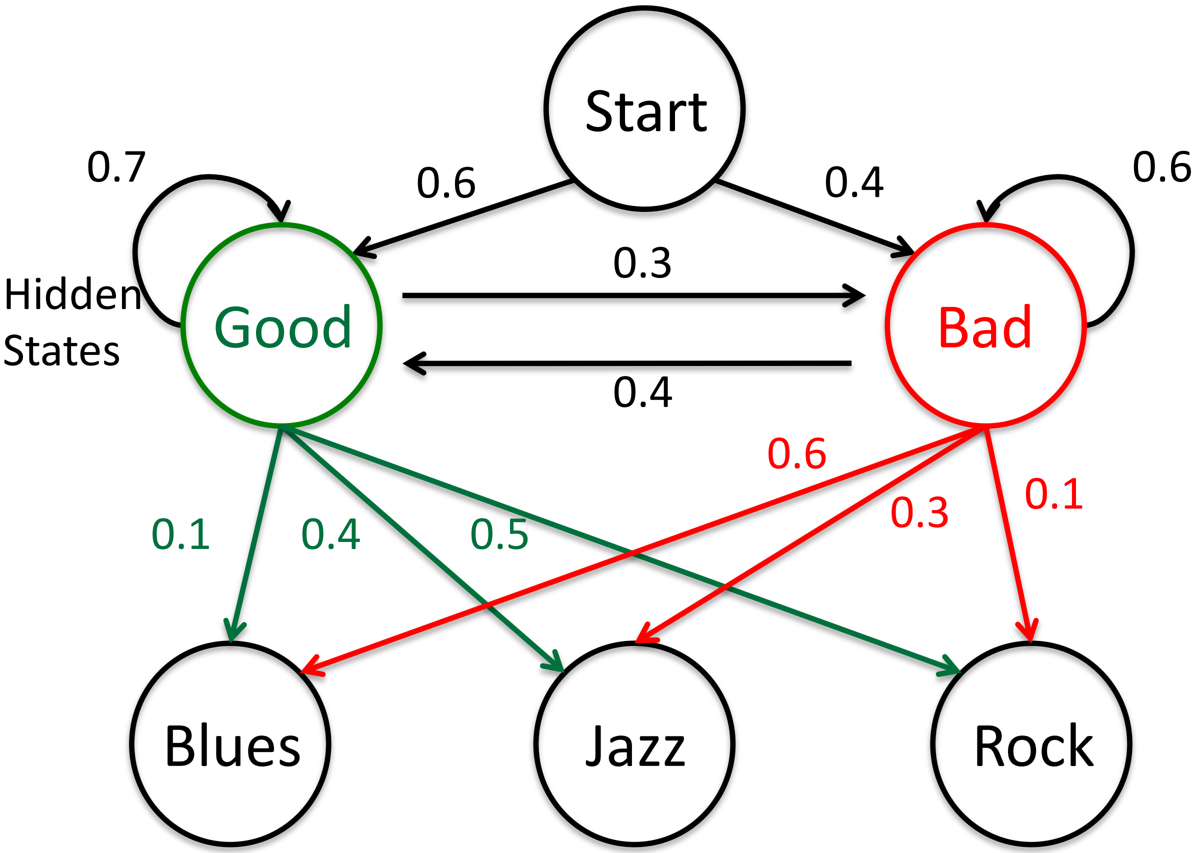
So is today a good day to ask for a raise?

So far we have no music observation



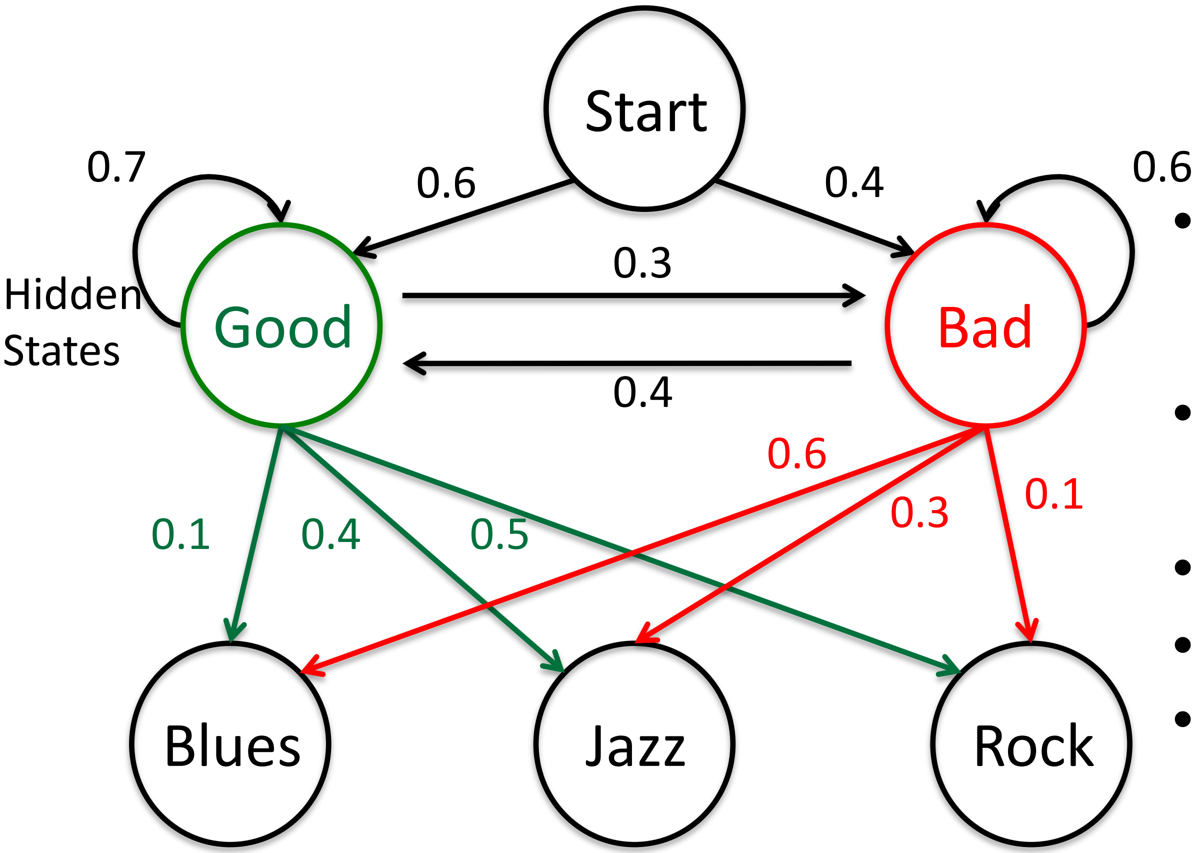
By observing music, we might be able to get a better sense of the boss's mood!

Observe Rock music



Bayes theorem can give us the actual probabilities of each hidden state

Observe Rock music



88% likely to be in good mood

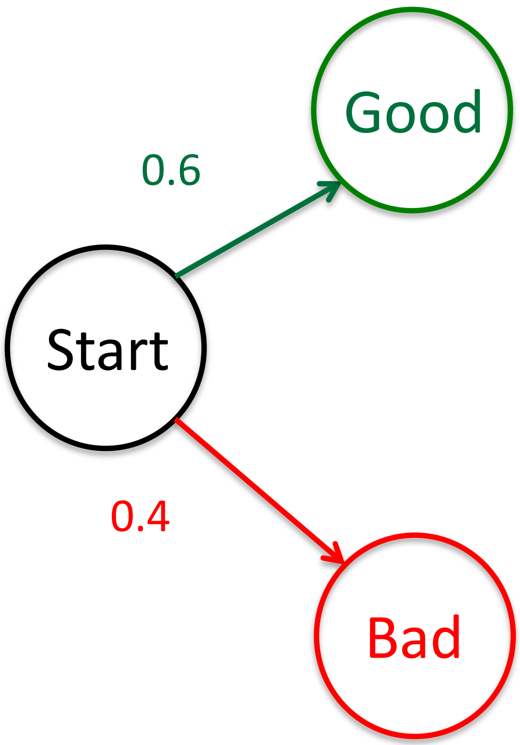
G=Good, B=Bad, R=Rock

- Given the boss is playing Rock music, use Bayes Theorem:
- $P(A|B) = \frac{P(B|A)*P(A)}{P(B)}$
- $P(G|R) = \frac{P(R|G)*P(G)}{P(R)}$
- $P(R|G) = 0.5$
- $P(G) = 0.6$
- $P(R)=0.6*0.5+0.4*0.1 = 0.34$
- $P(G|R) = 0.5*0.6/0.34 = 0.88$

Agenda

1. Pattern matching vs. recognition
2. From Finite Automata to Hidden Markov Models
3. Decoding: Viterbi algorithm
4. Training

We can estimate the most likely hidden state based on observations

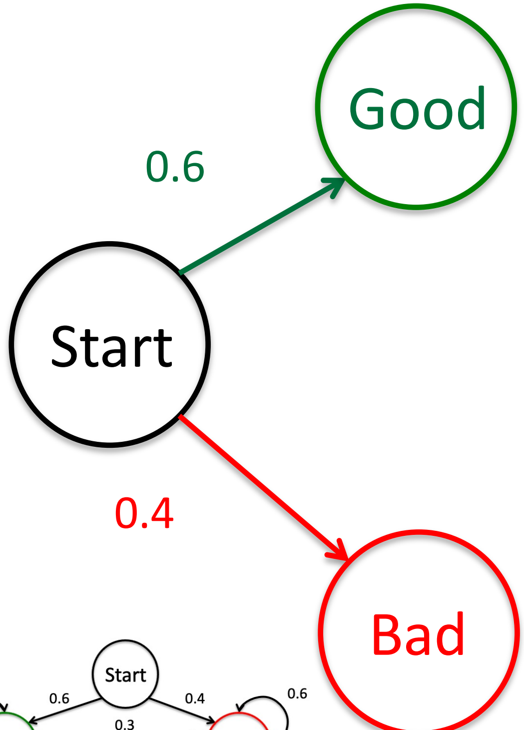


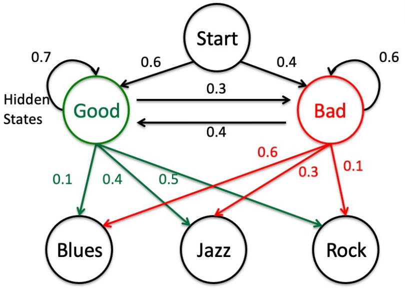
- Viterbi algorithm reconstructs most likely historical states given a set of observations
  - Computes "forward" the most likely state given each observation
  - Once most likely state computed for all observations, back track to find most likely sequence of states
  - Can update its prior estimates based on new observations

- Closely related Forward algorithm computes probability of being in all states as observations made

We can estimate the most likely hidden state based on observations

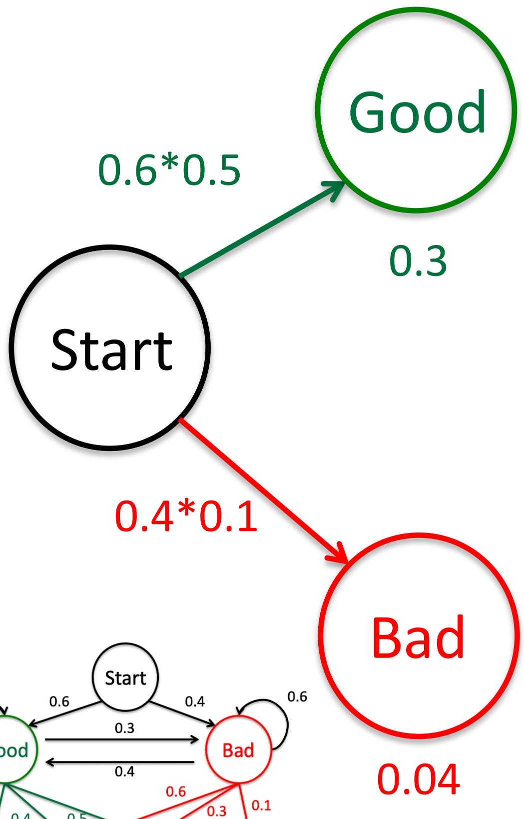
No observations yet

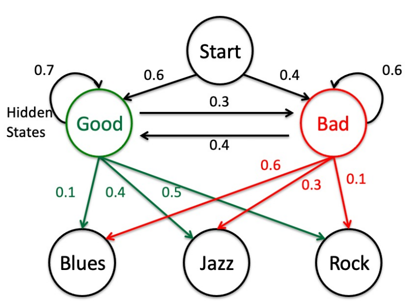




We can estimate the most likely hidden state based on observations

Day 1: Observe Rock

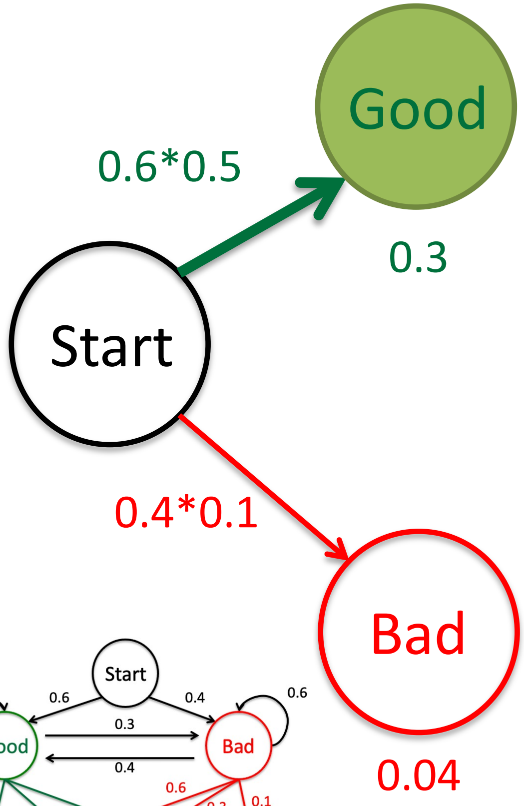


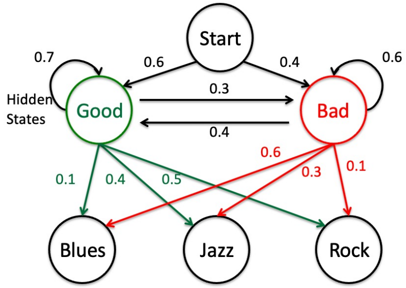


We can estimate the most likely hidden state based on observations

Day 1: Observe Rock

Most likely State has highest score

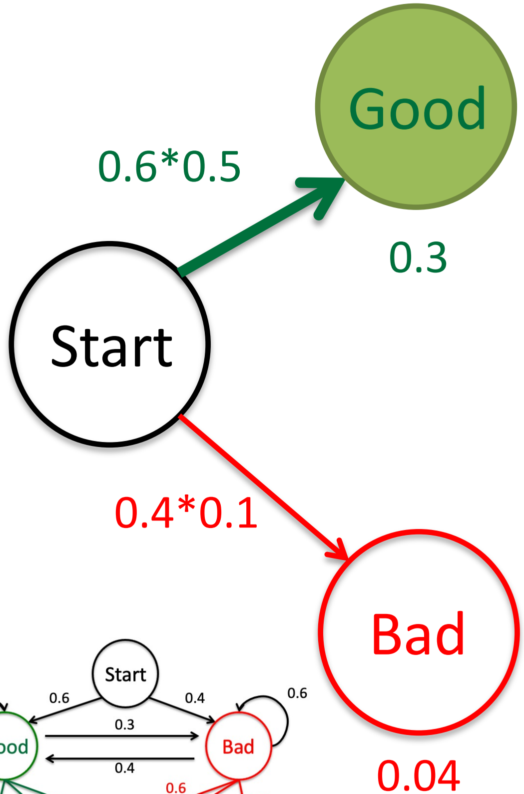


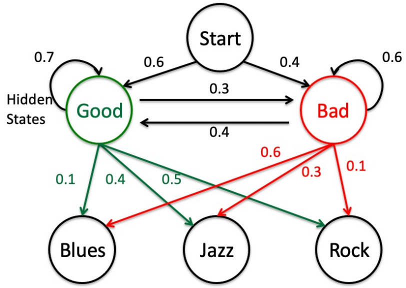


We can estimate the most likely hidden state based on observations

Day 1:
Observe
Rock

Day 2:
Observe
Jazz

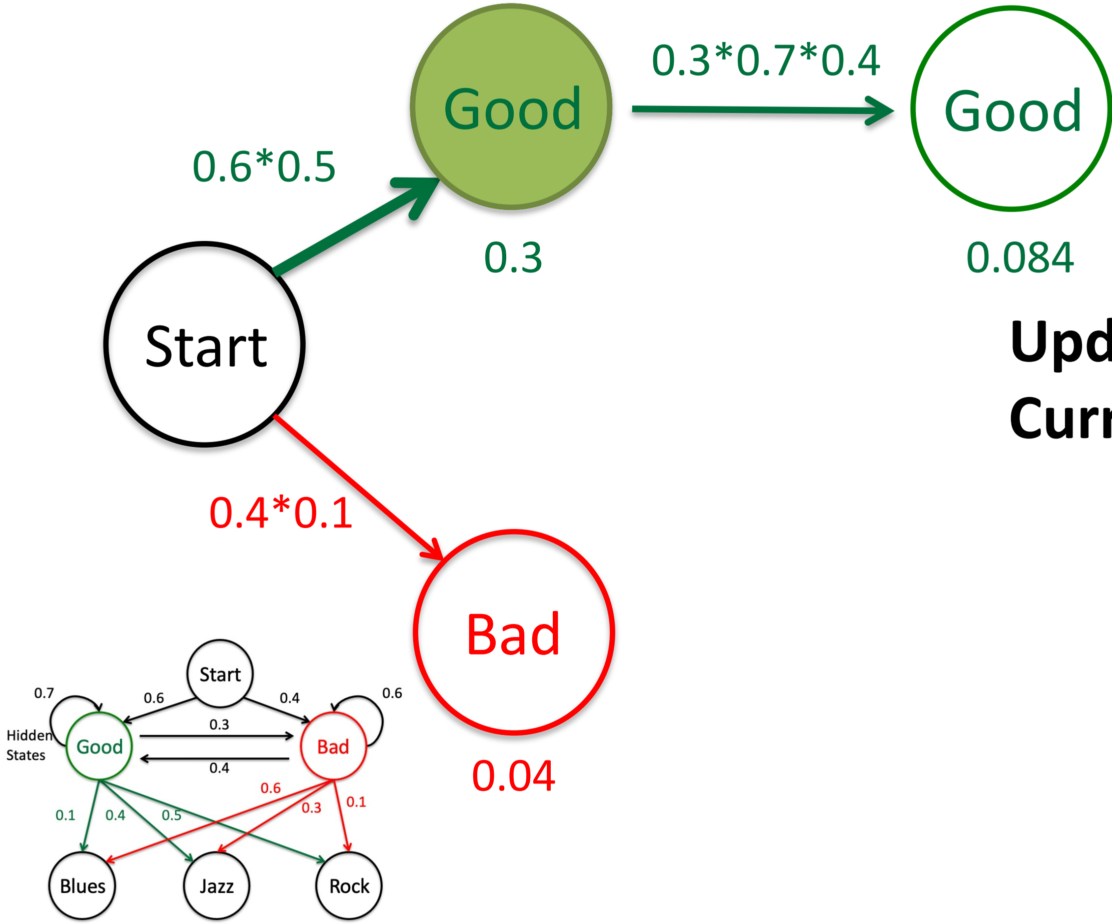




We can estimate the most likely hidden state based on observations

Day 1:
Observe
Rock

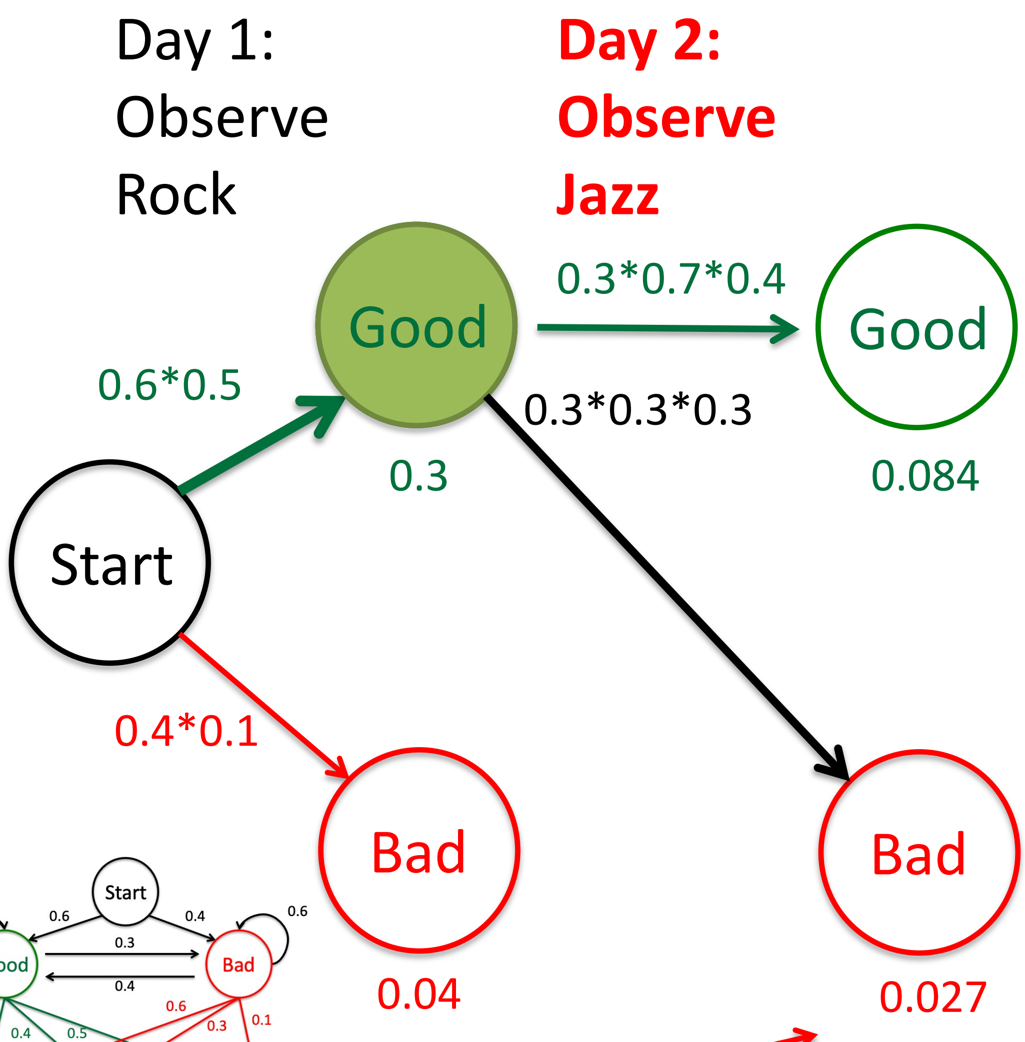
Day 2:
Observe
Jazz



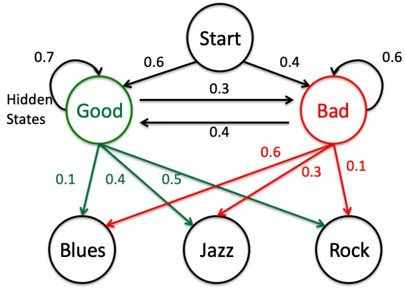
Update rule on new observation:
Current* Transition* Observation

Most likely state has highest value

We can estimate the most likely hidden state based on observations



Do the same for possible transition from Good to Bad



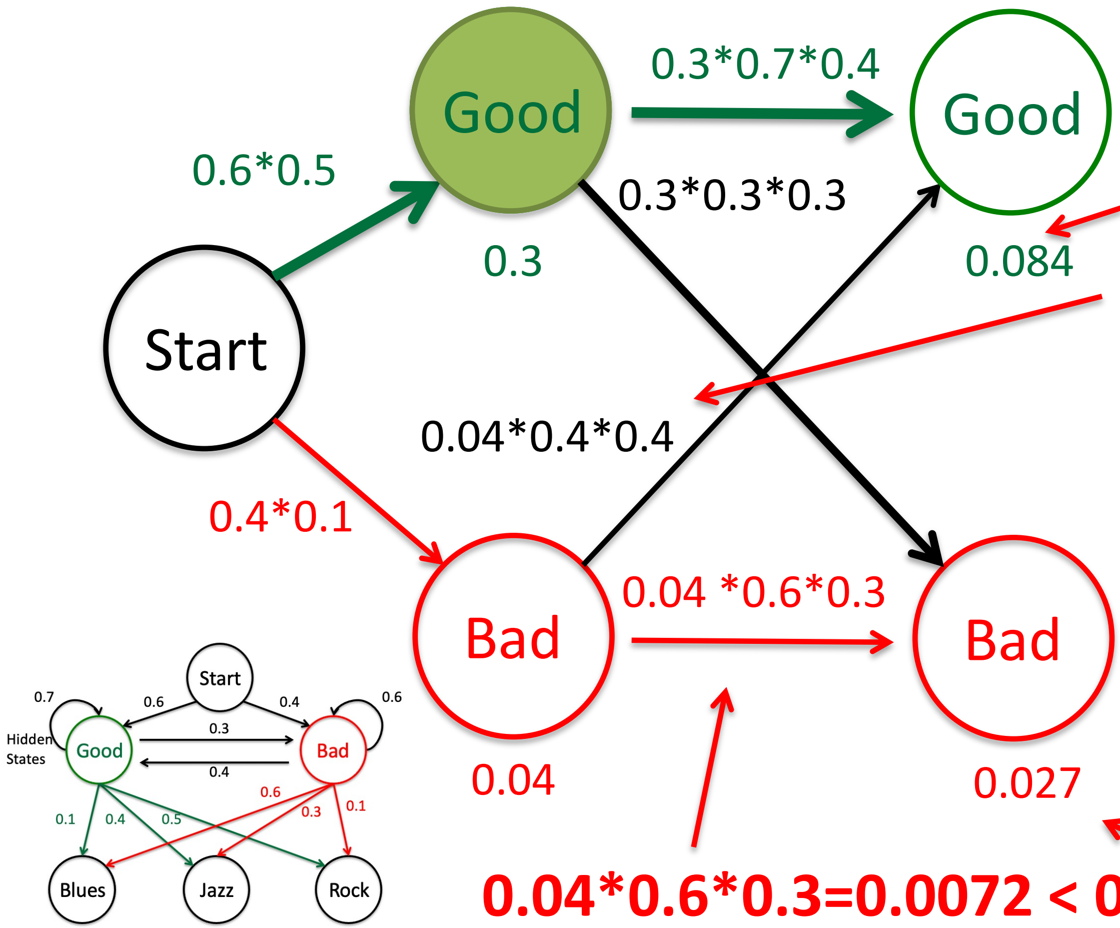
New current estimate for Bad if Good yesterday

We can estimate the most likely hidden state based on observations

Day 1: Observe Rock

Day 2: Observe Jazz

- **Repeat process for estimate from Bad State**
- **Keep highest estimate as most likely State**



0.04*0.4*0.4=0.0064 < 0.084

Keep 0.084 as most likely

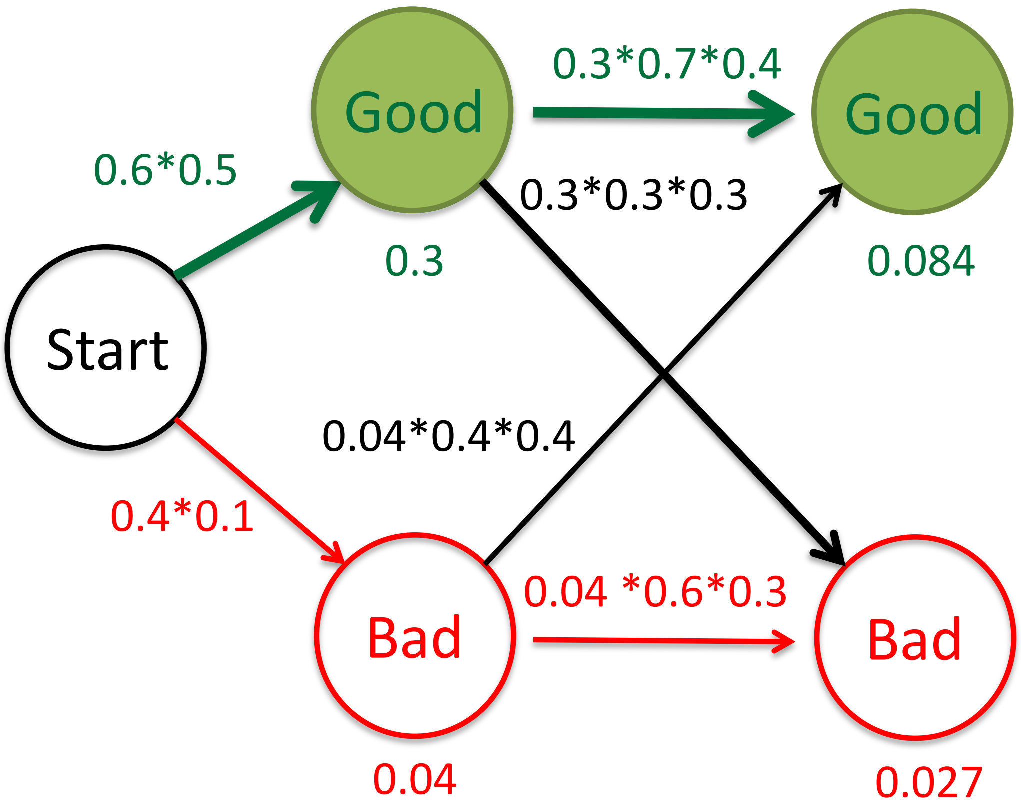
Sum for Forward algorithm

0.04*0.6*0.3=0.0072 < 0.027 so keep 0.027

We can estimate the most likely hidden state based on observations

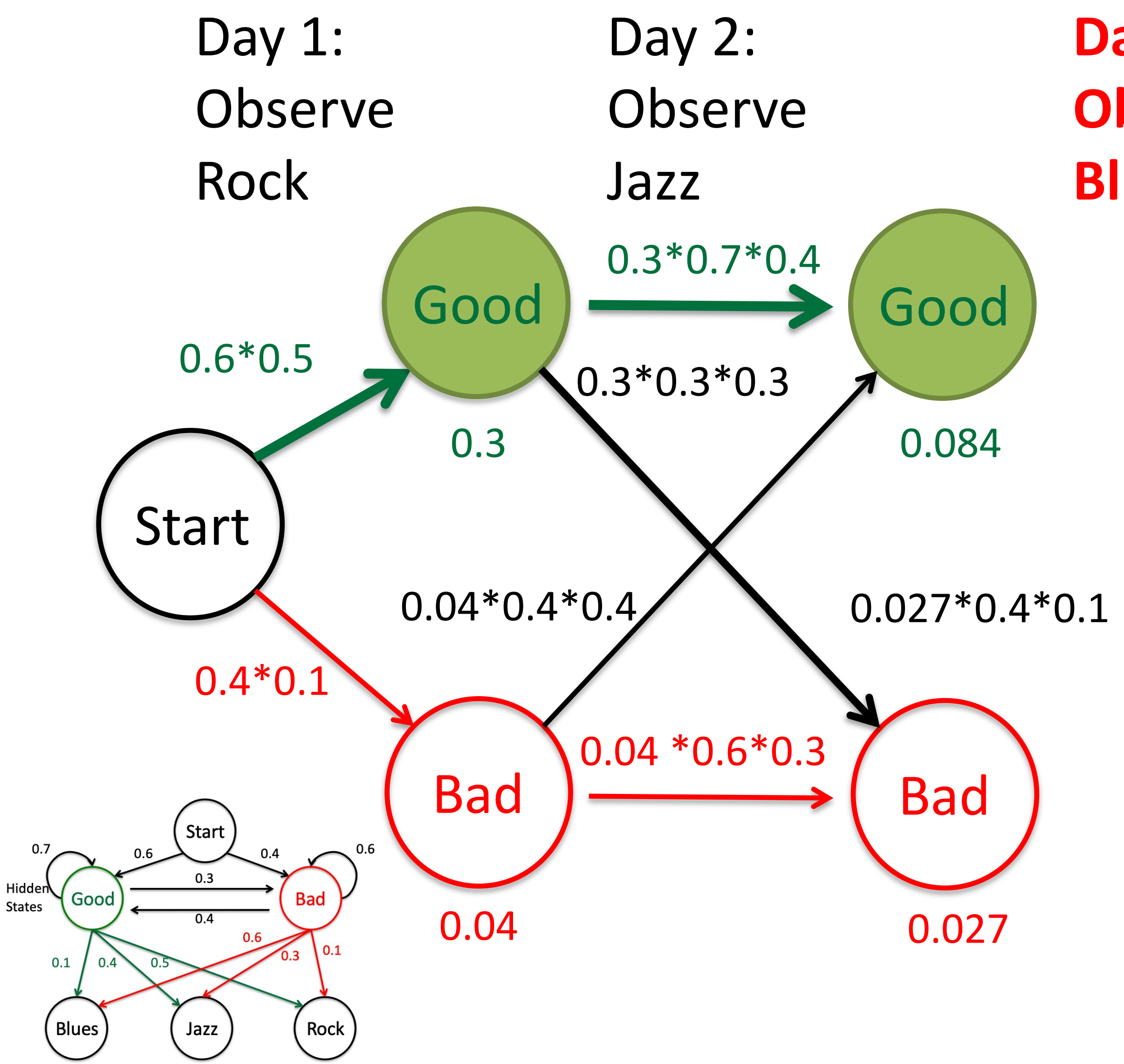
Day 1:
Observe
Rock

Day 2:
Observe
Jazz

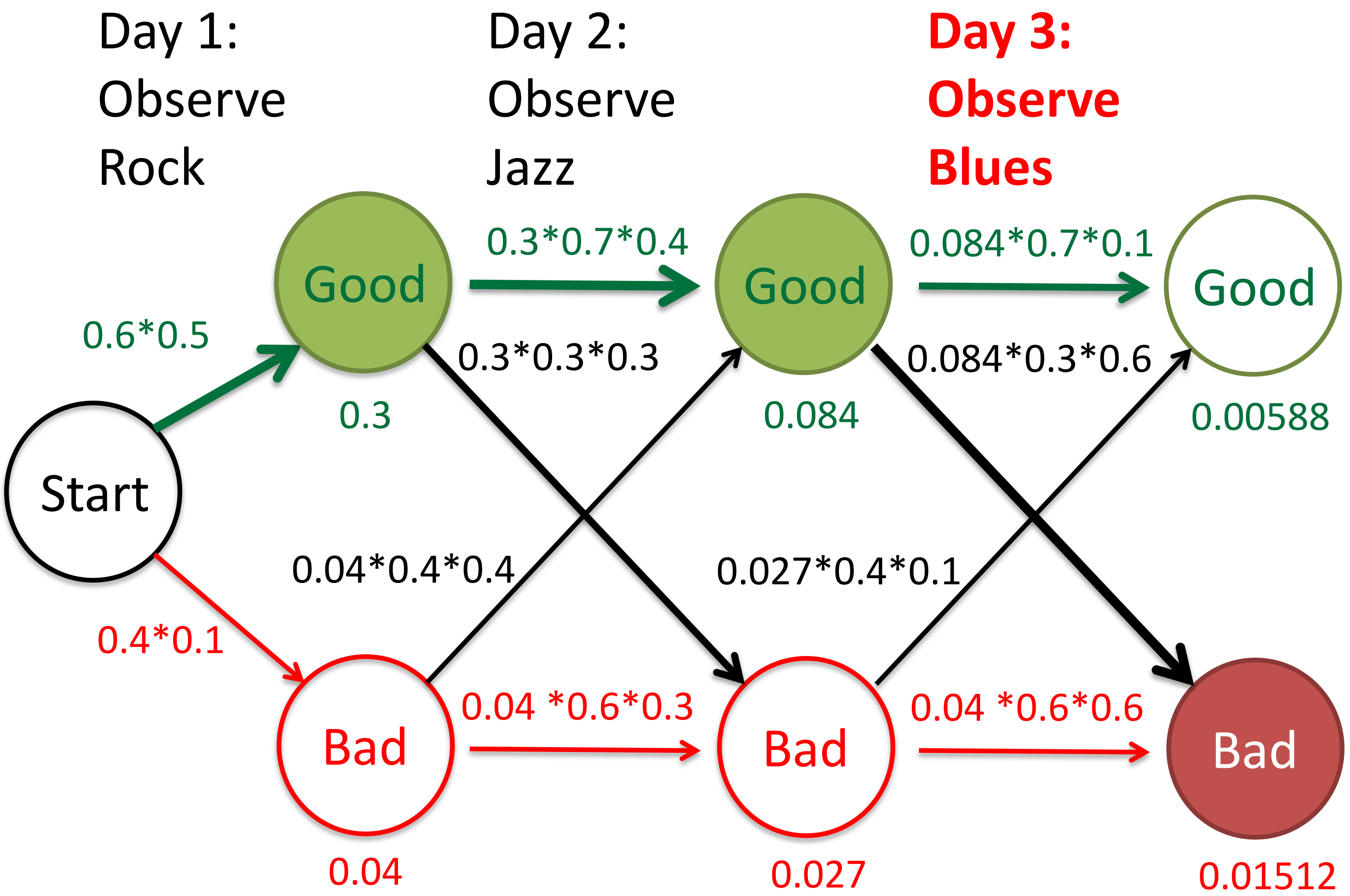


NOTE: score gets smaller with each observation!

We can estimate the most likely hidden state based on observations

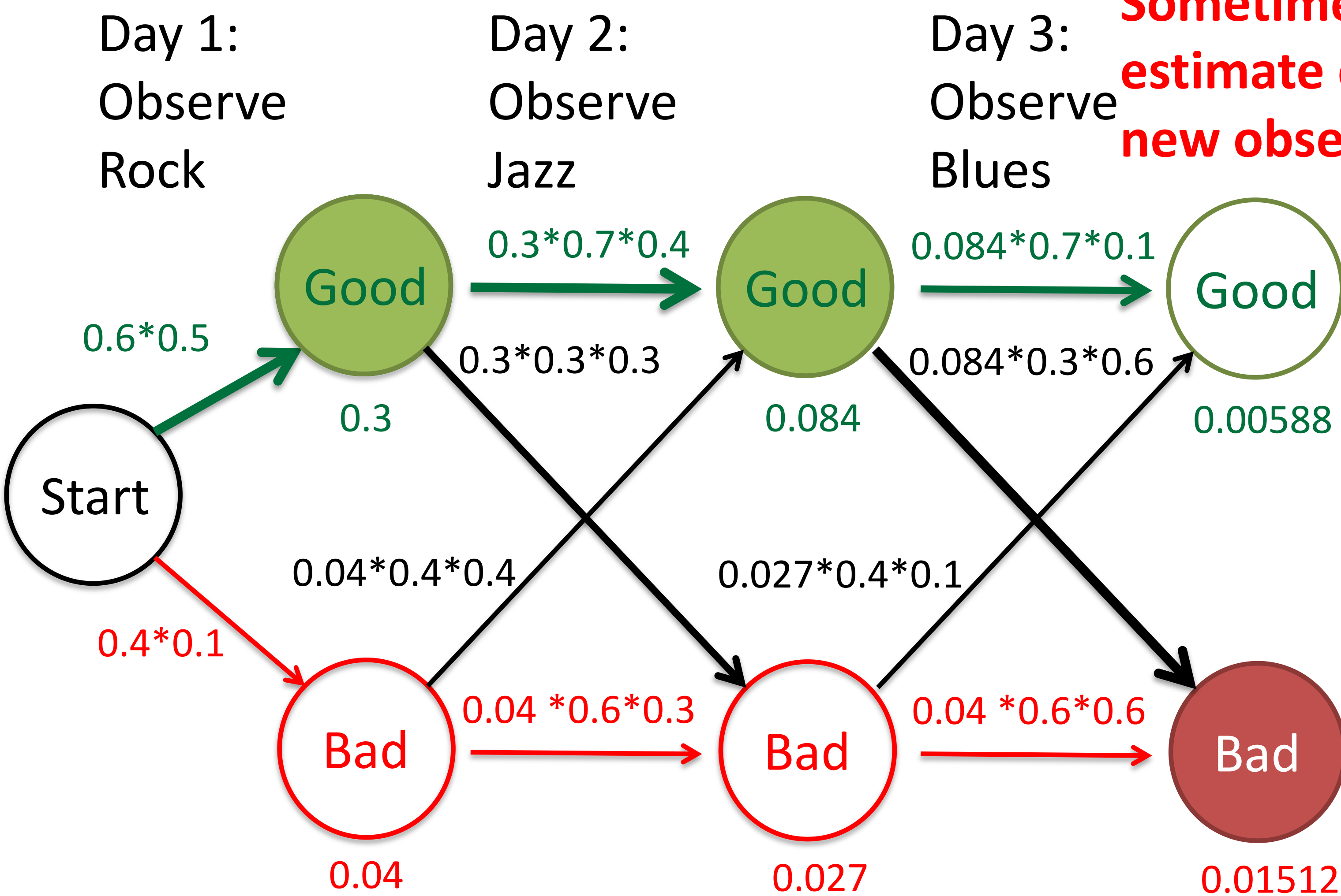


We can estimate the most likely hidden state based on observations

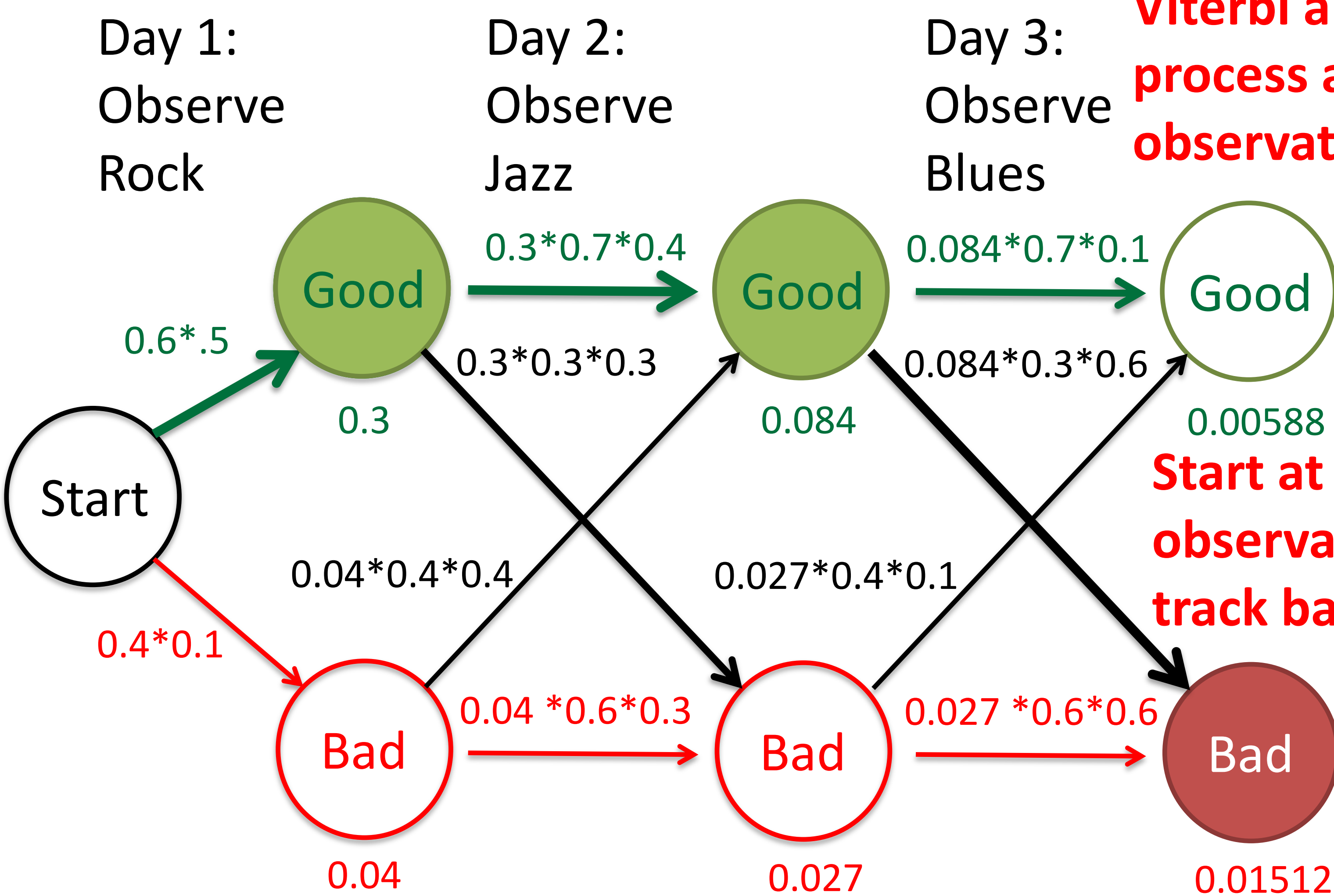


We can estimate the most likely hidden state based on observations

Sometimes path estimate changes on new observations



Viterbi algorithm back tracks to find most likely state sequence given observations



Viterbi algorithm: process all observations

Start at last observation and track back to start

Given observations of {Rock, Jazz, Blues}

The boss's mood mostly likely was {Good, Good, Bad}

Viterbi allow us to determine the most likely sequence of state transitions

Key points

We can't directly observe the hidden state so we can't know the true state with certainty

If there is something we *can* observe, we might be able to *infer* the true state with greater accuracy than guessing

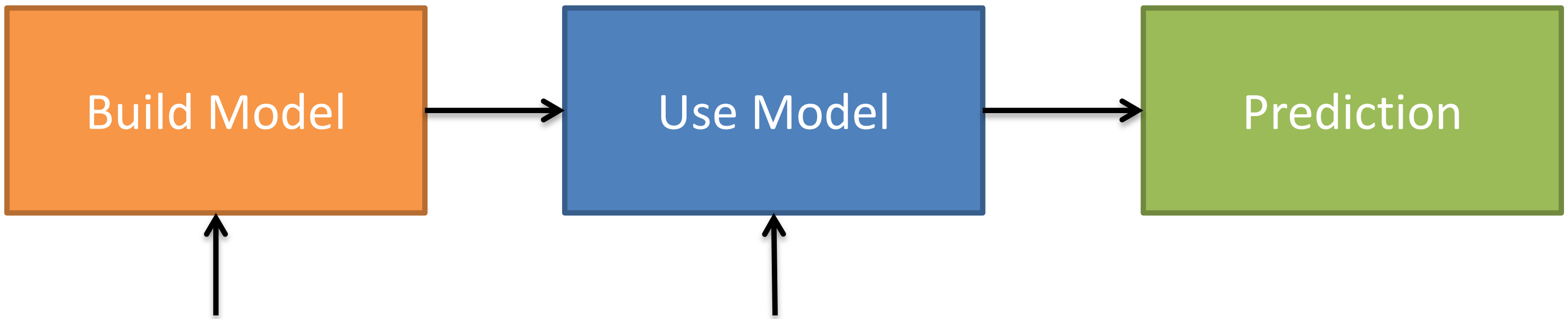
Given a sequence of observations we can determine the most likely state transitions over time

Agenda

1. Pattern matching vs. recognition
2. From Finite Automata to Hidden Markov Models
3. Decoding: Viterbi algorithm
4. Training

First we build a model, then we use it to make predictions on new data

Simplified machine learning pipeline



Training data annotated with actual outcome (e.g., weather was Hot, I ate 3 ice cream cones)

Want many samples of training data to learn system's behavior

New data not seen in training (e.g., I ate 2 ice cream cones, what was the weather?)

Predict outcome of new data (e.g., based on behavior in the training data, the weather was most likely Hot)

To build an HMM we start with previous observations called training data

Annotated training data gives transition probabilities

Situation:

Have a diary with of number of ice cream cones eaten each day when the weather was Hot or Cold

Diary provides the *annotated* training data to build a HMM

Later we will use the model to make predictions (e.g., given the number of cones eaten on a different set of days, predict weather for those days)

Cones eaten is observable, weather is the hidden State

Identify observable States (cones eaten) and count number of times each occurs

Annotated training data gives transition probabilities

Diary entries:

1. Hot day today! I chowed down **three** whole cones.
2. Hot again. But I only ate **two** cones; need to run to the store and get more ice cream.
3. Cold today. Still, the ice cream was calling me, and I ate **one** cone.
4. Cold again. Kind of depressed, so ate a **couple** cones despite the weather.
5. Still cold. Only in the mood for **one** cone.
6. Nice hot day. Yay! Was able to eat a cone each for **breakfast, lunch, and dinner**.
7. Hot but was out all day and only had enough cash on me for **one** ice cream.
8. Brrrr, the weather turned cold really quickly. Only **one** cone today.
9. Even colder. Still ate **one** cone.
10. Defying the continued coldness by eating **three** cones.

Real world: normally have to pre-process data to get something like:

1 | Hot | 3 cones

2 | Hot | 2 cones

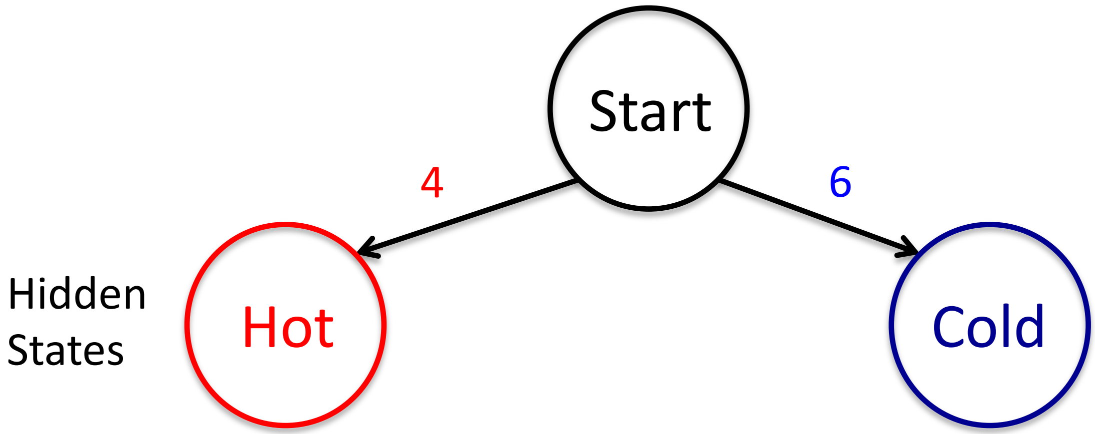
3 | Cold | 1 cone

Hidden states: Hot (4 days) or Cold (6 days)

Observations: 1, 2, or 3 ice cream cones eaten

Begin at Start, add vertex for each hidden State with counts from training data

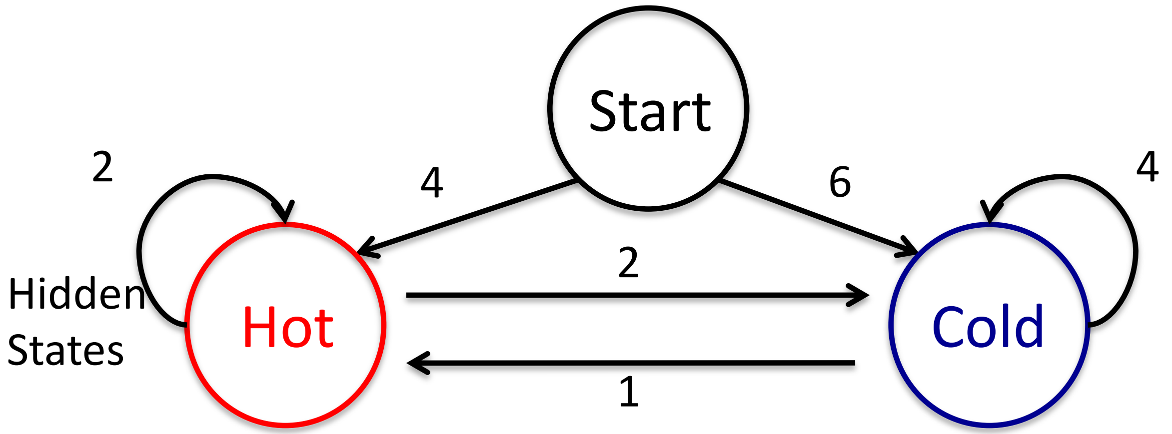
Count observations: 4 Hot days, 6 Cold days



1 | Hot | 3 cones
2 | Hot | 2 cones
3 | Cold| 1 cone
4 | Cold | 2 cones
5 | Cold | 1 cone
6 | Hot | 3 cones
7 | Hot | 1 cone
8 | Cold | 1 cone
9 | Cold | 1 cone
10 | cold | 3 cones

Add transitions between hidden States using count of next day's hidden State

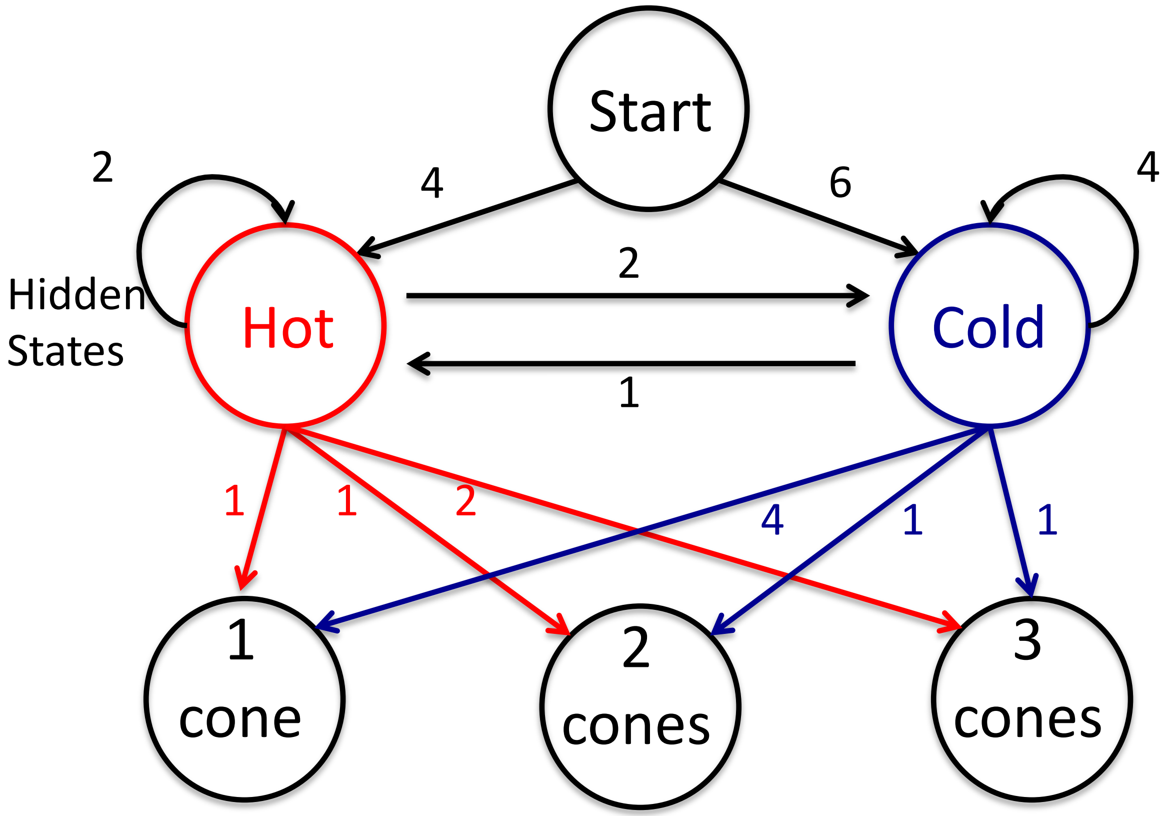
Count observations: transitions between hidden states (e.g., Hot->Hot)



1 | Hot | 3 cones
2 | Hot | 2 cones
3 | Cold| 1 cone
4 | Cold | 2 cones
5 | Cold | 1 cone
6 | Hot | 3 cones
7 | Hot | 1 cone
8 | Cold | 1 cone
9 | Cold | 1 cone
10 | cold | 3 cones

For each hidden State, count the number of occurrences of each observation

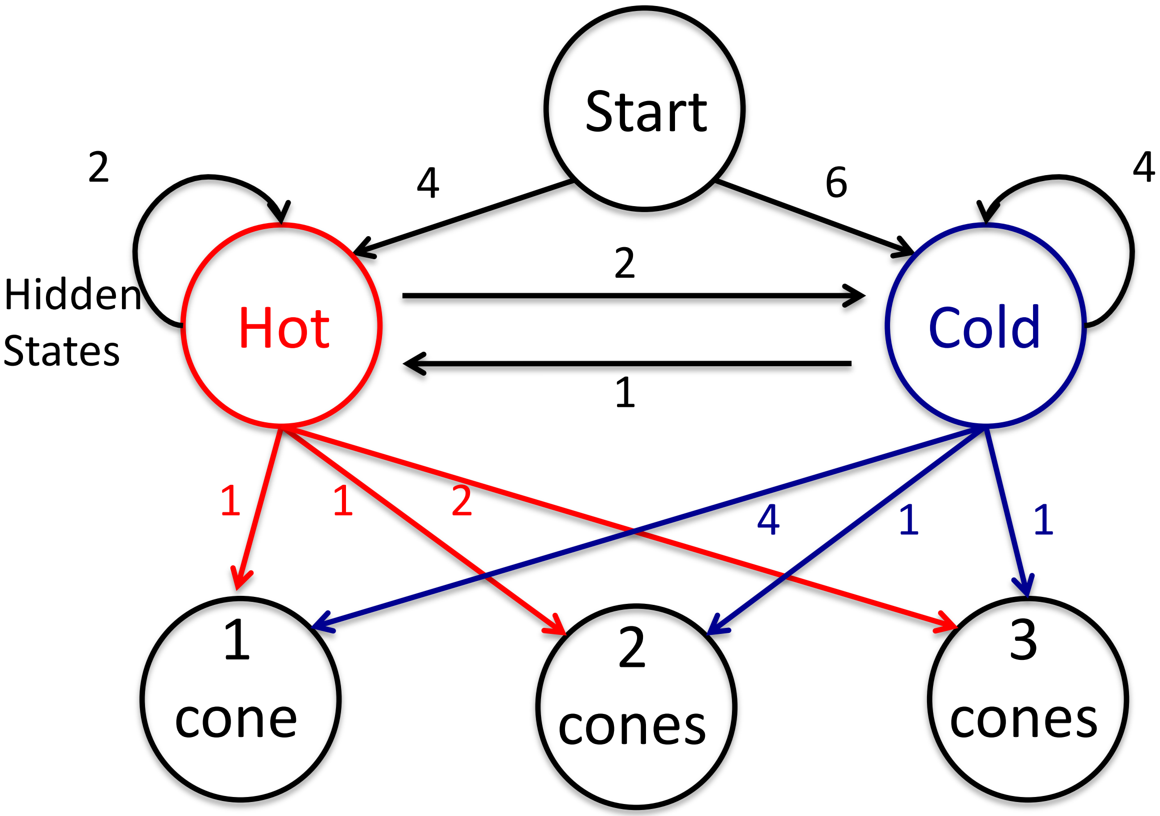
Count observations: cones eaten when Cold



1 | Hot | 3 cones
2 | Hot | 2 cones
3 | Cold| 1 cone
4 | Cold | 2 cones
5 | Cold | 1 cone
6 | Hot | 3 cones
7 | Hot | 1 cone
8 | Cold | 1 cone
9 | Cold | 1 cone
10 | cold | 3 cones

Convert observations counts into probabilities by dividing by total count

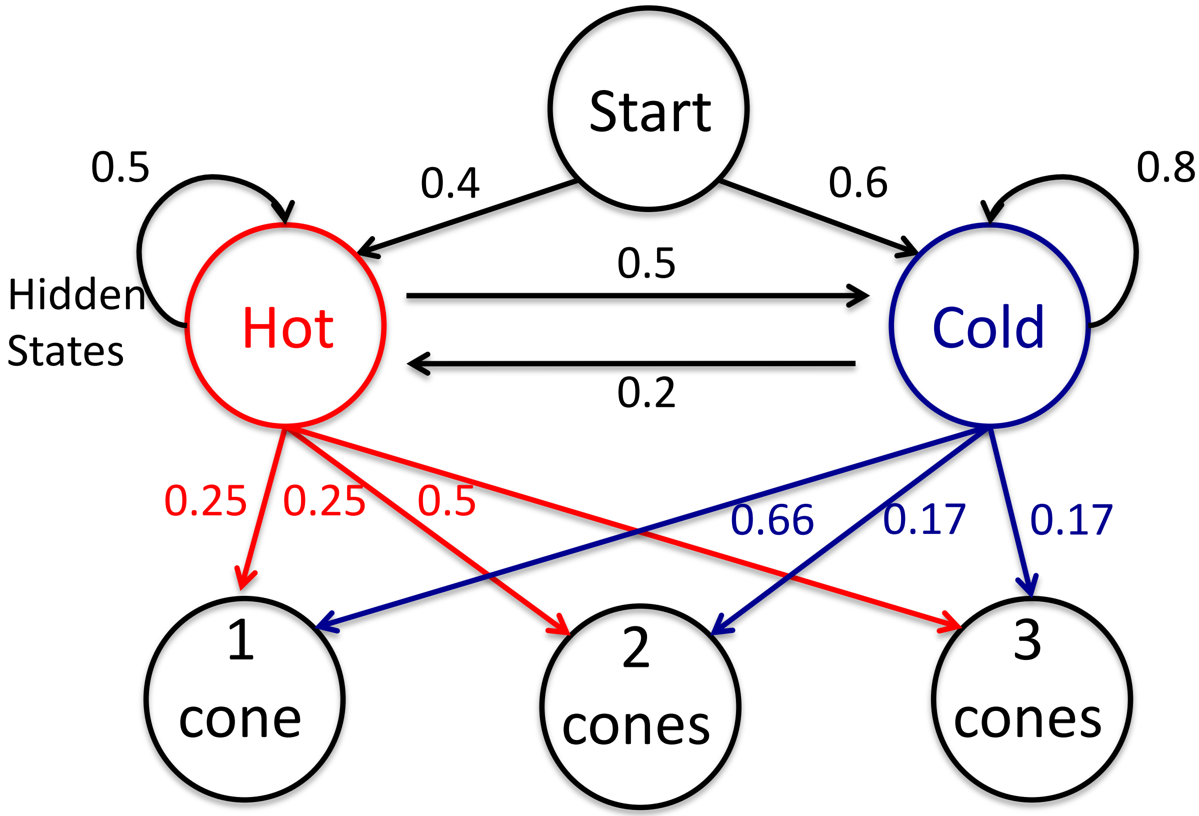
Convert to probabilities



Probability = count/total count

Convert observations into probabilities by dividing count by total count

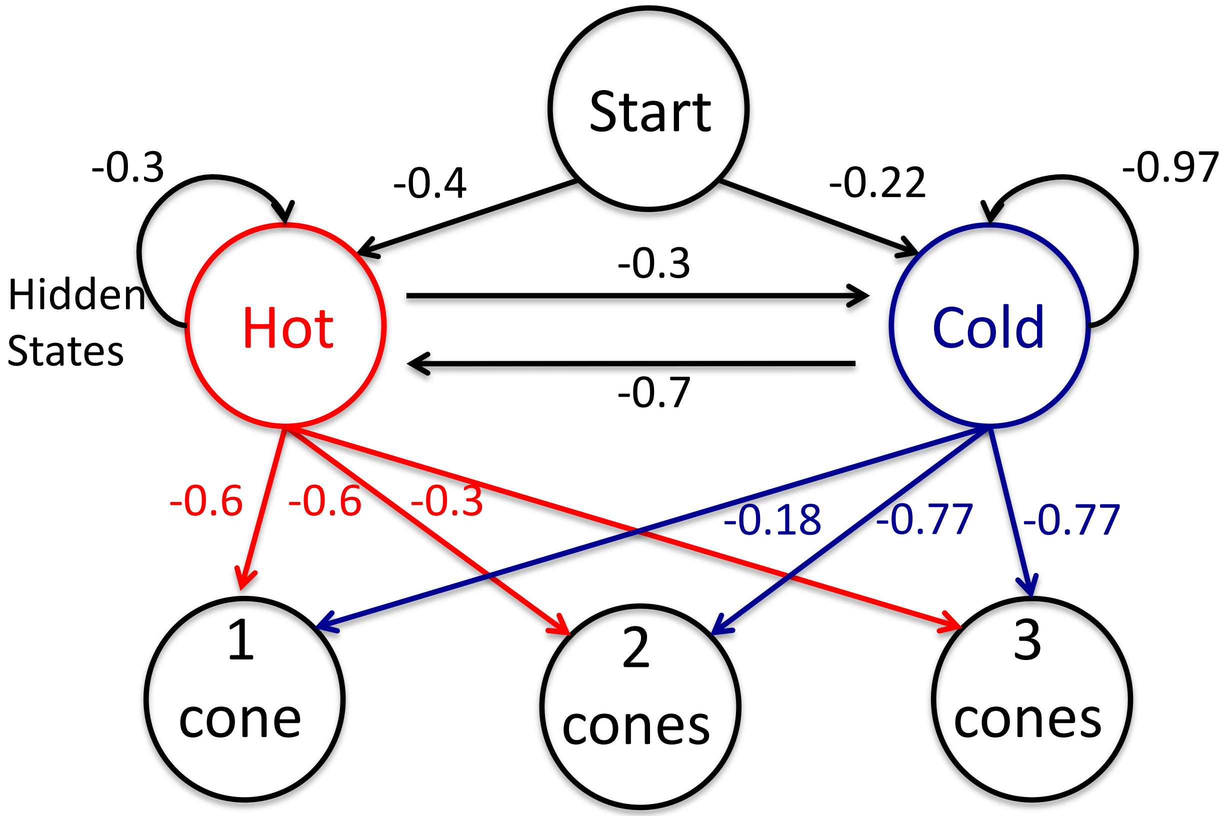
Probabilities based on observations



Problem in using probabilities in Viterbi algorithm: repeatedly multiplying numbers less than 1 quickly leads to numerical precision problems

Use logarithms to help with numerical precision problem

Log probabilities based on observations



A fact about logarithms can help us avoid precision issues:

log(mn) = log(m) + log(n)

To calculate score, add logs of each factor instead of multiplying probabilities

Take log (base 10 here, natural log in PS-5) of each probability

Negative numbers are ok, we will soon choose largest score (least negative)

Model built: given number of cones eaten, calculate most likely weather on each day

New set of observations

Day 1:
Two cones

Weather
Hot or Cold?

Day 2:
Three cones

Weather
Hot or Cold?

Day 3:
Two cones

Weather
Hot or Cold?

Observations {Two cones, three cones, two cones}

Begin at Start State with 0 current score

#	Observation	nextState	currrentState	currScore + transScore + observation	nextScore
Start	n/a	Start	n/a	0	0

Observations {Two cones, three cones, two cones}

First observation is two cones eaten, calculate score for each possible next State

#	Observation	nextState	currrentState	currScore + transScore + observation	nextScore
Start	n/a	Start	n/a	0	0
0	Two cones	Cold	Start	0-0.22-0.77	-0.99
		Hot	Start	0-0.4-0.6	-1.0

Best guess is first day is Cold

Observations {Two cones, three cones, two cones}
Most likely {Cold} (largest score)

Next observation is three cones eaten, calculate score for each possible next State

#	Observation	nextState	currrentState	currScore + transScore + observation	nextScore
Start	n/a	Start	n/a	0	0
0	Two cones	Cold	Start	0-0.22-0.77	-0.99
		Hot	Start	0-0.4-0.6	-1.0
1	Three cones	Cold	Cold	-0.99-0.97-0.77	~~-2.73~~
		Cold	Hot	-1-0.3-0.77	-2.07
		Hot	Cold	-0.99-0.7-0.3	~~-1.99~~
		Hot	Hot	-1-0.3-0.3	-1.6

Observations {Two cones, three cones, two cones}
Most likely {Hot Hot }

Next observation is two cones eaten, calculate score for each possible next State

#	Observation	nextState	currrentState	currScore + transScore + observation	nextScore
Start	n/a	Start	n/a	0	0
0	Two cones	Cold	Start	0-0.22-0.77	-0.99
		Hot	Start	0-0.4-0.6	-1.0
1	Three cones	Cold	Cold	-0.99-0.97-0.77	~~-2.73~~
		Cold	Hot	-1-0.3-0.77	-2.07
		Hot	Cold	-0.99-0.7-0.3	~~-1.99~~
		Hot	Hot	-1-0.3-0.3	-1.6
2	Two cones	Cold	Cold	-2.07-0.97-0.77	~~-3.81~~
		Cold	Hot	-1.6-0.3-0.77	-2.67
		Hot	Cold	-2.07-0.7-0.6	~~-3.37~~
		Hot	Hot	-1.6-0.3-0.6	-2.5

Observations {Two cones, three cones, two cones}
Most likely {Hot Hot Hot }

Because estimates can change, start at end and work backward to find most likely path

#	Observation	nextState	currrentState	currScore + transScore + observation	nextScore
Start	n/a	Start	n/a	0	0
0	Two cones	Cold	Start	0-0.22-0.77	-0.99
		Hot	Start	0-0.4-0.6	-1.0
1	Three cones	Cold	Cold	-0.99-0.97-0.77	~~-2.73~~
		Cold	Hot	-1-0.3-0.77	-2.07
		Hot	Cold	-0.99-0.7-0.3	~~-1.99~~
		Hot	Hot	-1-0.3-0.3	-1.6
2	Two cones	Cold	Cold	-2.07-0.97-0.77	~~-3.81~~
		Cold	Hot	-1.6-0.3-0.77	-2.67
		Hot	Cold	-2.07-0.7-0.6	~~-3.37~~
		Hot	Hot	-1.6-0.3-0.6	-2.5

Previous came from Hot

Back track to largest where nextState is Hot

Most likely nextState at end was Hot

Observations {Two cones, three cones, two cones}

Most likely {Hot Hot Hot }

The weather was most likely Hot, Hot, Hot

Best estimates of hidden State given new set of observations

Day 1:
Two cones

Weather
Hot

Day 2:
Three cones

Weather
Hot

Day 3:
Two cones

Weather
Hot

Observations {Two cones, three cones, two cones}
Most likely {Hot Hot Hot }

PS-5 due on 5/23 at 11:59pm ET

Training

Input

train-sentences

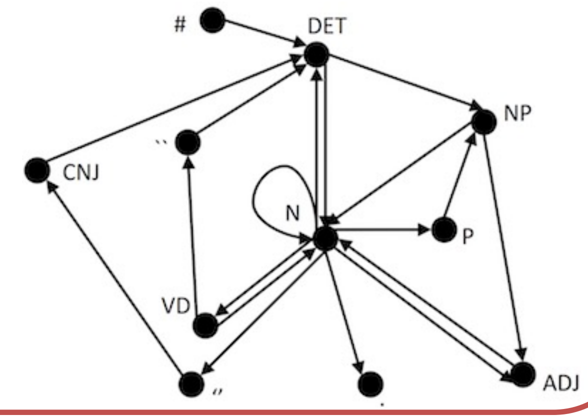
your work is beautiful .

train-tags

PRO N V ADJ .

Training

Trained HMM



Testing

Input

```
The Fulton County Grand
Jury said Friday an
investigation of
Atlanta's recent
primary election
produced `` no evidence
'' that any
irregularities took
place .
```

Trained HMM

Output

The/DET Fulton/NP County/N Grand/ADJ Jury/N said/VD Friday/N an/DET investigation/N of/P Atlanta's/NP recent/ADJ primary/N election/N produced/VD ``/`` no/DET evidence/N ''/'' that/CNJ any/DET irregularities/N took/VD place/N ./.

Summary

- Hidden Markov models for recovering the most likely hidden state given a sequence of observations
  - Markov property: it doesn’t matter how we got to a state, the current state is all we need to predict the next state
  - Modeling similar to finite automata
  - Viterbi Algorithm to find the most likely sequence
  - Training is necessary to build the model

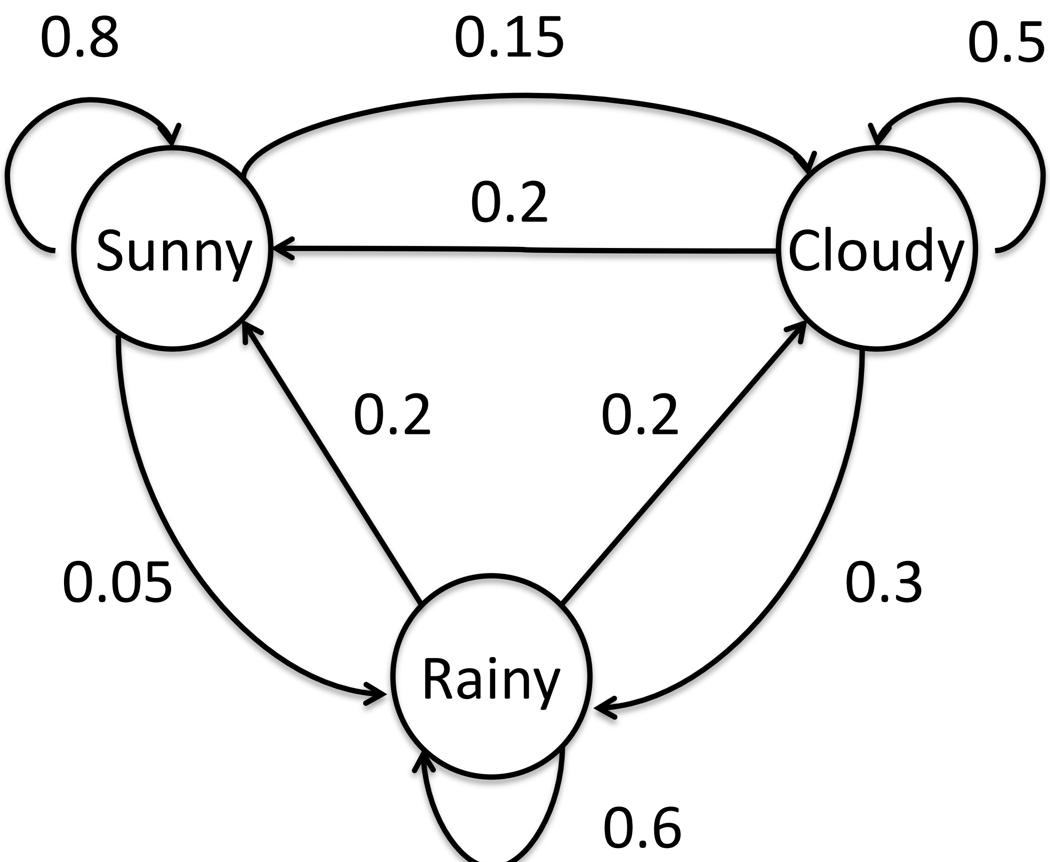
Additional Resources

Weather model

ANNOTATED SLIDES

Markov property suggests it doesn't really matter how we got into the current State

Given current State, can predict likelihood of future states



Given that we can observe the state we are in, it doesn't really matter how we got there:

- Probability of weather at time n, given the weather at time n-1, and at n-2, and n-3 …
- Is approximately equal to the probability of weather at time n given *only* the weather at n-1
- $P(w_n | w_{n-1}, w_{n-2}, w_{n-3}) \approx P(w_n | w_{n-1})$

Markov property: it doesn't matter how we got to a state, the current state is all we need to predict the next state

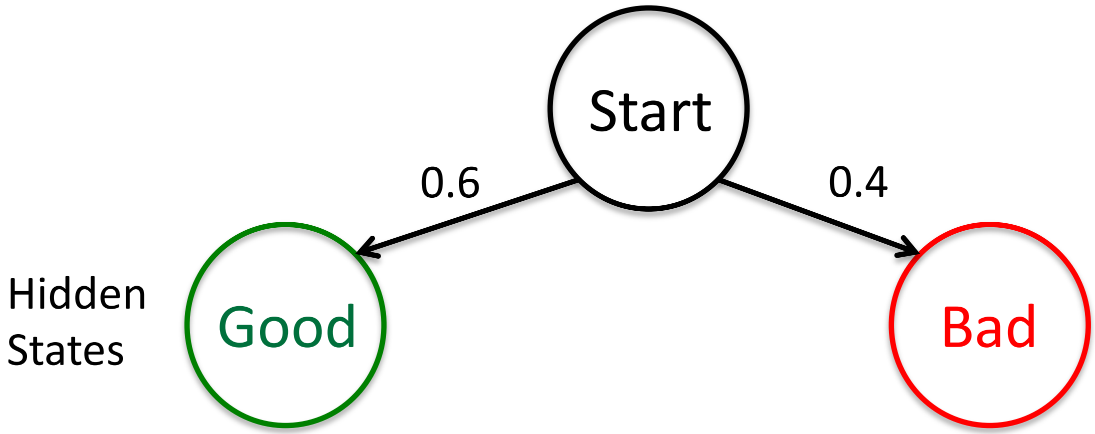
Adapted from: https://pdfs.semanticscholar.org/b328/2eb0509442b80760fea5845e158168daee62.pdf

Good/bad mood example

ANNOTATED SLIDES

Want to ask the boss for raise when the boss's state is a Good mood

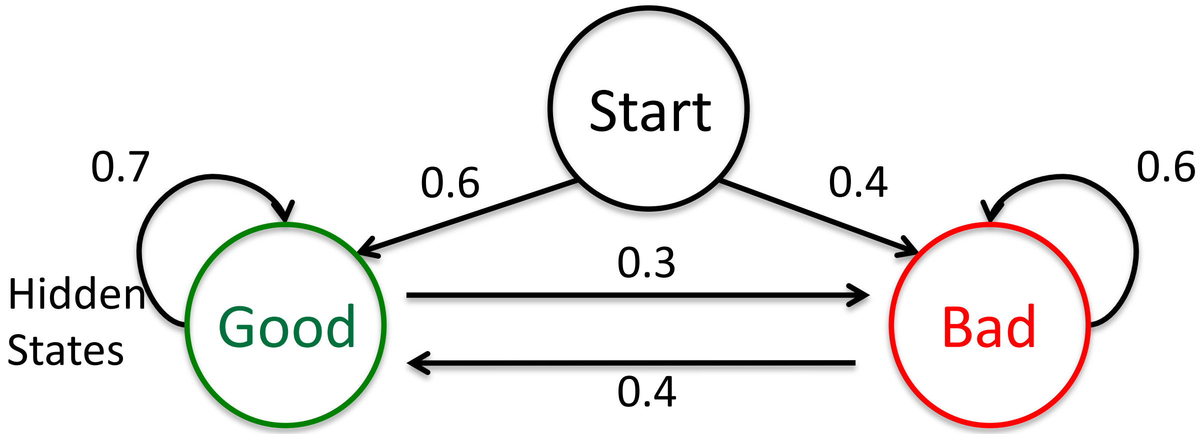
Gather stats about likelihood of states



- Can't know boss's mood for sure simply by looking (state is hidden)
- Want to know current state (Good or Bad)
- Could ask everyday and record statistics about it
- Assume boss answers truthfully:
  - Ask 100 times
  - 60 times good
  - 40 times bad
- Boss slightly more likely to be in good mood (60% chance)

In addition to states, find likelihood of *transitioning* from one state to another

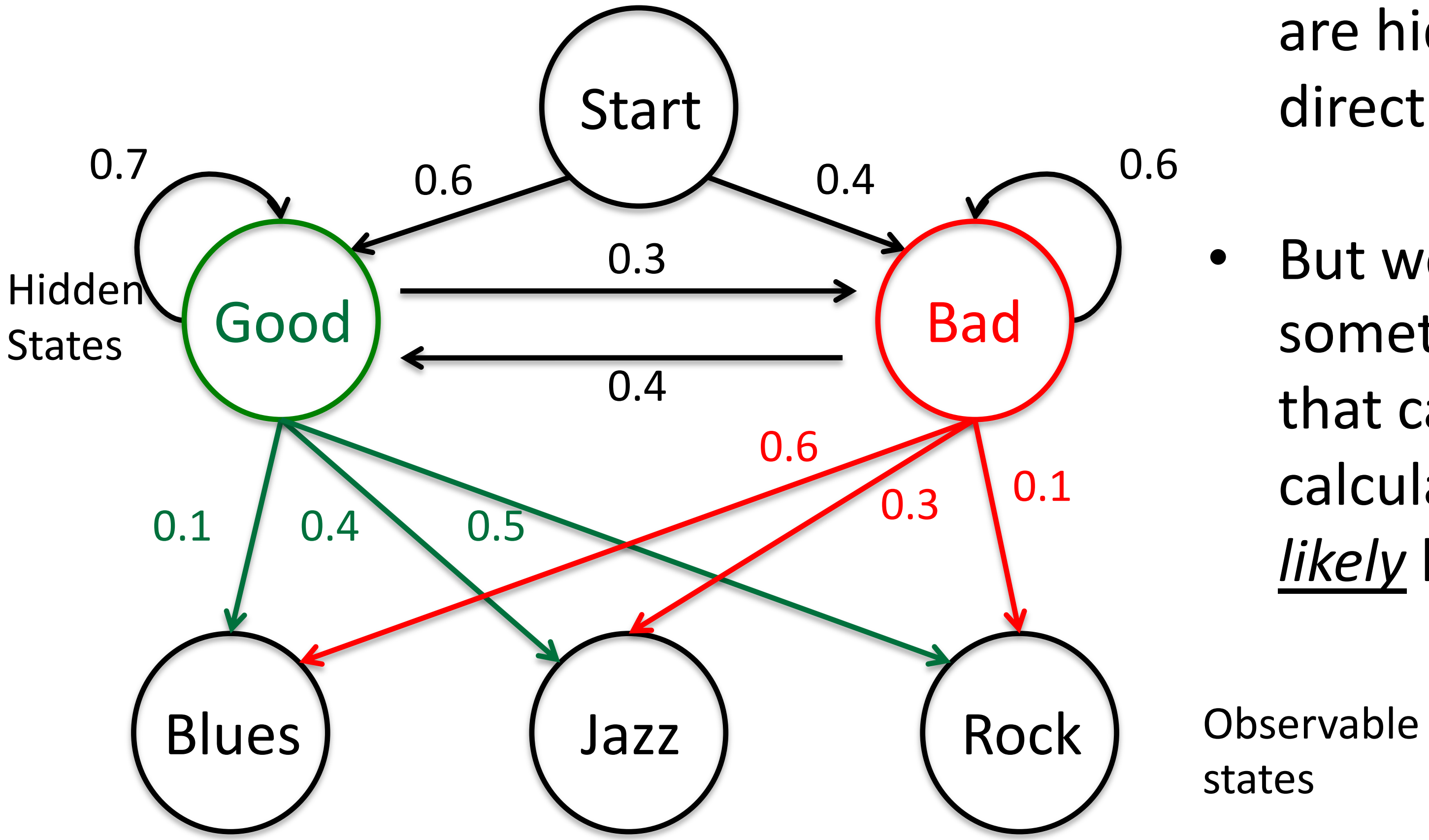
Gather stats about state transitions



- Watch boss on day after asking about mood, ask again next day
- Calculate probability of staying in same mood or ***transitioning*** to another mood (hidden state)
- Similar to how weather transitioned states

This is a Hidden Markov Model (HMM)

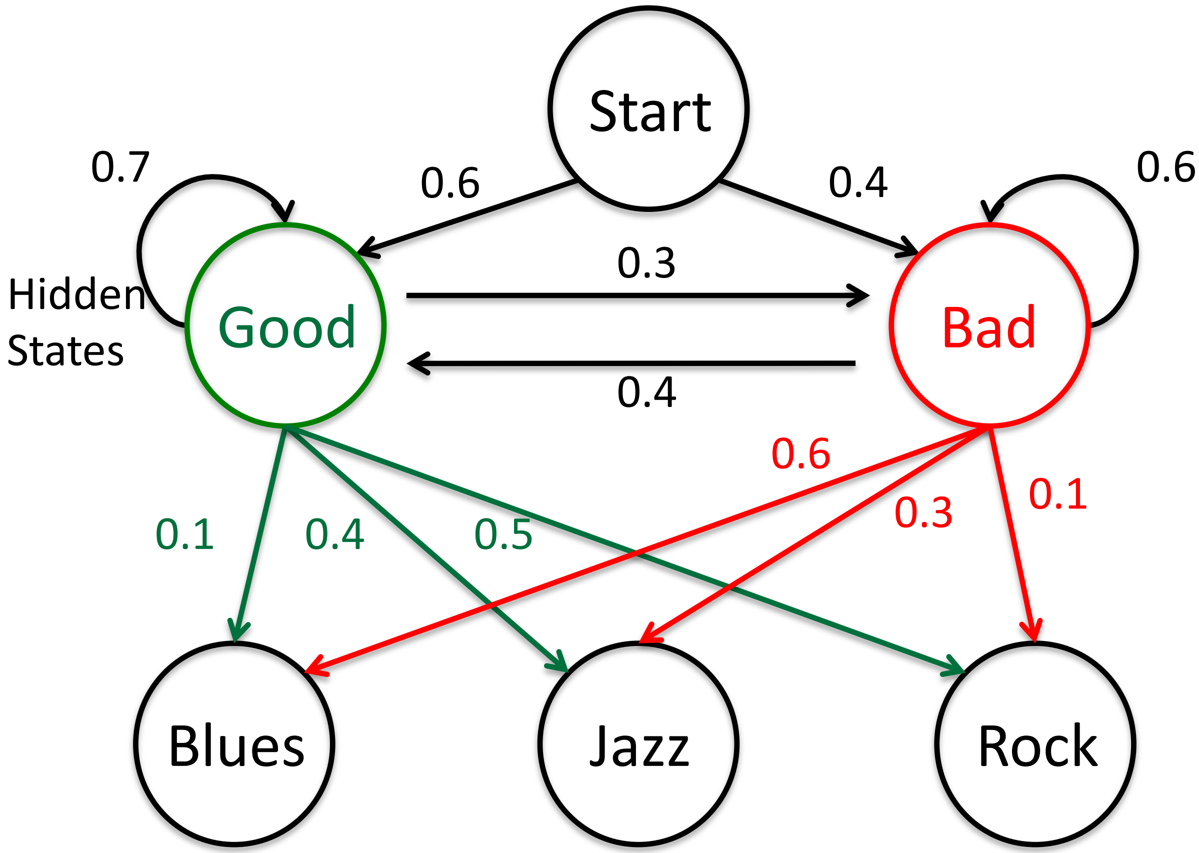
Hidden Markov Model



- States (boss's mood) are hidden, can't be directly observed
- But we *can* observe something (music) that can help us calculate the most *likely* hidden state

So is today a good day to ask for a raise?

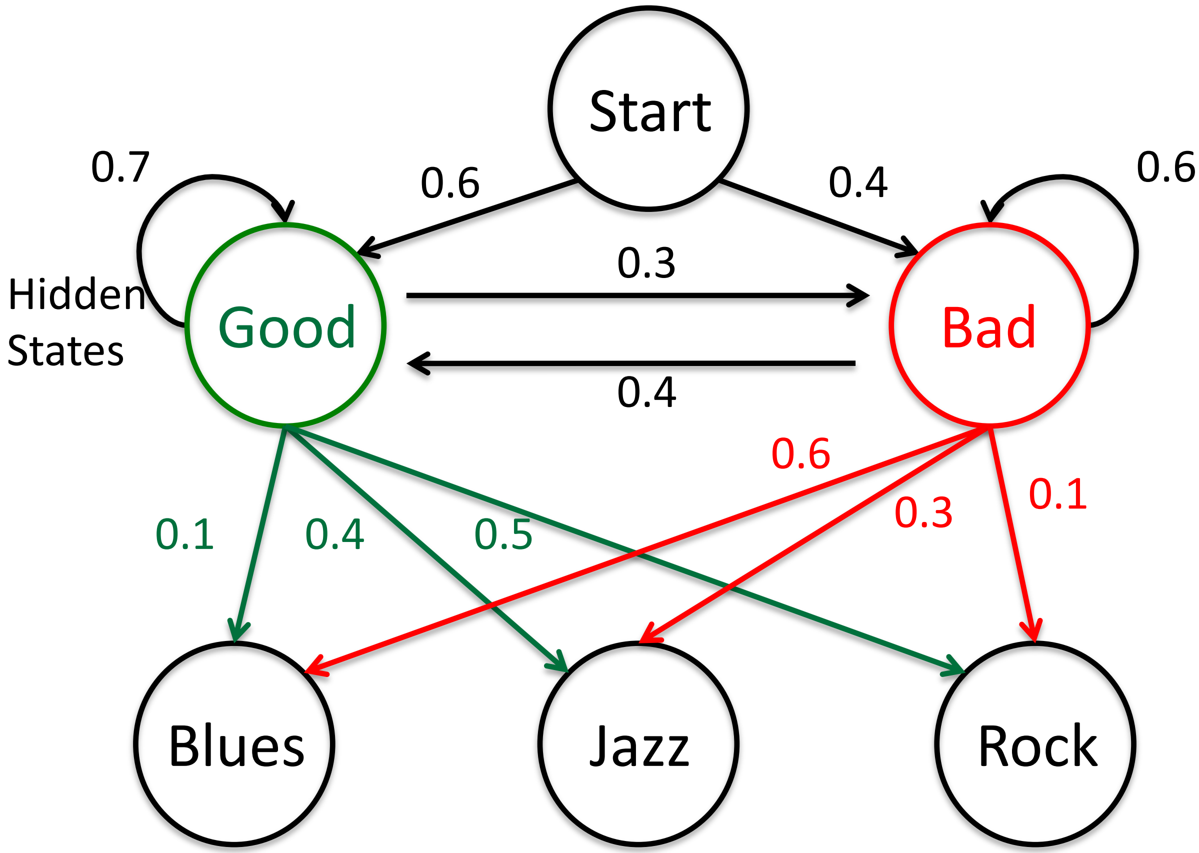
So far we have no music observation



- Given no other information, it's a pretty good bet the boss in Good mood
- Good mood = 0.6
- Bad mood = 0.4
- Yes, on any given day boss is slightly more likely to be in a good mood

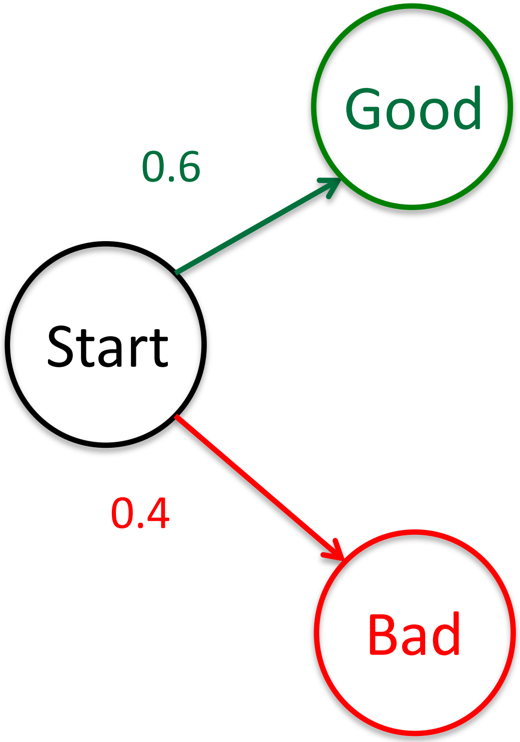
By observing music, we might be able to get a better sense of the boss's mood!

Observe Rock music



- Say today we observe the boss is playing Rock music
- Should we ask for a raise?
- Good mood = 0.6*0.5 = 0.3
- Bad mood = 0.4*0.1 = 0.04
- Most likely a good day to ask!

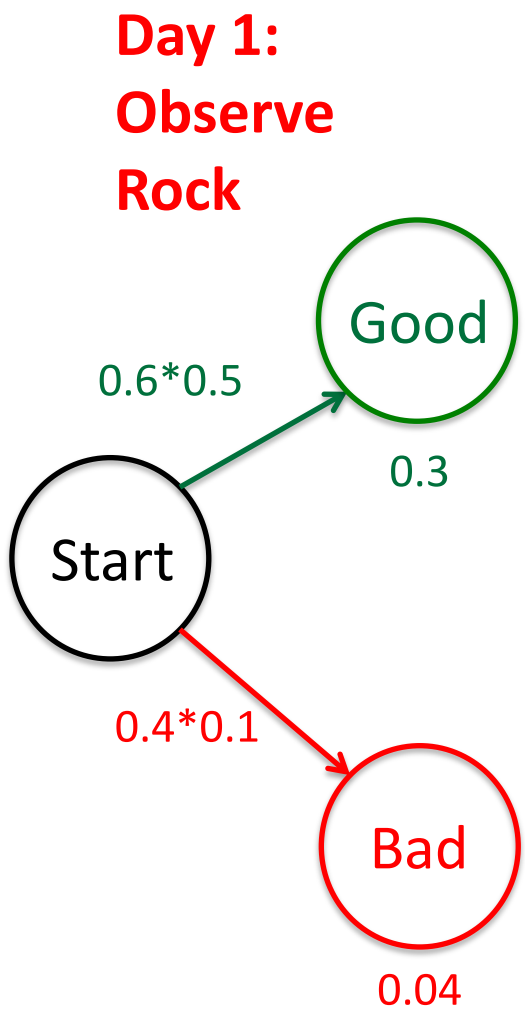
We can estimate the most likely hidden state based on observations



Given no observations, can make a guess at true state

Guess state with highest score

We can estimate the most likely hidden state based on observations



If we make an observation, we might be able to increase our accuracy

Multiply previous score by likelihood of observation

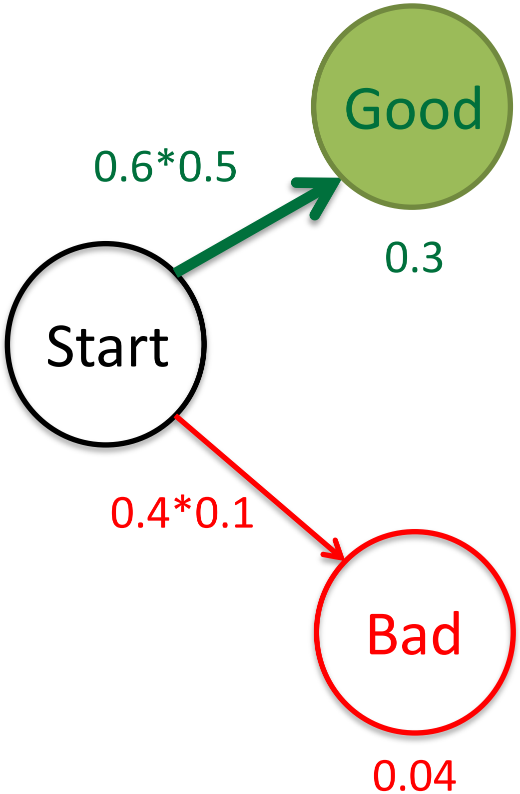
Most likely in a Good mood (~8X more likely)

Ask for a raise?
Yes!

We can estimate the most likely hidden state based on observations

Day 1: Observe Rock

Most likely State has highest score



If we make an observation, we might be able to increase our accuracy

Multiply previous score by likelihood of observation

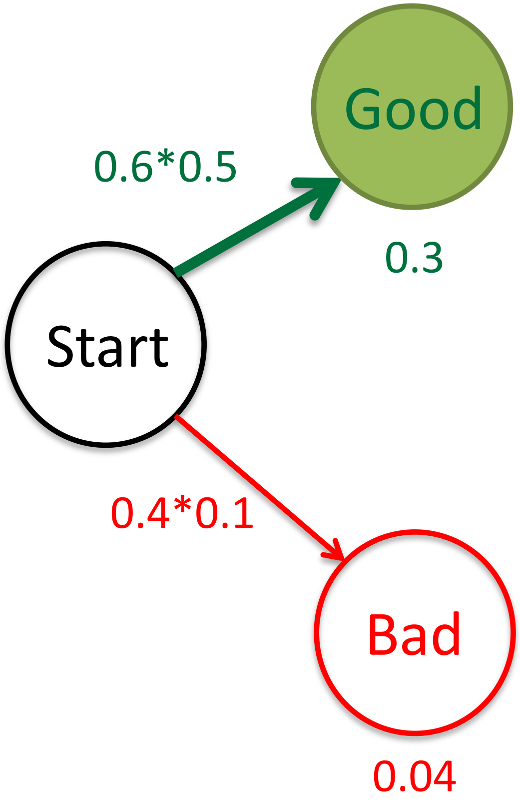
Most likely in a Good mood (~8X more likely)

Ask for a raise?
Yes!

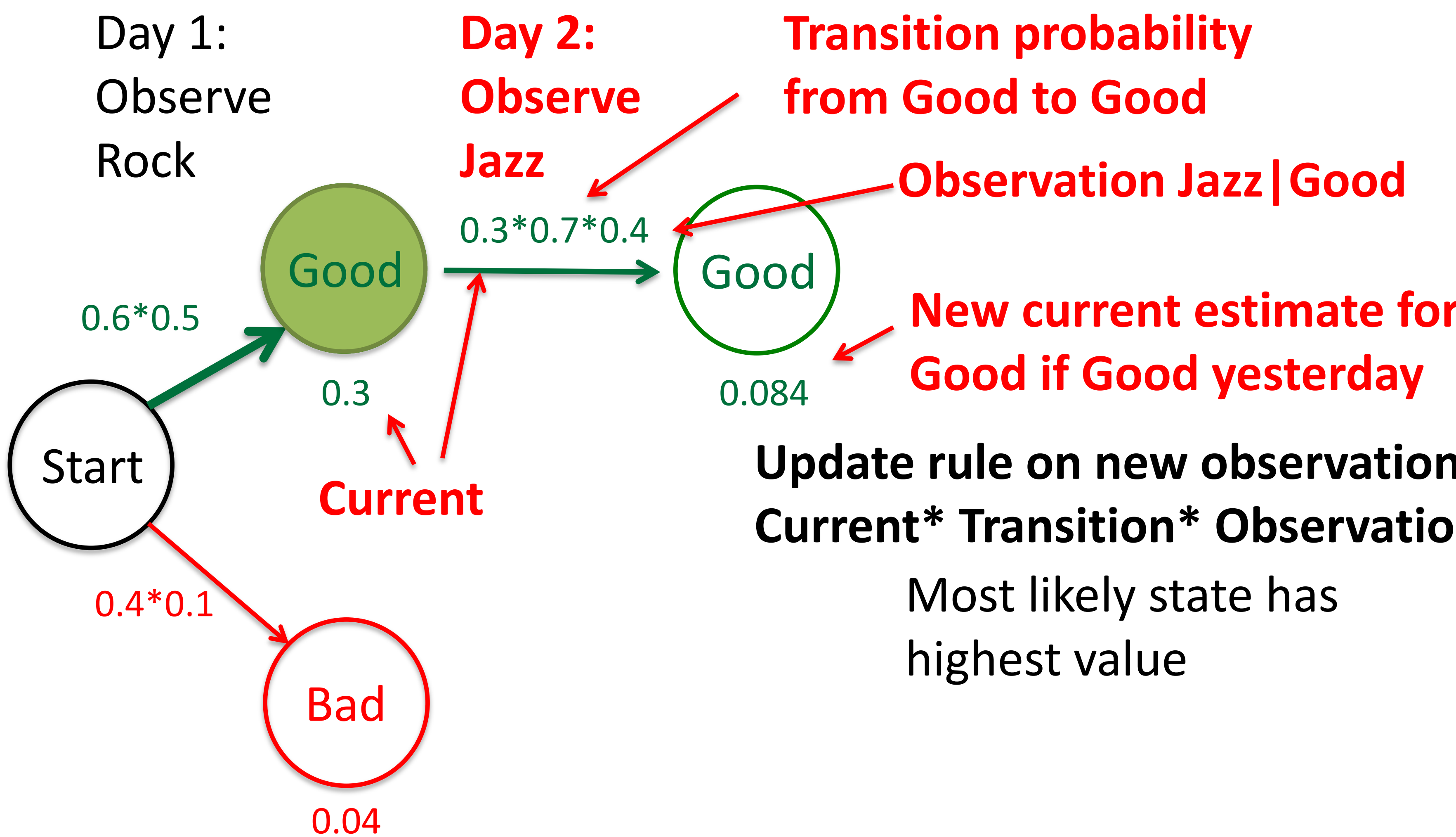
We can estimate the most likely hidden state based on observations

Day 1:
Observe
Rock

Day 2:
Observe
Jazz



We can estimate the most likely hidden state based on observations



Update rule on new observation:
Current* Transition* Observation

Most likely state has highest value

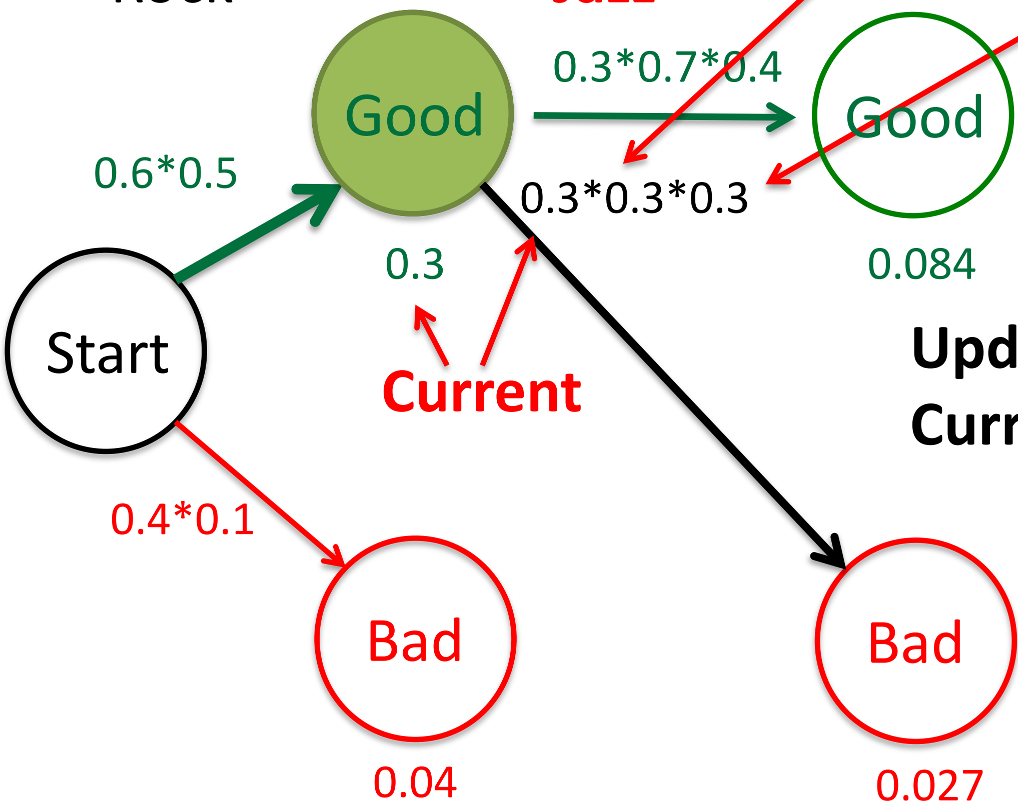
We can estimate the most likely hidden state based on observations

Day 1:
Observe
Rock

Day 2:
Observe
Jazz

Transition probability from Good to Bad

Observation Jazz|Bad



Do the same for possible transition from Good to Bad

Update rule on new observation:
Current* Transition* Observation

Most likely state has highest value

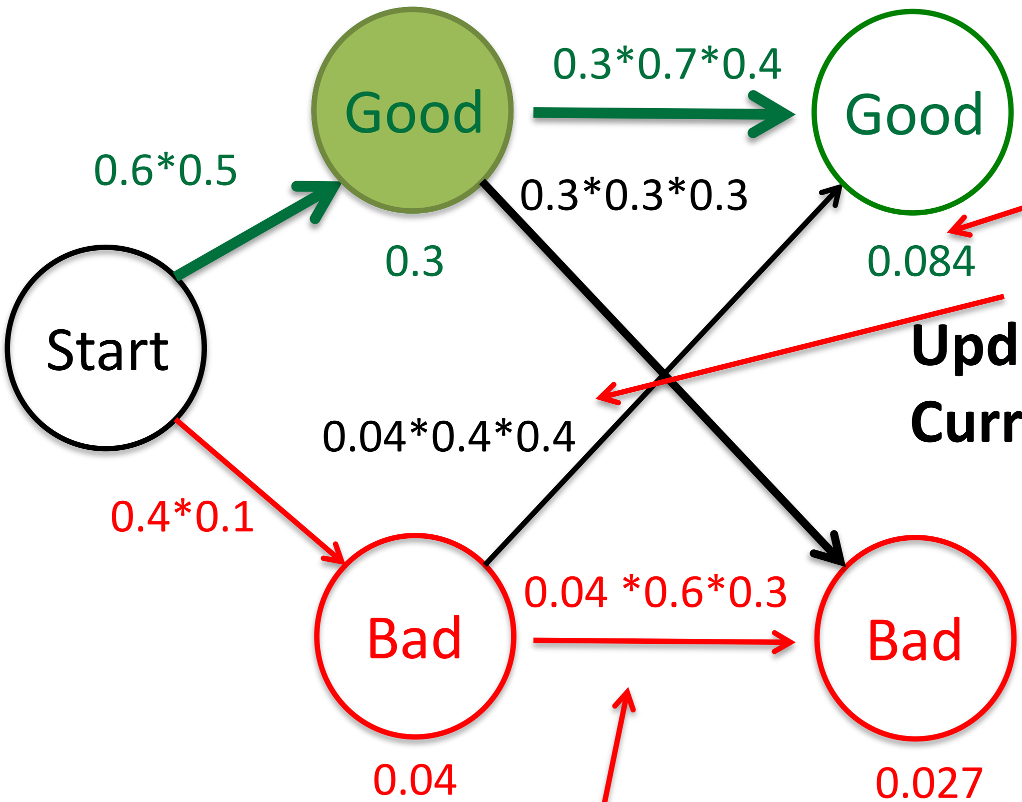
New current estimate for Bad if Good yesterday

We can estimate the most likely hidden state based on observations

Day 1:
Observe
Rock

Day 2:
Observe
Jazz

- **Repeat process for estimate from Bad State**
- **Keep highest estimate as most likely State**



0.04*0.4*0.4=0.0064 < 0.084
Keep 0.084 as most likely

Update rule:
Current* Transition* Observation

Sum for Forward algorithm

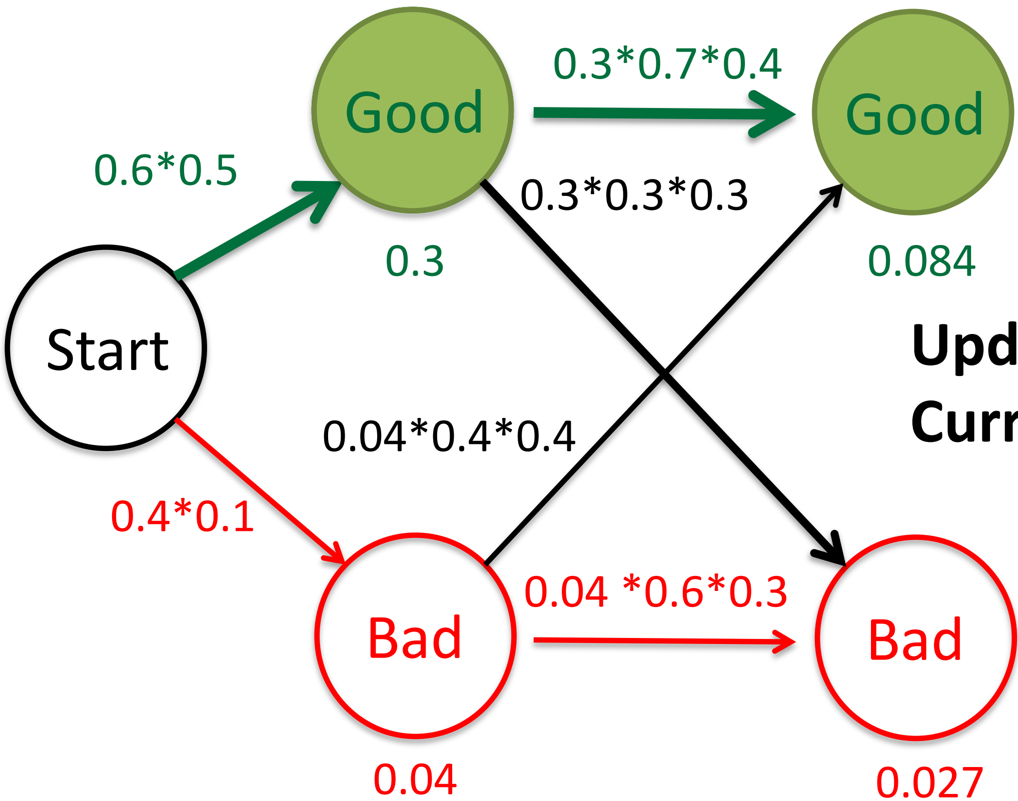
Most likely state has highest value

0.04*0.6*0.3=0.0072 < 0.027 so keep 0.027

We can estimate the most likely hidden state based on observations

Day 1:
Observe
Rock

Day 2:
Observe
Jazz



- **Most likely current State has highest score**
- **Most likely path given Observations of Rock then Jazz was Good mood yesterday, Good mood today**

Update rule:
Current* Transition* Observation

Most likely state has highest value

- **Now only about 3X more likely to be in Good mood**
- **Previously 8X more likely**
- **Structure called a trellis**

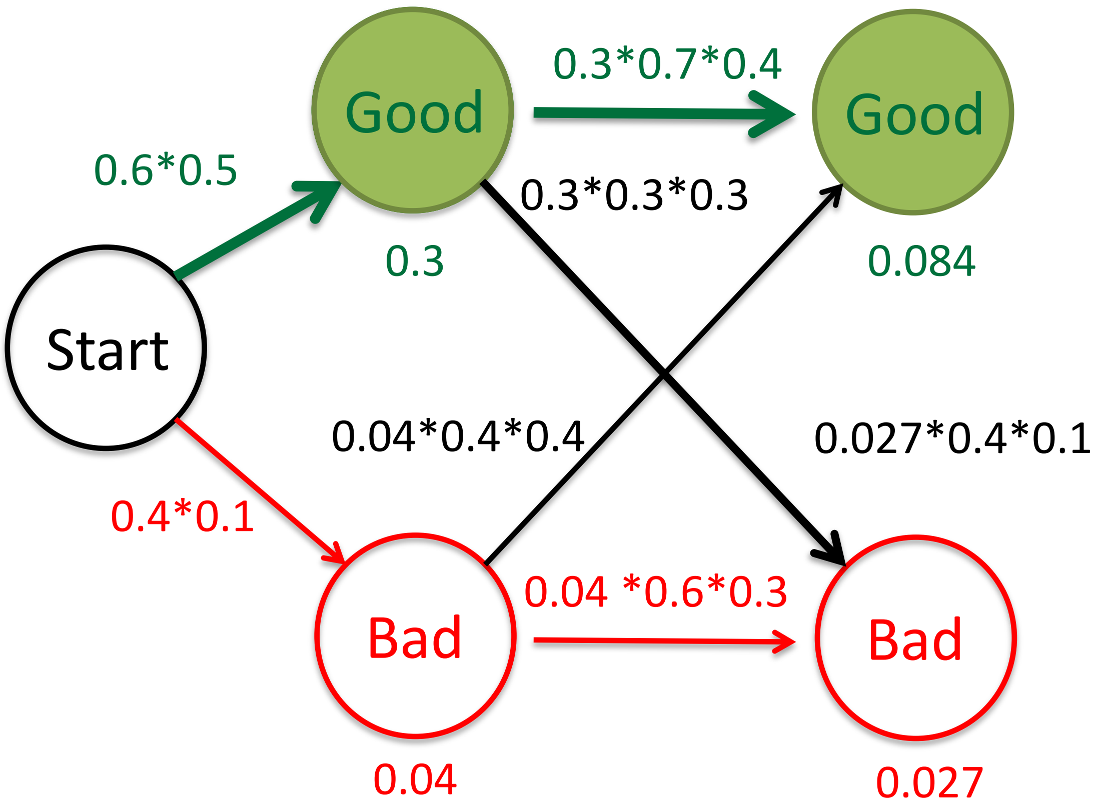
NOTE: score gets smaller with each observation!

We can estimate the most likely hidden state based on observations

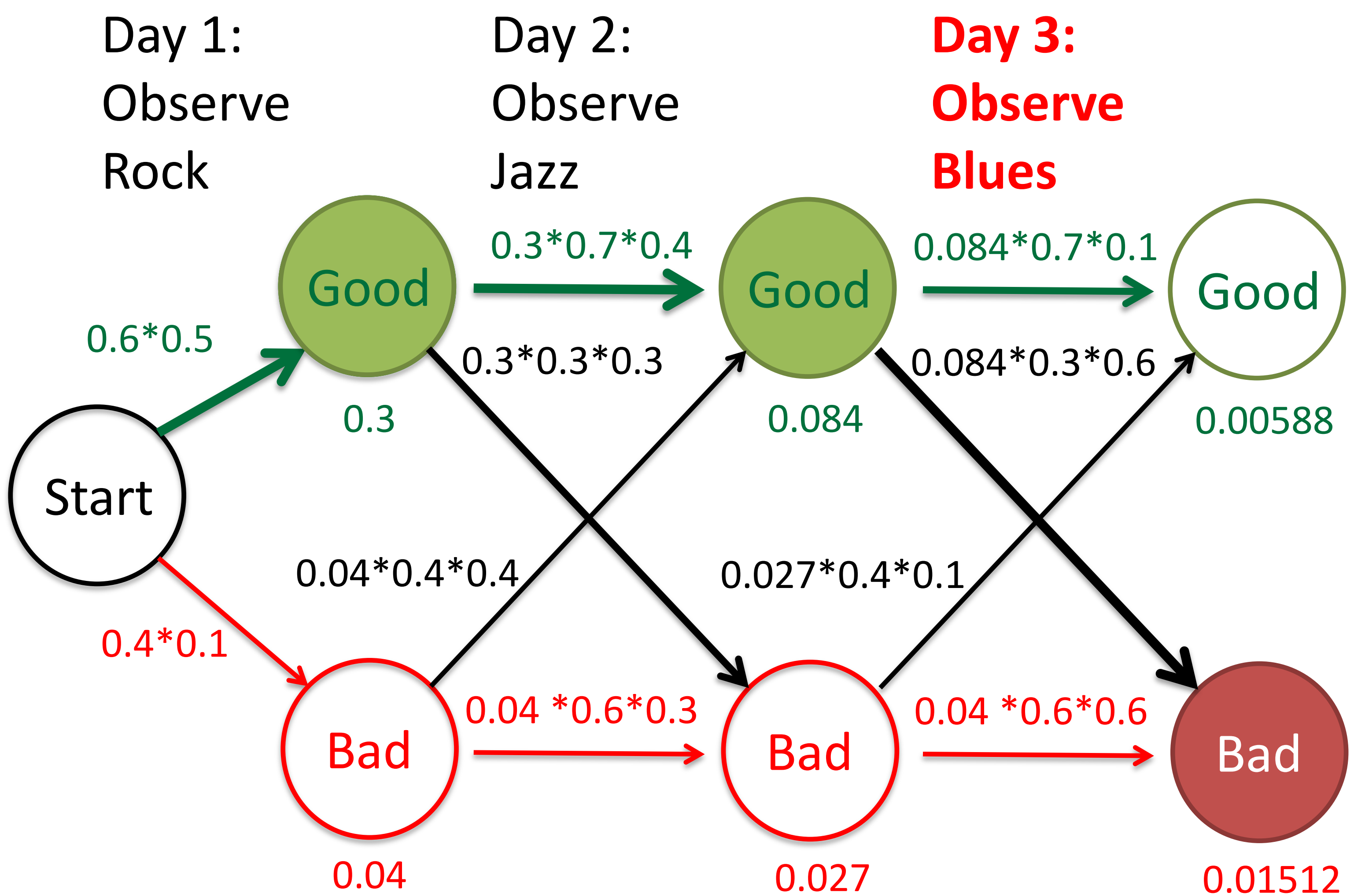
Day 1: Observe Rock

Day 2: Observe Jazz

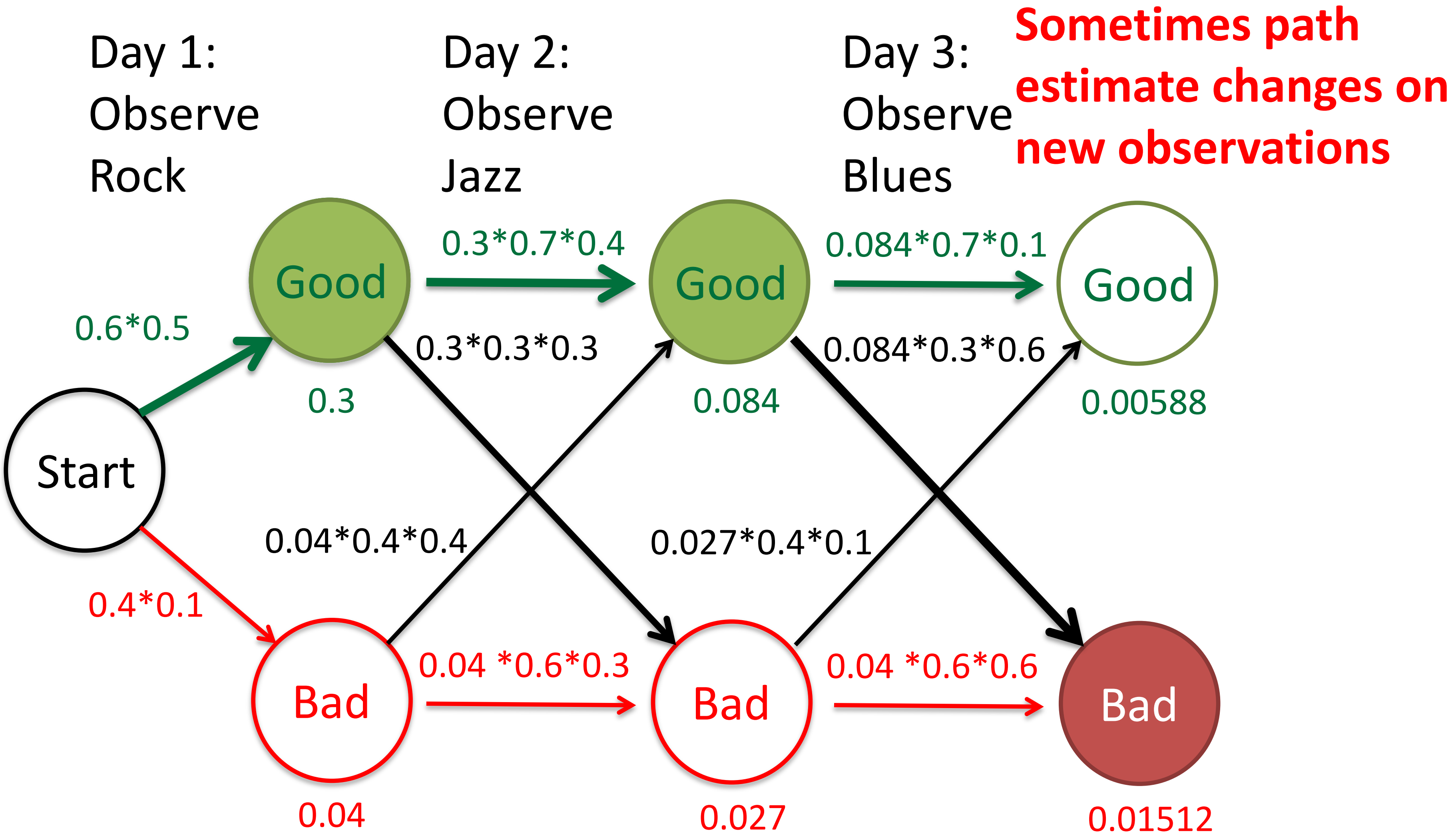
Day 3: Observe Blues



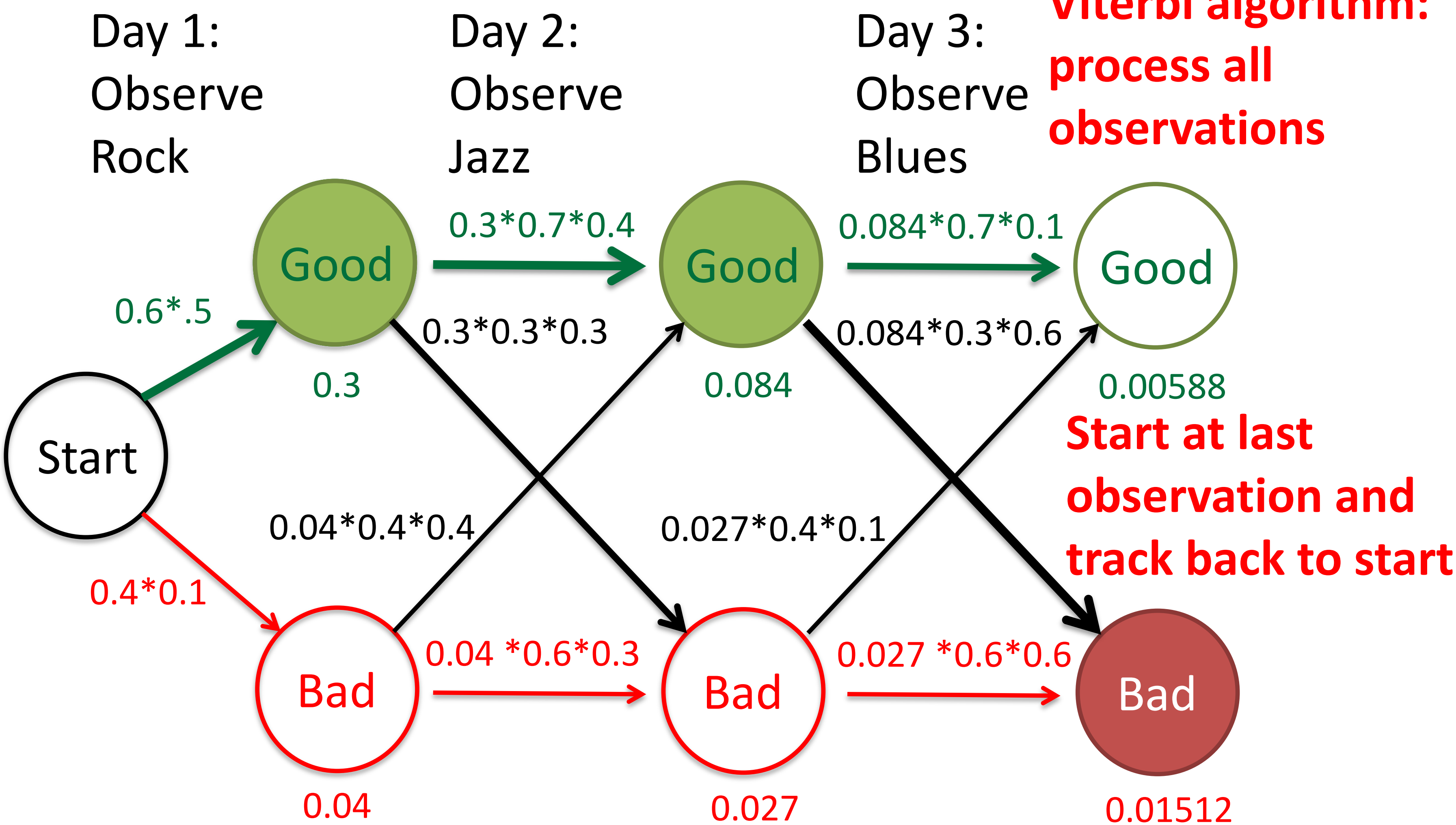
We can estimate the most likely hidden state based on observations



We can estimate the most likely hidden state based on observations



Viterbi algorithm back tracks to find most likely state sequence given observations



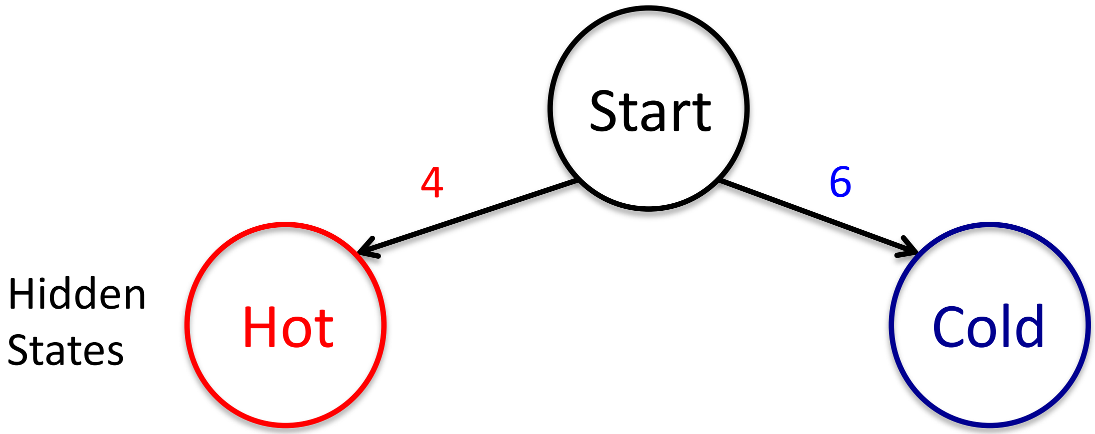
Given observations of {Rock, Jazz, Blues}
The boss's mood mostly likely was {Good, Good, Bad}

Temperature/cones example

ANNOTATED SLIDES

Begin at Start, add vertex for each hidden State with counts from training data

Count observations: 4 Hot days, 6 Cold days

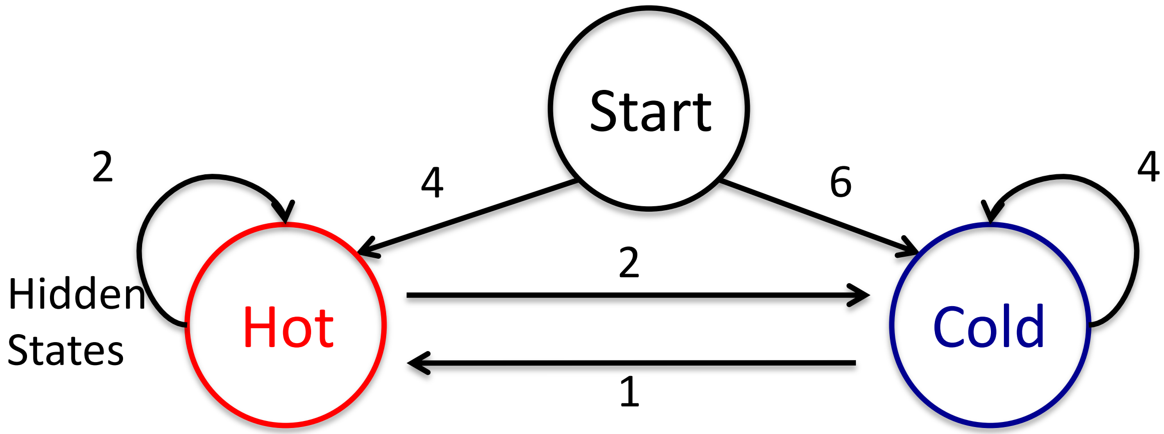


There were a total of 10 observations:

- **4 Hot days**
- **6 Cold days**

Add transitions between hidden States using count of next day’s hidden State

Count observations: transitions between hidden states (e.g., Hot->Hot)



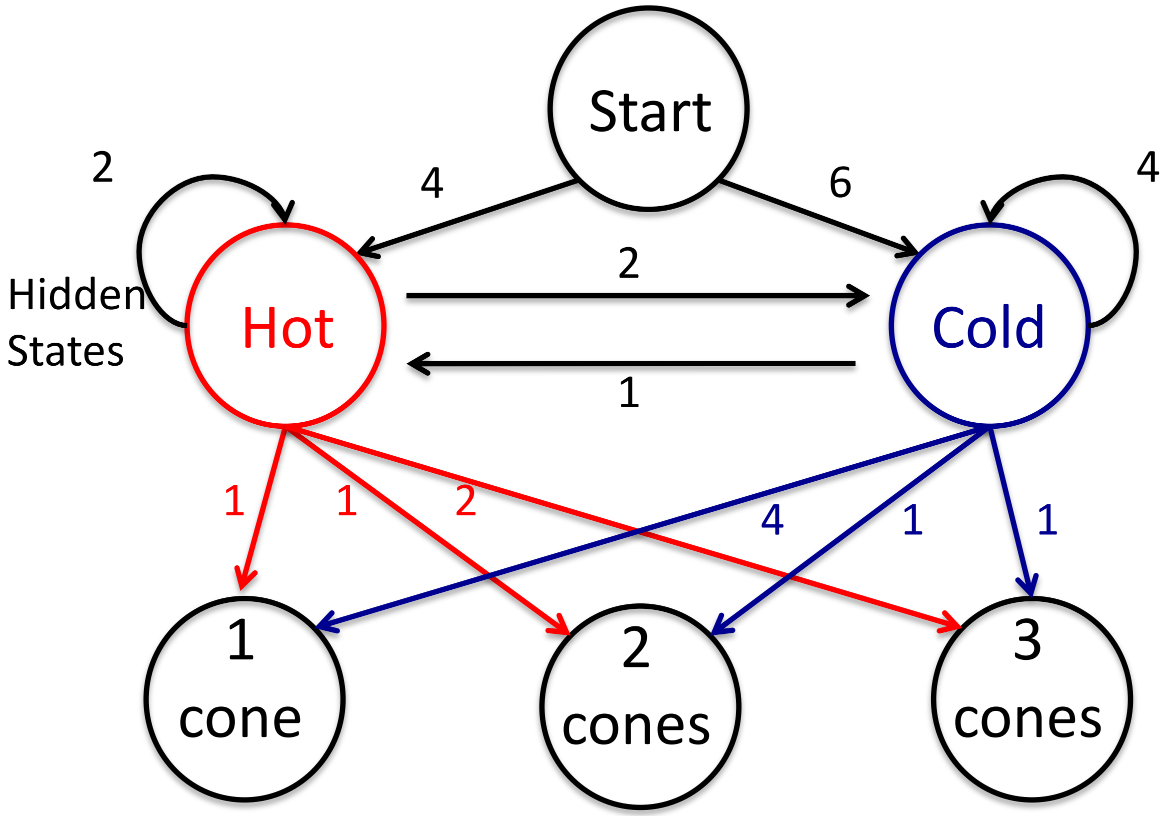
When it was Hot:
- **How many times was the next day also Hot (2)**
- **How many times was the next day Cold (2)**

When it was Cold:
- **How many times was the next day also Cold (4)**
- **How many times was the next day Hot (1)**

Note: one fewer Cold transitions because last day was Cold and no observation for the following day

For each hidden State, count the number of occurrences of each observation

Count observations: cones eaten when Cold



From each hidden State count how many times we see each observation

Hot:

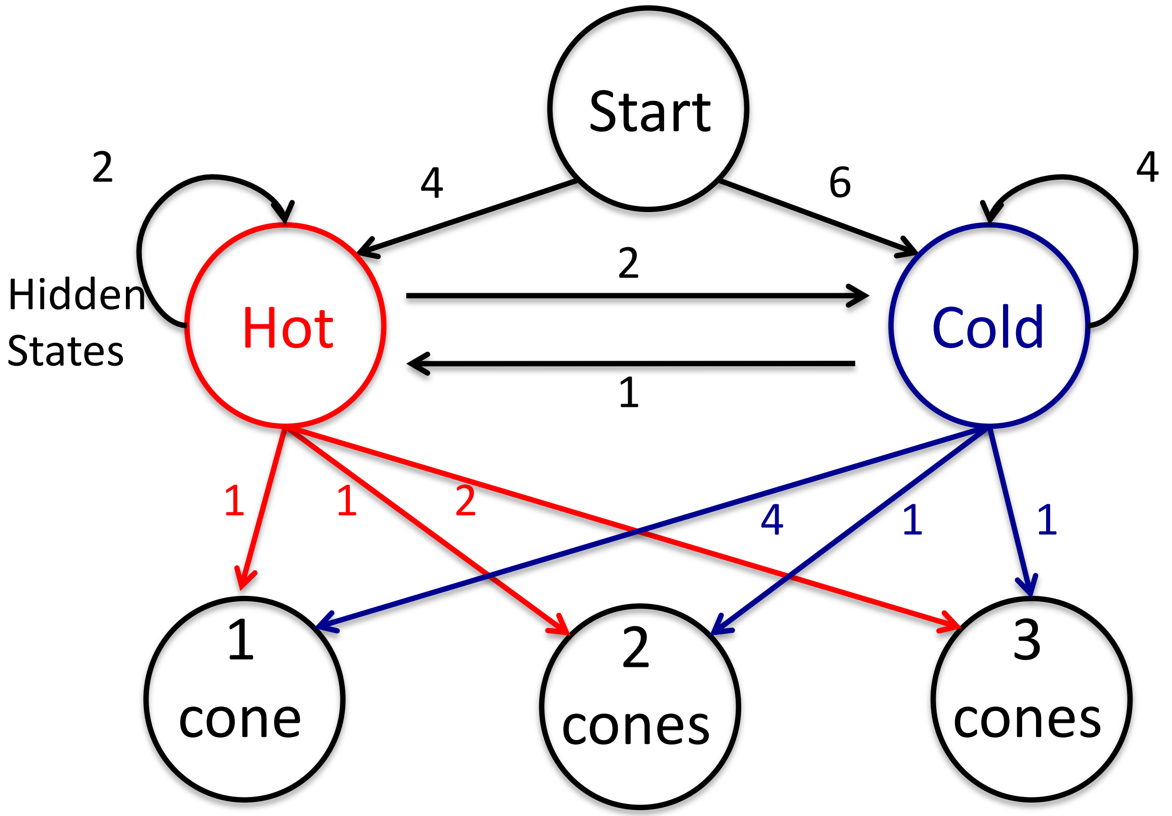
- **1 cone seen 1 time**
- **2 cones seen 1 time**
- **3 cones seen 2 times**

Cold

- **1 cones seen 4 times**
- **2 cones seen 1 time**
- **3 cones seen 1 time**

Convert observations counts into probabilities by dividing by total count

Convert to probabilities



Probability = count/total count

Example from Hot days:
Total of 4 cones eaten when Hot

- **1 cone eaten 1 time**
- **2 cones eaten 1 time**
- **3 cones eaten 2 times**
- **Total 4 cones eaten**

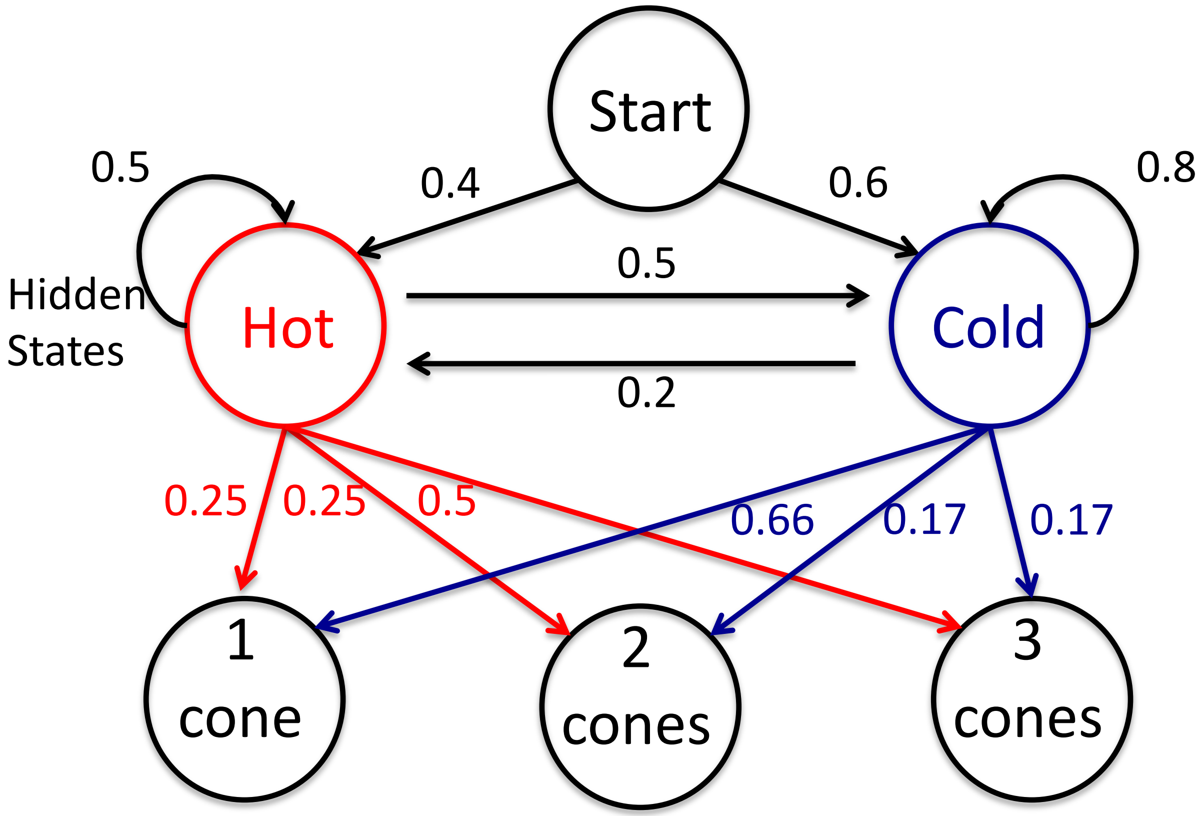
Probability:

- **1 cone = 1/4 = 0.25**
- **2 cones = 1/4 = 0.25**
- **3 cones = 2/4 = 0.5**

Convert all transitions to probabilities

Convert observations into probabilities by dividing count by total count

Probabilities based on observations



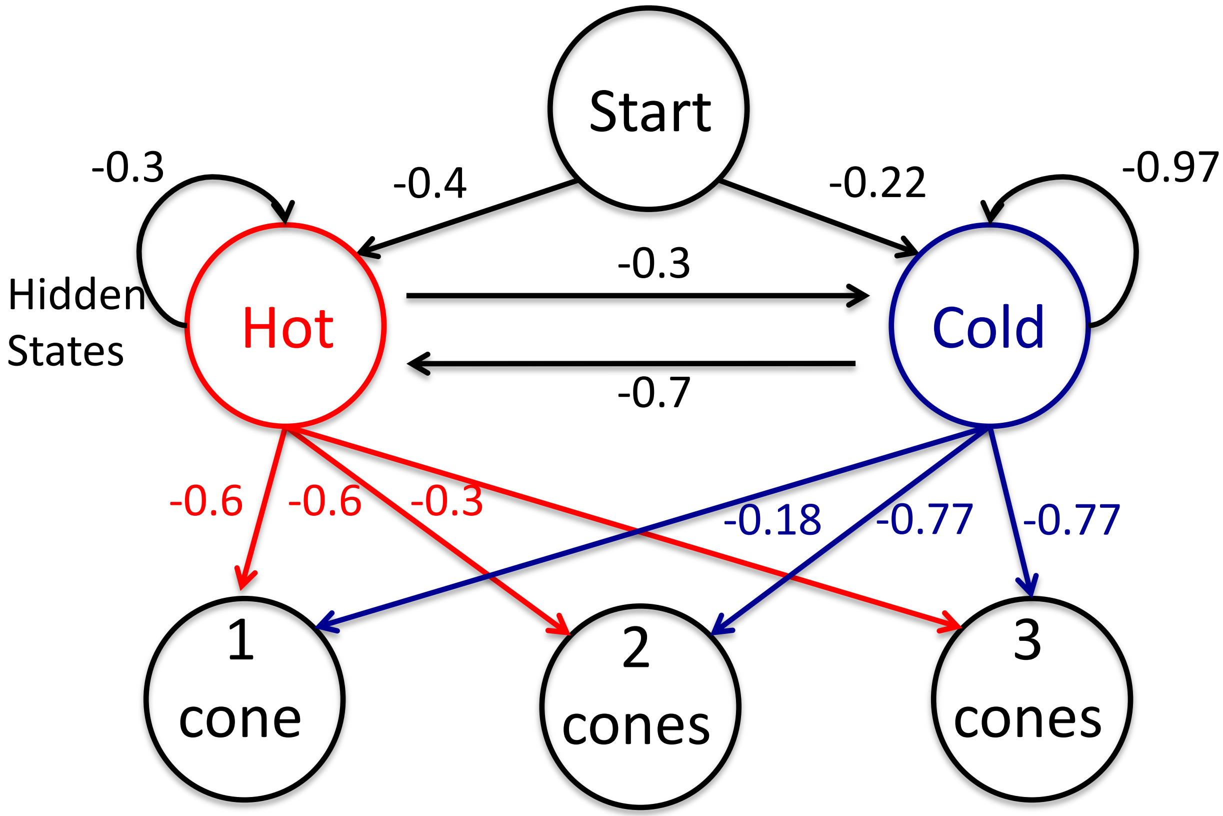
All counts now converted into probabilities

We would like to use the probabilities in the update rule covered previously: (current*transition*observation)

Problem: repeatedly multiplying numbers less than 1 quickly leads to numerical precision problems

Use logarithms to help with numerical precision problem

Log probabilities based on observations



A fact about logarithms can help us avoid precision issues:

log(mn) = log(m) + log(n)

To calculate score, add logs of each factor instead of multiplying probabilities

Take log (base 10 here, natural log in PS-5) of each probability

Negative numbers are ok, we will soon choose largest score (least negative)

Begin at Start State with 0 current score

#	Observation	nextState	currrentState	currScore + transScore + observation	nextScore
Start	n/a	Start	n/a	0	0

Observations {Two cones, three cones, two cones}

First observation is two cones eaten, calculate score for each possible next State

#	Observation	nextState	currrentState	currScore + transScore + observation	nextScore
Start	n/a	Start	n/a	0	0
0	Two cones	Cold	Start	0-0.22-0.77	-0.99
		Hot	Start	0-0.4-0.6	-1.0

Best guess is first day is Cold

Could transition to Cold or to Hot from Start, keep track of both possibilities

Calculate nextScore for each hidden State by adding logarithms

Store nextScore for each hidden State, largest score is most likely (Cold)

Observations {Two cones, three cones, two cones}
Most likely {Cold} (largest score)

Next observation is three cones eaten, calculate score for each possible next State

#	Observation	nextState	currrentState	currScore + transScore + observation	nextScore
Start	n/a	Start	n/a	0	0
0	Two cones	Cold	Start	0-0.22-0.77	-0.99
		Hot	Start	0-0.4-0.6	-1.0
1	Three cones	Cold	Cold	-0.99-0.97-0.77	~~-2.73~~
		Cold	Hot	-1-0.3-0.77	-2.07
		Hot	Cold	-0.99-0.7-0.3	~~-1.99~~
		Hot	Hot	-1-0.3-0.3	-1.6

Current State could be Cold or Hot, next State could be Cold or Hot, keep track of all possibilities

Calculate nextScore for each hidden State by adding logarithms

Keep largest score for each nextState
Largest most likely (Hot)
Prior was also Hot
Estimate of prior day changed from Cold to Hot

Observations {Two cones, three cones, two cones}
Most likely {Hot Hot }

Next observation is two cones eaten, calculate score for each possible next State

#	Observation	nextState	currrentState	currScore + transScore + observation	nextScore
Start	n/a	Start	n/a	0	0
0	Two cones	Cold	Start	0-0.22-0.77	-0.99
		Hot	Start	0-0.4-0.6	-1.0
1	Three cones	Cold	Cold	-0.99-0.97-0.77	~~-2.73~~
		Cold	Hot	-1-0.3-0.77	-2.07
		Hot	Cold	-0.99-0.7-0.3	~~-1.99~~
		Hot	Hot	-1-0.3-0.3	-1.6
2	Two cones	Cold	Cold	-2.07-0.97-0.77	~~-3.81~~
		Cold	Hot	-1.6-0.3-0.77	-2.67
		Hot	Cold	-2.07-0.7-0.6	~~-3.37~~
		Hot	Hot	-1.6-0.3-0.6	-2.5

Current State could be Cold or Hot, next State could be Cold or Hot, keep track of all possibilities

Largest most likely (Hot) Prior was also Hot then Prior prior also Hot

Observations {Two cones, three cones, two cones}

Most likely {Hot Hot Hot }

Because estimates can change, start at end and work backward to find most likely path

#	Observation	nextState	currrentState	currScore + transScore + observation	nextScore
Start	n/a	Start	n/a	0	0
0	Two cones	Cold	Start	0-0.22-0.77	-0.99
		Hot	Start	0-0.4-0.6	-1.0
1	Three cones	Cold	Cold	-0.99-0.97-0.77	~~-2.73~~
		Cold	Hot	-1-0.3-0.77	-2.07
		Hot	Cold	-0.99-0.7-0.3	~~-1.99~~
		Hot	Hot	-1-0.3-0.3	-1.6
2	Two cones	Cold	Cold	-2.07-0.97-0.77	~~-3.81~~
		Cold	Hot	-1.6-0.3-0.77	-2.67
		Hot	Cold	-2.07-0.7-0.6	~~-3.37~~
		Hot	Hot	-1.6-0.3-0.6	-2.5

Previous came from Hot

Back track to largest where nextState is Hot

Observations {Two cones, three cones, two cones}
Most likely {Hot Hot Hot }

Most likely nextState at end was Hot