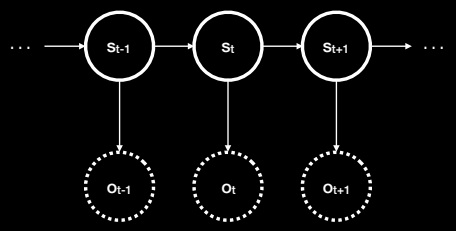
Viterbi Decoders

Cam Allen

1

Very useful application of Viterbi Algorithm for Hidden Markov Models.

Hidden Markov Models



2

Sequence of states at discrete time steps. The hidden part means we can't see the states directly. Only can see observations. We typically assume we know how the system transitions from one state to next, plus what observations are likely given each state.

The Problem

- Have: Sequence of observations $\{o_1, o_2, \ldots, o_n\}$
- Want: Sequence of states $\{s_1, s_2, \ldots, s_n\}$

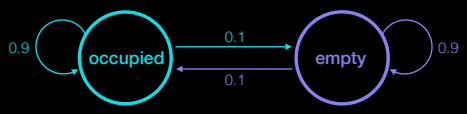
- Viterbi Algorithm:
  Find most-likely explanation (sequence of states) for observations about a Hidden Markov Model

3

Example: Office Lighting

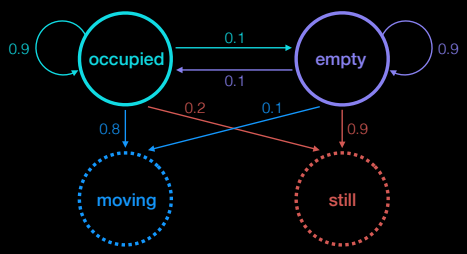
4

Example: Office Lighting



5

Example: Office Lighting



6

Example: Office Lighting

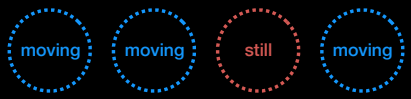
7

Observe:

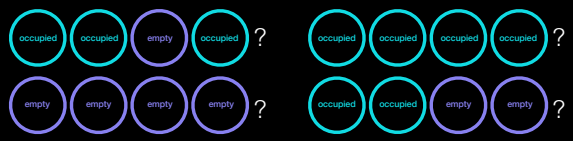
Example: Office Lighting

8

Observe:

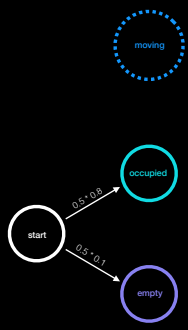


Explanation:

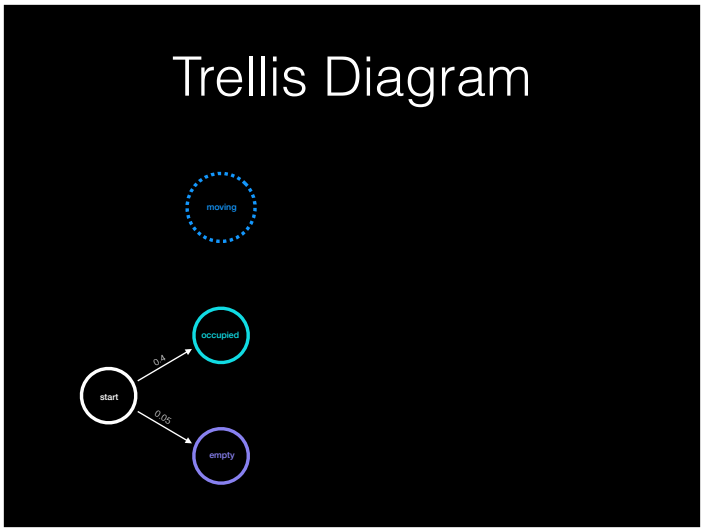


Trellis Diagram

9

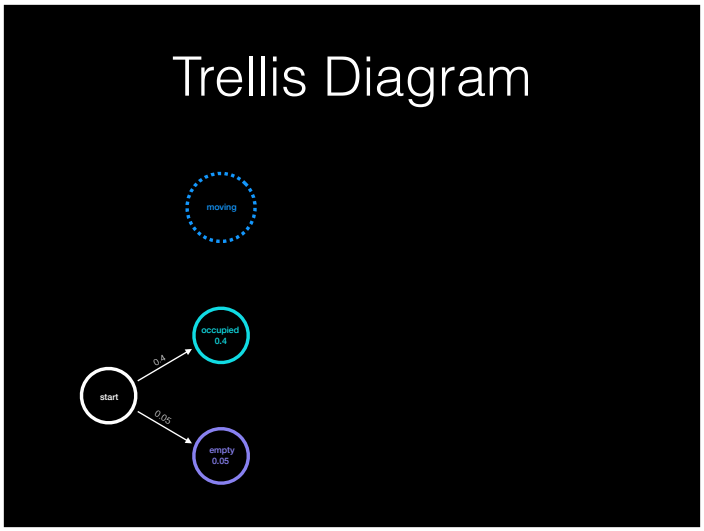


In order to visualize how the Viterbi Algorithm works, we'll make use of something called a trellis diagram. Observations on the top. States and transitions are below that. Here we're multiplying the probability of each state (50/50) times P(our_observation | state).



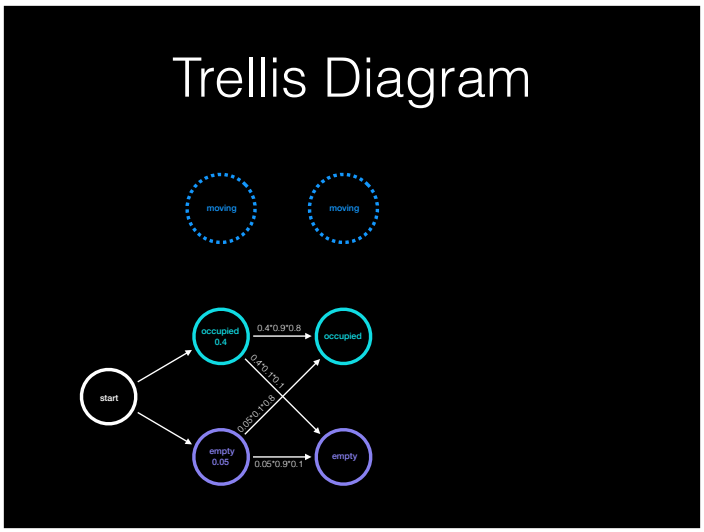
10

More specifically, we're using Bayes' Rule. Usually when you do this, you normalize by the probability of the observation, but here we only care about the maximum, so I'm skipping that part.



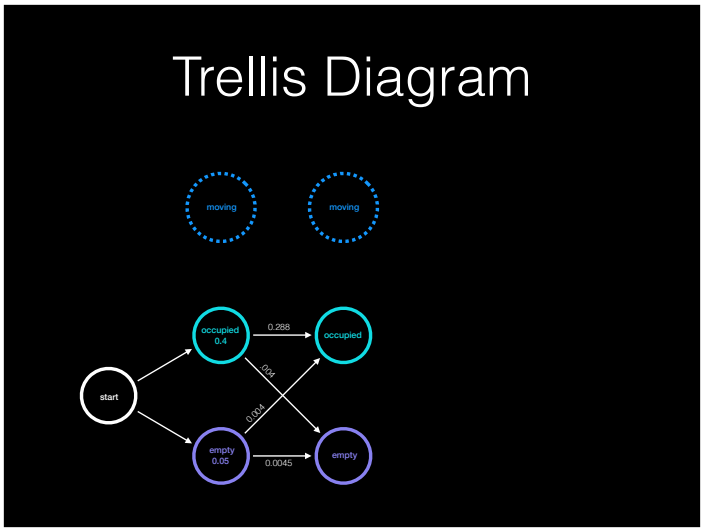
11

We fill in the states with their probabilities, and then we move on to the next time step.

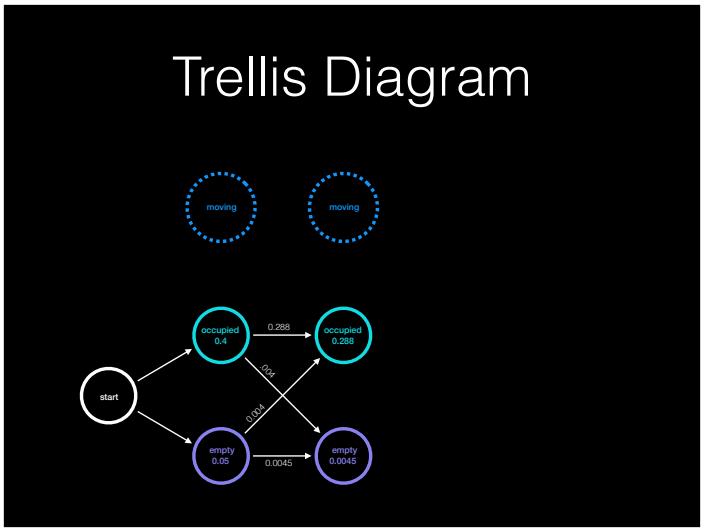


12

We have the same observation again, so, starting with the probabilities we just calculated, we multiply by the probability of each state transition, times the probability of the observation given the state we end up in.

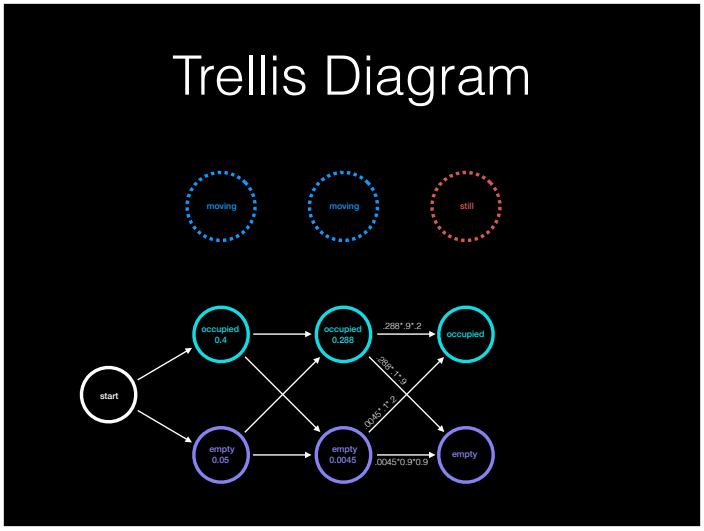


13



14

And then we update our state probabilities again.

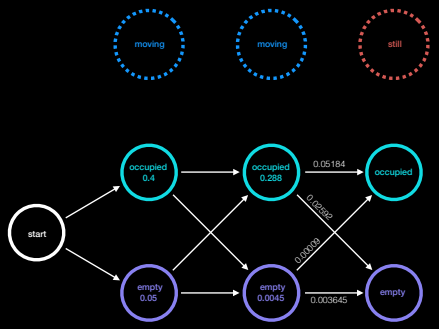


15

In the next time step we get a different observation, so the calculations change slightly

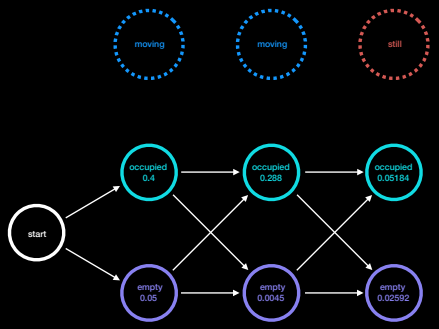
16

Trellis Diagram



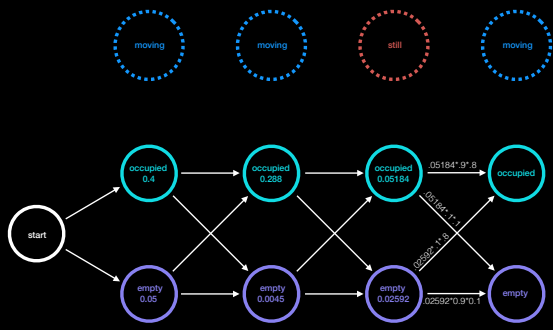
17

Trellis Diagram

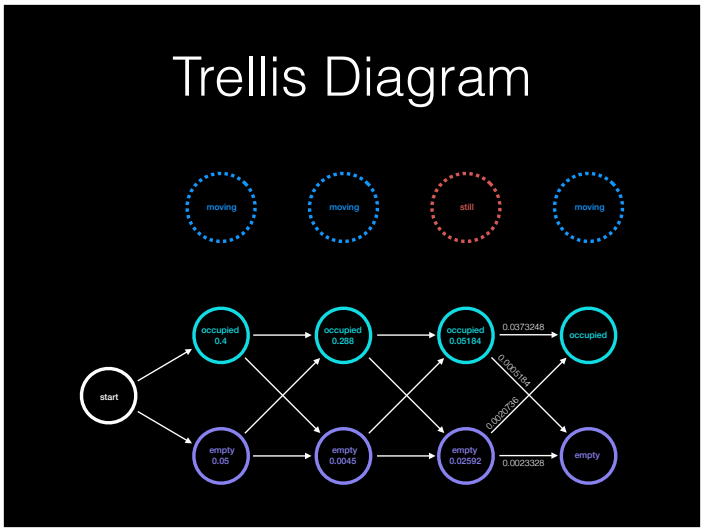


18

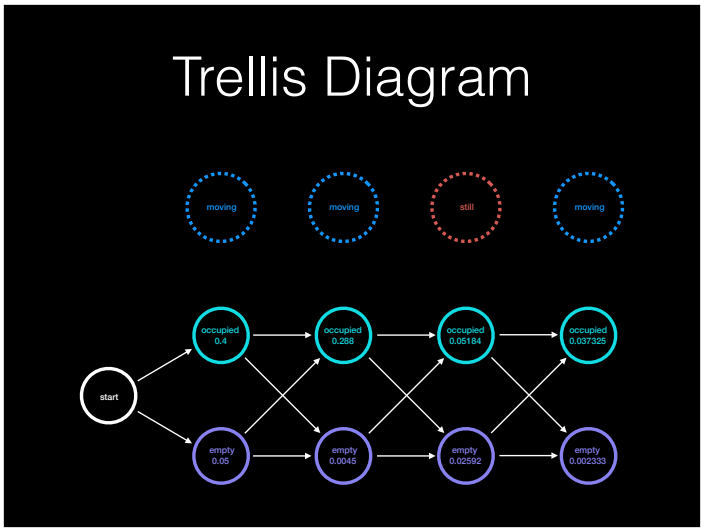
Trellis Diagram



19

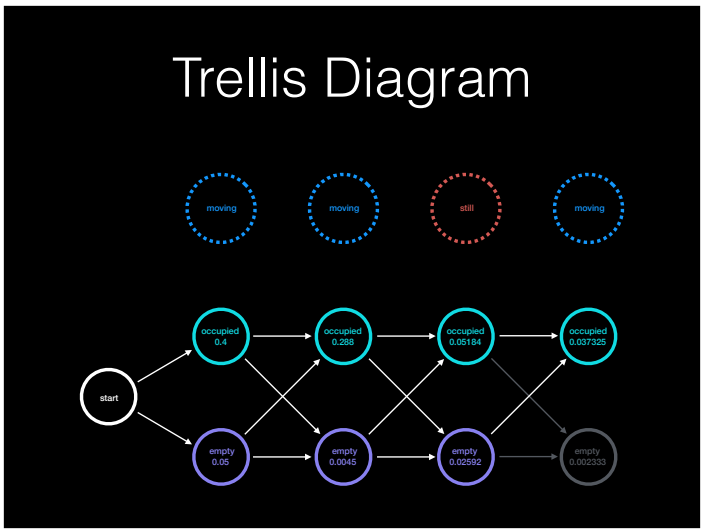


20

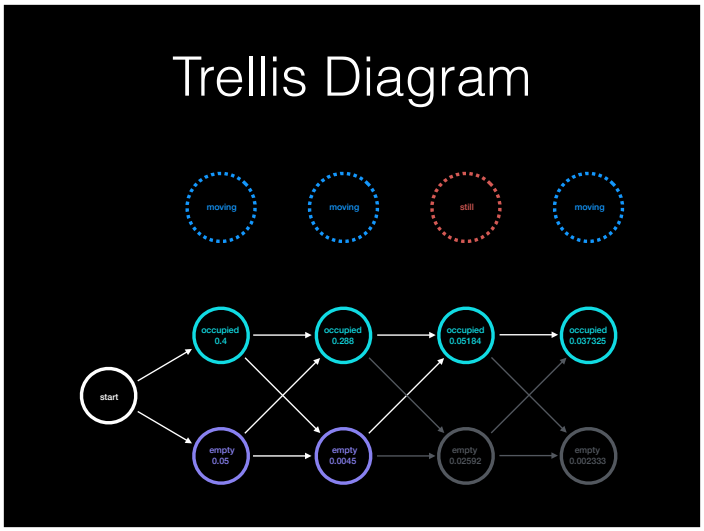


And finally we finish our trellis diagram. Now we look at the last two states and see which has the higher probability. That's where we end up.

21



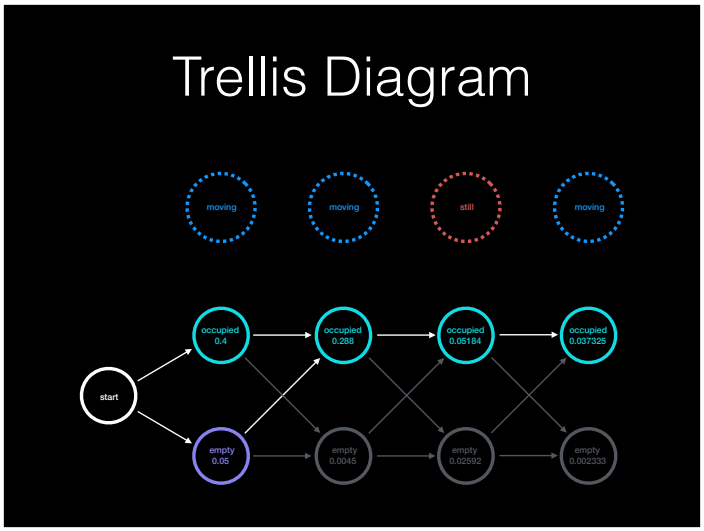
We cross out the other state, and move backwards.



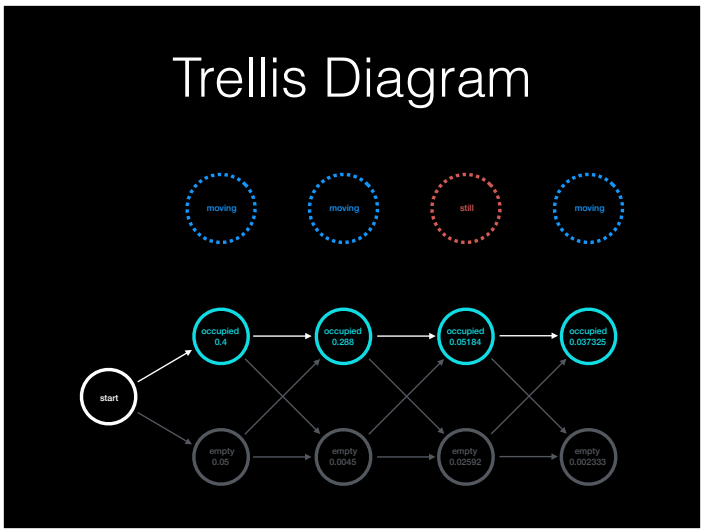
22

At each time step, we pick the state with the highest probability

---



23



24

Until finally we have our entire path. This is called the Viterbi path, and it's the most likely explanation for the observations.

Example: Office Lighting

Observe:

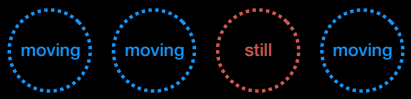
Explanation:

25

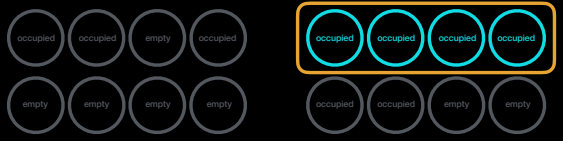
Now we can answer our question

Example: Office Lighting

Observe:



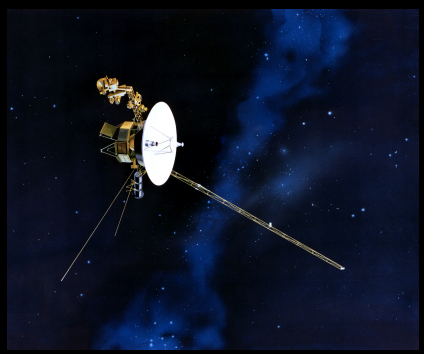
Explanation:



26

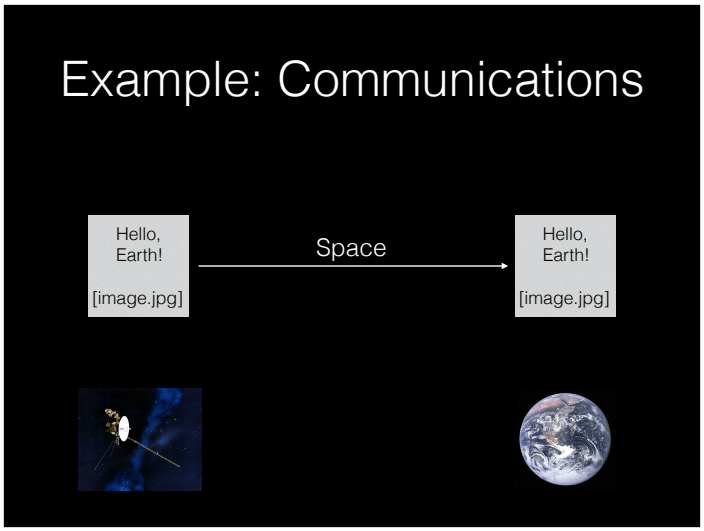
So basically, if the lab had a lighting controller like this, maybe the lights would stop randomly shutting off while I'm working causing me to wave my arms around frantically.

Example: Communications



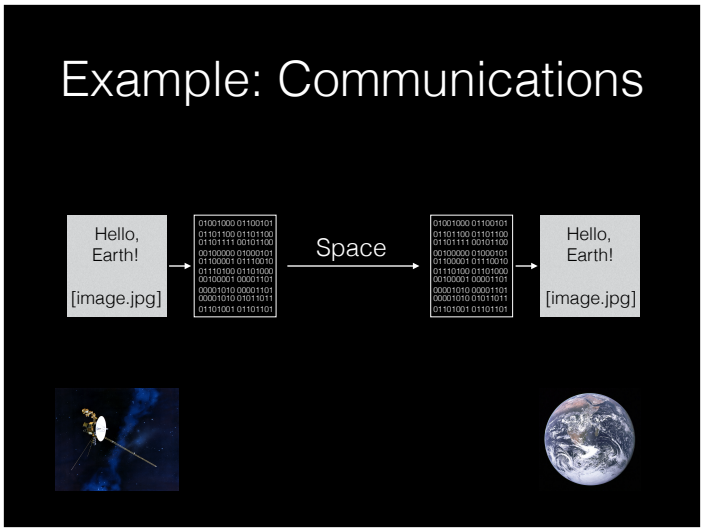
27

Another great use for the Viterbi Algorithm is in communications. This is a picture of Voyager 1 – one of several deep-space probes that used the algorithm for communications to and from Earth



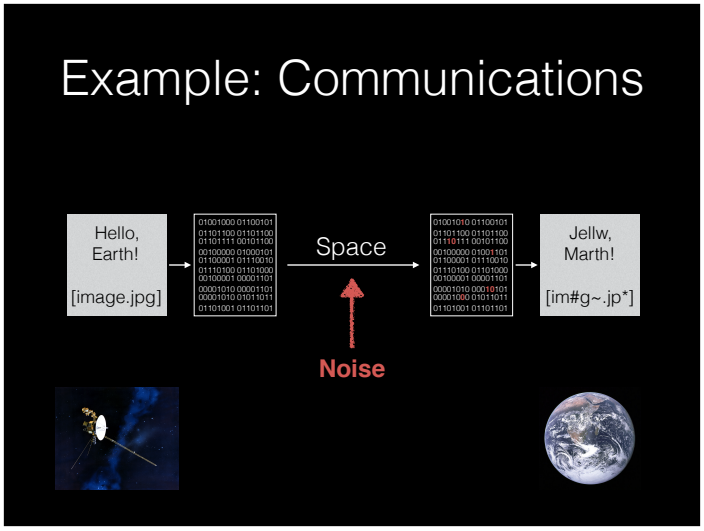
28

Typically communications works like this. You have some message, and you want to send through space back to Earth.



29

Normally we first convert the message to binary

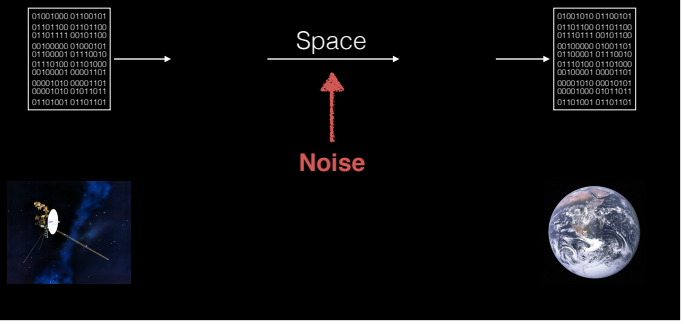


30

The problem is that usually the message gets corrupted by noise along the way

31

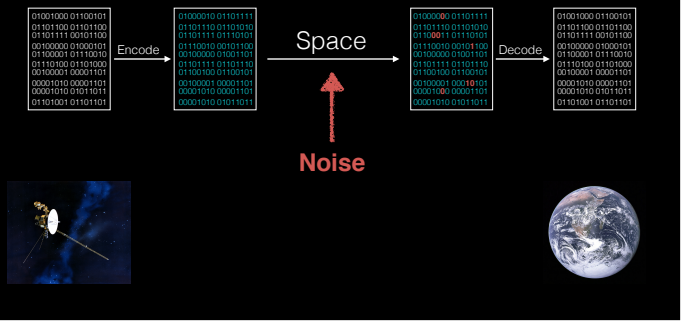
Example: Communications



32

Example: Communications

- Error-Correcting Codes

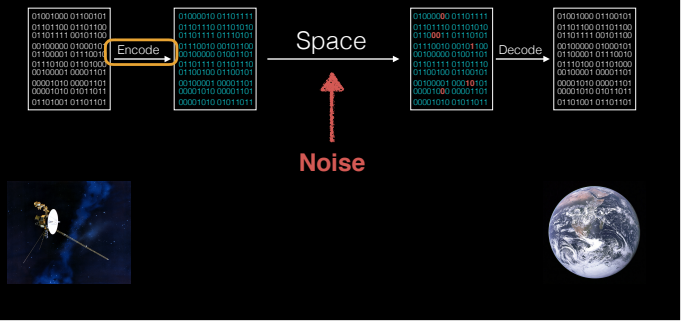


To solve this problem, we'll use something called an error correcting code. Basically, instead of transmitting the white bits, we send them through an encoder to get the blue bits. The blue bits contain extra information that helps recover the message.

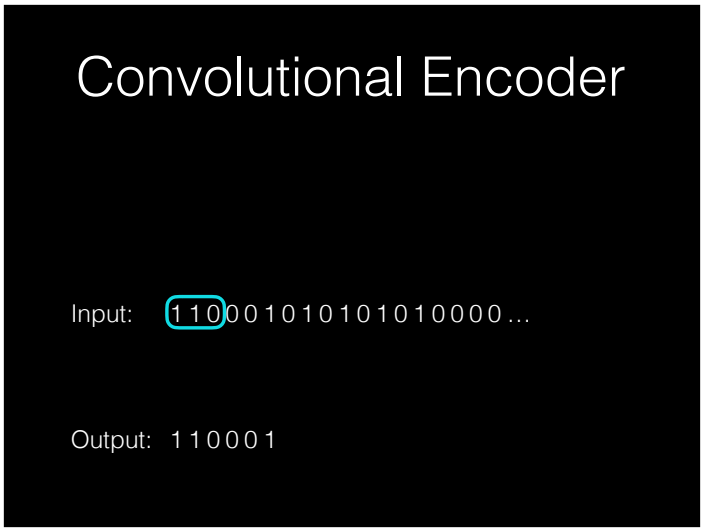
33

Example: Communications

- Error-Correcting Codes

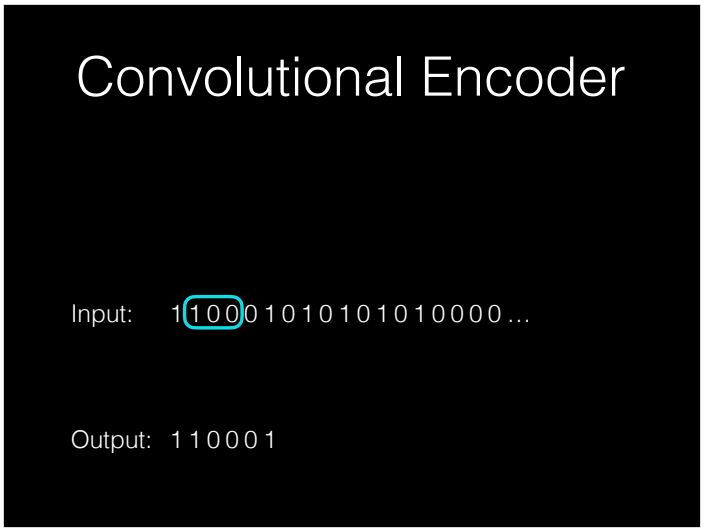


I'll get to the decoder in a minute, but first I'll explain what's going on in the encoder.

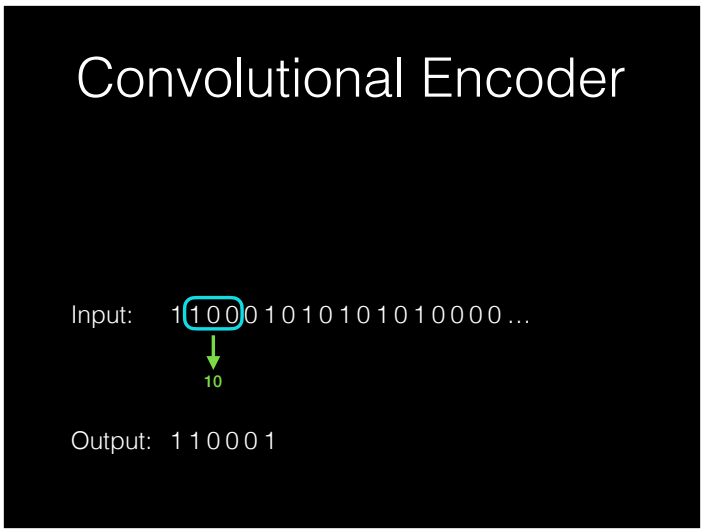


34

The encoder operates on groups of 3 bits at a time.

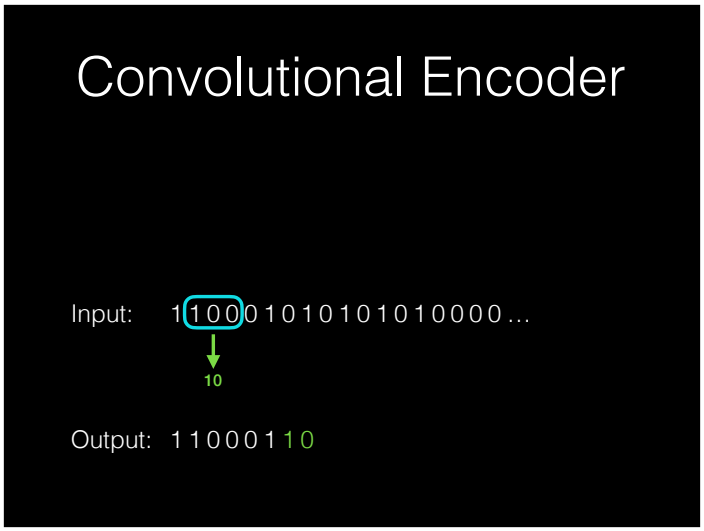


35



36

Convolutional Encoder

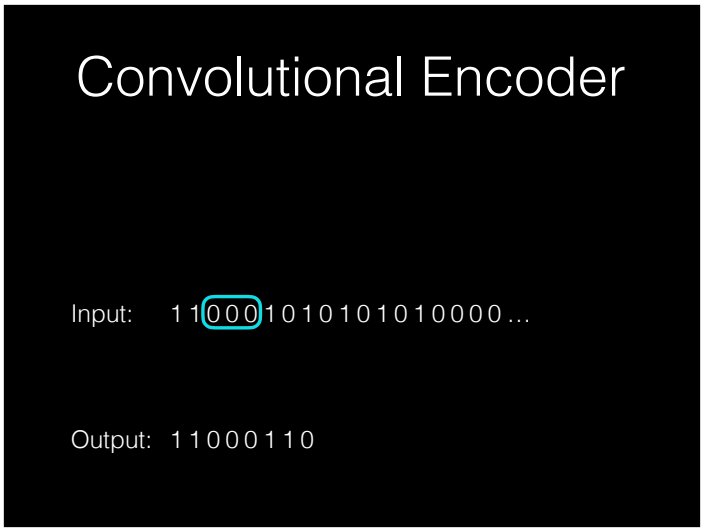


37

You slide this 3-bit window over by 1, and it produces 2 output bits

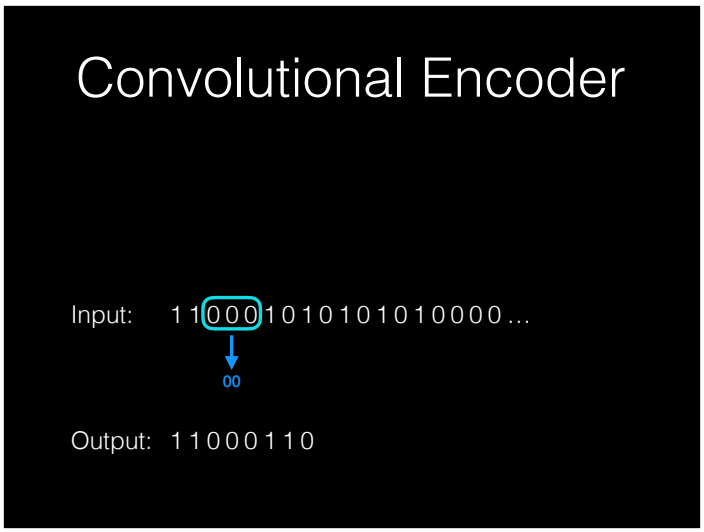
---

Convolutional Encoder

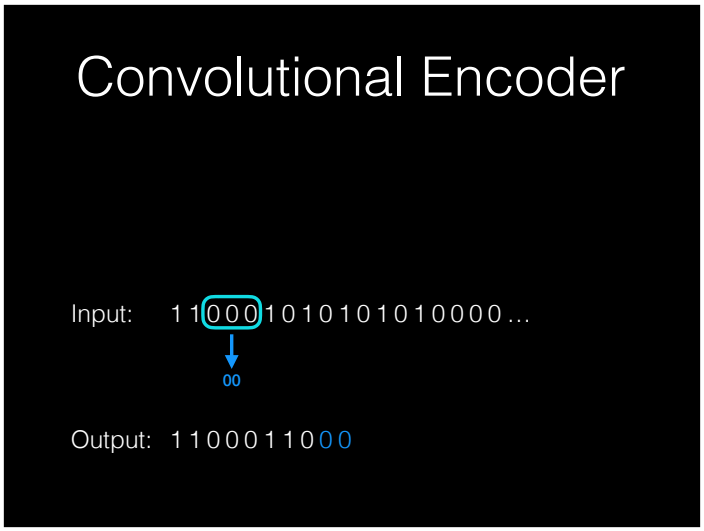


38

Convolutional Encoder

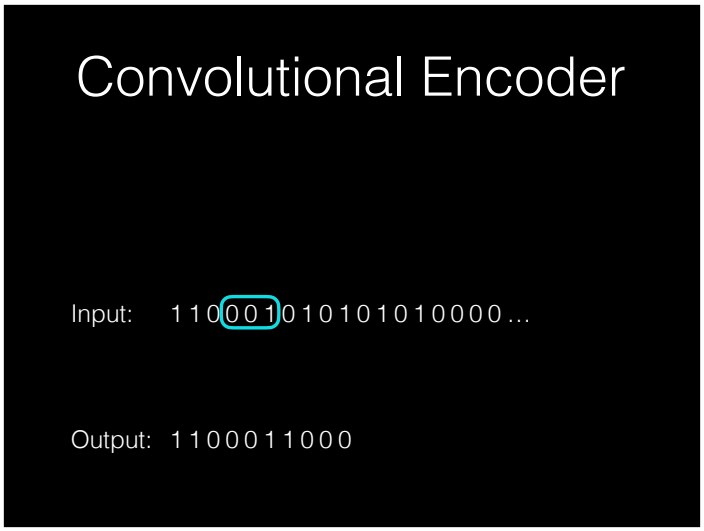


39

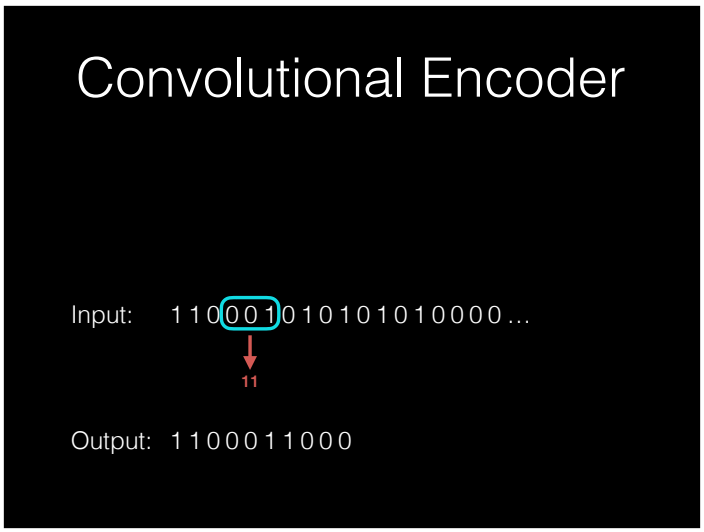


40

Slide it over by one more bit, and you get another 2 bits out. This is a rate 1/2 encoder, because it produces twice as many bits of output as it gets input.



41



42

43

Convolutional Encoder

Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …

11

Output: 1 1 0 0 0 1 1 0 0 0 1 1

44

Convolutional Encoder

Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …

Output: 1 1 0 0 0 1 1 0 0 0 1 1

45

Convolutional Encoder

Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …

11

Output: 1 1 0 0 0 1 1 0 0 0 1 1

Convolutional Encoder

Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …

11

Output: 1 1 0 0 0 1 1 0 0 0 1 1 1 1

46

Convolutional Encoder

Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …

Output: 1 1 0 0 0 1 1 0 0 0 1 1 1 1

47

Convolutional Encoder

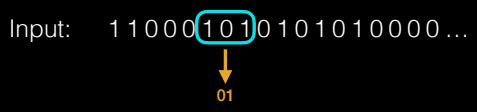
Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …

01

Output: 1 1 0 0 0 1 1 0 0 0 1 1 1 1

48

Convolutional Encoder



Output: 1 1 0 0 0 1 1 0 0 0 1 1 1 1 0 1

49

Convolutional Encoder

Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …

Output: 1 1 0 0 0 1 1 0 0 0 1 1 1 1 0 1

50

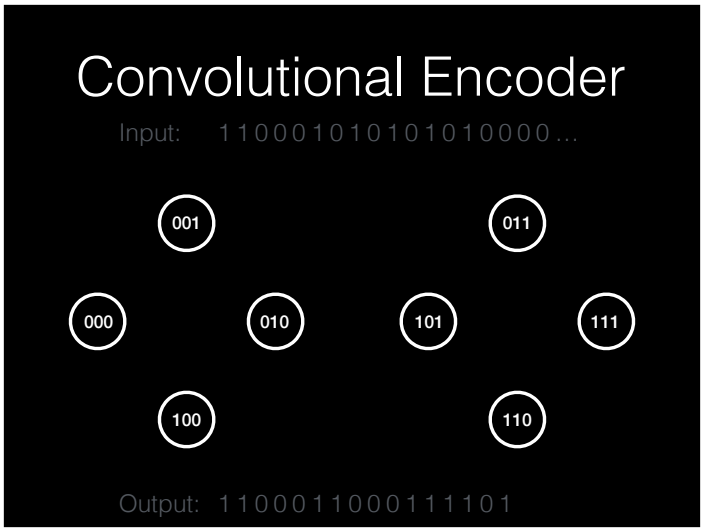
If we move this stuff out of the way, we can draw a state diagram for what's going on here

Convolutional Encoder

Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …

Output: 1 1 0 0 0 1 1 0 0 0 1 1 1 1 0 1

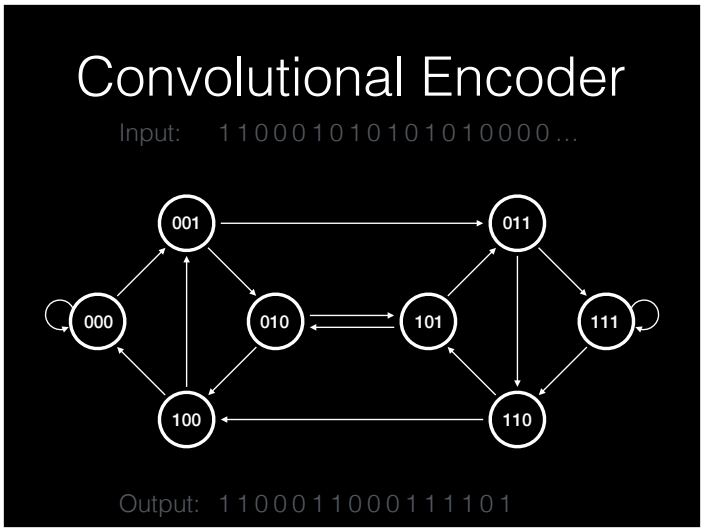
51



52

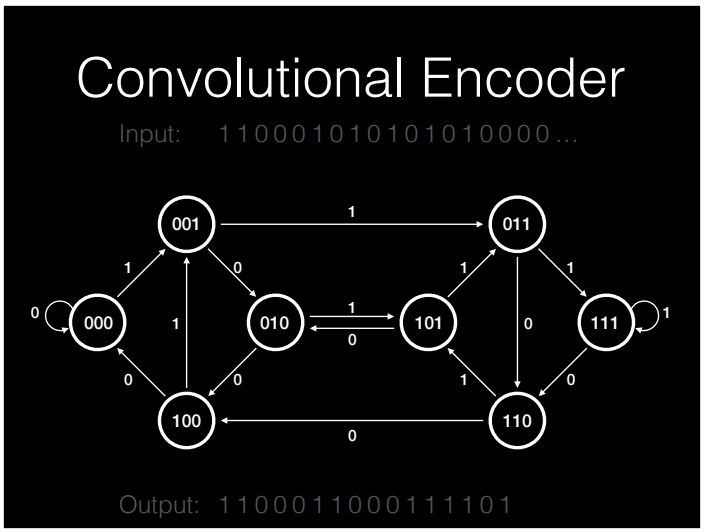
Each of these states corresponds to something our 3-bit window might see

---



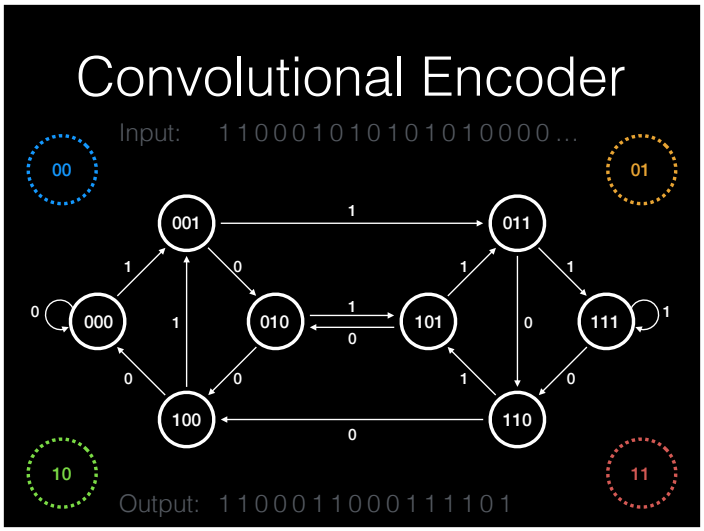
53

We can draw in the transitions with arrows

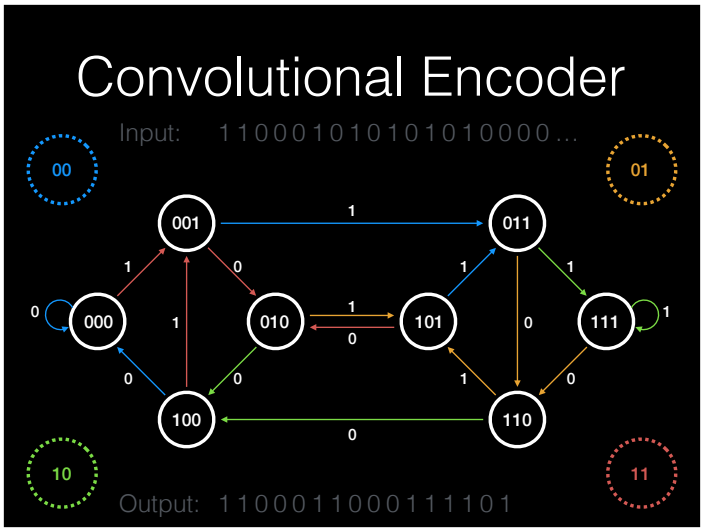


54

Each transition represents one of the two possible input bits.

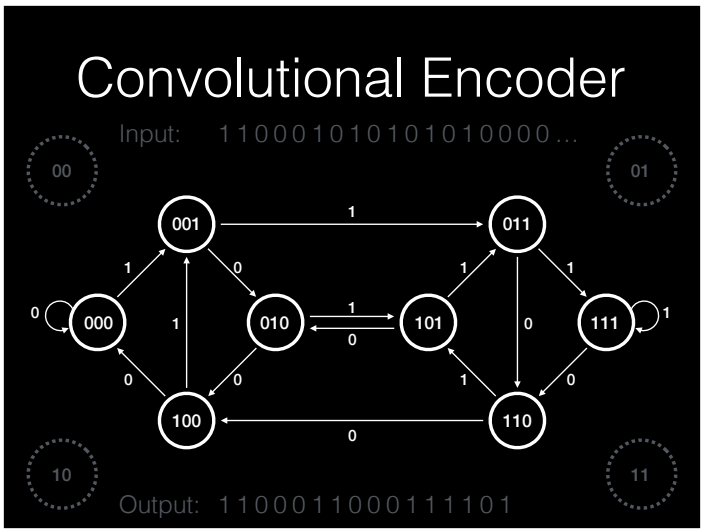


55

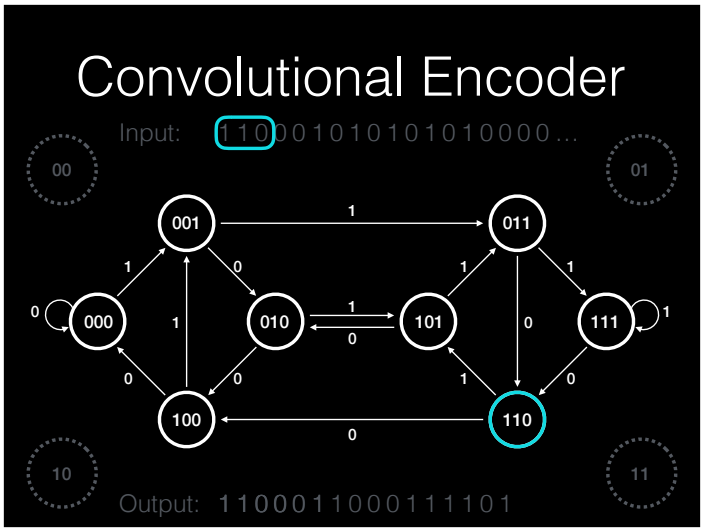


56

Each transition also produces two output bits



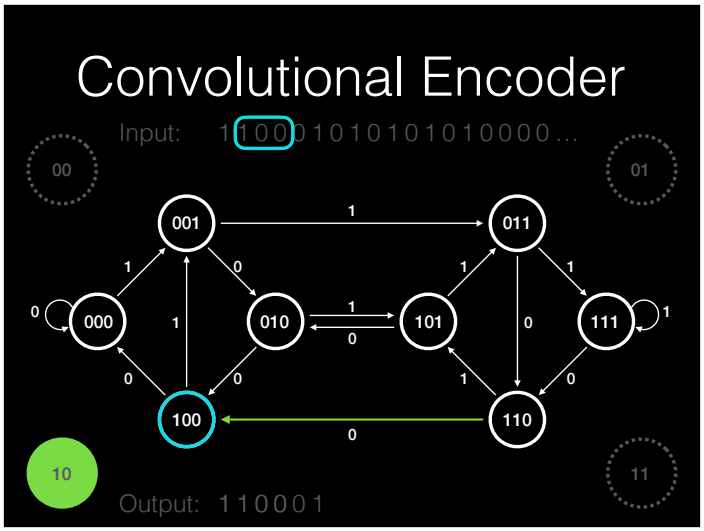
57



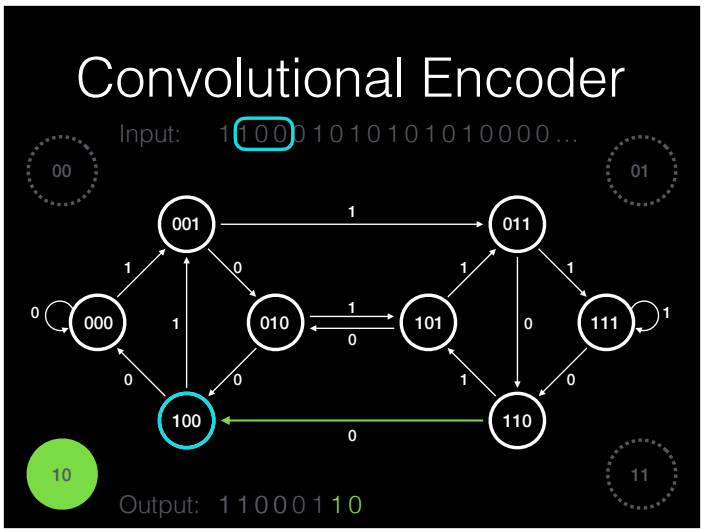
58

Now if we start in state 110, we can re-run our input and visualize the encoder state

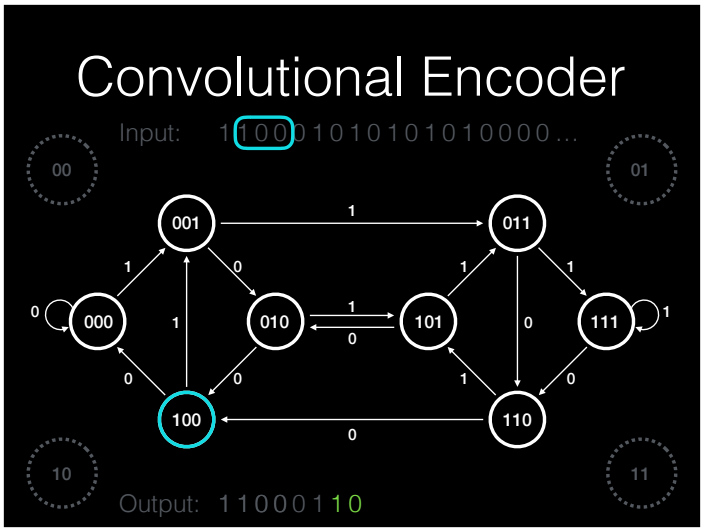
---



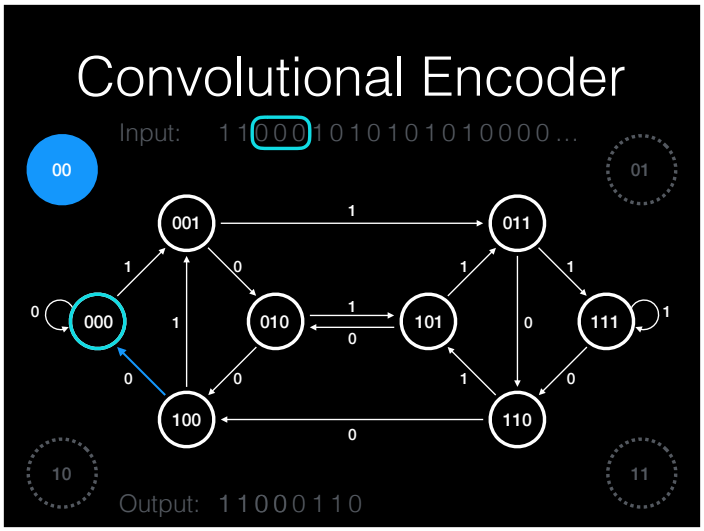
59



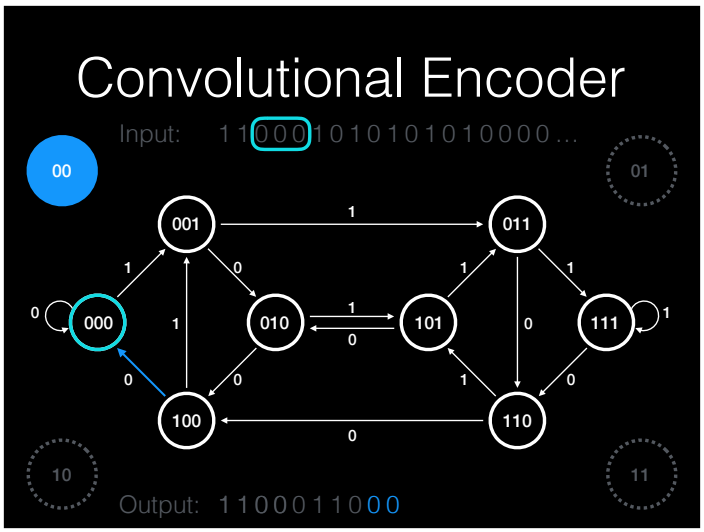
60


Convolutional Encoder
Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …
00
01
10
11
001
011
000
010
101
111
100
110
Output: 1 1 0 0 0 1 1 0

61

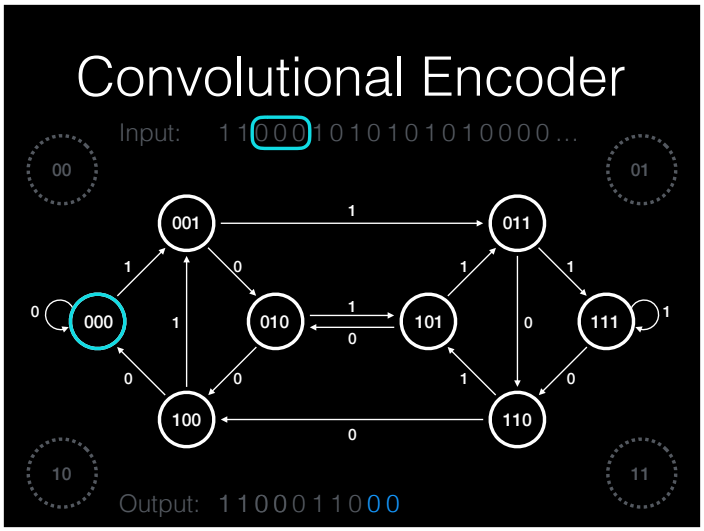

Convolutional Encoder
Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …
00
01
10
11
001
011
000
010
101
111
100
110
Output: 1 1 0 0 0 1 1 0

62

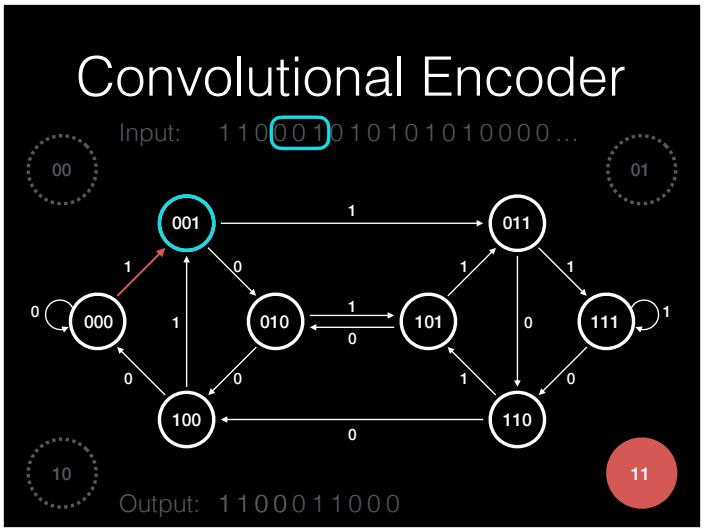

Convolutional Encoder
Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …
00
01
10
11
001
011
000
010
101
111
100
110
Output: 1 1 0 0 0 1 1 0 0 0

63

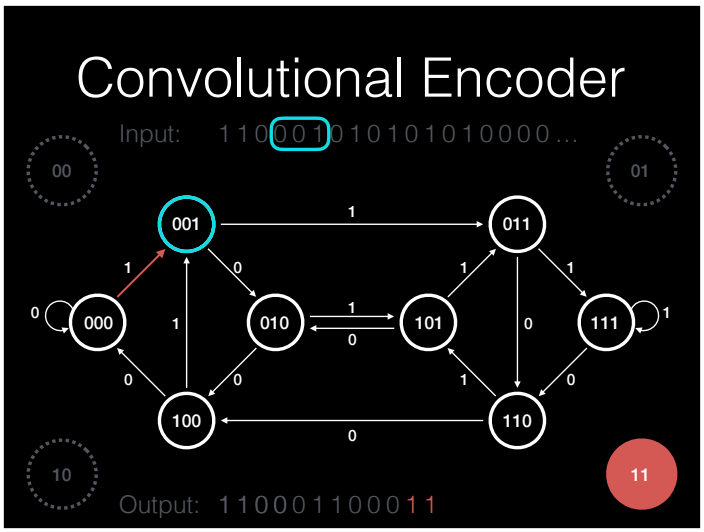
64

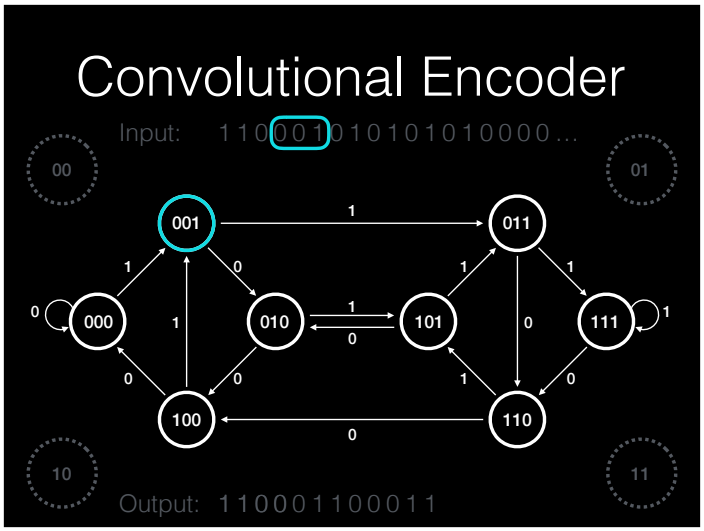

Convolutional Encoder
Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …
00
01
10
11
001
011
000
010
101
111
100
110
Output: 1 1 0 0 0 1 1 0 0 0

65

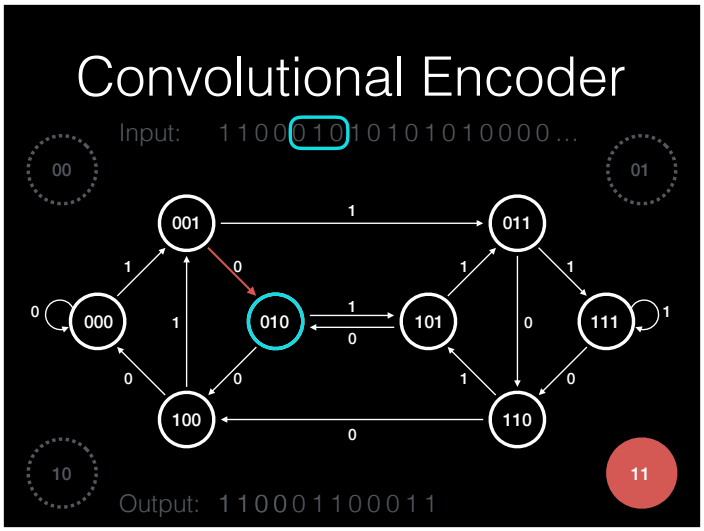

Convolutional Encoder
Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …
00
01
10
11
001
011
000
010
101
111
100
110
Output: 1 1 0 0 0 1 1 0 0 0

66

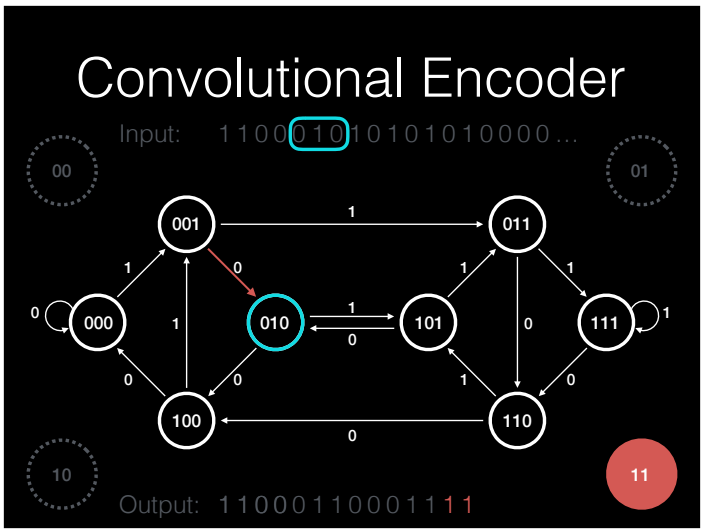

Convolutional Encoder
Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …
00
01
10
11
001
011
000
010
101
111
100
110
Output: 1 1 0 0 0 1 1 0 0 0 1 1


Convolutional Encoder
Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …
00
01
10
11
001
011
000
010
101
111
100
110
Output: 1 1 0 0 0 1 1 0 0 0 1 1

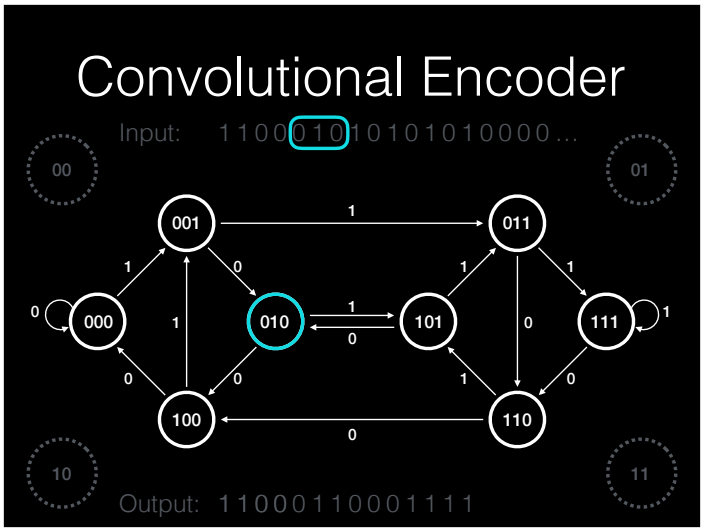
67


Convolutional Encoder
Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …
00
01
10
11
001
011
000
010
101
111
100
110
Output: 1 1 0 0 0 1 1 0 0 0 1 1

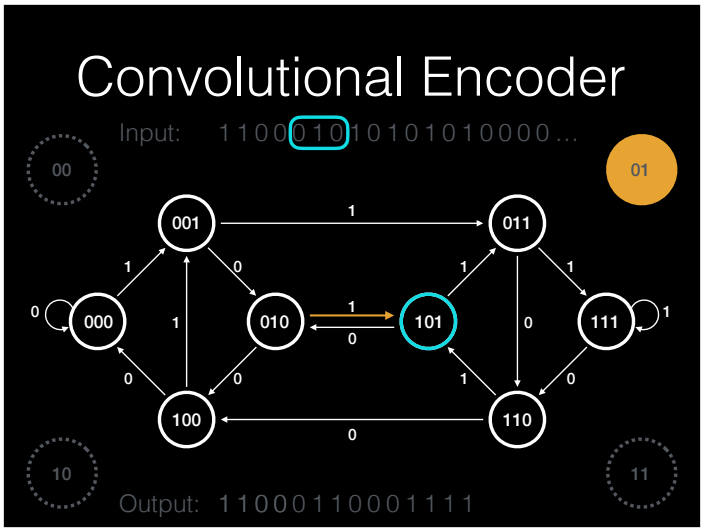
68


Convolutional Encoder
Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …
00
01
10
11
001
011
000
010
101
111
100
110
Output: 1 1 0 0 0 1 1 0 0 0 1 1 1 1

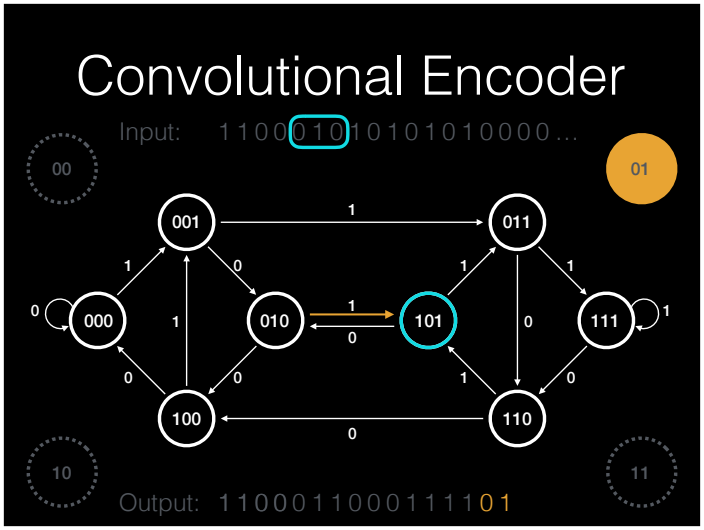
69


Convolutional Encoder
Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …
00
01
10
11
001
011
000
010
101
111
100
110
Output: 1 1 0 0 0 1 1 0 0 0 1 1 1 1

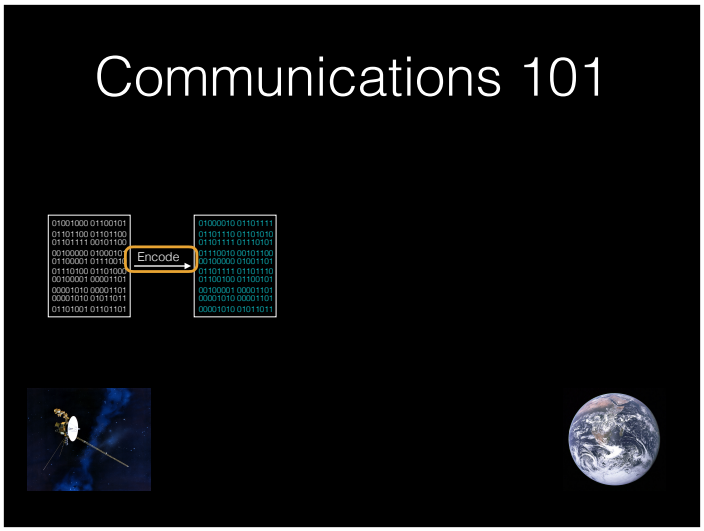
70


Convolutional Encoder
Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …
00
01
10
11
001
011
000
010
101
111
100
110
Output: 1 1 0 0 0 1 1 0 0 0 1 1 1 1

71

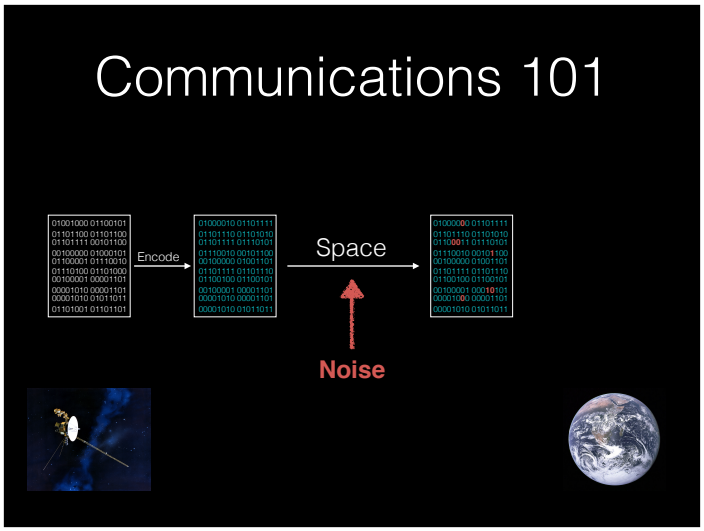

Convolutional Encoder
Input: 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 …
00
01
10
11
001
011
000
010
101
111
100
110
Output: 1 1 0 0 0 1 1 0 0 0 1 1 1 1 0 1

72



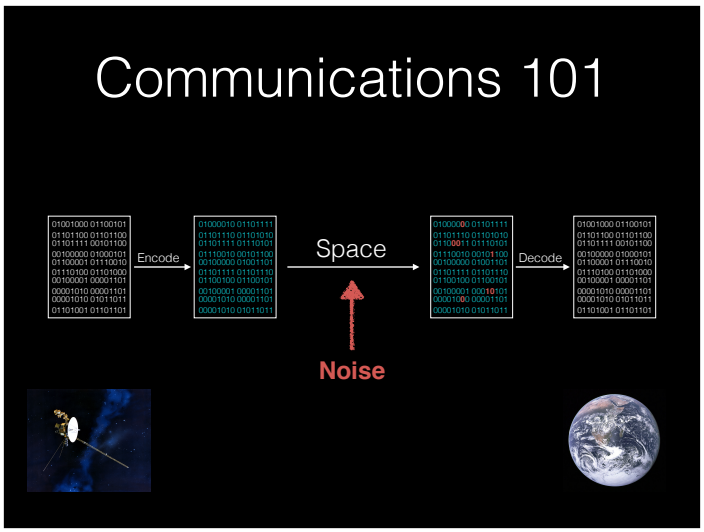
73

So that's the encoder



74

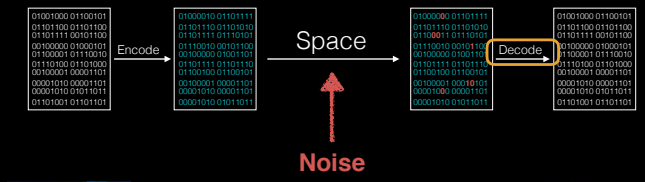
We still need to get our message through space

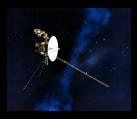


75

And then decode the noisy results to get our message back out.

Communications 101

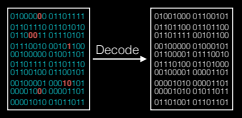




76

Now we'll look at the decoder.

Communications 101



77

The Problem

Have:

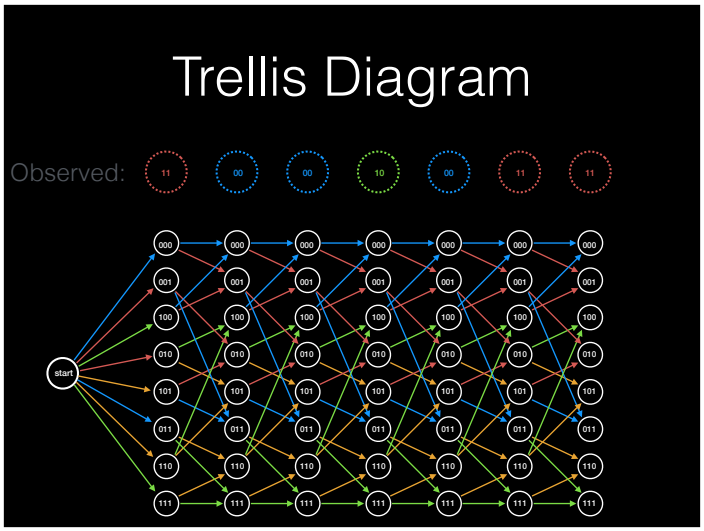
Want:

Sequence of observations

Sequence of states

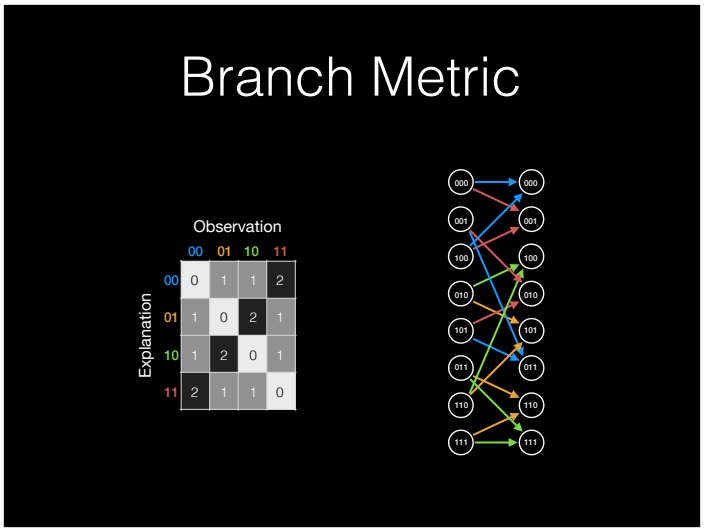
- Viterbi Decoder:
  Find most-likely original transmitted message for a given set of noisy observed bits

78



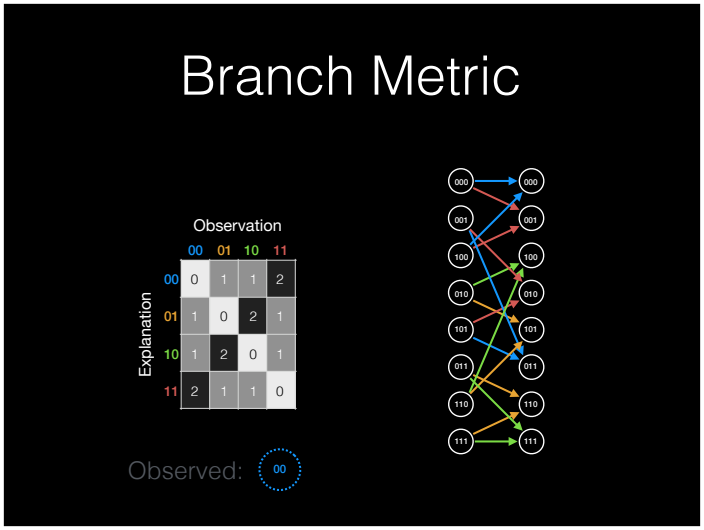
79

We’re going to do the same thing as before, but this time the trellis diagram is a lot more complex.



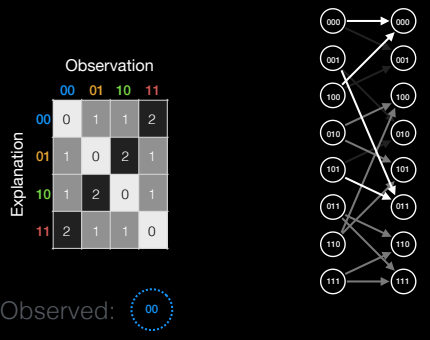
80

We’re also going to make use of a few optimizations. Instead of computing conditional probabilities, we’re going to just compute the Hamming distance between the observation and each explanation. Order stays the same, it’s just simpler.



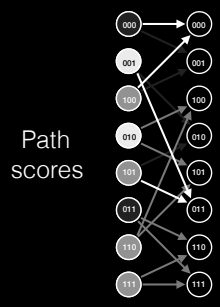
81

Branch Metric



82

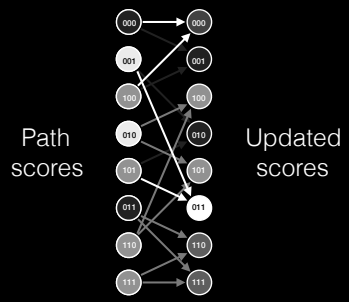
Path Metric



83

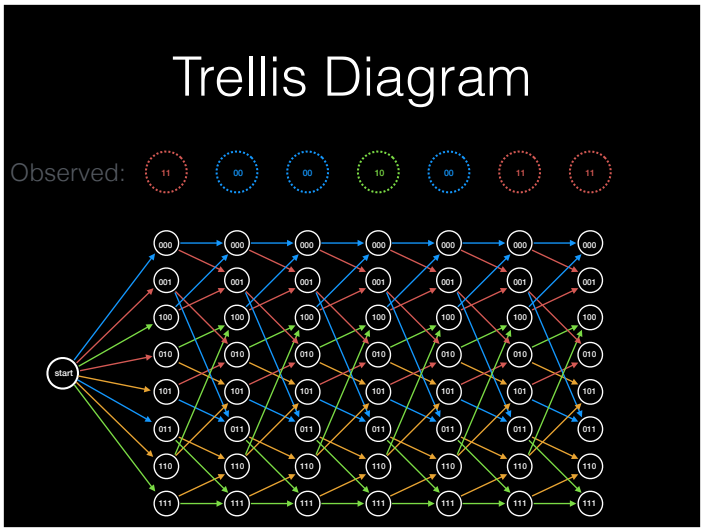
The other optimization is that instead of multiplying out our probabilities, we'll just add the Hamming errors for each branch. Again, all we care about is the most likely path, and this will give the same result. We take our path scores, and add the branch metrics.

Path Metric



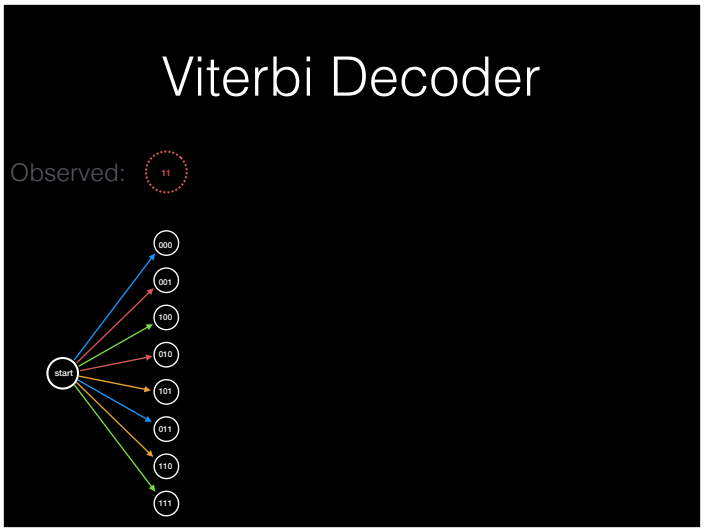
84

Then we compare the results, and select the better score as our next path metric.



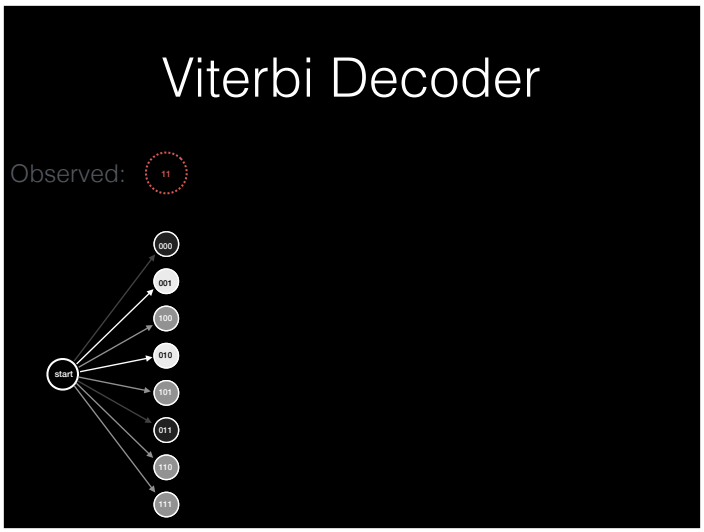
85

Now we can go back to the trellis diagram, and start calculating.



86

At the first time step, we observe a '11'.



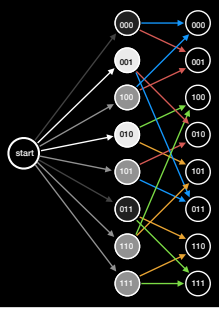
87

We compute initial branch metrics, and those give us our starting path metrics.

88

Viterbi Decoder

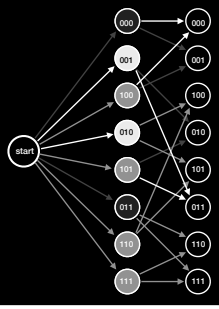
Observed: 11 00



89

Viterbi Decoder

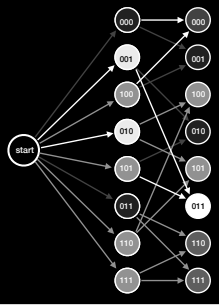
Observed: 11 00



90

Viterbi Decoder

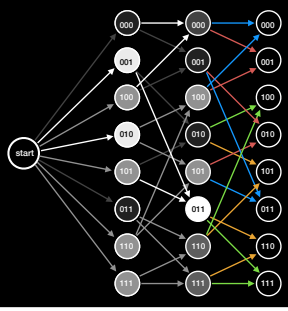
Observed: 11 00



Viterbi Decoder

91

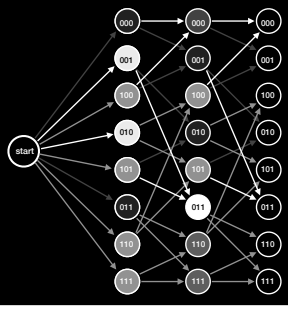
Observed: 11 00 00



Viterbi Decoder

92

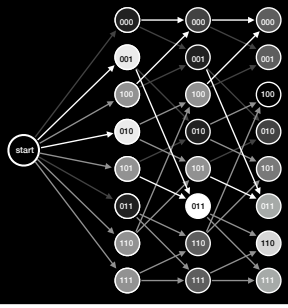
Observed: 11 00 00



Viterbi Decoder

93

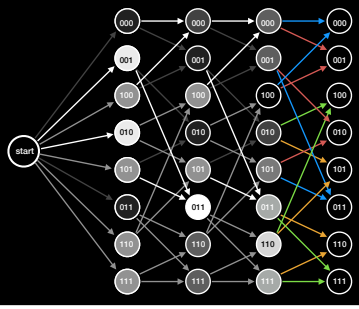
Observed: 11 00 00



94

Viterbi Decoder

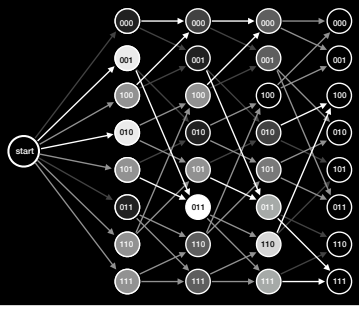
Observed: 11 00 00 10



95

Viterbi Decoder

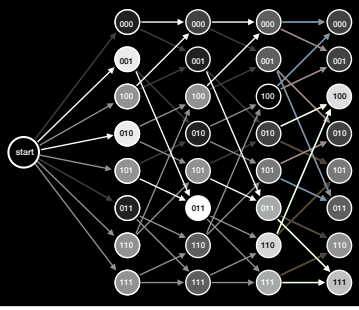
Observed: 11 00 00 10



96

Viterbi Decoder

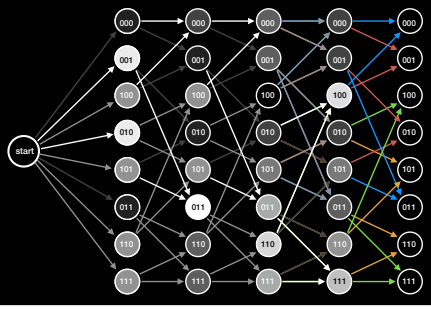
Observed: 11 00 00 10



97

Viterbi Decoder

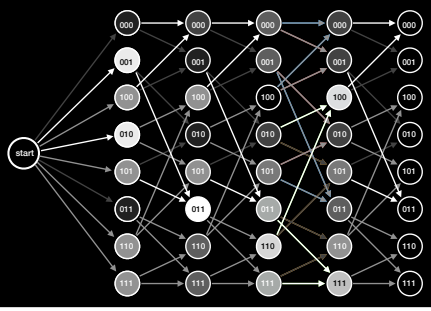
Observed: 11 00 00 10 00



98

Viterbi Decoder

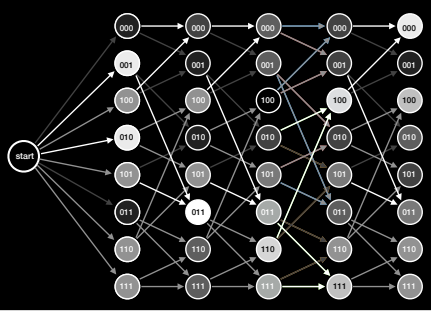
Observed: 11 00 00 10 00

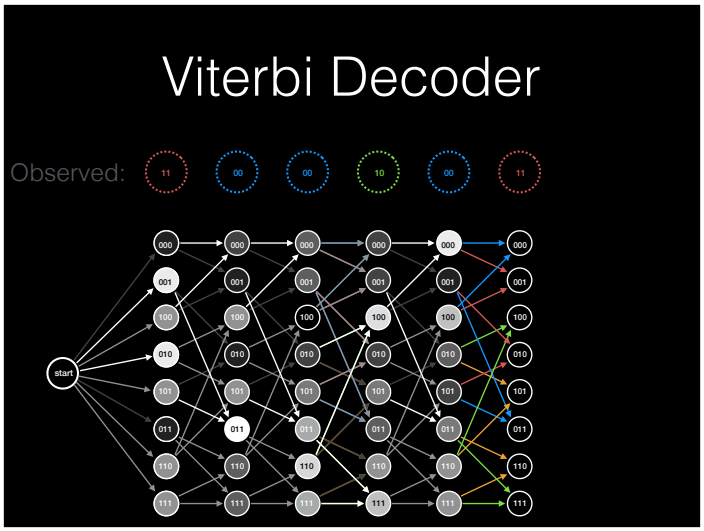


99

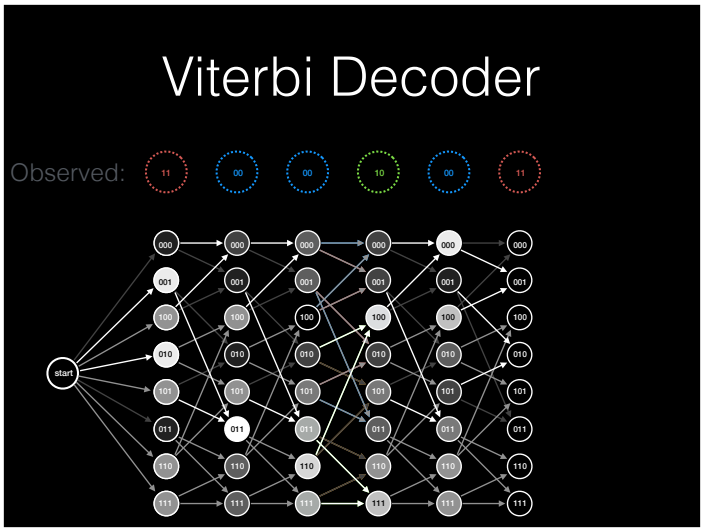
Viterbi Decoder

Observed: 11 00 00 10 00

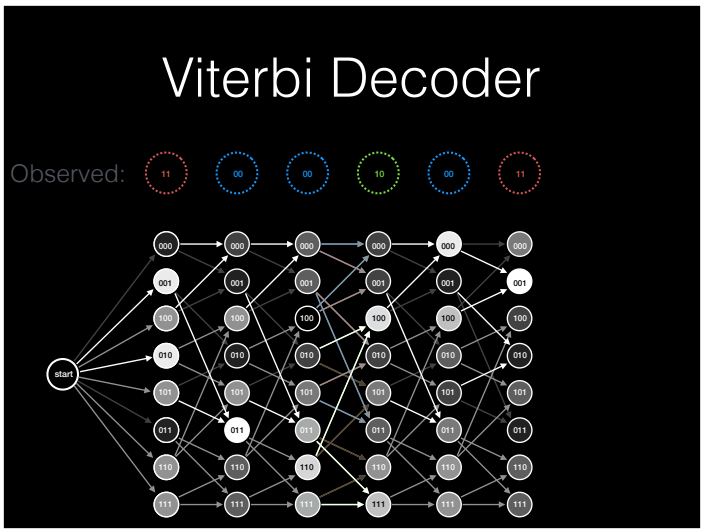



Viterbi Decoder
Observed:
11
00
00
10
00
11
start
000
001
100
010
101
011
110
111

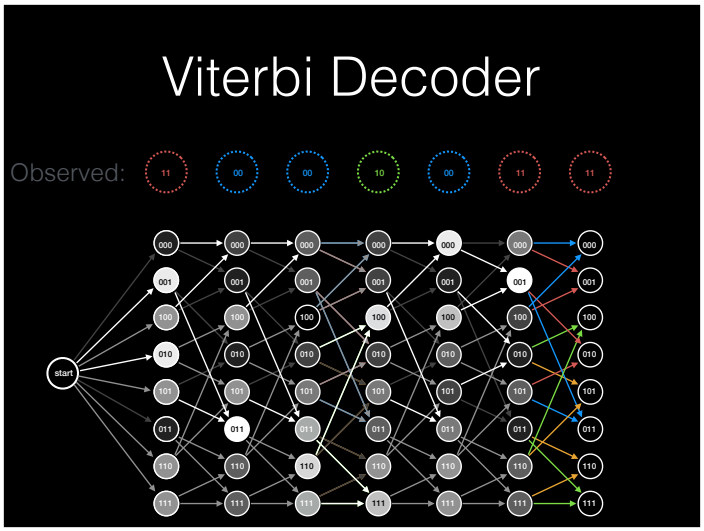
100


Viterbi Decoder
Observed:
11
00
00
10
00
11
start
000
001
100
010
101
011
110
111

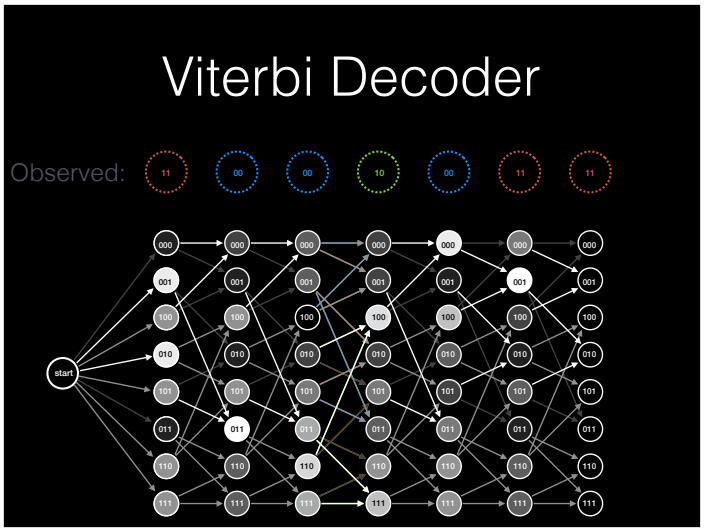
101


Viterbi Decoder
Observed:
11
00
00
10
00
11
start
000
001
100
010
101
011
110
111

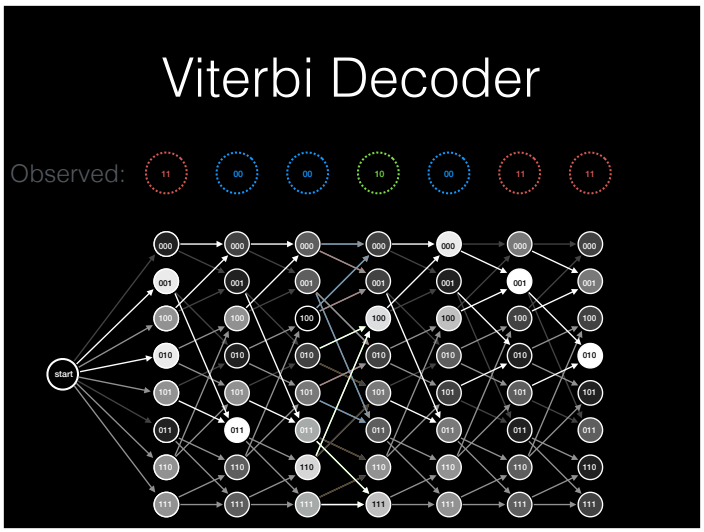
102



103

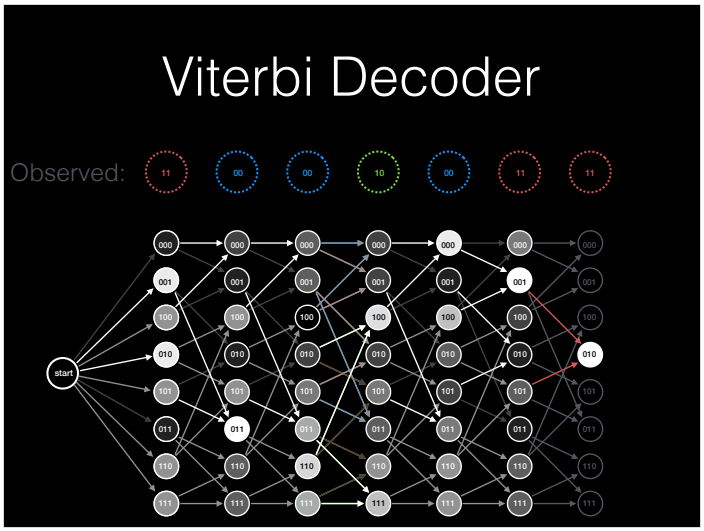


104

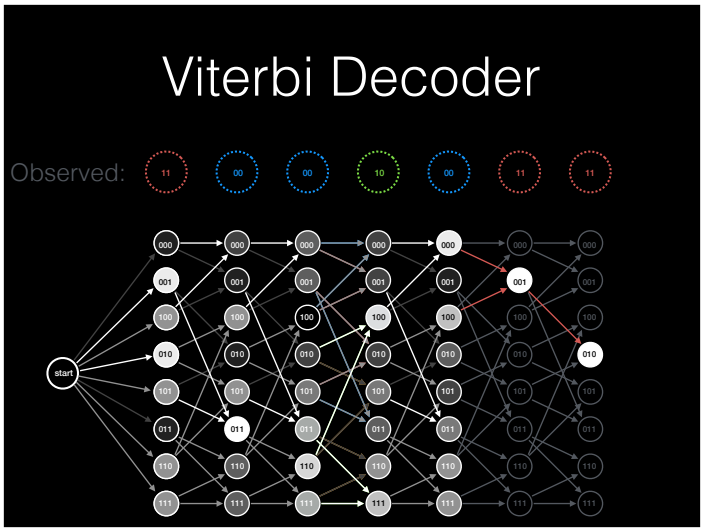


105

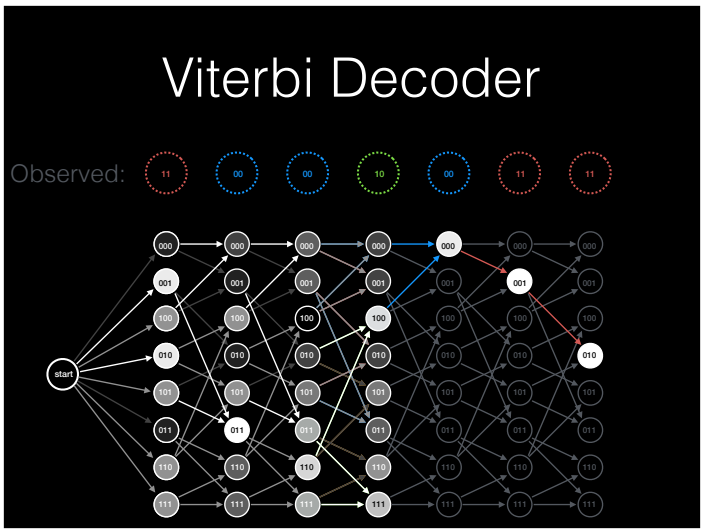
Now our trellis is complete, and we can trace back through it to find the optimal path.


Viterbi Decoder
Observed: 11 00 00 10 00 11 11
start

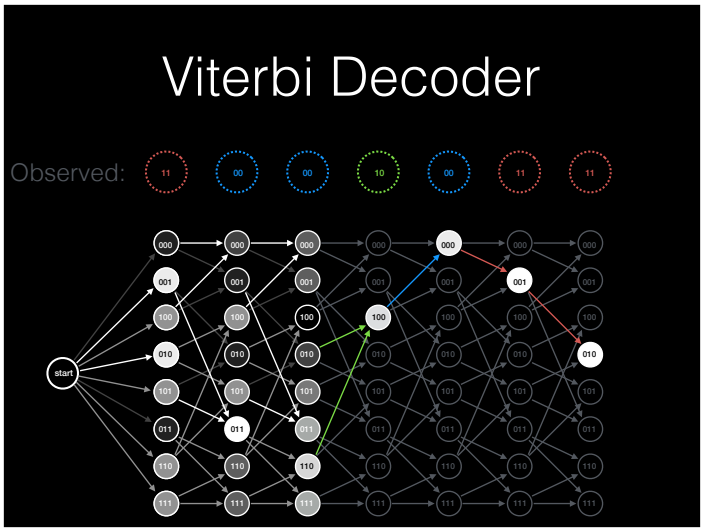
106


Viterbi Decoder
Observed: 11 00 00 10 00 11 11
start

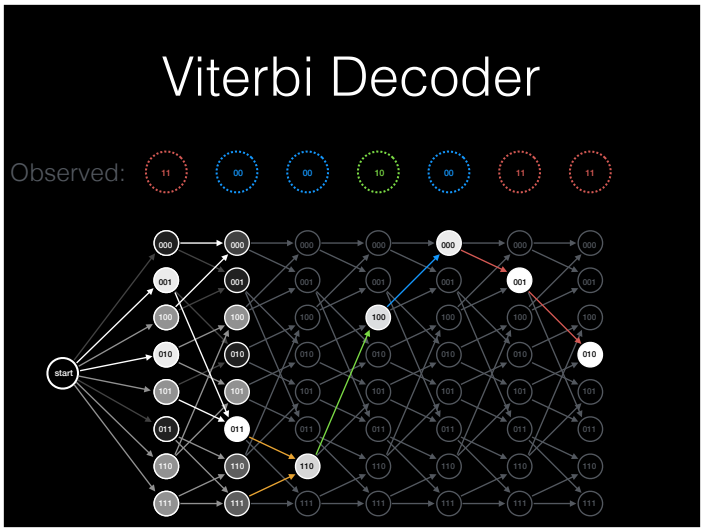
107


Viterbi Decoder
Observed: 11 00 00 10 00 11 11
start

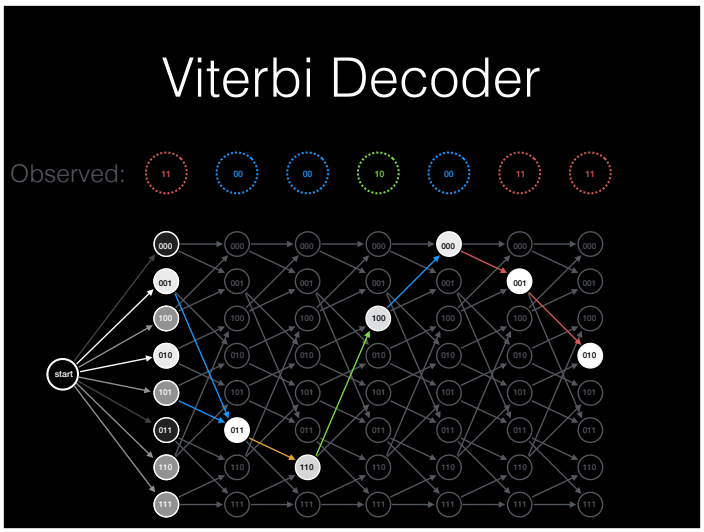
108


Viterbi Decoder
Observed: 11 00 00 10 00 11 11
start

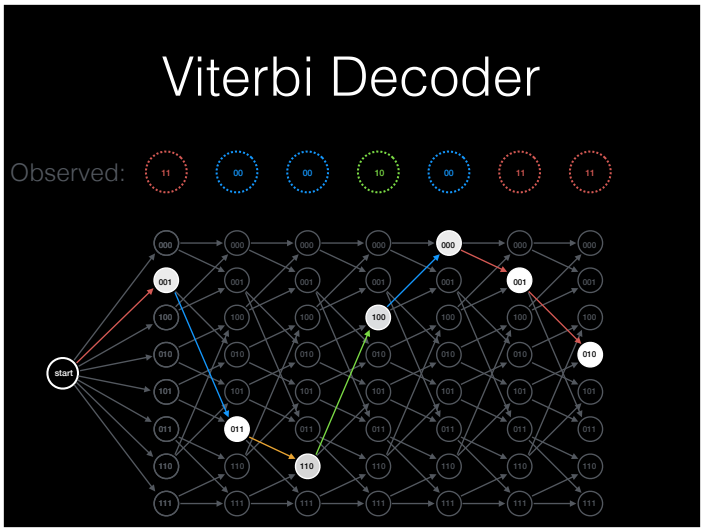
109


Viterbi Decoder
Observed: 11 00 00 10 00 11 11
start

110

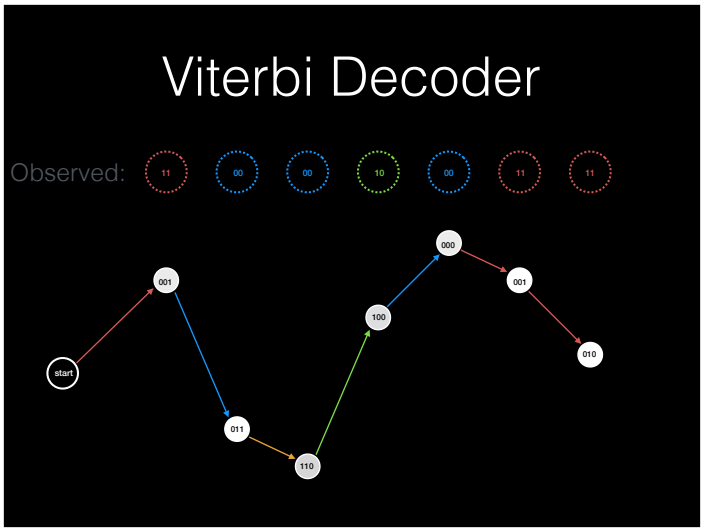

Viterbi Decoder
Observed: 11 00 00 10 00 11 11
start

111

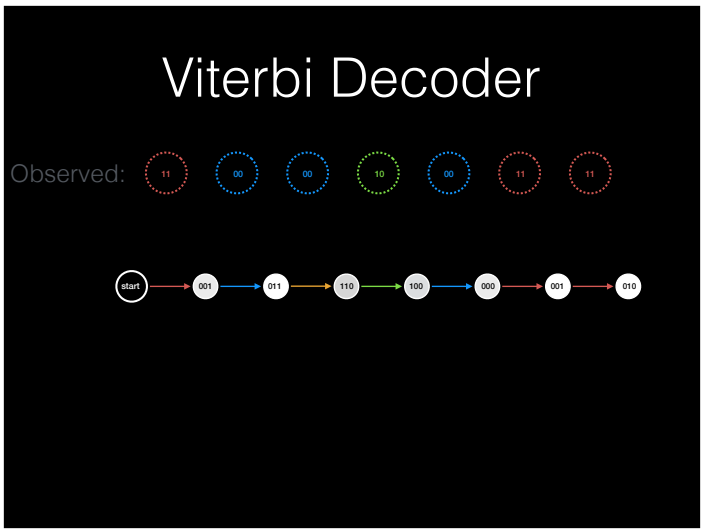


112

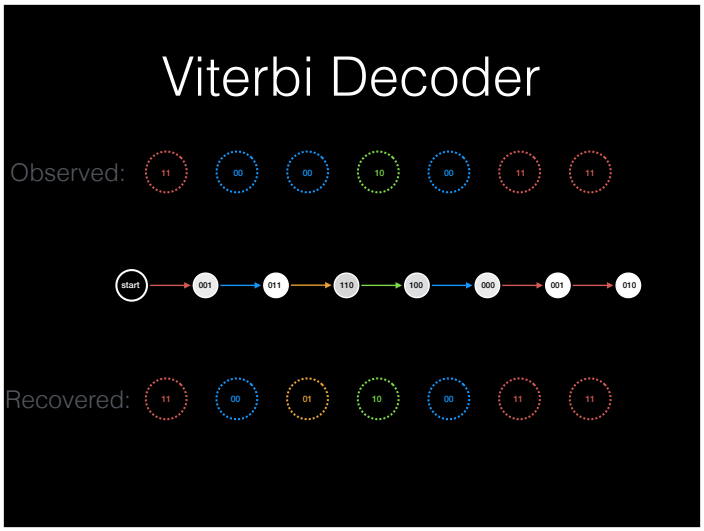
This is the most likely path through the trellis



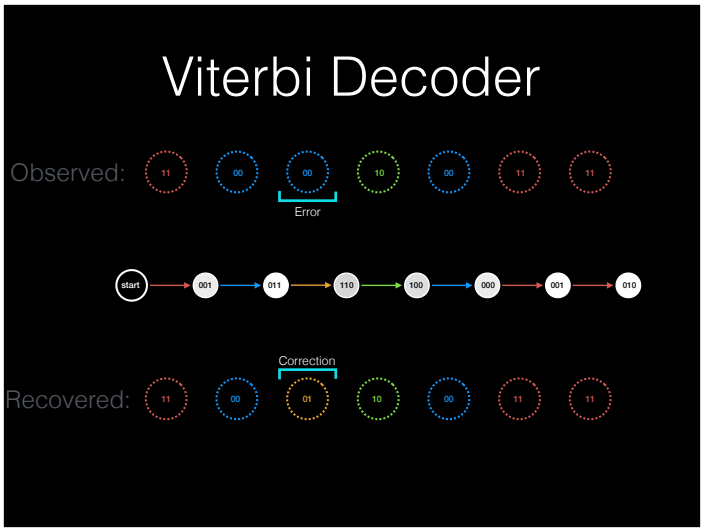
113



114

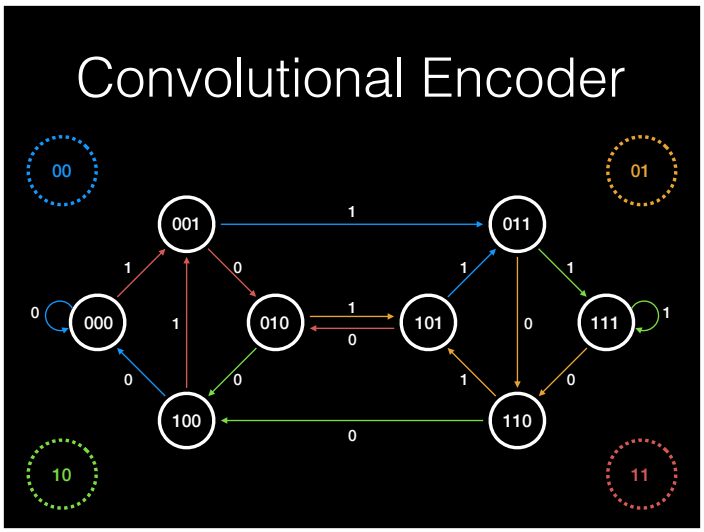


115



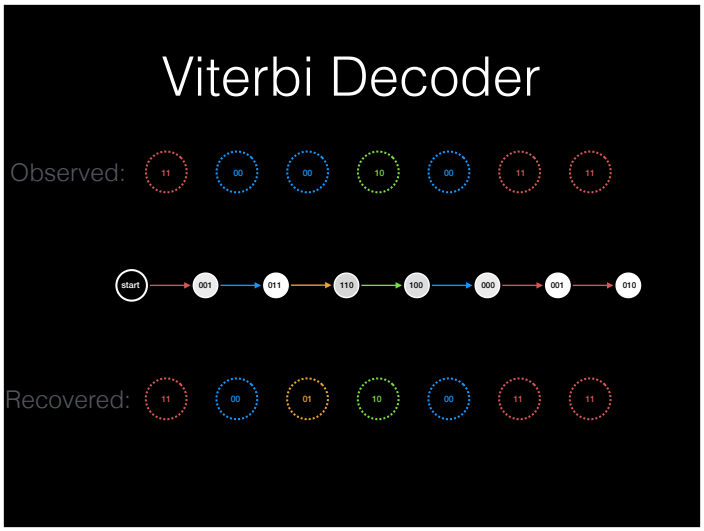
116

You can see the error correcting part is working. The 3rd observation "00" was most likely caused by a "01" transition instead. Now, each one of these arrows corresponds to a bit from the original message.

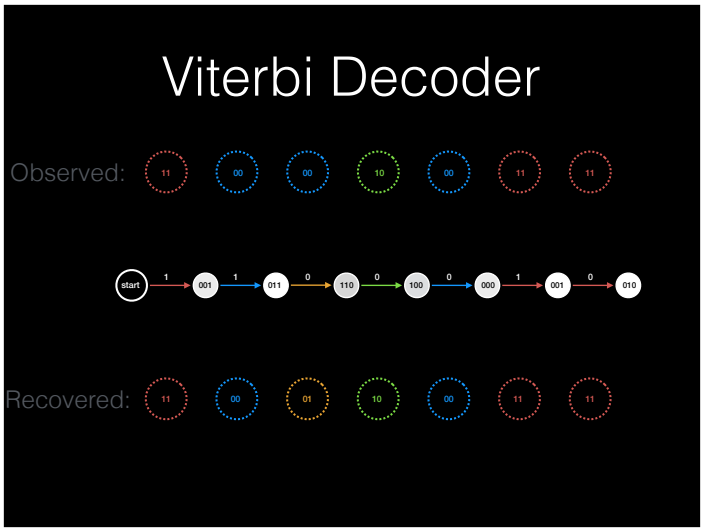


117

We can go back to the encoder state diagram to see which bit goes with each arrow.

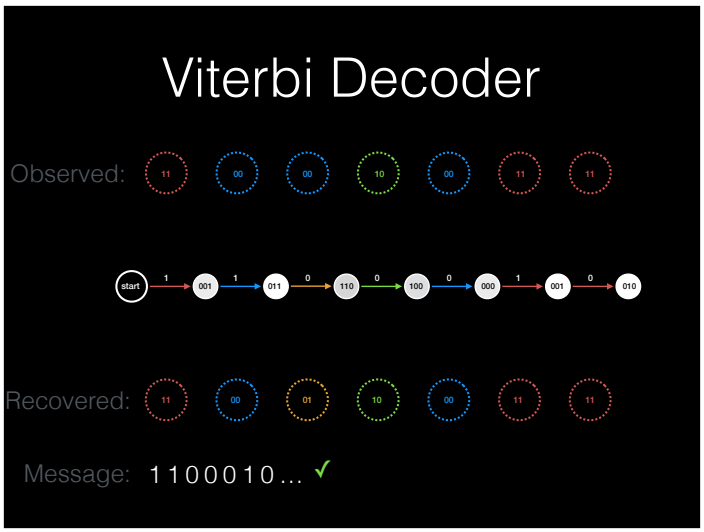


118



119

And then we can fill in the arrows to see what our message was.

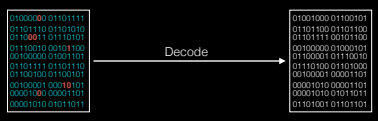


120

And if you don’t remember, this message matches what we originally sent.

Viterbi Decoder

Have: Want:



Sequence of observations Sequence of states

121

So that's the basic version of the Viterbi Decoder… There are a few variations.

Streaming Decoder

Have: Want:

Streaming observations

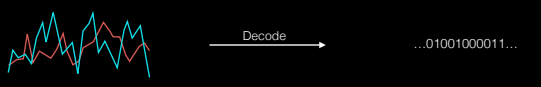
Original message stream

122

Streaming decoders allow you to do a 1-way streaming link, like for continuous sensor data

Soft-Decision Decoder

Have: Want:



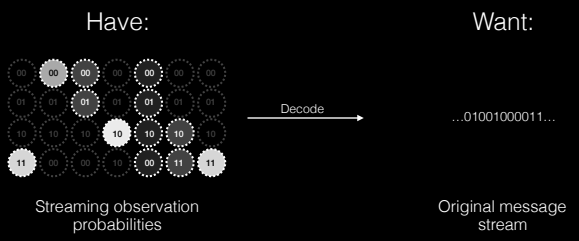
Voltage levels

Original message stream

123

Soft decision decoders are more accurate. Here we have voltage levels from our antenna, and we want our original message stream. We could draw a line through the middle – call everything above it a 1, everything below a 0. (or…)

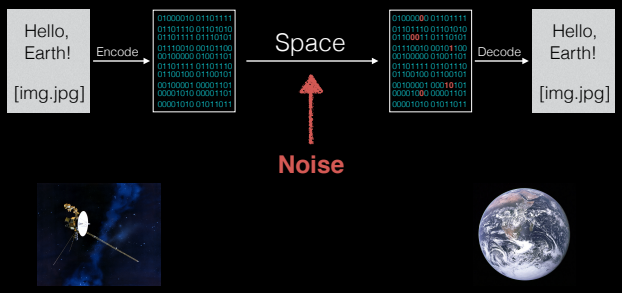
Soft-Decision Decoder



124

We can represent our observations as probability distributions. This allows you to give observations more weight when they have less uncertainty

Example: Communications



125

Remember, the goal is to be able to send messages robustly through a noisy channel

Data Rates

Voyager

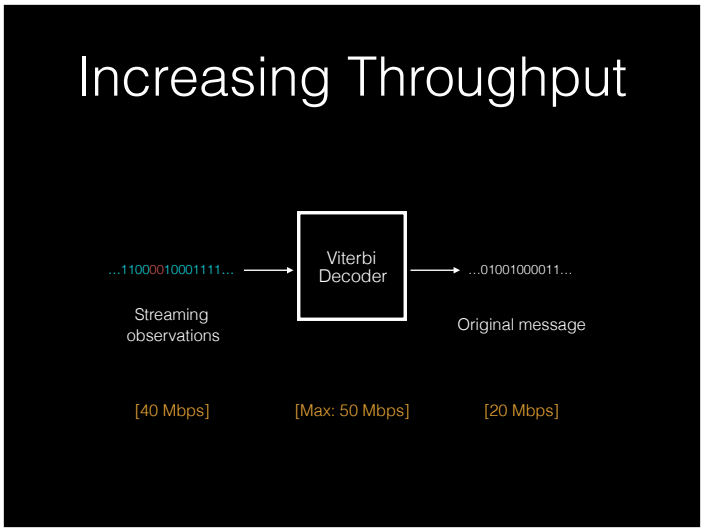
- Low rate: 160 bits/sec
- High rate: 115.2 kbps

Modern Satellites

- Much higher rates (~200 Mbps)

126

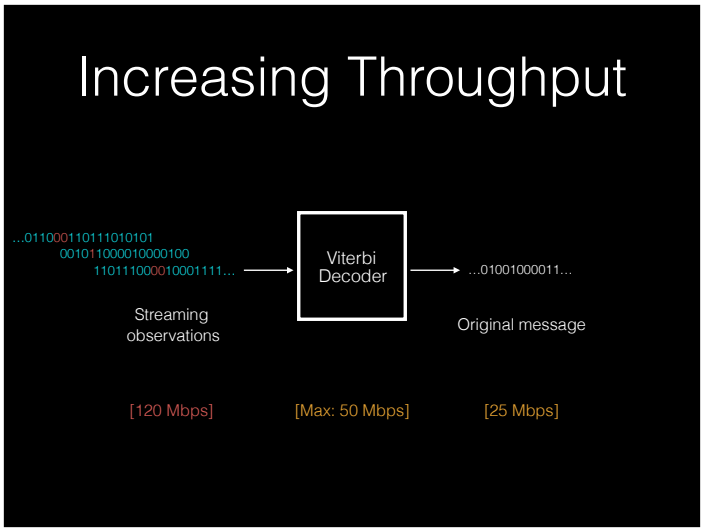
I worked on a project before coming to Duke where we needed to increase the throughput of a Viterbi decoder so we could handle higher data rates.



127

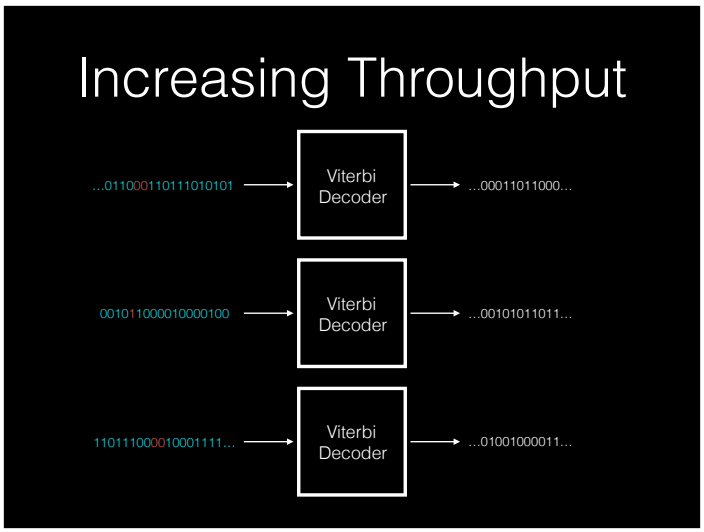
The problem was that our decoder could only handle data rates up to 50 Mbps

---



128

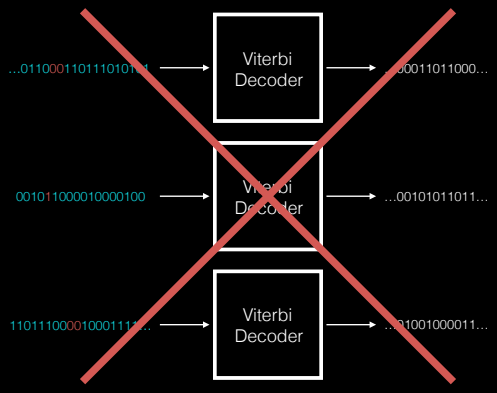
The final version needed to sustain several hundred Mbps.



129

Normally what you'd do in a situation like this is you'd just split the data as it comes in, send to different decoders, and combine the outputs

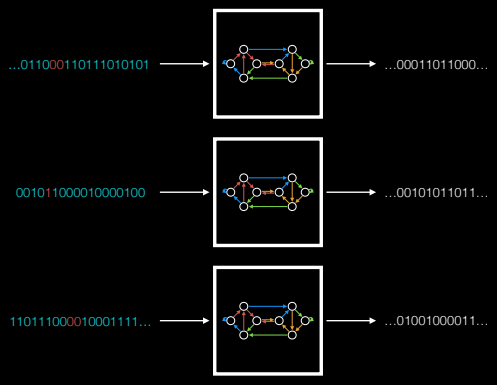
Increasing Throughput



130

Not going to work here

Increasing Throughput



131

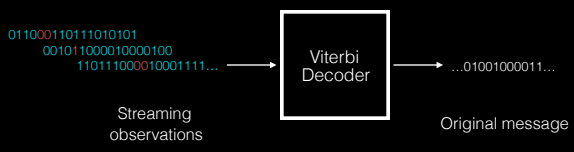
Remember, each decoder needs to keep track of state histories. Splitting up the data won't give you the right results.

Increasing Throughput

- Common parallelization techniques insufficient
  - Hysteresis
  - Streaming data, not block data
- Performance requirements → soft decisions
  - Published decoders used hard decisions
- Not enough time to rewrite existing decoder

132

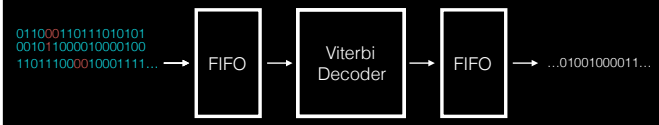
Increasing Throughput



133

To solve it, we ended up doing something similar to our original plan.

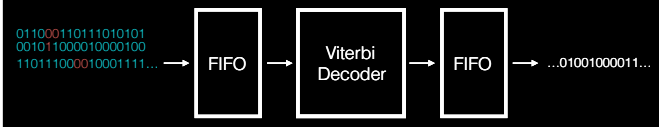
Increasing Throughput



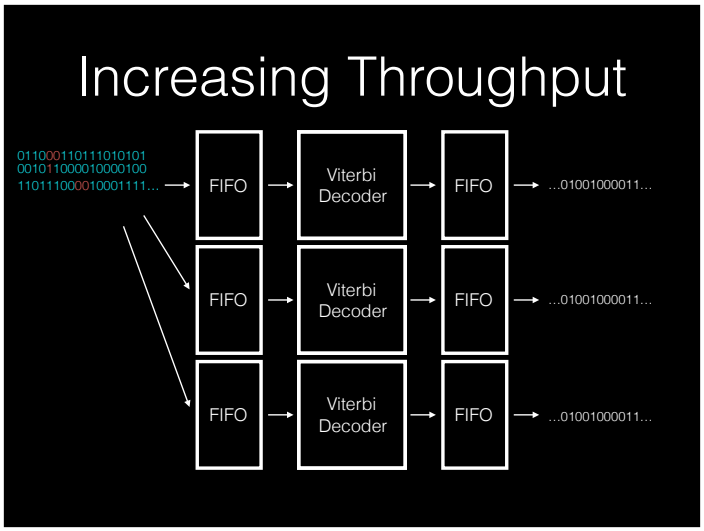
134

But we added these FIFOs on each end

Increasing Throughput

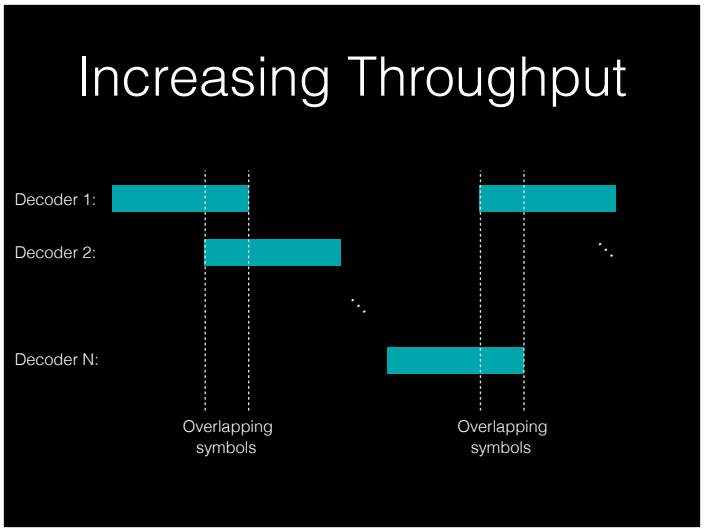


135



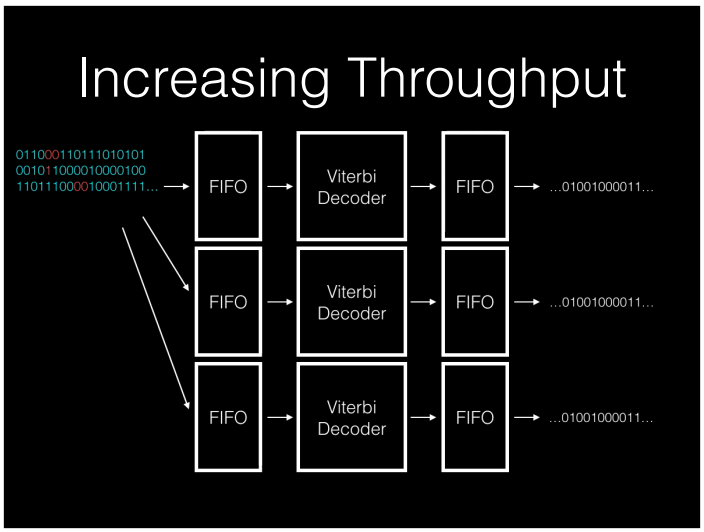
136

And we used multiple decoders, but we didn't split the data up like we did before…



137

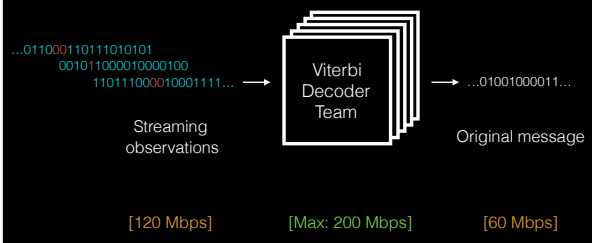
Instead, I came up with a novel method which sent overlapping, consecutive samples to multiple decoders at same time, and then rotated batches of samples through the decoders in sequence



138

Doing this effectively initializes each with enough state history information to recover a piece of the original message. By combining the pieces, we could reproduce the entire message

Results



139

Using this method, we able to meet arbitrarily high throughput requirements, and meet our project deadline, without needing to rewrite our existing decoder

---

References

- MIT Lecture Notes – Digital Communication Systems
  - Lecture 8 – Convolutional Coding
  - Lecture 9 – Viterbi Decoding of Convolutional Codes
- Wikipedia
  - Viterbi Algorithm
  - Viterbi Decoder
  - Voyager 1

140
