Unit 10: Convolutional Codes

EL-GY 6013: DIGITAL COMMUNICATIONS
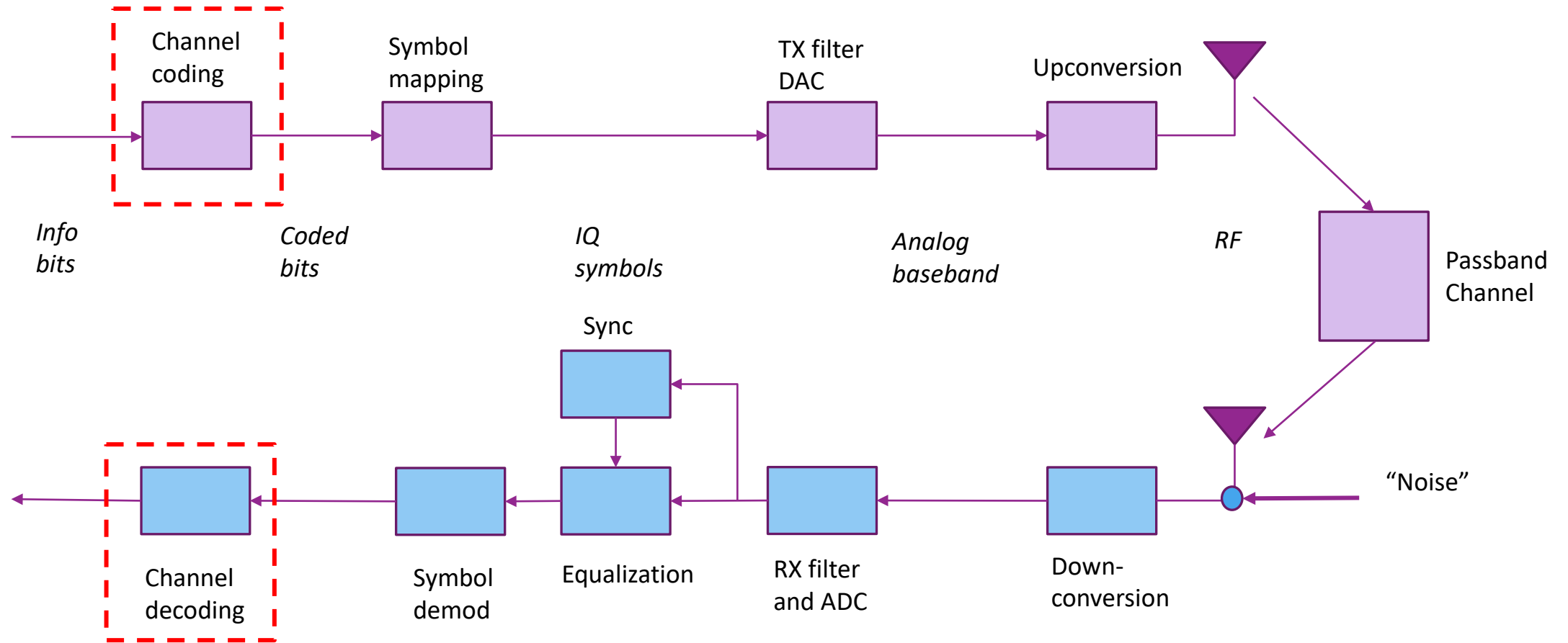
PROF. SUNDEEP RANGAN

Learning Objectives

❑Encode data using a convolutional code for given a generator polynomial

❑Compute the rate of the code including tail bits

❑Represent the code via a trellis diagram and finite state machine

❑Compute the branch metrics of a code for a memoryless channel

❑Decode the code via the Viterbi algorithm

This Unit

Outline

➡ Convolutional codes: encoding and representations

❑ Tree, trellis and state diagrams

❑ Decoding with branch metrics

❑ Viterbi decoding

Convolutional Codes History

❑ Generates a stream of coded bits from uncoded bits
- Block codes form by terminating the stream

❑ Output stream created by binary FIR filters

❑ Developed originally by Elias (1955)
- Key challenge was decoders. Much study in 1960s.
- Practical, optimal decoders developed by Viterbi,1967.

❑ Can perform within 2-3 dB of Shannon limit.

❑ Instrumental in Pioneer Space program (along with RS codes)

❑ Most widely-used code in industry today: WiFi, cellular
- In the mid 1990s, longer block length cellular codes were replaced with Turbo codes,
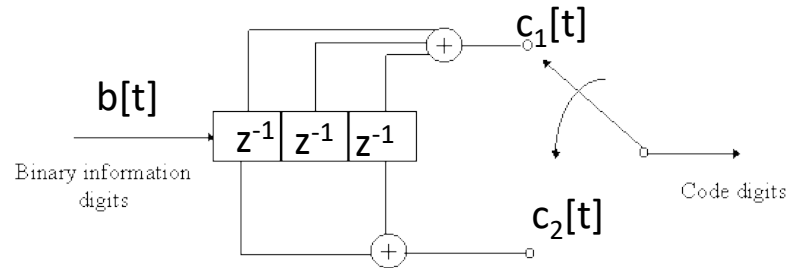- But, these use convolutional codes as basis

Convolutional Codes

❑Encode data through parallel binary (usu. FIR) filters

❑Example convolutional code:
- Rate = ½ (two output bits $(c_1[t], c_2[t])$ for each input bit b[t].
- Constraint length K=3 (size of shift register)
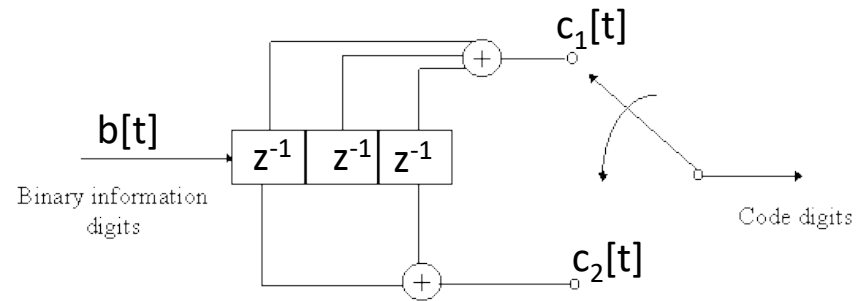
$$c_1[t] = b[t] + b[t-1] + b[t-2]$$
$$c_2[t] = b[t] + b[t-2]$$

Convolutional Codes
Block Implementation

$$c_1[t] = b[t] + b[t-1] + b[t-2]$$
$$c_2[t] = b[t] + b[t-2]$$

❑ To implement as block code:
- Start with L input bits b[0],b[1],…,b[L-1]
- Append K-1 zero b[L]=b[L+1]=…=b[L+K-2]=0 (called tail bits)
- Generate output bits from each branch
  - $c_j[0], c_j[1], … , c_j[L+K-2]$, j=1,…,N where N = number of branches
- Final codeword is concatenation of branch output
  - $\mathbf{c}=(c_1[0], c_1[1], … , c_1[L+K-2],…, c_N[0], c_N[1], … , c_N[L+K-2])$

❑ Effective rate: $R = \dfrac{L}{N(L+K-1)} \approx \dfrac{1}{N}$ for large L
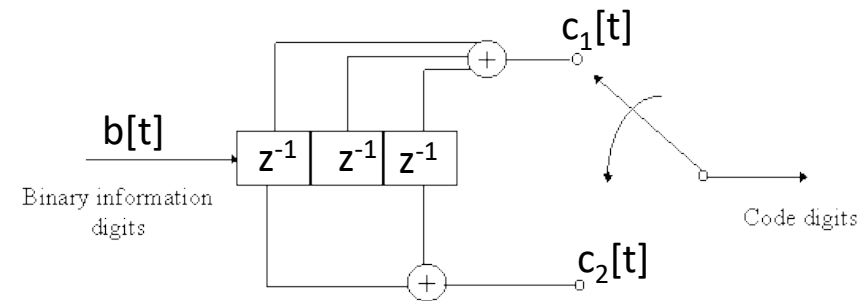
Convolutional Codes
Encoding Example

❑ Encode message **b** = [1 0 1]

❑ Branch outputs **c**1=[11011] c2=[11001]

❑ Final output **c**=[11,10,00,10,11] (interleaved)

❑ Rate = 3/10

t	b[t]	c1[t]	c2[t]
0	1	1	1
1	0	1	0
2	1	0	0
3	0	1	0
4	0	1	1

Tail bits

$$c_1[t] = b[t] + b[t-1] + b[t-2]$$
$$c_2[t] = b[t] + b[t-2]$$

Generator Polynomials: Binary Form
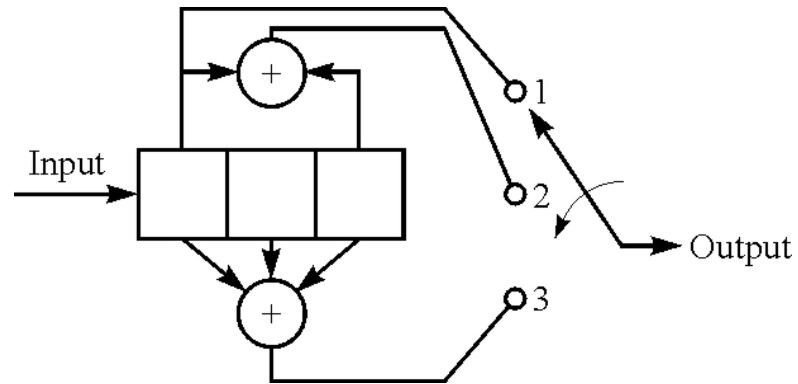
Rate 1/3, K=3 example:

$$c_1[t] = b[t]$$
$$c_2[t] = b[t] + b[t-1]$$
$$c_2[t] = b[t] + b[t-1] + b[t-2]$$

❑Code polynomials: Binary vector of filter coefficients

g_1 =   [100]

g_2 =   [101]

g_3 =   [111]

Generator Polynomials: Octal Form

❑For large constraint lengths (large K), binary form is inefficient
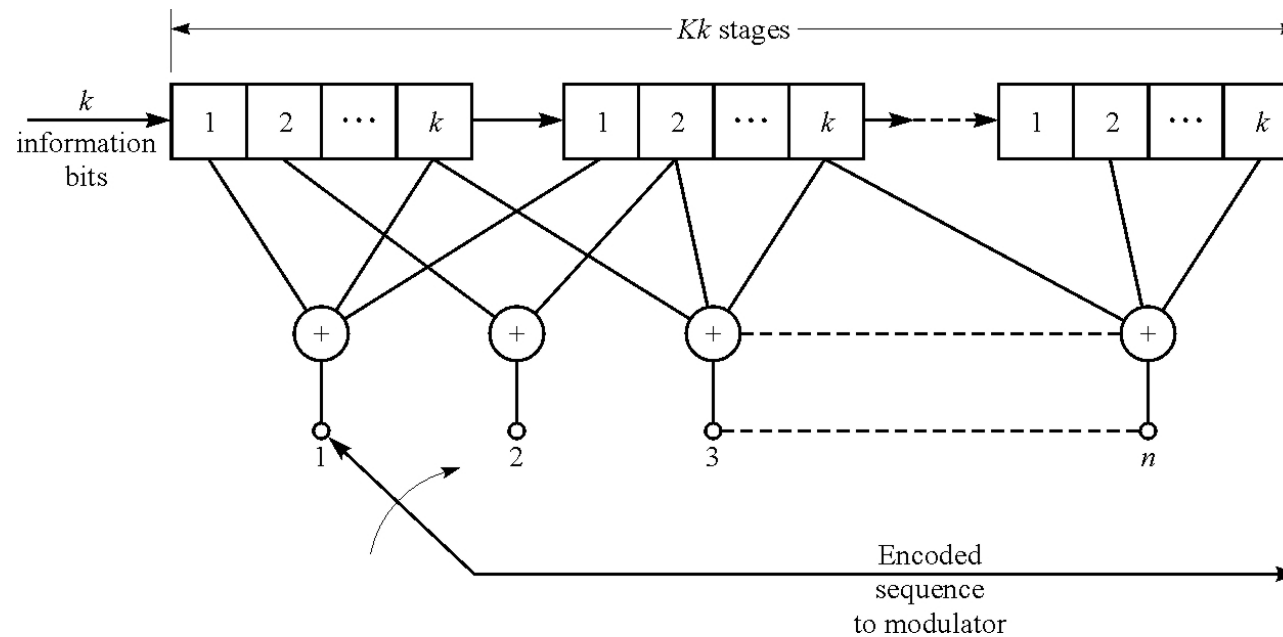 ◦ Engineers often use octal form
  ◦ Base 8, each digit 0...7
  ◦ Each digit represents three bits

❑Example:

g_1 =   [1 011] g_1 =   [13]

g_2 =   [1 101]   ➡ g_2 =   [15]

g_3 =   [1 010] g_3 =   [12]

Binary   Octal

Multiple Inputs

❑Examples up to now are R=1/n

❑Can extend to rate R=k/n
◦ Add k bits at a time

In-Class Exercise

Convolution Code In-Class Exercises

Building an Encoder

We will look at a simple R=1/2 convolution code:

$$c(t,1) = b(t) + b(t-1) + b(t-2)$$
$$c(t,2) = b(t) + b(t-2)$$

This encoder can be represented with a generator matrix G as shown.

```
G = [1 1 1; 1 0 1];
nin = 1;                % number of input streams
nout = size(G,1);       % number of output streams
conLen = size(G,2);     %cconstraint length
```

We will build a simple encoder using the convolution method. The convolution method in MATLAB uses real values, so we will use a modulo to convert back to .

Outline

❑ Convolutional codes: encoding and representations

➡ Tree, trellis and state diagrams

❑ Decoding with branch metrics

❑ Viterbi decoding

Convolutional Codes as State Machines

❑Convolutional codes have memory
- Stored in contents of shift registers
- Each input bit changes memory contents
- Output bits are a function of memory and input

❑Three common ways to represent evolution of the memory:
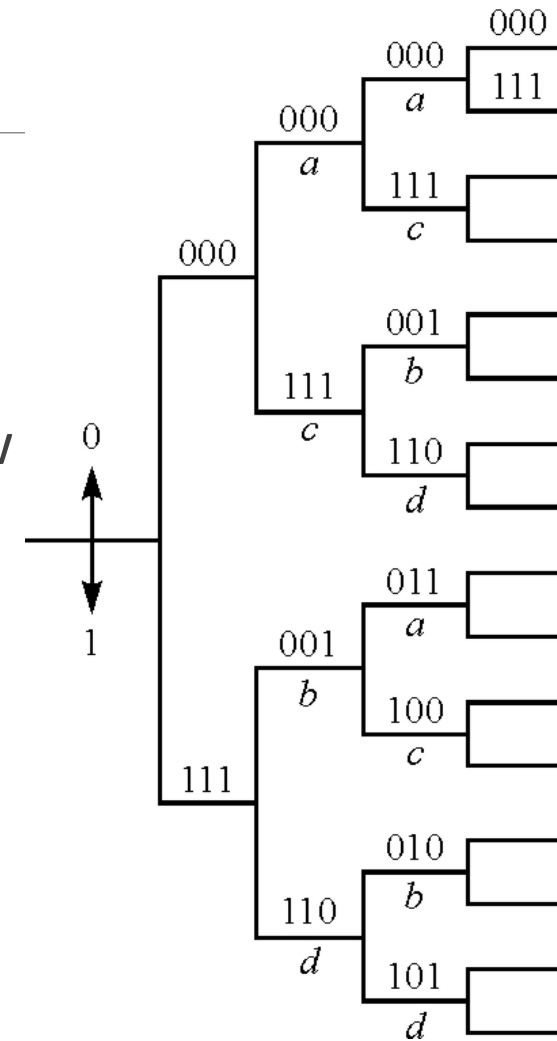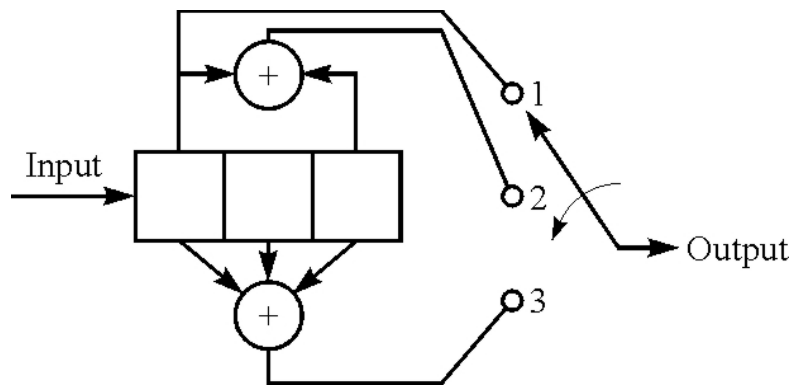- Tree diagram
- Trellis diagram
- State diagram

Encoder States

❑ State of the encoder determined by contents of shift register

❑ Only need to look at most recent K-1 bits

  ◦ Last bit will be shifted out
  ◦ There are 2^{K-1} states
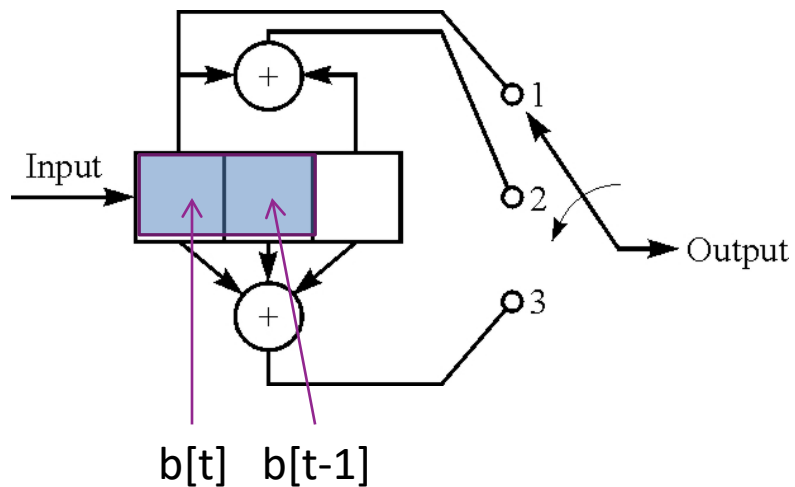
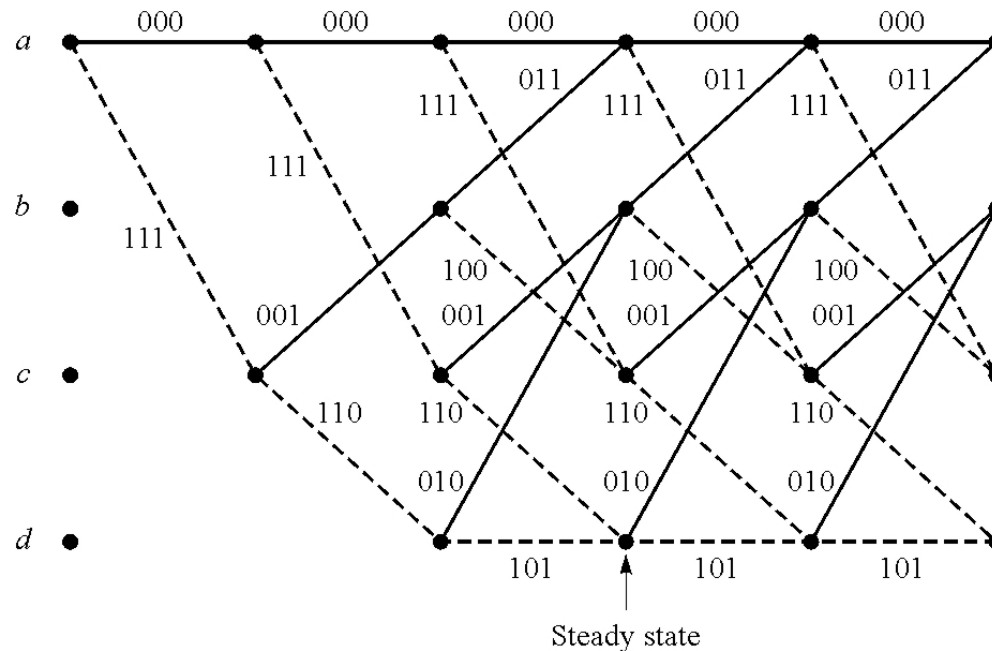| State label	$b[t]$	$b[t-1]$
a	0	0
b	0	1
c	1	0
d	1	1

Tree Diagram

❑Two branches for each input bit 0 or 1.

❑Branches labeled by output bits (n bits)
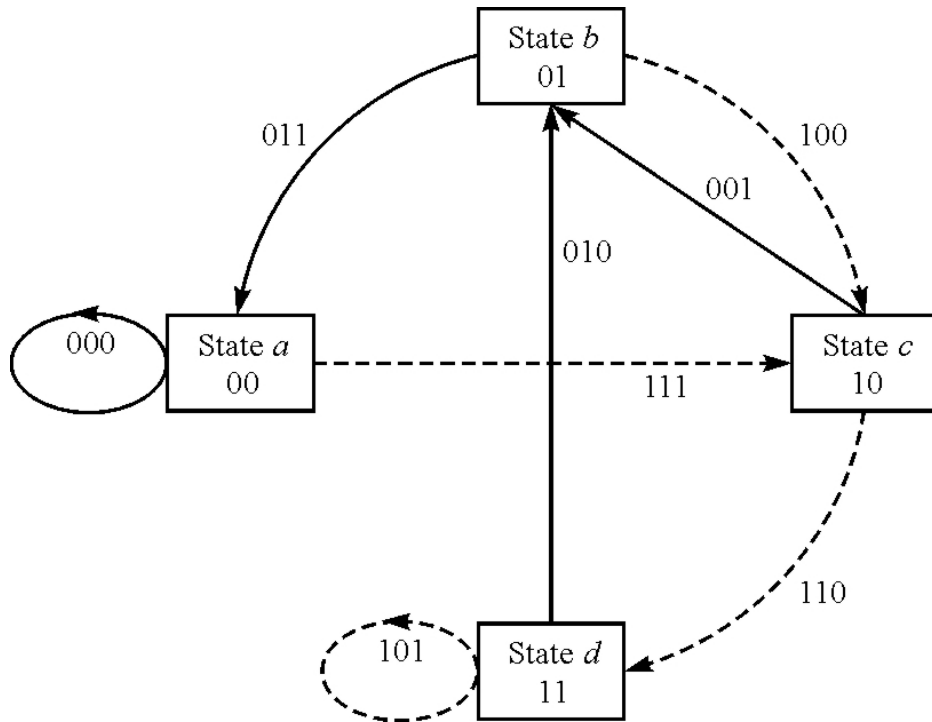
❑Difficult to use since branches infinitely grow

Trellis Diagram

☐ Show trajectory through the states

☐ Solid lines: paths with b[t]=0, Dash lines: b[t]=1

Labels are the outputs along each path

State Diagram

- One node per state

- Solid lines:
  - Transitions with b[t]=0

- Dashed lines:
  - Transitions with b[t]=1

- Labels indicate outputs on each state

State Machine Functional Description

❑ Finite state machine:
  ◦ $x[t]$: Sequence of states, $x[t] \in \{0, \ldots, 2^{K-1} - 1\}$
  ◦ $b[t]$: Input sequence
  ◦ $c[t]$: Output sequence

❑ Iterative generating sequence:

$$x[t+1] = f(x[t], b[t]) \quad \longleftarrow \text{State function}$$
$$c[t] = h(x[t], b[t]) \quad \longleftarrow \text{Output function}$$

❑ Initial condition: $x[0]$

In-Class Exercise

Representing the State Transition Function

We wish to represent the convolutional encoder as a finite state machine. To this end, we will represent the states by a single decimal number:

$$x(t) = 2*b(t-2) + b(t-1) + 1$$

This state runs from `x(t) = 1,...,nstate` where `nstate = 2^{convLen-1}`

To represent the state transition, we create a matrix `xnextMat`

- `xnextMat(i,1)`= the value of `x(t+1)` when `b(t)=0` and `x(t)=i`
- `xnextMat(i,2)` = the value of `x(t+1)` when `b(t)=1` and `x(t)=i`

```
ninVal = 2^nin;         % number of input values per time step
nstate = 2^(conLen-1);  % number of states

% Intiialize matrix
xnextMat = zeros(nstate,ninVal);

% TODO:  Write a for loop to fill in the xnextMat matrix
```

We next create a matrix to represent the output:

- `coutMat(k,i,j) = output c(t,k)  when x(t)=i and b(t)=j+1`

Outline

❑Convolutional codes: encoding and representations

❑Tree, trellis and state diagrams

➡❑Decoding with branch metrics

❑Viterbi decoding

ML Estimation

❑ Assume the following dimensions:

◦ N outputs per time steps: $c[t] = (c_1[t], \ldots, c_N[t])$

◦ T time steps (including tail bits!)

❑ Channel model:

◦ For each bit $c_i[t]$, we make some observation $r_i[t]$

◦ Output is probabilistic $P(r_i[t]|c_i[t])$

◦ Assume all outputs are independent:

$$P(\boldsymbol{r}|\boldsymbol{c}) = \prod_{t=0}^{T-1} \prod_{i=1}^{N} P(r_i[t]|c_i[t])$$

❑ Find ML estimate:

$$\hat{c} = \arg\max_{c} P(r|c)$$

◦ Maximum over all sequences

LLR Review

❑Before studying convolutional codes, consider simple code:
- ◦ Codeword has N output bits: $\boldsymbol{c} = (c_1, \ldots, c_N)$
- ◦ Receive N symbols $r = (r_1, \ldots, r_N)$. Each r_i may be binary, discrete, real-valued or complex-valued
- ◦ Assume a memoryless channel:

$$P(r|c) = \prod_{i=1}^{N} P(r_i|c_i)$$

❑Define LLR: $L_i = \log \dfrac{P(r_i|c_i = 1)}{P(r_i|c_i=0)}$

❑Theorem: The ML codeword maximizes the LLR sum value function:

$$\hat{c} = \arg \max_c J(c), \qquad J(c) = \sum_{i=0}^{N-1} c_i L_i$$

NYU | TANDON SCHOOL OF ENGINEERING

Example 1: Binary Symmetric Channel

❑Binary symmetric channel: Output $r_i = 0$ or 1

❑Error probability $p < 1/2$

$$P(r_i|c_i) = \begin{cases} 1-p & \text{if } c_i = r_i \text{ (no error)} \\ p & \text{if } c_i \neq r_i \text{ (error)} \end{cases}$$

❑Branch metric: After appropriate shifting:

$$L_i = \begin{cases} A & \text{if } c_i = r_i \text{ (no error)} \\ 0 & \text{if } c_i \neq r_i \text{ (error)} \end{cases}, \qquad A = \log\frac{1-p}{p}$$

❑ML estimate value function:

$$J(c) = \sum_{i=0}^{N-1} c_i L_i = A\sum_{i=0}^{N-1} c_i 1_{\{c_i=r_i\}} = A(\# \text{ correct bits})$$

◦ Maximizes number of correct bits

Example 2: QPSK Channel

❑Binary modulation (on a real or imaginary dimension):

$$r_i = s_i + w_i, \qquad w_i \sim N(0, N_0/2), \qquad s_i = \begin{cases} A & c_i = 1 \\ -A & c_i = 0 \end{cases}$$

❑Gaussian distribution: $\log p(r_i|s_i) = -\frac{1}{N_0}(r_i - s_i)^2 - \frac{1}{2}\log\frac{N_0}{2}$

❑LLR is: $L_i = \log p(r_i|s_i = A) - \log p(r_i|s_i = -A) = -\frac{1}{N_0}[(r_i - A)^2 - (r_i + A)^2] = \frac{4A}{N_0}r_i$

❑LLR sum value function:

$$J(c) = \sum_i L_i c_i = \frac{4A}{N_0}\sum_i r_i c_i$$

Branch Metric

☐ Log-likelihood ratio: $L_i[t] = \log \dfrac{P\big(r_i[t]\big|c_i[t] = 1\big)}{P\big(r_i[t]\big|c_i[t] = 0\big)}$
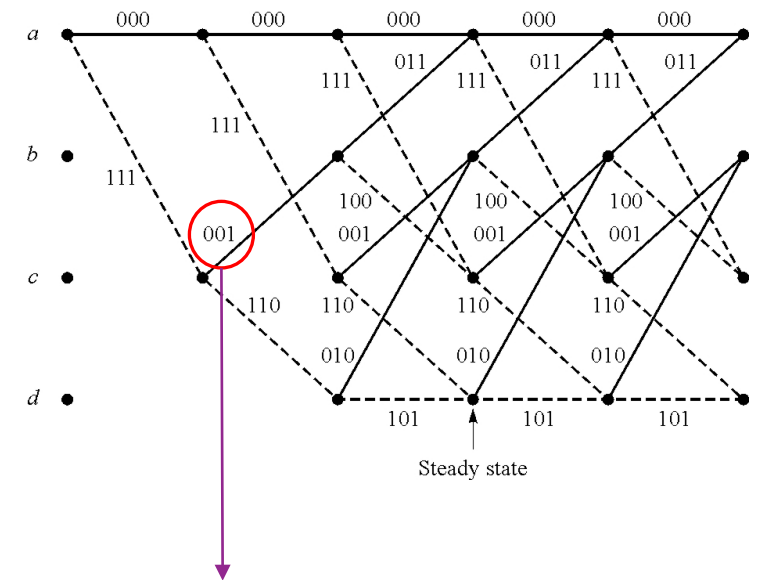
- Depends on received symbol $r_i[t]$

☐ Each branch has an output set of bits $c[t] = (c_1[t], \ldots, c_N[t])$

☐ Define the branch metric:

$$\mu_t(x_{t+1}, x_t) = \sum_{i=1}^{N} c_i[t]\, L_i[t]$$

- $c_i[t]$ is the coded outputs on the branch

☐ Describes likelihood that sequence passed through the branch
- Branches with higher branch metrics are more likely

$c[t] = (0,0,1) \Rightarrow$
$\mu_t = 0L_1[t] + 0L_2[t] + L_3[t] = L_3[t]$
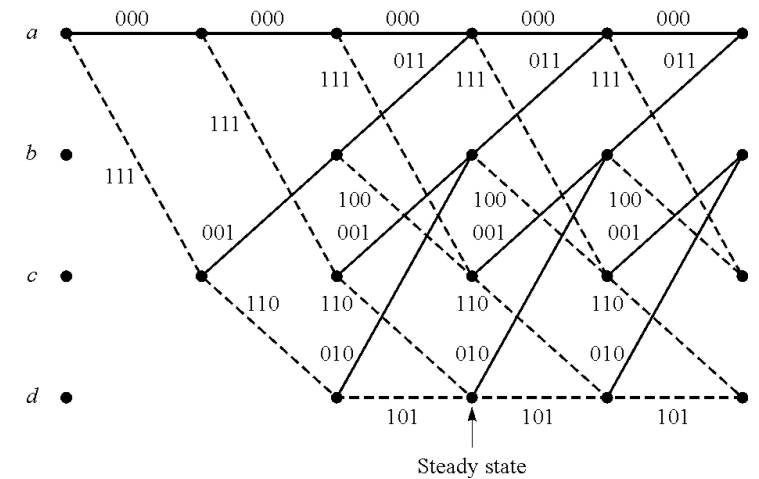
Branch Metric Maximization

❑ Each branch in trellis has a branch metric $\mu_t(x_{t+1}, x_t) = \sum_{i=1}^{N} c_i[t] L_i[t]$

❑ We can write LLR sum value function as:

$$J(c) = \sum_{t=0}^{T-1} \sum_{i=1}^{N} c_i[t] L_i[t] = \sum_{t=0}^{T-1} \mu_t(x_{t+1}, x_t)$$

❑ Interpretation:
  ◦ Each code sequence is a path in the trellis
  ◦ Value function is the sum of branch metrics on the path

Conclusion: ML estimate is the codeword with the highest total path

Example

□ Consider toy Trellis (not a real conv code)

□ Input: $b[t], t = 0,1,2$

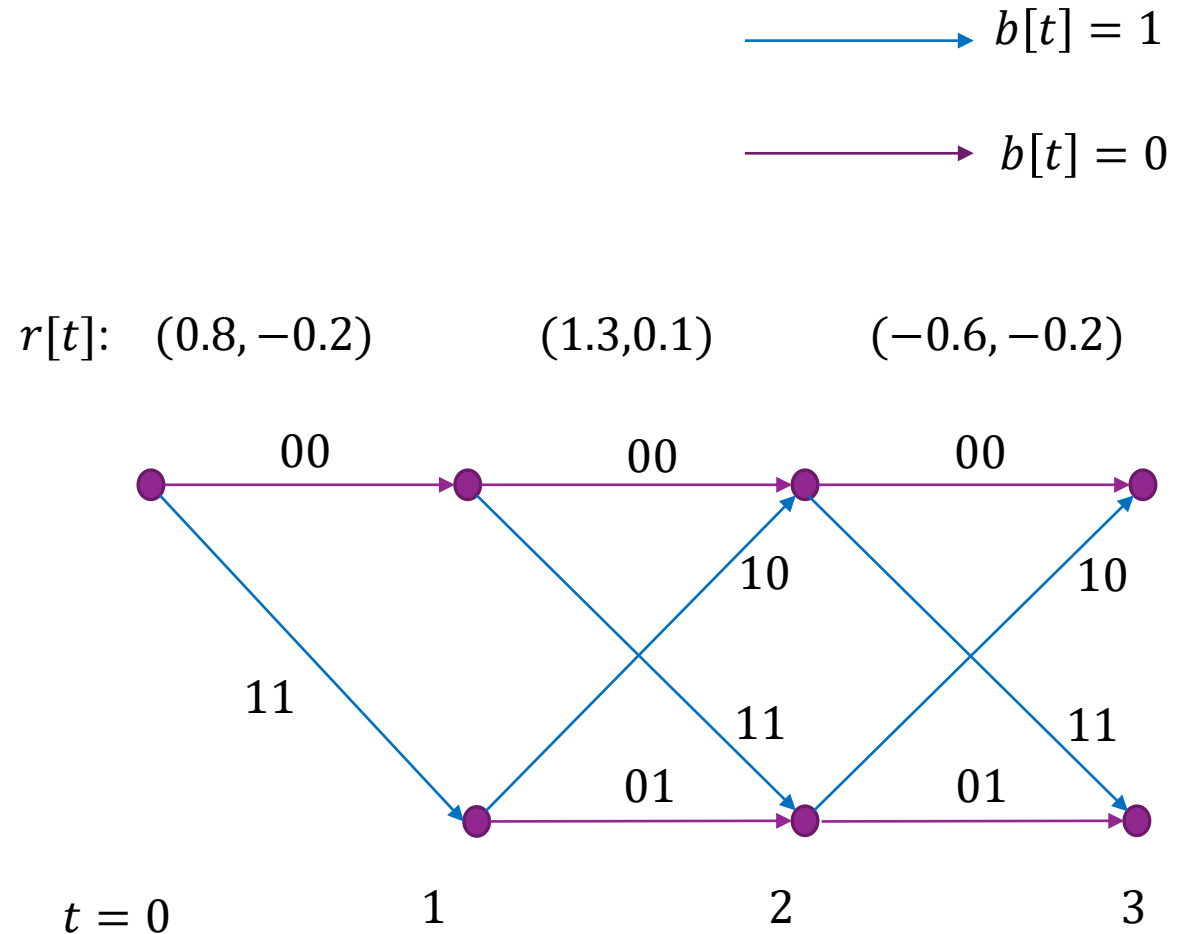□ Output: $c[t] = (c_1[t], c_2[t])$
  ◦ Shown on branches

□ RX symbols: $r[t] = (r_1[t], r_2[t])$
  ◦ Values shown above Trellis

□ Assume QPSK channel value function:

$$J(c) = \sum_t \sum_i r_i c_i$$

  ◦ Constant is ignored

$\longrightarrow \quad b[t] = 1$

$\longrightarrow \quad b[t] = 0$

$r[t]: \quad (0.8, -0.2) \qquad (1.3, 0.1) \qquad (-0.6, -0.2)$

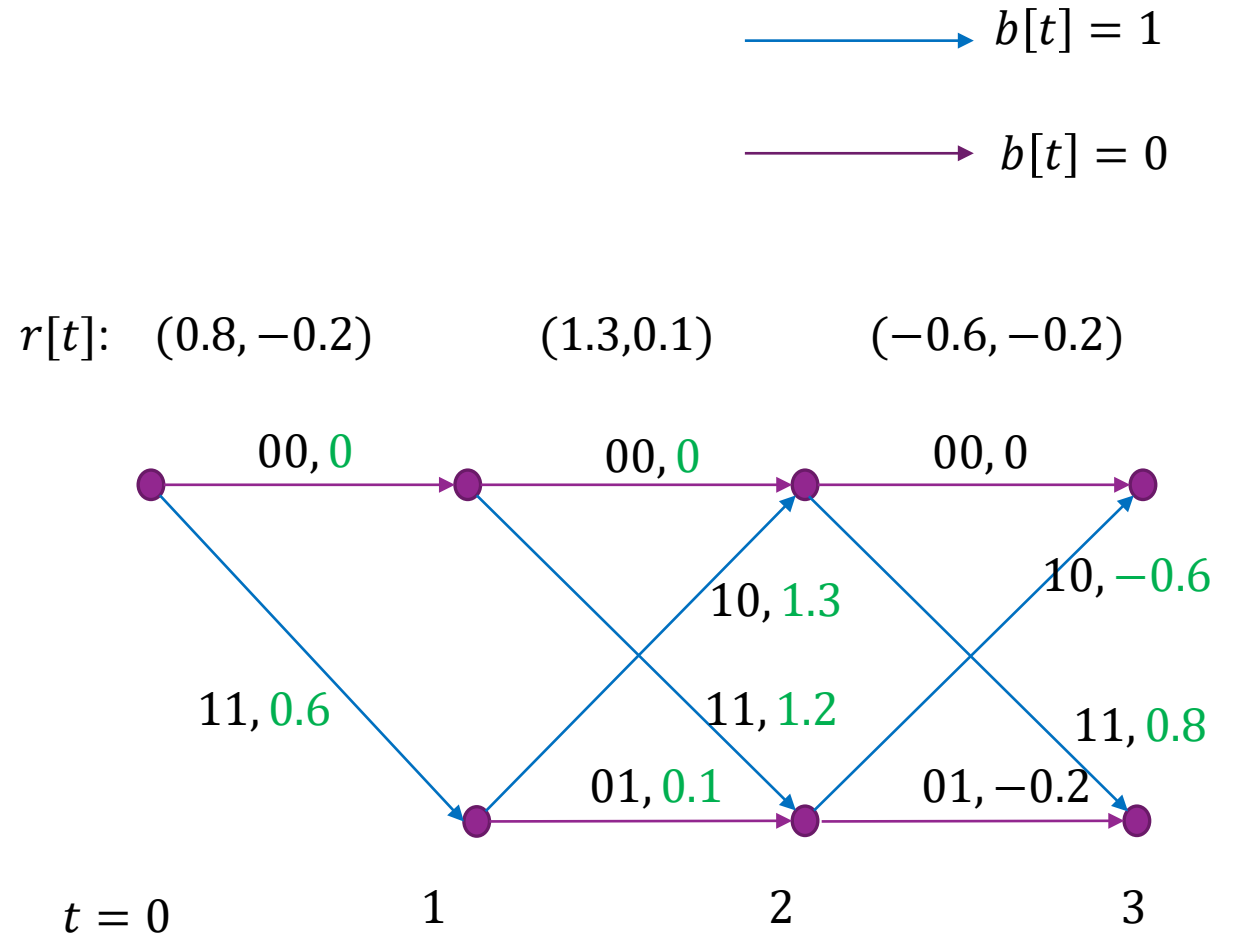$t = 0 \qquad\qquad 1 \qquad\qquad 2 \qquad\qquad 3$

Step 1: Compute Branch Metrics

☐ For QPSK channel, branch metric is

$$\mu_t(x_{t+1}, x_t) = \sum_i r_i[t] c_i[t]$$

☐ On each branch we have labeled:
(Output Bits, branch) = $(c[t], \mu_t)$

$\xrightarrow{\hspace{2cm}} b[t] = 1$

$\xrightarrow{\hspace{2cm}} b[t] = 0$

$r[t]$: $(0.8, -0.2)$ $(1.3, 0.1)$ $(-0.6, -0.2)$

00, 0   00, 0   00, 0

10, 1.3

10, -0.6

11, 0.6

11, 1.2

11, 0.8

01, 0.1   01, -0.2

$t = 0$ 1   2   3

Step 2: Find the Maximizing Path

□ For this simple Trellis, we can do this manually

□ Path is shown with red nodes 🔴

□ Solution:
  ◦ Coded sequence: (11), (10), (00)
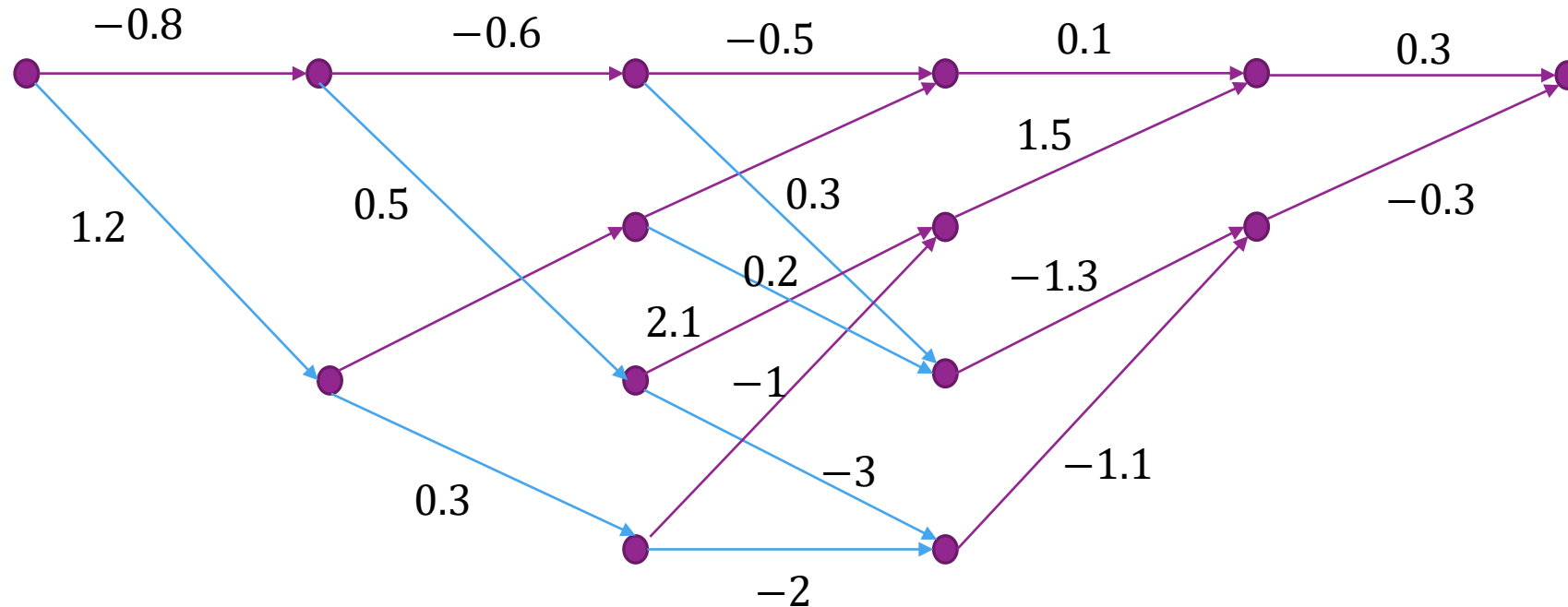 ◦ Bits: (1,1,0)

$b[t] = 1$

$b[t] = 0$
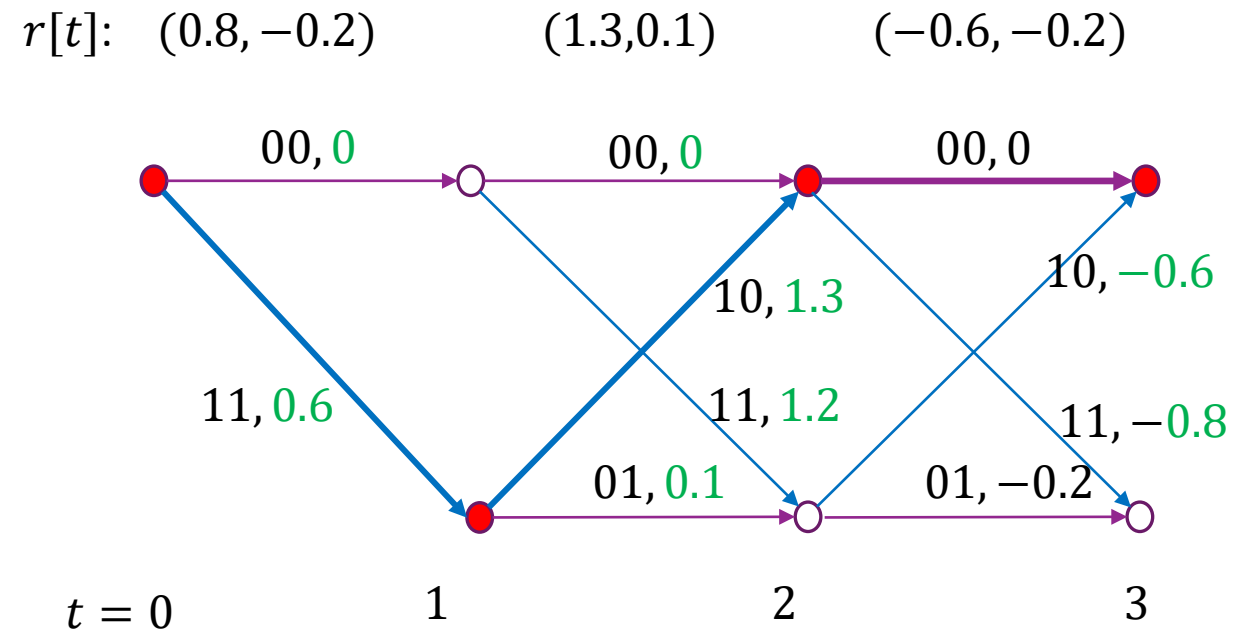
Outline

❑ Convolutional codes: encoding and representations

❑ Tree, trellis and state diagrams

❑ Decoding with branch metrics
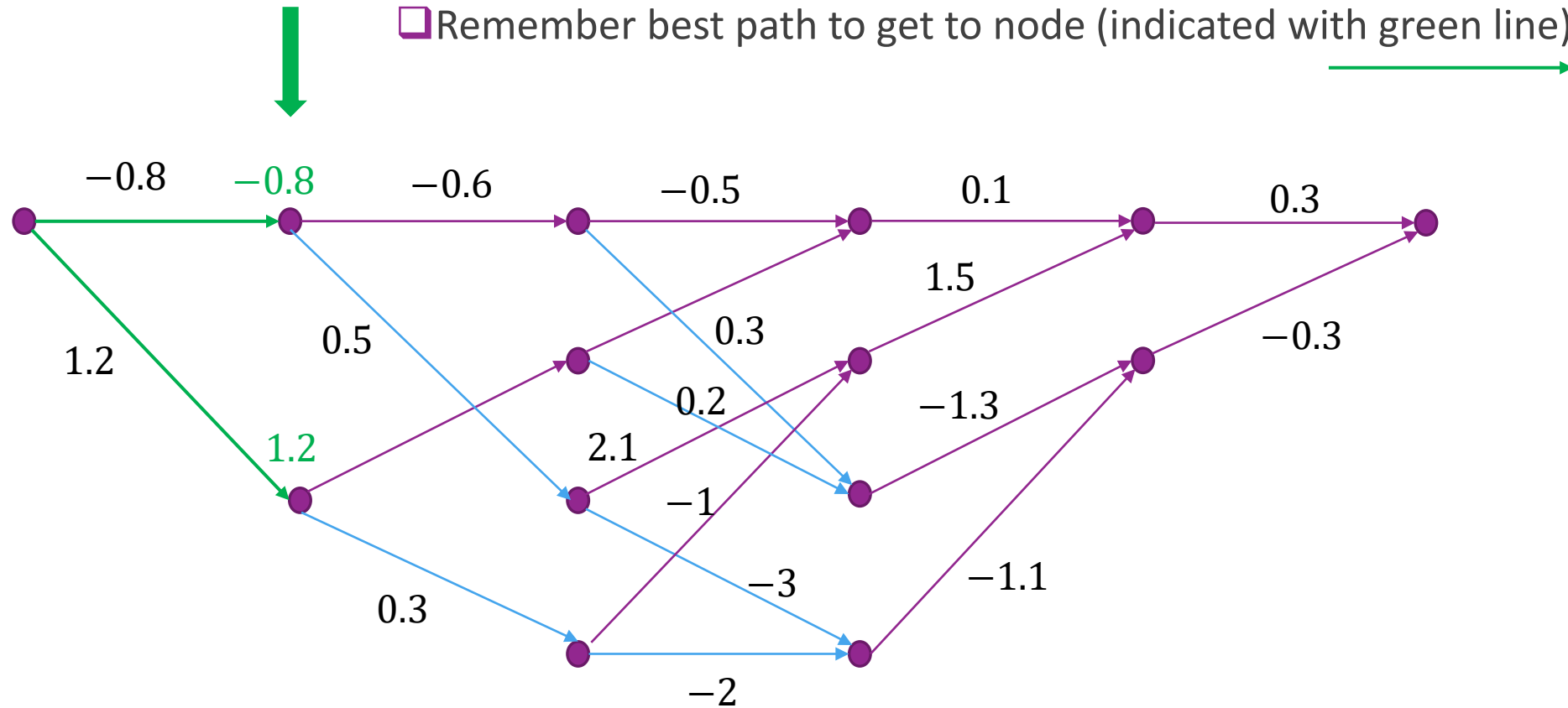
➡️ ❑ Viterbi decoding

Viterbi Algorithm by Example

❑Iterative solution for finding maximum path in a graph

❑Key idea: Find the max path one time step at a time

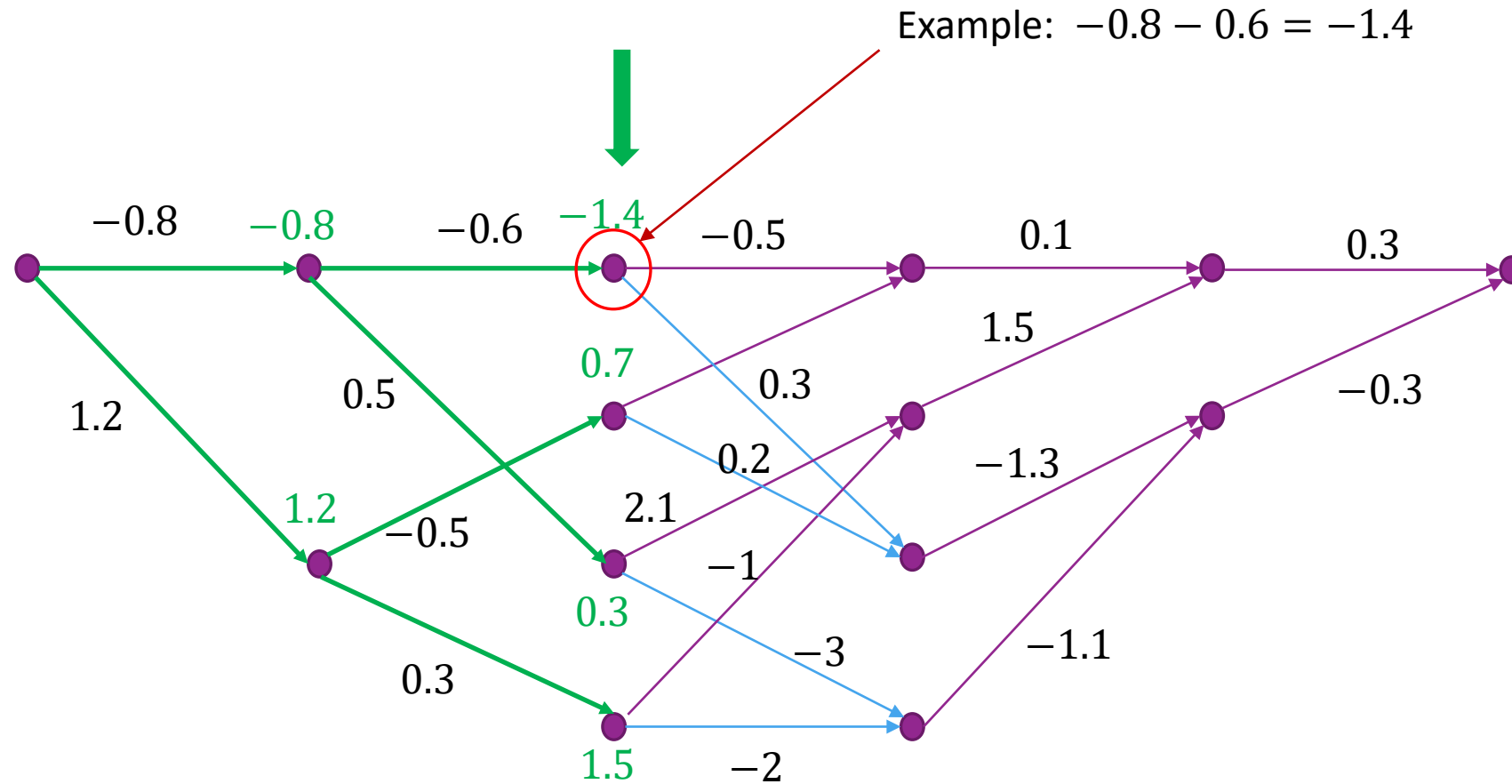❑We will illustrate by example: Shown are the branch metrics

Ex: Time 1

☐ Find total value to get to each node at time 1

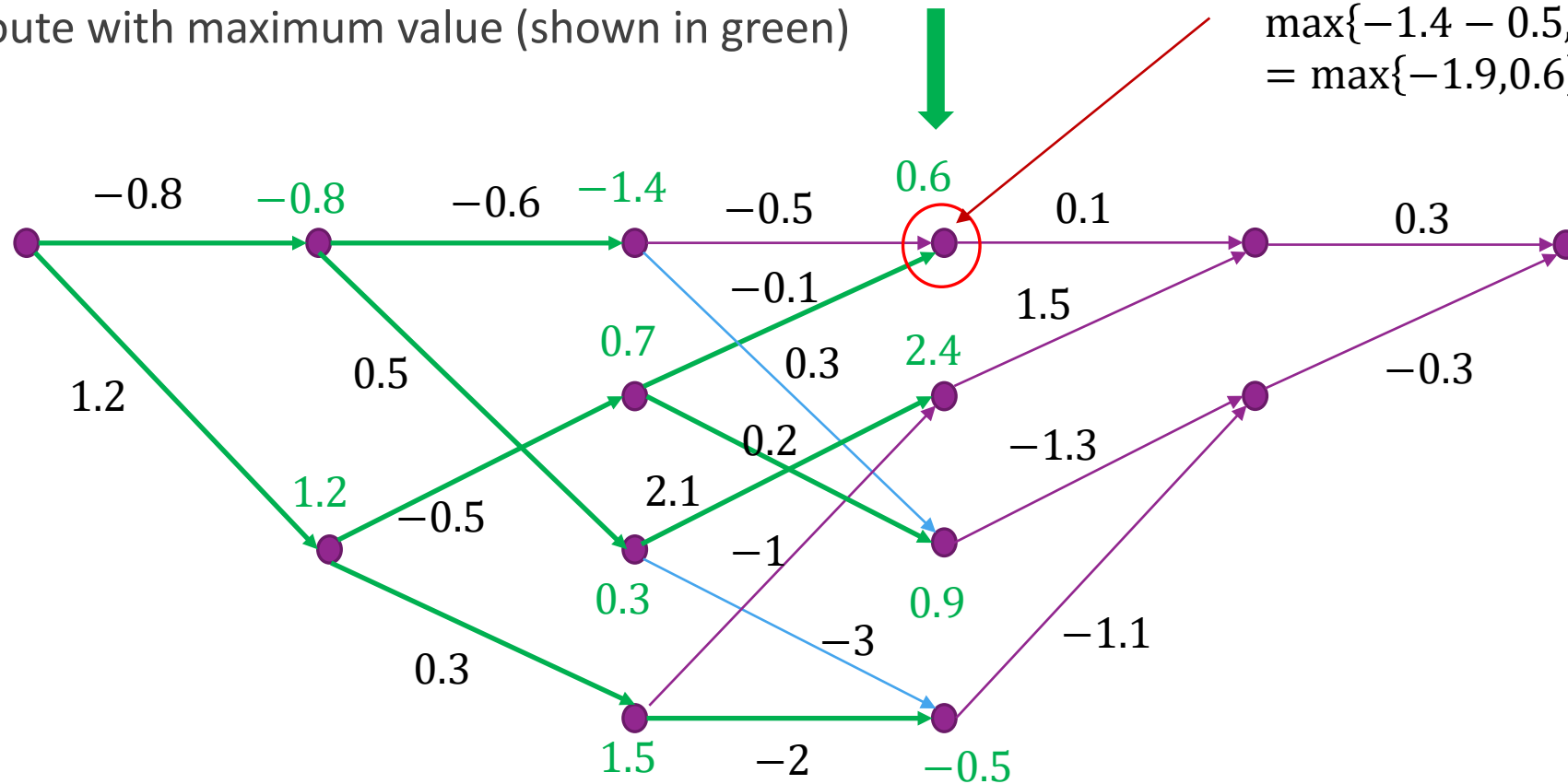☐ Remember best path to get to node (indicated with green line)
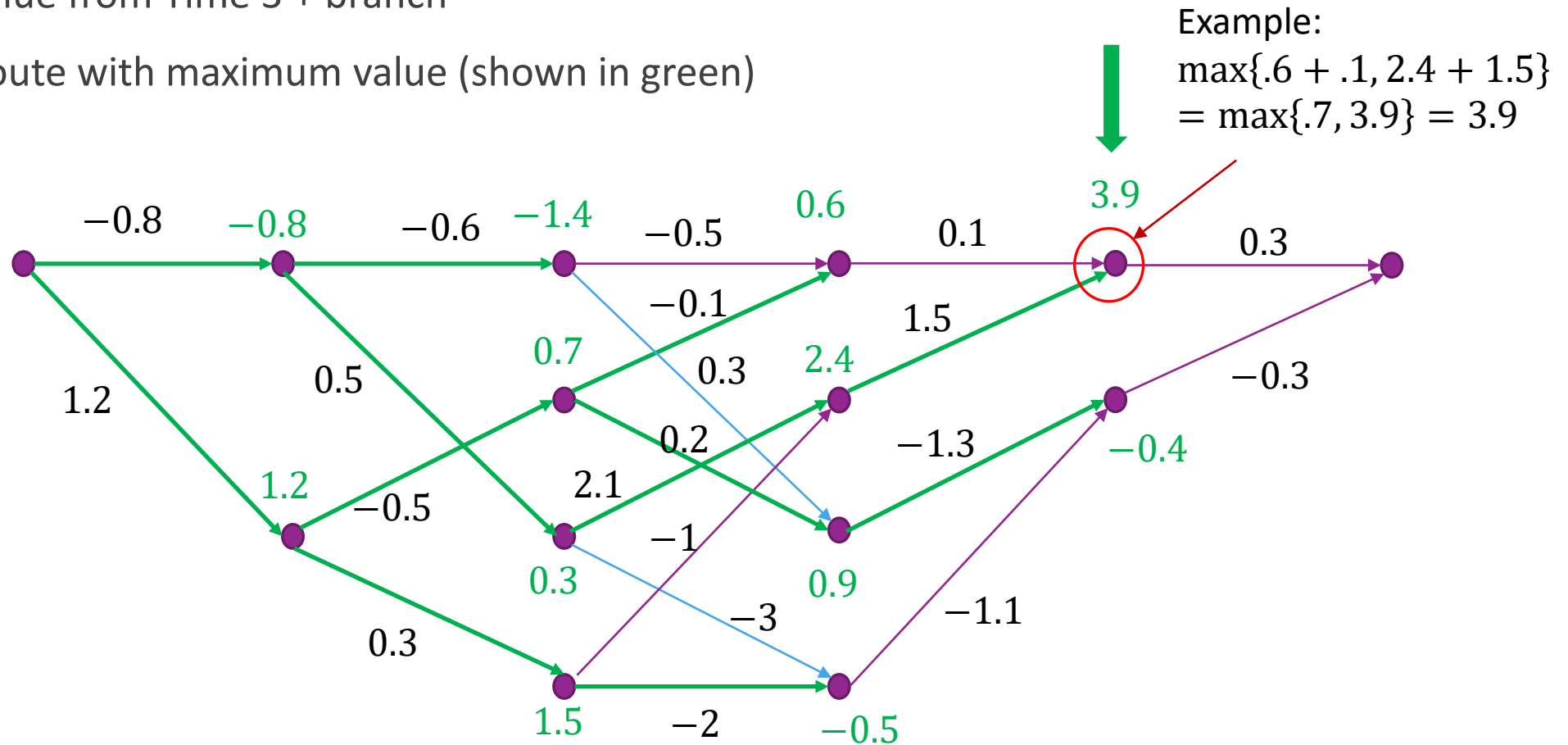
Ex: Time 2

☐ Add value from Time 1 + branch

Example: $-0.8 - 0.6 = -1.4$

Ex: Time 3

☐ Add value from Time 2 + branch

☐ Take route with maximum value (shown in green)

Example:
$$\max\{-1.4 - 0.5, 0.7 - 0.1\}$$
$$= \max\{-1.9, 0.6\} = 0.6$$

Ex: Time 4

□ Add value from Time 3 + branch

□ Take route with maximum value (shown in green)

Example:
$\max\{.6 + .1, 2.4 + 1.5\}$
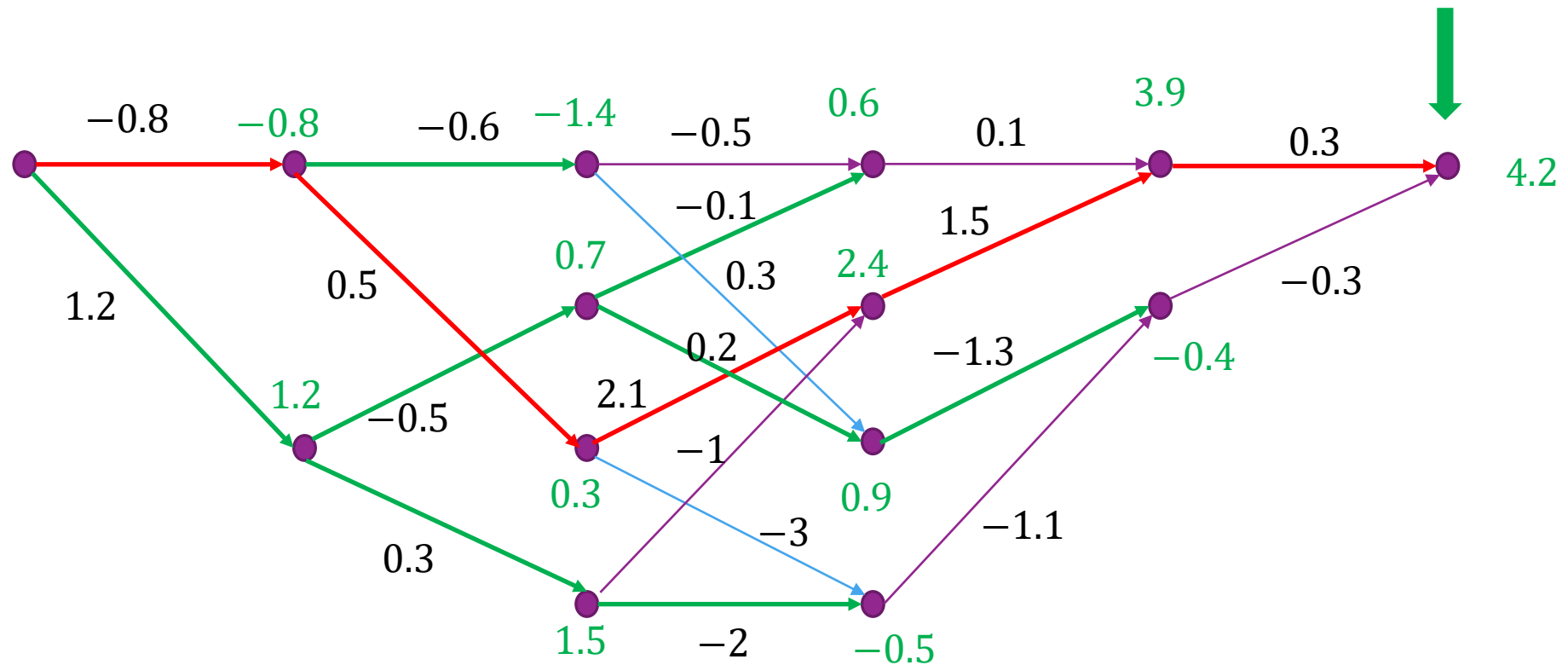$= \max\{.7, 3.9\} = 3.9$

Ex: Time 5

□ Add value from Time 4 + branch

□ Take route with maximum value (shown in green)
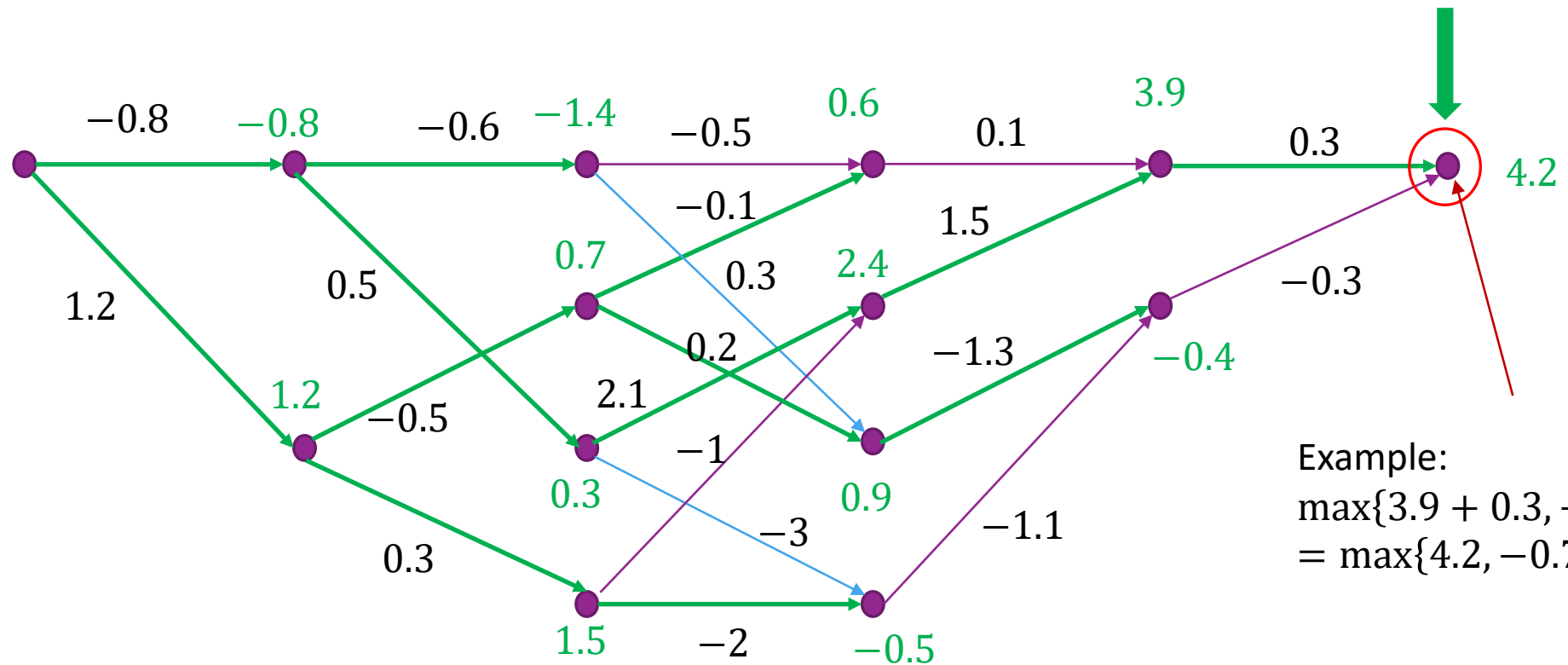
Example:
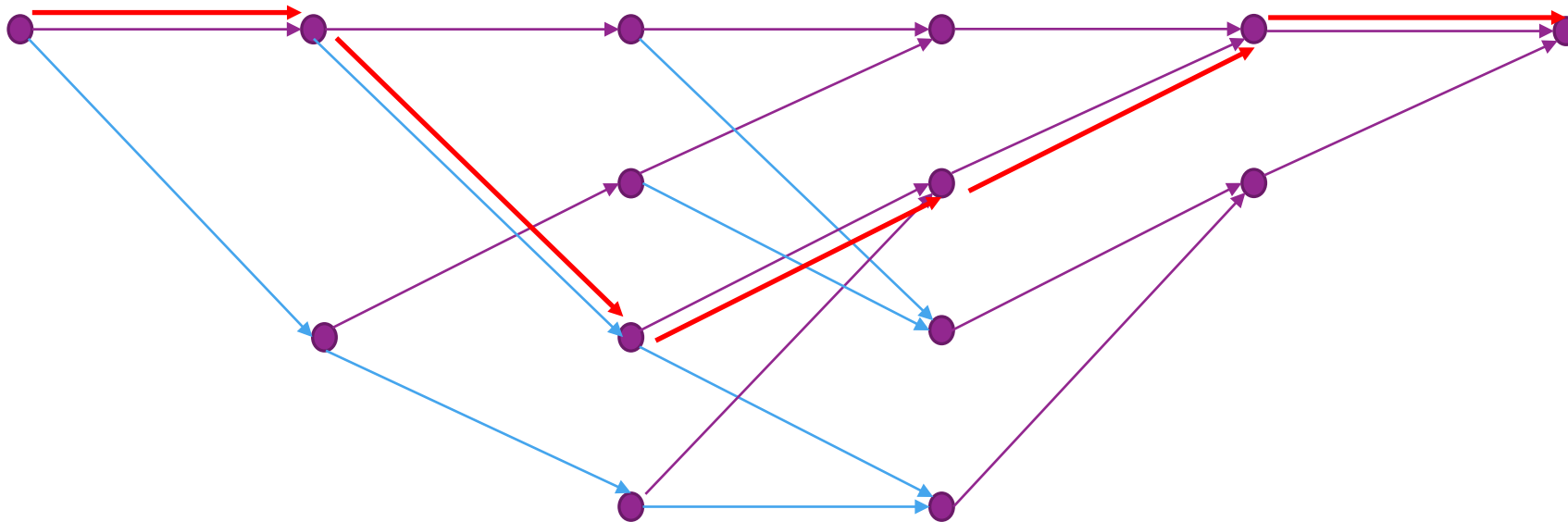$\max\{3.9 + 0.3, -0.4 - 0.3\}$
$= \max\{4.2, -0.7\} = 4.2$

Ex: Trace back path

☐ Trace back maximum path. Shown in red

Ex: Read off bits

☐ Read off the bits from the maximum path

☐ ML input bits = (0,1,0,0,0)

Complexity

❑Update of each node requires maxima over 2^k branches

❑There are $2^{k(K-1)}$ states, so complexity / time is $O(2^{kK})$

❑Total complexity is $O(T2^{kK})$

❑Storage is also $O(T2^{kK})$. (From the paths)

❑Summary:
◦ Viterbi algorithm is linear in block length.
◦ Can have very long block lengths (often T in the 1000s)
◦ But, complexity is exponential in constraint length
◦ Practical decoders limited to $K = 7$ or $K = 9$.
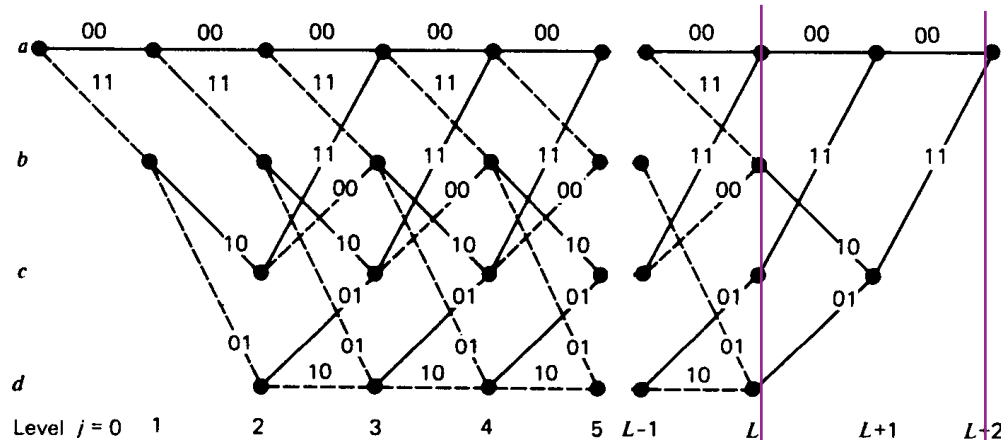
Terminating the Trellis

Figure 11.15 Trellis for the convolutional encoder of Fig. 11.13a.

During tail bits only paths with zero bit inputs.

- ❑ Recall tail bits are zero:
  $$\boldsymbol{b}(L) = \cdots = \boldsymbol{b}(L + K - 2) = 0.$$

- ❑ Limits the paths at the end of the trellis

- ❑ Viterbi algorithm should only be done on the zero path.

- ❑ Very important to not forget the bits at the end.

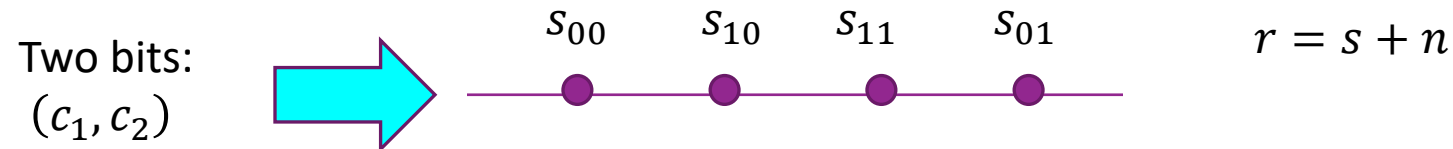- ❑ Otherwise, final bits are not protected.

Pruning the Path Memory

❑In current algorithm, path $P_t(x_t)$ grows to full block length.

❑Adds storage: Storage is O$\left(T2^{kK}\right)$. Linear in *T*

❑Adds delay. No bits can be determined until code is fully decoded.

❑Many practical implementations:
- Store some finite length δ of each surviving path.
- Can make decision on bit input $\boldsymbol{b}(t)$ after $\mathbf{r}(t + \delta)$.
- Rule of thumb: Good performance if $\delta \geq 5K$.
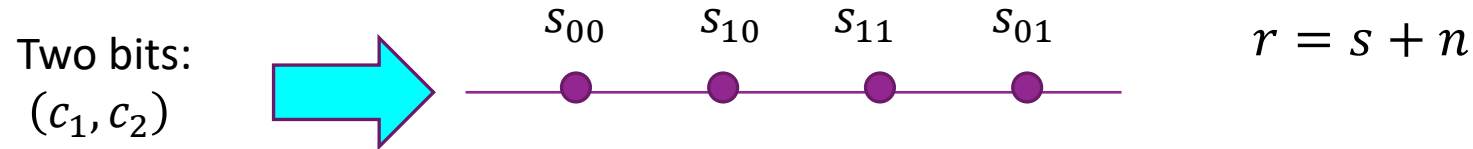
Rate Matching Convolutional Codes

❏Convolutional codes have limited rates, usu. ½ or 1/3.

❏Obtain other rates through
  ◦ Puncturing: Remove coded bits to increase rate
  ◦ Repetition: Repeat coded bits to decrease rate

❏Puncture / repeat pattern is important (see Proakis)
  ◦ Try to spread out modified bits

❏For punctured bits, set corresponding LLRs to zero
  ◦ Viterbi decoder just ignores those bits

❏For repeated bits, add the corresponding LLRs
  ◦ Viterbi decoder will increase confidence on that branch

High Order Constellations

❑Higher order constellations (eg. 16- or 64-QAM)

❑Each constellation is a point is a function of multiple bits.

❑Likelihood does not factorize
- Each symbol r(t) depends on multiple bits

❑Example 4-PAM (or one dimension of 16-QAM):
- Each symbol likelihood depends on two bits: $p(r|c_1, c_2)$

Two bits:
(c_1, c_2)

s_{00} s_{10} s_{11} s_{01}

$r = s + n$

High Order Constellations

Two bits: (c_1, c_2) ➡️ s_{00} s_{10} s_{11} s_{01}
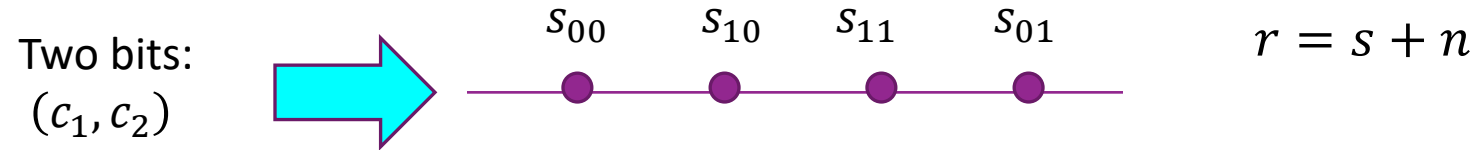
$$r = s + n$$

☐ To create LLRs for individual bits use total probability rule:

$$p(r|c_1) = \frac{1}{2}\big(p(r|c_1, c_2 = 0) + p(r|c_1, c_2 = 1)\big)$$
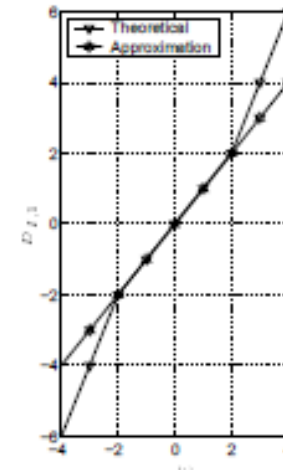
☐ Resulting bitwise LLR:

$$LLR \text{ for } c_1 = \log\frac{p(r|c_1, c_2 = 1,0) + p(r|c_1, c_2 = 1,1)}{p(r|c_1, c_2 = 0,0) + p(r|c_1, c_2 = 0,1)}$$
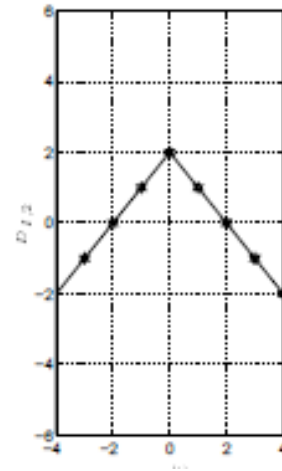
High Order Constellations

Two bits: (c_1, c_2)

s_{00} s_{10} s_{11} s_{01}

$r = s + n$

❑ LLRs can have irregular shapes

❑ Not simple linear function as in BPSK / QPSK case

❑ Often use approximations

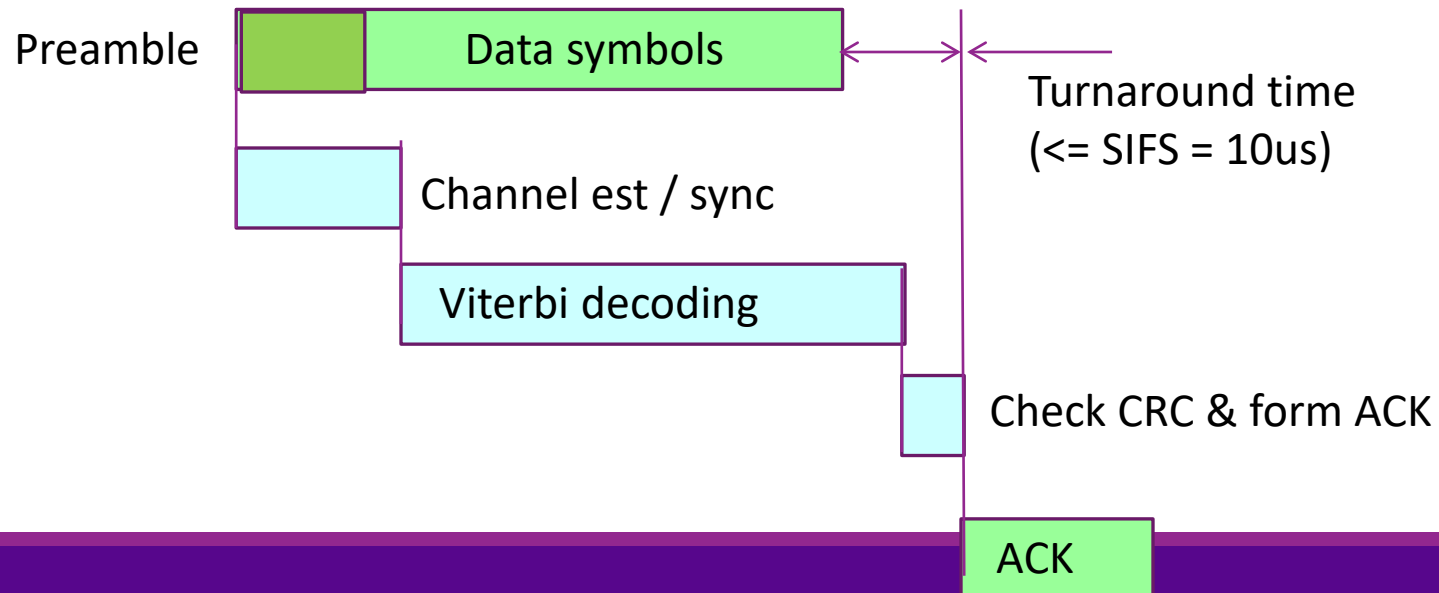❑ More info: Caire, Taricco and Biglieri, "Bit-Interleaved Coded Modulation," 1998.

LLR for c2   LLR for c1

Convolutional Codes in WiFi

❑ 802.11 uses R=1/2 K=7 code.

❑ Length adjusted to packet size

❑ Higher rates (R=2/3 and ¾) achieved through puncturing

❑ Enables decoding as data arrives for ACK fast turnaround

Preamble

Data symbols

Turnaround time
(<= SIFS = 10us)

Channel est / sync

Viterbi decoding

Check CRC & form ACK

ACK

Convolutional Codes in LTE

❑Convolutional codes in LTE used for:
- ◦ Control channels (payload typ 20-40 bits +CRC), and
- ◦ Short (< 128 bit) data frames

❑Larger payloads encoded with turbo codes (discussed later)

❑Uses rate=1/3 base convolutional code with K=7.

❑Higher rates achieved via puncturing

❑Control channels use a sophisticated technique called tail biting to reduce loss on the tail bits
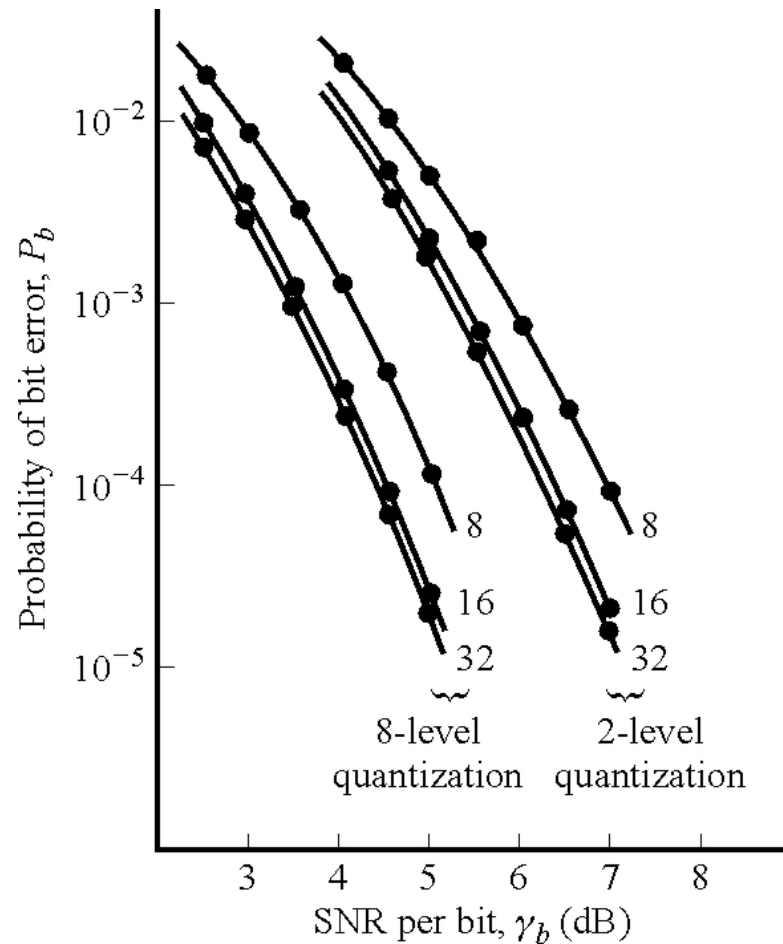
Decoding Performance
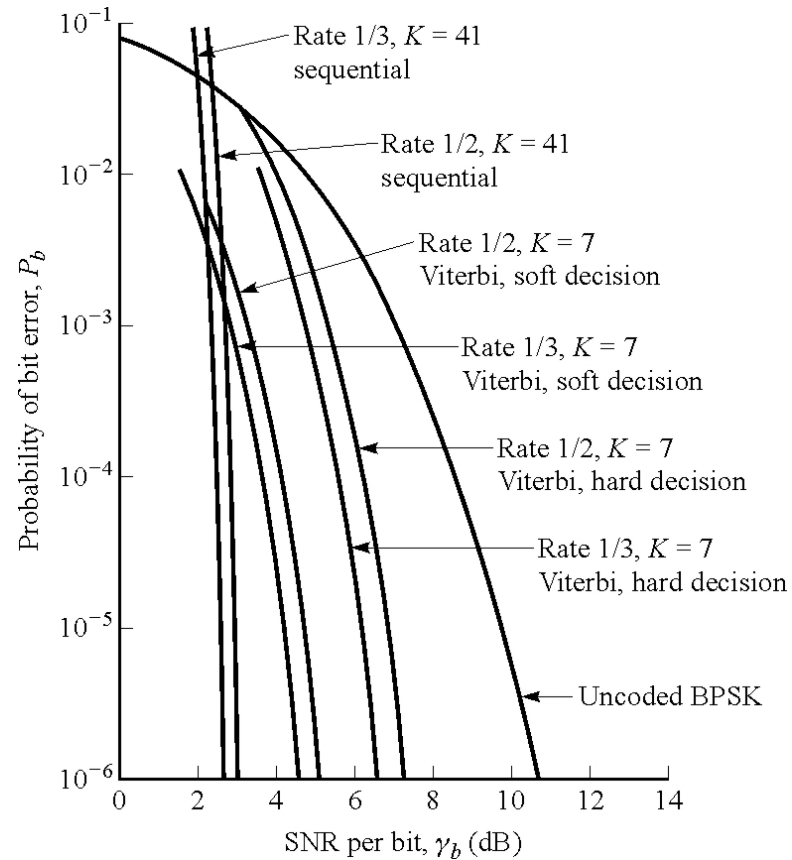
❑Hard decision loses approximately 1.5 to 2 dB

❑Constraint length K=7 is sufficient for very sharp error performance

Different Constraint Lengths
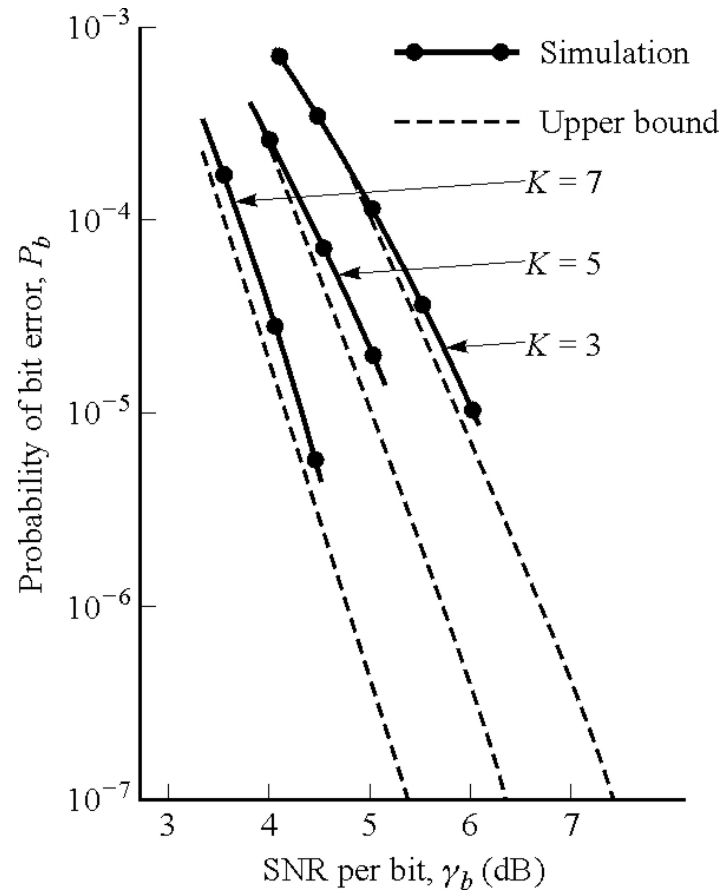
- ❑Approximate 1 dB improvement between K=3, 5 and 7

- ❑Higher constraint lengths become computationally intractable

- ❑Recall decoding complexity is exponential in K

LLR Quantization

- ❑ Minimal gains after 16 bit quantization of branch metrics

- ❑ Recall HD is equivalent to 1 bit quantization

- ❑ Most commercial implementations use 6-bit LLRs

MATLAB Convolution Tools

Demo Convolutional Code

The Viterbi decoder built in the in-class exercise is very slow. Fortunately, MATLAB has excellent tools for encoding and decoding

- Perform convolutional encoding with the MATLAB comm toolbox commands
- Extract LLRs from the qamdemod command
- Perform convolution decoding from the soft LLRs
- Measure the block error rate as a funciton of the SNR

We illustrate with a simple convolutional coder over an AWGN channel. We use a standard constraint length K=7 code:

```
% Create the convolutional encoder and decoder with constraint length K=7
trellis = poly2trellis(7,[171 133]);
convEnc = comm.ConvolutionalEncoder('TrellisStructure', trellis, 'TerminationMethod','Terminated');
convDec = comm.ViterbiDecoder('TrellisStructure', trellis, 'TerminationMethod','Terminated');

% Modulation
bitsPerSym = 2;    % QPSK
M = 2^bitsPerSym;

% Number of information bits per block
nbits = 1000;
```

❑MATLAB comm toolbox
  ◦ General convolution encoders & decoders
  ◦ Efficient implementation
  ◦ Excellent for testing

❑See demo

In-Class Exercise

Building a Simple Viterbi Decoder

We conclude with building a simple decoder. This decoder is slow but illustrates the ideas.

To start, let's encode some random data.

```
% Number of bits
nbits = 100;

% TODO:  Generate random bits
%    b = ...


% TODO:  Create output bits.
%    c = ...
```