Lecture 09.

Note Title 2/8/2012

* Viterbi Algorithm (VA)

An efficient ML decoder for the convolutional codes.

Goal: $\arg\max_{\overline{p}} \prod_{t=1}^{T} f_t(\overline{p[t]})$

Step 1: Compute the metric values for each branch / path segment based on the observation $\overline{y_{obs}}$

Step 2:

$$a_S[0] = \begin{cases} 1 & \text{if } S = 00 \\ 0 & \text{for all other } S \end{cases}$$

$$a_S[t+1] = \max_{S' \text{ that goes to } S} \left\{ a_{S'}[t] \, m_{S',S} \right\}$$

Step 3:

$$\max_{\overline{p}} \prod_{t=1}^{T} f_t(\overline{p[t]}) = \max_{S} a_S[T]$$

The above procedure describes how to find $\max_R \prod_{t=1}^{T} f_t(\overline{p[t]})$. But we are more interested in

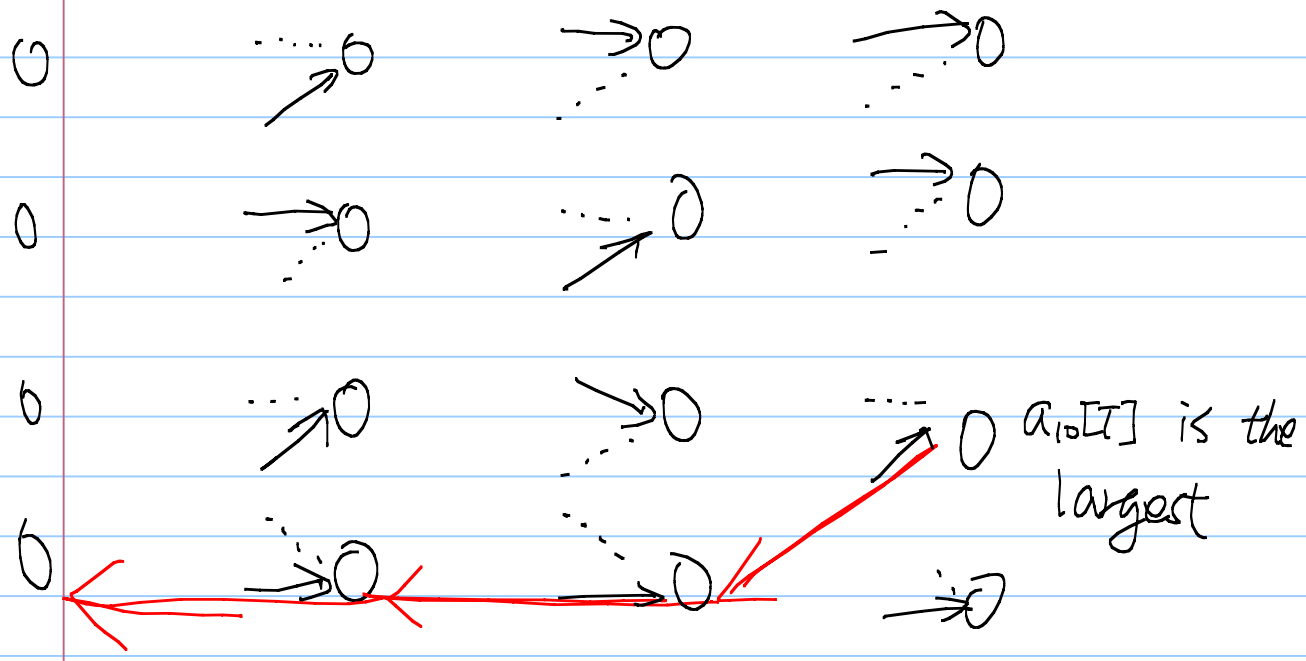
$$\boxed{\operatorname{argmax}_R \prod_{t=1}^{T} f_t(\overline{p[t]}).}$$

which tells us the most likely codeword

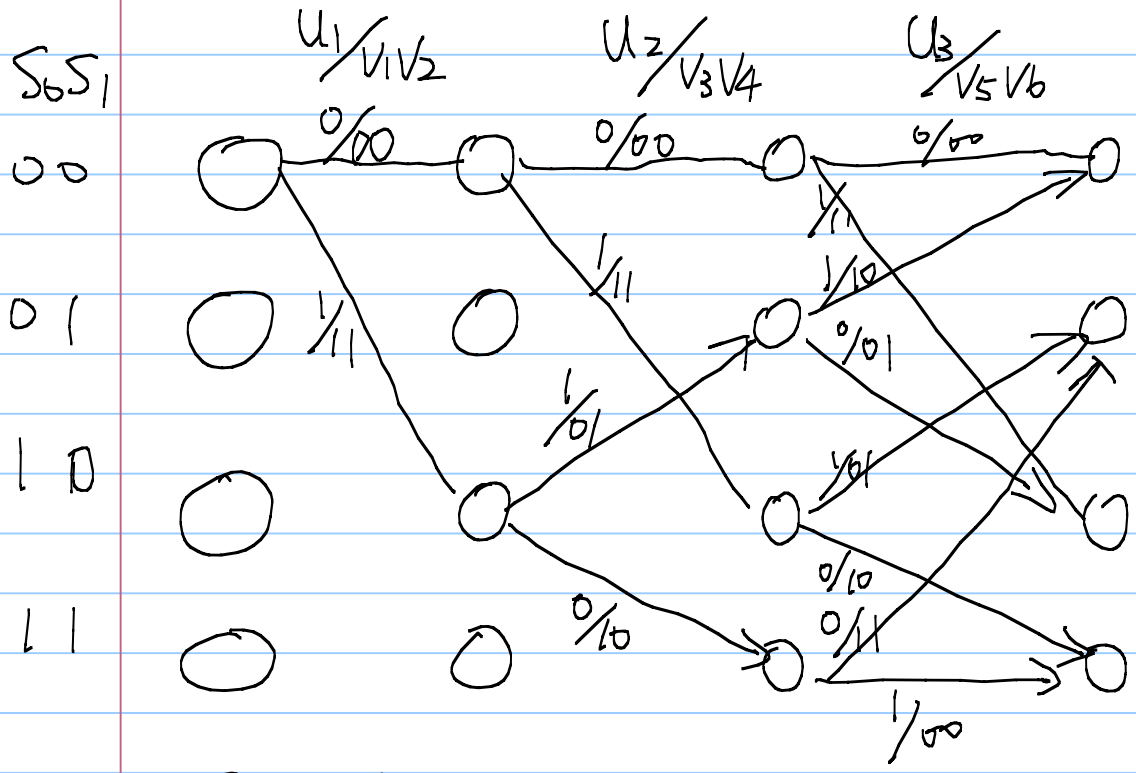
Q: How to find the most likely codeword?

Ans: Similar to the Dijkstra algorithm.

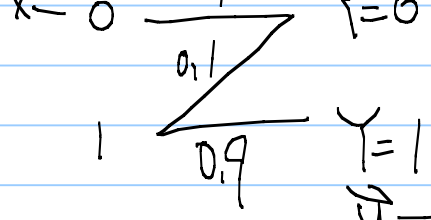
We memorize the choices of the max operations and then trace it backward from the destination back to the origin.


$a_{10}[i]$ is the largest

An illustrating example based on the running example code. $T=3$



Channel: Z-channel

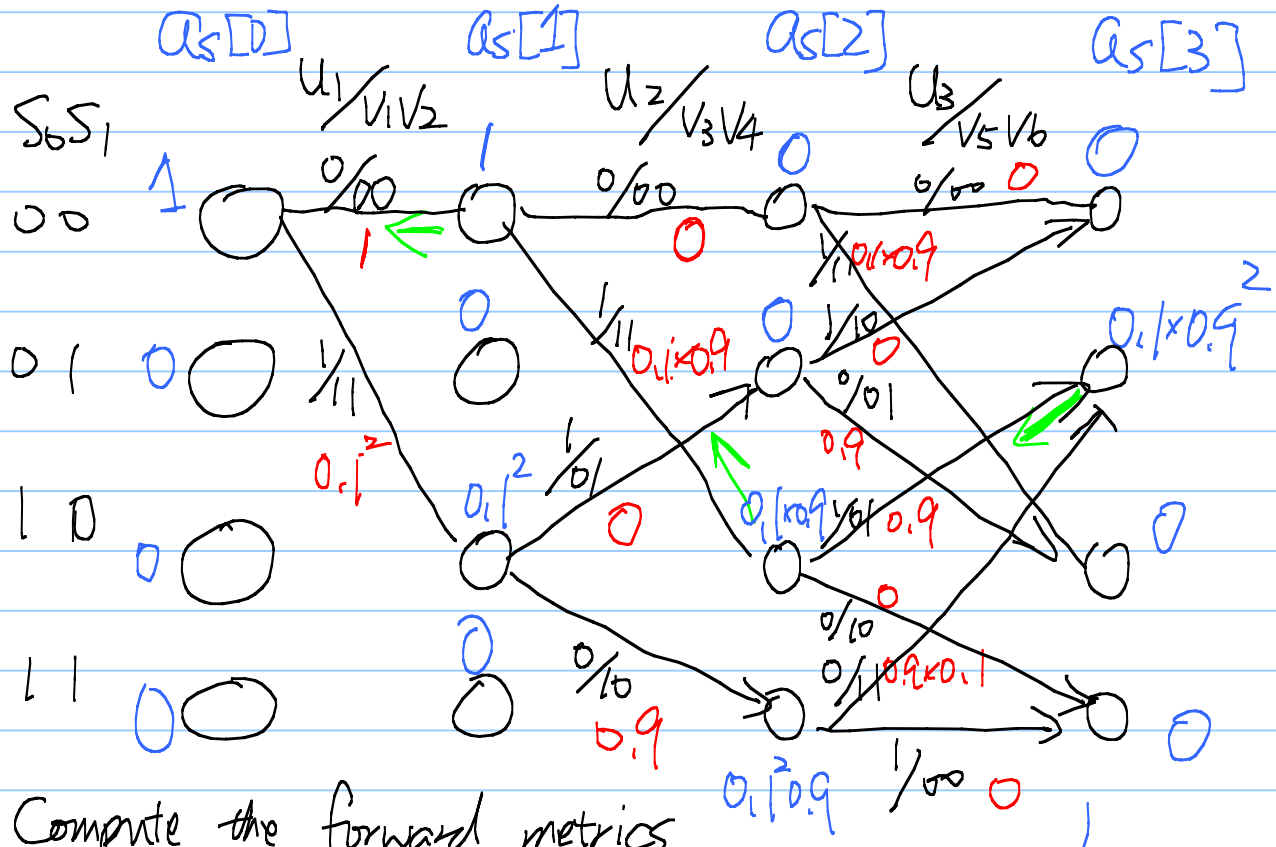


$\underline{y}=$ The observation is 00 10 01

Q: Use the VA to find the optimal codeword.

Ans: ① Initialize the partial objective function $f_t(\cdot)$ (or the metric for each segment).

Observation: 00 10 01



② Compute the forward metrics

$$a_{00}[3] = \max(\underline{0\times0}, \underline{0\times0}) = 0$$

$$a_{01}[3] = \max(\underline{0.1\times0.9}\times0.9, \underline{0.1^2\times0.9}\times0.1\times0.9)$$

$$= 0.1\times0.9^2$$

$$a_{10}[3] = \max(\underline{0\times0.1\times0.9}, 0\times0.1\times0.9) = 0$$

$$a_{11}[3] = \max(\underline{0.1\times0.9\times0}, 0.1^2\times0.9\times0) = 0$$

③ Backward trace

The most likely codeword

$\vec{V} = 00\ 11\ 01$

Summary of convolutional codes:

① Encoding: the info bits steer the path

② Decoding: Iteratively compute the maximizing partial paths & then trace it backward to find p^*

③ The use of the path metric $f_t(\cdot)$ is an abstraction of the prob quantities. There is no need to worry about the prob meaning of $a_s[t]$

Several variants of the Viterbi decoder

① Multiplication is hard, addition is easy

$$\max_p \prod_{t=1}^{T} f_t(\overline{p[t]}) \equiv \max_p \sum_{t=1}^{T} \log f_t(\overline{p[t]})$$

We only need to change the metrics used in the VA. & the corresponding update rule

$$a'_{00}[0] = \log(a_s[0]) = \log(1) = 0, \quad a'_s[0] = \log(0) = -\infty \text{ for } s \neq 00.$$

$$m'_{s',s} = \log(m_{s',s}) = \log\left(P_{y_{t-1}y_t|v_{t-1}v_t}\right)$$

$$a'_s[t+1] = \max_{s' \text{ that reaches } s} \left\{ a'_{s'}[t] + m'_{s',s} \right\}$$

The new VA maximizes

$$\max_{p} \sum_{t=1}^{I} \log f_t(\overline{p[t]})$$

② Many people prefer minimization over maximization

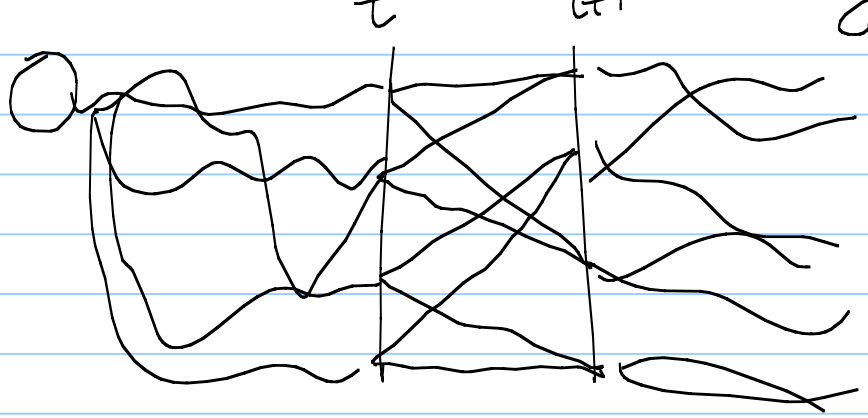
$$\Rightarrow m''_{s',s} = -m'_{s',s} = -\log\left(P_{y_{2t-1},y_{2t}|V_{2t-1},V_{2t}}\right)$$

$$a''_s[0] = \begin{cases} 0 & \text{if } s=0 \\ \infty & \text{otherwise} \end{cases}$$

$$a''_s[t+1] = \min_{s' \text{ that reaches } s} \left\{ a''_{s'}[t] + m'_{s',s} \right\}$$

③ VA: One-way parsing the trellis structure.

Two-way VA: Parse the trellis from both ends. (Trading the hardware complexity for decoding speed.)



$\frac{1}{2}$ the decoding time

Forward direction

$a_{00}[0] = 1.$ $\quad a_S[0] = 0$ if $S \neq 00$

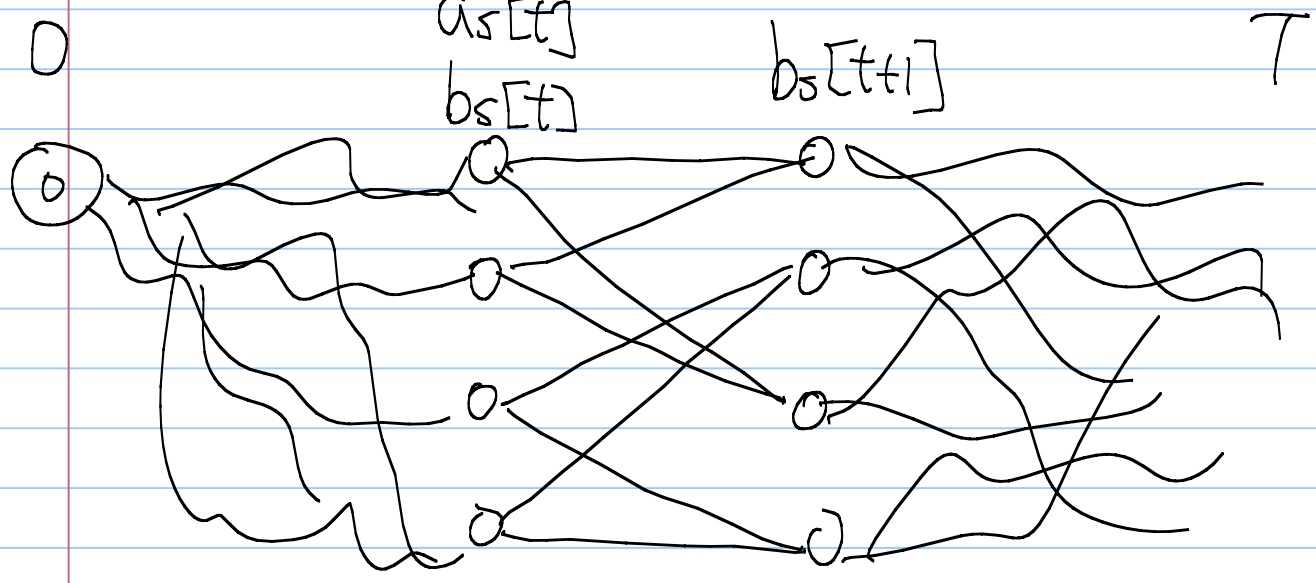
$$a_S[t+1] = \max_{\substack{S' \text{ that} \\ \text{reaches} \\ S}} \{ a_{S'}[t] \cdot m_{S',S} \}$$

Backward direction

$b_S[T] = 1$ for $S = 00, 01, 10, 11$

$$b_S[t-1] = \max_{\substack{S' \text{ that is} \\ \text{reachable by } S}} \{ b_{S'}[t] \cdot m_{S,S'} \}$$

When the forward & backward computations meet



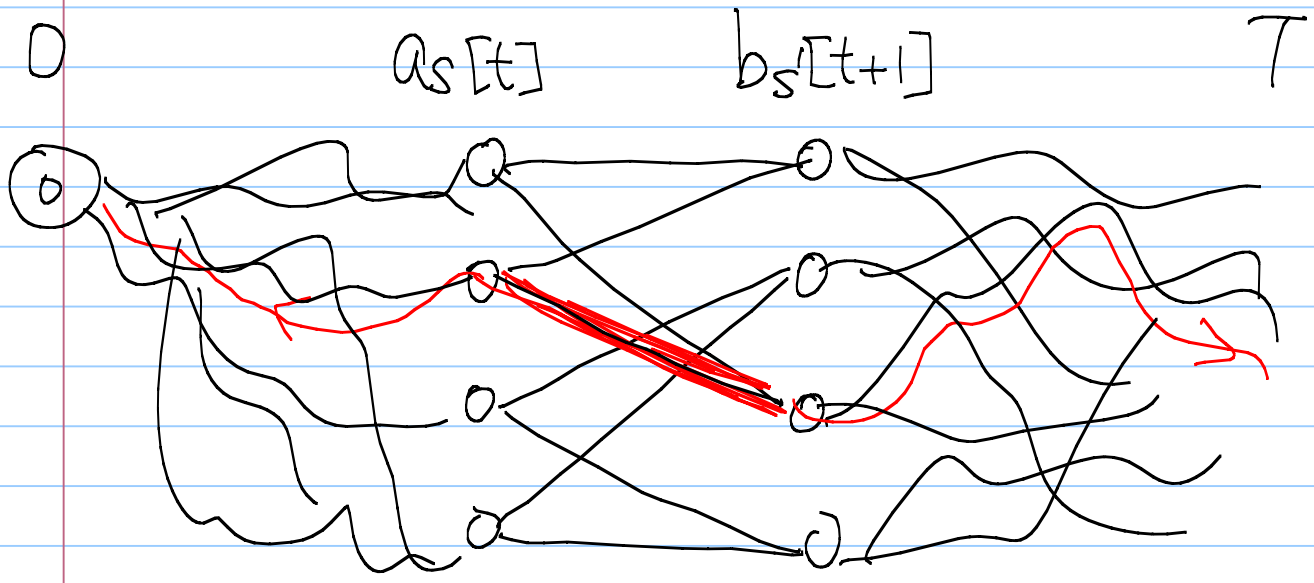
$$\max_{p} \prod_{t=1}^{T} f_t(\overline{p[t]}) = \max_{s} \left\{ a_s[t]\ b_s[t] \right\}$$

total $\leq |\text{States}|^2$ terms.

Q: How to find the $\operatorname{argmax}_{p} \prod_{t=1}^{T} f_t(\overline{p[t]})$

A: Record the choices of the max operations during the computation of $a_s[t+1]$ & $b_s[t-1]$

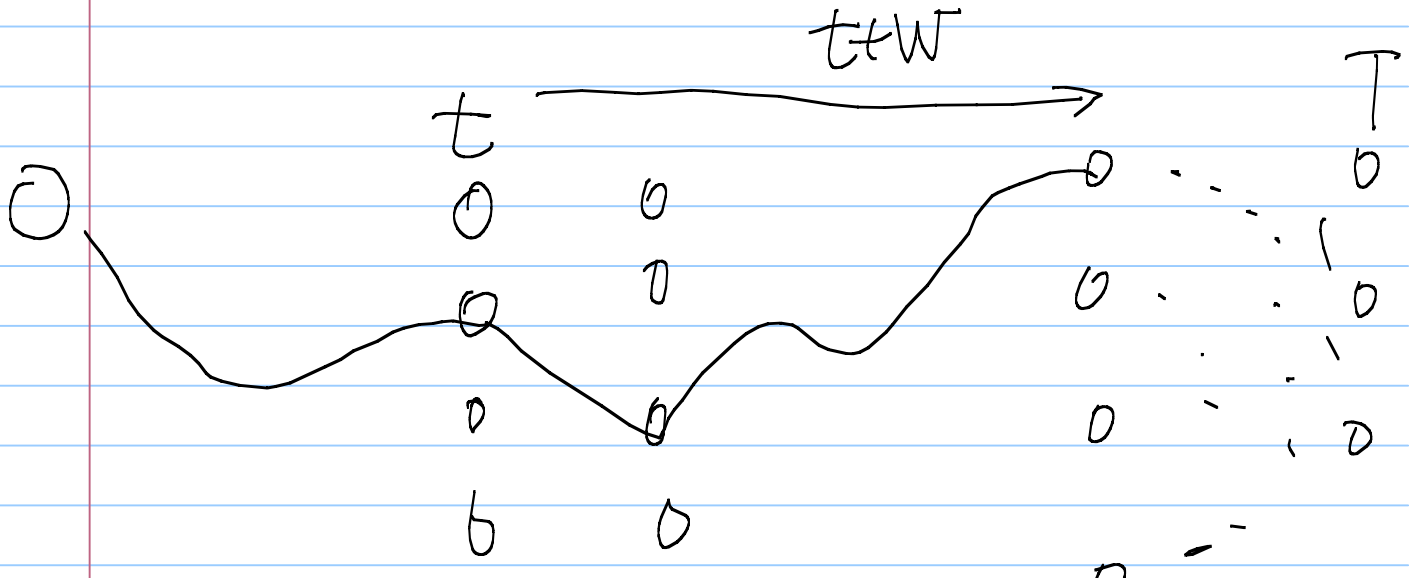
Then trace the path both forward & backward.



④ VA with delay constraints (Window-based VA)

One drawback of the VA is that it needs to parse the entire trellis structure. ⇒ Need to wait for all $y_1, y_2 \cdots, y_n$ to arrive before decoding ⇒ longer delay

One solution is to sacrifice the performance by focusing on some window



Use the data we have had until $t+W$ to decide the best partial path, & the best V_{t-1}, V_t values. Repeat it for different t.

Generally the window size is chosen as

$5 \times$ (the number of delay units)

In some sense, The window-based VA solves a different maximization problem. Q: Write down the new objective function of a window-based VA for our running example

Ans: Each $(\hat{v}_{2t-1}\ \hat{v}_{2t})$ is the t-th coordinate of the following maximization problem

$$\left(\arg\max_{v_1 v_2 \cdots v_{2(t+W)}} \prod_{s=1}^{(t+W)} P_{Y_{2s-1}Y_{2s}|V_{2s-1}V_{2s}}\left(y_{2s-1}, y_{2s} \middle| v_{2s-1}, v_{2s}\right)\right)$$

$$= \arg\max_{v_1, v_2 \cdots v_{2(t+W)}} P_{\vec{Y}_1^{2t+W}|\vec{V}_1^{2t+W}}\left(\vec{y}_1^{2t+W} \middle| \vec{v}_1^{2t+W}\right)$$

the decision based on partial observation

⑤ Use the VA as a <u>minimum distance decoder</u>

$$\text{Viterbi} \equiv \max_{\bar{p}} \prod_{t=1}^{T} f_t(\overline{p[t]})$$

$$\equiv \min_{\bar{p}} \sum_{t=1}^{T} h_t(\overline{p[t]})$$

where $h_t(\cdot) = -\log(f_t(\cdot))$

We can choose $h_t(\cdot)$ to be the distance function instead of the likelihood

$$h_t(\cdot) = \left| \overline{V[t]} - \overline{y_{obs}[t]} \right|$$

The VA thus becomes a <u>minimum distance decoder</u>.

Example: ① For binary symmetric channels.

$h_t(\cdot)$ is the Hamming distance

② For binary-input Laplacian channels

$h_t(\cdot)$ is the Euclidean distance

③ For binary-input white Gsn channels.

$h(t) = (\vec{x} - \vec{y})^2$ is the square error.

Variants: ① product $\xrightarrow{\log}$ sum ② minimization ③ 2-way VA. ④ Window-based scheme ⑤ Use it as a channel-model-indep. min distance decoder

⑥ Taking into account when we have non-uniform prior distribution.

Example: Suppose now that each codeword is not equally probable

$$P(\vec{V}=\vec{v}) \propto (q)^{\#\text{ of 1s}} (1-q)^{\#\text{ of 0s}}$$

Q: How to use the VA to find the MAP decoder?

Ans: Our goal is to find

$$\underset{\overline{p}}{\operatorname{argmax}} \left(\prod_{t=1}^{T} P_{\overline{y[t]}|\overline{p[t]}}(\overline{y_{obs}[t]}|\overline{p[t]}) \cdot q^{\#\text{ of 1s}} (1-q)^{\#\text{of 0s}} \right)$$

$$\equiv \underset{\overline{p}}{\operatorname{argmax}} \prod_{t=1}^{T} \left(P_{\overline{y[t]}|\overline{p[t]}}(\overline{y_{obs}[t]}|\overline{p[t]}) q^{\#\text{ of 1s in } \overline{p[t]}} (1-q)^{\#\text{ of 0s in } \overline{p[t]}} \right)$$

$$\equiv \underset{\overline{p}}{\operatorname{argmax}} \prod_{t=1}^{T} f_t'(\overline{p[t]}) \quad **$$

The only change is thus to initialize the $f_t(\cdot)$ function in a different way

* The VA is an efficient codeword-based ML decoder.

Q: Can we also design an efficient bit-based ML decoder?