ELE-32 Introdução a Comunicações

Aula 3- Código Convolucional

October 17, 2018

1 Introdução

Os códigos de canal vistos até o momento são códigos de bloco: convertem um bloco com K bits de informação em um bloco com N bits transmitidos. Alternativamente, a codificação de canal pode ser feita processando sequências de bits.

Um codificador convolucional pode ser visto como uma máquina que a cada instante processa k bits de entrada e gera $n > k$ bits de saída e que possui memória, de forma que os bits a serem gerados no próximo instante dependem do estado(memória) atual do codificador. A princípio é possível gerar uma sequência infinita de bits a partir de uma sequência finita de bits. Um exemplo de uma máquina destas está mostrado na figura 1.

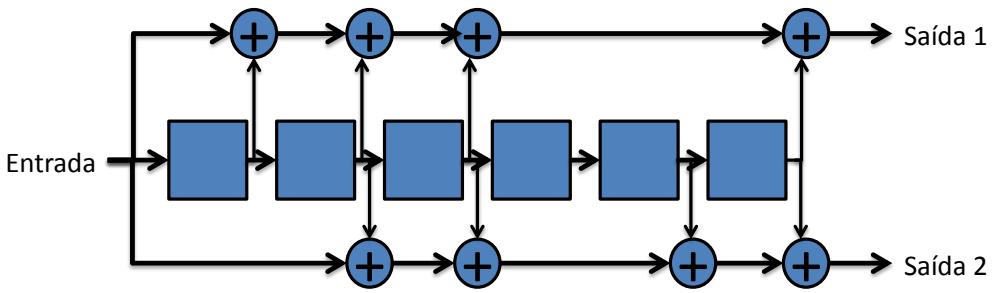

Figure 1: Codificador convolucional a ser implementado.

Este codificador possui 6 memórias. Cada memória pode valer 0 ou 1. Logo, há $2^6 = 64$ estados possíveis. Há 1 bit de conteúdo (informação) para cada 2 bits transmitidos. Neste caso, a taxa deste codificador é 1/2.Há então três parâmetros importantes:

1. k - o número de bits de informação que entram no codificador a cada instante;
2. n - o número de bits gerados a cada instante pelo codificador;
3. m - o número de memórias (no nosso caso binárias) que o codificador possui.

Estes códigos podem ser descritos de várias formas. Uma delas é através da descrição polinomial semelhante à notação utilizada para códigos cíclicos. A diferença é que agora os polinômios de entrada e saída podem ter grau infinito. Nesta representação temos a matriz geradora $\mathbf{G}(D)$, que agora é uma matriz de poliômios. Por exemplo, para a figura acima teríamos:

$$\mathbf{G}(D) = [1 + D + D^2 + D^3 + D^6 \qquad 1 + D^2 + D^3 + D^5 + D^6] \tag{1}$$

A relação entre entrada e saída é dada por:

$$\mathbf{v}(D) = u(D)\mathbf{G}(D) \tag{2}$$

onde $u(D)$ neste exemplo tem dimensão 1×1 e representa o polinômio de entrada (com grau potencialmente infinito). Pode haver mais do que uma entrada, mas isso não é comum.

Como $\mathbf{v}(D) = [v_1(D), v_2(D)]$ tem dimensão 1×2 , podemos formar a sequência de saída $v(D)$ (unidimensional) interpolando os termos da seguinte forma:

$$v(D) = \sum_{i=1}^{n} v_i(D^n)D^{i-1} \tag{3}$$

Outra descrição possível é através do diagrama de estados. Para apresenta-la de forma mais simples, utilizaremos o codificador abaixo:

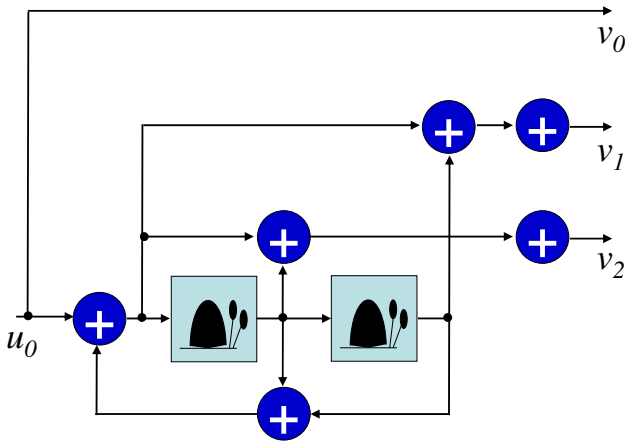

Figure 2: Codificador recursivo.

Este codificador é recursivo, isto é, há uma realimentação do próprio estado na definição do estado no próximo estado. A descrição polinomial deste codificador e mais detalhes sobre a teoria de códigos convolucionais podem ser visto em http://www.ele.ita.br/~manish/eet49/Documents/EET49-20120530-Notas%20de%20aula%2010.pdf.

O diagrama de estados é um diagrama que apresenta todos os estados possíveis do codificador e os conecta com setas rotuladas pelos bits de informação que causam aquela transição e pela consequente saída caso haja aquela transição. Este diagrama pode auxiliar na tarefa de codificação pois é mais simples ler uma tabela com as informações correspondentes do que calcular o polinômio de saída instante a instante.

Por último, podemos definir ainda a seção de treliça, que nada mais é do que o diagrama de estados dividido em dois instantes: o inicial e o final. Todas as setas partem de estados dos instantes iniciais e terminam em estados dos instantes finais, com os mesmos rótulos do diagrama de estados. Esta representação será utilizada para realizar a decodificação via algoritmo de Viterbi.

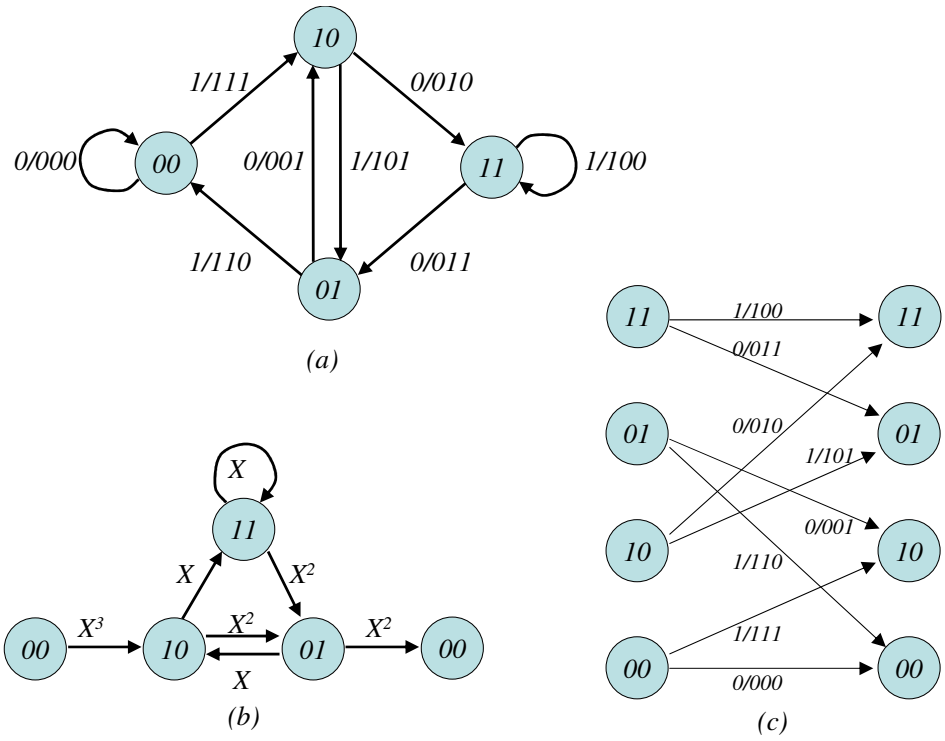

Figure 3: Representações para um código convolucional:(a)-Diagrama de estados. (b)-Diagrama de estados modificado. (c)-Seção de treliça.

2 Decodificação: Algoritmo de Viterbi

- O algoritmo de Viterbi usa a treliça do código para realizar a decodificação de um código convolucional
- O algoritmo armazena métricas de percurso (custo) para tomar a decisão sobre qual sequência foi transmitida. A métrica do percurso é obtida através da soma das métricas dos ramos que, concatenados, formam o percurso.
- Há algumas possibilidades para a métrica:
  - No caso de transmissão através um canal BSC, pode-se utilizar a distância de Hamming entre o que foi recebido e os rótulos binários de saída dos ramos. A distância de Hamming entre dois

vetores binários com o mesmo comprimento é o número de posições em que estes dois vetores diferem de valor. Pode-se usar também a probabilidade de cada ramo ter sido transmitido dado o valor recebido naquele instante. Por exemplo, ao recebermos a sequência 111, a probabilidade de ter transmitido a sequência 111 é $(1-p)^3$, de p^3 para a sequência 000 e de $p(1-p)^2$ para a sequência 101, por exemplo.

  - No caso de transmissão através de um canal Gaussiano, pode-se utilizar a distância Euclidiana quadrática entre o símbolo recebido e os símbolos associados aos ramos de saída.

- A cada instante, o número de percursos possíveis cresce exponencialmente. Se, por exemplo, o codificador tiver k entradas, n saídas e m memórias, há 2^{kt} sequências possíveis do instante 0 até o instante t.

- Por outro lado, somente um pequeno conjunto de sequências são as mais prováveis. O algoritmo de Viterbi aproveita-se deste fato para reduzir a complexidade de decodificação. Em um dado instante, há vários percursos que levam ao mesmo estado σ_i^t. Dentre os vários possíveis, um deles será mais provável do que os outros. Logo, se o percurso mais provável passa pelo estado σ_i no instante t, o caminho que levou até este estado deve ser o mais provável. Assim, em cada instante, é necessário armazenar, pra cada estado, somente o caminho mais provável de chegar até ele. Isto é, precisamos armazenar 2^m caminhos, o que é uma redução de complexidade se comparado com o número de sequências possíveis(2^{kt}).

- O algoritmo de Viterbi pode ser descrito em palavras desta forma:
  1. Inicie todos os estados com custo igual a zero. Caso o estado inicial seja obrigatoriamene um estado em particular(por exemplo estado nulo), somente este terá custo zero; os outros terão custo inicial infinito.
  2. Dado o símbolo recebido, calcule o custo de cada uma das transições possíveis no instante atual*
  3. Para cada estado futuro, calcule o custo de todos os percursos que chegam ao estado somando o custo do estado anterior com o custo do ramo que causa a transição entre estados. O caminho sobrevivente para um estado futuro é aquele com menor custo. Este custo torna-se também o custo do estado.
  4. Enquanto houver símbolos a serem processados, retorne ao passo 2.
  5. Escolha o estado final que tem o menor custo e o caminho que leva a ele, obedecendo a restrições sobre o estado final, se houver†.

- O algoritmo de Viterbi também pode ser descrito matematicamente. Para isto precisamos apresentar algumas variáveis:
  - σ_i^k: i-ésimo estado no instante k, $i = 0, 1, 2, ..., 2^m - 1$ e $k = 0, 1, 2, ..., K$
  - $C(\sigma_i^k)$: custo do i-ésimo estado no instante k
  - $\rho_{i,j}^k$: ramo que causa a transição de σ_i^k para σ_j^{k+1}
  - $C(\rho_{i,j}^k)$: custo do ramo que causa a transição de σ_i^k para σ_j^{t+1}‡.
  - $s(\rho_{i,j}^k)$: símbolo de entrada associado ao ramo $\rho_{i,j}^k$.

*No instante inicial, por exemplo, o único estado possível é o nulo. Logo, todas as transições devem sair deste estado.

†É possível por projeto garantir que o último estado seja igual ao estado nulo, por exemplo

‡Pode haver mais de um ramo que causa a mesma transição entre estados. Neste caso haverá transições paralelas. Por simplicidade de notação omitimos esta situação

- $t(\rho_{i,j}^k)$: símbolo de saída (transmitido) associado ao ramo $\rho_{i,j}^k$.
- r^k símbolo recebido no k-ésimo instante.
- $\mathbf{s}_i^k = [s_0, s_1, s_2, ..., s_{k-1}]$: Sequência de símbolos que formam o caminho sobrevivente que chega ao estado σ_i^k

- Desta forma, o algoritmo de Viterbi é:

1. Iniciação de variáveis:
$$\begin{gathered} C(\sigma_0^0) = 0 \\ C(\sigma_i^0) = \infty, i = 1, 2, 3, ..., 2^m - 1 \\ k = 0 \\ \mathbf{s}_0^0 = \emptyset \end{gathered} \tag{4}$$
2. Cálculo dos custos dos ramos:
$$C(\rho_{i,j}^k) = d(r^k, t(\rho_{i,j}^k)) \tag{5}$$
3. Cálculo dos custos dos estados futuros e seleção de caminho sobrevivente.
$$\begin{aligned} i(j)* = &\quad \arg\min_i \left\{ C(\sigma_i^k) + C(\rho_{i,j}^k) \right\} \\ C(\sigma_j^{k+1}) = &\quad C(\sigma_{i(j)*}^k) + C(\rho_{i(j)*,j}^k) \\ \mathbf{s}_j^{k+1} = &\quad [\mathbf{s}_{i(j)*}^k \quad s(\rho_{i(j)*,j}^k)] \end{aligned} \tag{6}$$
onde a última linha indica a concatenação do vetor $\mathbf{s}_{i(j)*}^k$ com o símbolo $s(\rho_{i(j)*,j}^k)$.
4. Se $k < K$(tamanho da sequência), $k = k + 1$. Retorne ao passo 2.
5. $\mathbf{s}* = \max_i \left\{ \mathbf{s}_i^K \right\}$

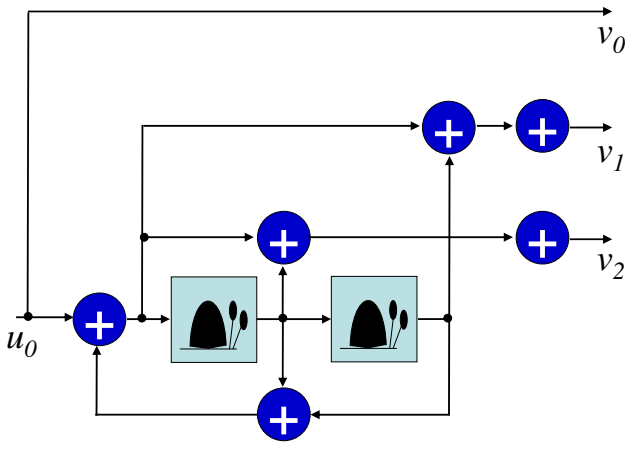

Figure 4: Codificador recursivo.

- A figura 5 mostra a evolução do algoritmo de Viterbi quando a sequência $000101100000\cdots$ é recebida, utilizando o codificador da figura 4. Na decodificação assumiu-se que a sequência $111101110000\cdots$ foi transmitida, correspondendo aos bits de informação $111000\cdots$. Note que, com o aumento de k, todos os outros caminhos continuarão a ter os seus pesos aumentados, enquanto que o caminho que prossegue com zeros manterá o seu peso igual a 3.

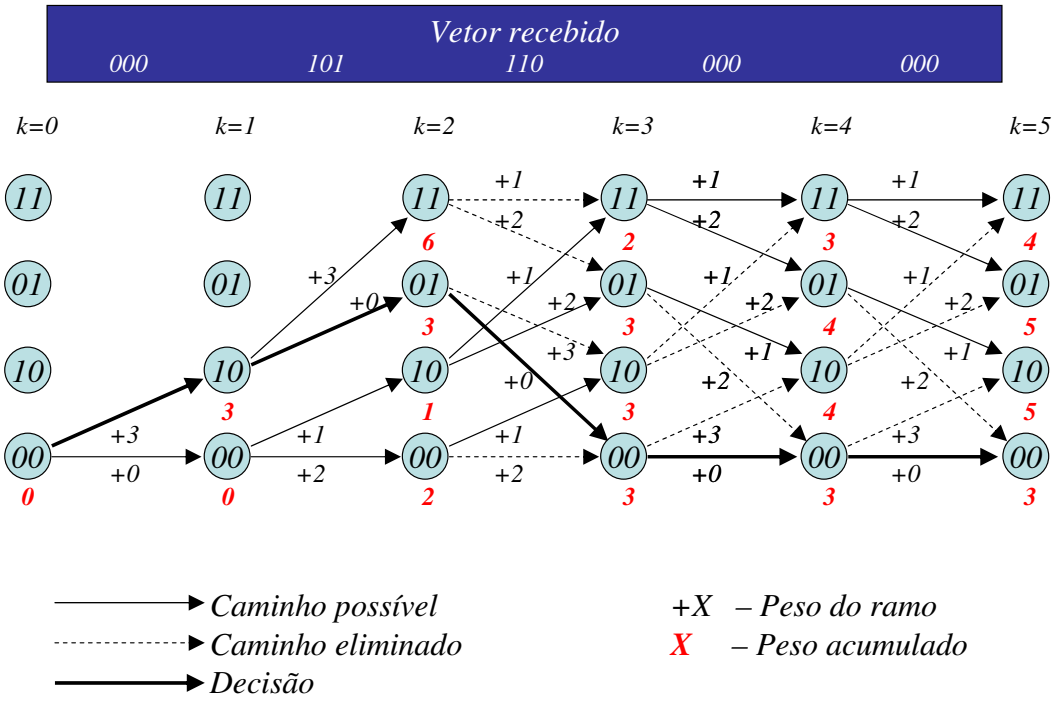

Figure 5: Execução do algoritmo de Viterbi para o exemplo.

m	$g_1(D)$	$g_2(D)$	$g_3(D)$
3	13	15	17
4	25	33	37
6	117	127	155

Table 1: Representação octal dos polinômios geradores a serem utilizados para gerar os códigos com taxa 1/3

- Um problema do algoritmo de Viterbi é que, para tomar a decisão, precisamos processar toda a sequência recebida, o que pode levar um tempo muito grande.
- Alternativamente, pode-se utilizar uma variante sub-ótima onde observa-se os sinais recebidos entre os instantes k e $k+T$ para se tomar a decisão pelo símbolo que foi transmitido no instante k. Isto é, após T instantes, a sequência mais provável dirá qual foi o símbolo transmitido no instante T.
- O valor de T pode ser escolhido por simulação.
- É possível que, ao utilizar o método sub-ótimo, a decisão final seja uma sequência de ramos que não seja possível. Isto pode acontecer por exemplo se a sequência vencedora no instante $k+T$ exija que o estado no instante $k+1$ seja diferente do estado no instante $k+1$ da sequência vencedora no instante $k+T+1$.

3 Atividade

O objetivo do laboratório de hoje e do próximo laboratório é implementar um codificador de canal convolucional e seu respectivo decodificador. Algumas atividades adicionais simples serão solicitadas no próximo laboratório.

O algoritmo deve, para cada um dos três códigos propostos:

1. Gerar 10000 bits de informação para cada valor de p do canal BSC.
2. Gerar um bloco 20000 bits codificados após passagem pelo codificador de canal. O estado inicial do codificador é obrigatoriamente o estado nulo.
3. Simular a passagem dos bits codificados por um canal BSC com parâmetro p semelhante aos laboratórios anteriores.
4. Decodificar a sequência recebida utilizando o algoritmo de Viterbi
5. Estimar a probabilidade de erro do bit de informação para cada valor de p do canal BSC

Realize esta codificação para os códigos com polinômios geradores descritos na tabela abaixo na forma octal com o bit mais significativo a esquerda. Por exemplo, para $m = 3, 1 \cdot 5 = 1 \cdot 101 = 1 + D + D^3$ e para $m = 6, 1 \cdot 5 \cdot 5 = 1 \cdot 101 \cdot 101$. Todos os códigos tem taxa 1/3.

A implementação deve seguir algumas regras:

- Não é permitido utilizar nenhuma tabela ou semelhante com mais do que 64 bits. Não é permitido utilizar várias tabelas com 64 bits ou menos com o intuito de contornar esta restrição.
- Não é permitido a utilização de implementações pré-existentes. É permitido a utilização de funções básicas pré-existentes como módulo, operações lógicas, etc.

Algumas atividades complementares serão solicitadas no próximo laboratório.

3.1 Desafio muito difícil

Implemente um código Turbo com concatenação paralela utilizando dois codificadores convolucionais com taxa 1/2. Cada um deles tem matriz geradora $\mathbf{G}(D) = [1, \frac{D}{1+D^2}]$. A primeira saída do segundo codificador é omitida. O tamanho do entrelaçador (permutador) deve ser de 1000 bits. Será necessário implementar o algoritmo BCJR. Utilize pelo menos 10 iterações internas. Compare com o desempenho do código com taxa 1/3.

4 Perguntas a serem respondidas no relatório

Haverá somente um relatório no bimestre, a ser entregue na data prevista para o R4. Este único relatório comporá a nota de R3 e R4. O relatório deve conter, em relação a esta parte:

1. Quais foram as maiores dificuldades em implementar o codificador convolucional?
2. Quanto tempo a sua solução demora para codificar cada bit de informação? Faça uma média.
3. Quais foram as maiores dificuldades em implementar o decodificador convolucional?
4. Como a probabilidade de erro de transmissão foi estimada? Qual é o seu valor? Como ela se compara com o valor de p escolhido? Como ela muda com m?x
5. Qual é o tamanho final do seu executável?
6. Quanto tempo a sua solução demora para decodificar cada bit? Faça uma média.
7. [Opcional-Bônus]É possível utilizar como métrica tanto a distância de Hamming entre os bits recebidos e os potencialmente transmitidos como a probabilidade de ter transmitido uma subsequência qualquer dados os bits recebidos quando estes passam por um canal BSC com parâmetro p. Qual apresenta melhor desempenho?

5 Referências

- https://en.wikipedia.org/wiki/Forward_error_correction
- https://en.wikipedia.org/wiki/Convolutional_code
- http://www.ele.ita.br/~manish/eet49/Documents/EET49-20120530-Notas%20de%20aula%2010.pdf
- http://www.ele.ita.br/~manish/eet49/Documents/EET49-20120806-Notas%20de%20aula%2012.pdf
- https://en.wikipedia.org/wiki/Turbo_code
- https://en.wikipedia.org/wiki/BCJR_algorithm