COMP2610/6261 - Information Theory

Lecture 16: Arithmetic Coding

Robert C. Williamson

Research School of Computer Science

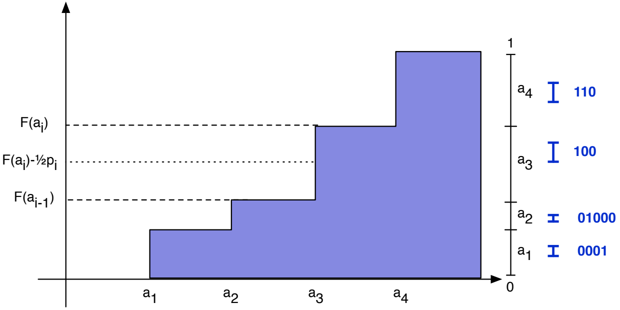
Australian National University

25 September, 2018

Interval Codes (Recap)

Shannon-Fano-Elias Coding method:

- Order the alphabet $\mathcal{A}$.
- Represent distribution **p** by cumulative distribution F
- Construct code by finding intervals of width $\frac{p_i}{2}$ that lie in each symbol interval $[F(a_{i-1}), F(a_i))$



Intervals and Prefix Codes (Recap)

The set of numbers in $[0, 1)$ that start with a given sequence of bits $\mathbf{b} = b_1 \ldots b_n$ form the interval

$$\left[0.\mathtt{b_1} \ldots \mathtt{b_n}, 0.\mathtt{b_1} \ldots \mathtt{b_n} + \frac{1}{2^n}\right) = [0.\mathtt{b_1} \ldots \mathtt{b_n}, 0.\mathtt{b_1} \ldots \mathtt{b_n} + 0.\mathtt{0} \ldots \mathtt{1})$$

This interval contains all binary strings for which $\mathtt{b_1 b_2 \ldots b_n}$ is a prefix

Prefix property (interval form): Once you pick a codeword $\mathtt{b_1 b_2 \ldots b_n}$, you cannot pick any codeword in the codeword interval

$$\left[0.\mathtt{b_1 b_2} \ldots \mathtt{b_n}, 0.\mathtt{b_1 b_2} \ldots \mathtt{b_n} + \frac{1}{2^n}\right)$$

Interval Coding Blocks

What if we apply SFE coding to blocks of an ensemble X?

Example: Let $\mathcal{A} = \{\texttt{aa}, \texttt{ab}, \texttt{ba}, \texttt{bb}\}$ with $\mathbf{p} = (0.2, 0.6, 0.1, 0.1)$.

Interval Coding Blocks

What if we apply SFE coding to blocks of an ensemble X?

Example: Let $\mathcal{A} = \{\texttt{aa}, \texttt{ab}, \texttt{ba}, \texttt{bb}\}$ with $\mathbf{p} = (0.2, 0.6, 0.1, 0.1)$.

x	p	$\bar{F}$	$\bar{F}_2$	ℓ	Code
`aa`	0.2	0.1	$0.0\overline{0011}_2$	4	0001
`ab`	0.6	0.5	0.1_2	2	10
`ba`	0.1	0.85	$0.110\overline{1100}_2$	5	11011
`bb`	0.1	0.95	$0.11\overline{1100}_2$	5	11110

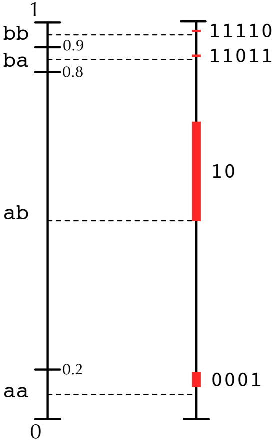
Extend to longer sequences

Interval Coding Blocks

What if we apply SFE coding to blocks of an ensemble X?

Example: Let $\mathcal{A} = \{\mathtt{aa}, \mathtt{ab}, \mathtt{ba}, \mathtt{bb}\}$ with $\mathbf{p} = (0.2, 0.6, 0.1, 0.1)$.

x	p	$\bar{F}$	$\bar{F}_2$	ℓ	Code
aa	0.2	0.1	$0.0\overline{0011}_2$	4	0001
ab	0.6	0.5	0.1_2	2	10
ba	0.1	0.85	$0.110\overline{1100}_2$	5	11011
bb	0.1	0.95	$0.11\overline{1100}_2$	5	11110



Extend to longer sequences

This works but:

- Need $P(\mathbf{x})$ for all $\mathbf{x}$
- Total $|\mathcal{A}|^N$ values for length N
- Huffman has similar complexity but shorter codes.

Arithmetic Coding: A Bird's Eye View

Basic idea of arithmetic coding follows SFE coding

	SFE Coding	**Arithmetic coding**
Input	Single outcome x_i	Sequence of outcomes $x_1 x_2 \ldots x_N$
Key step	Find symbol interval for x_i	Find symbol interval for $x_1 x_2 \ldots x_N$
Output	Binary string corresponding to chosen interval	Binary string corresponding to chosen interval

Arithmetic Coding: A Bird's Eye View

Basic idea of arithmetic coding follows SFE coding

	SFE Coding	**Arithmetic coding**
Input	Single outcome x_i	Sequence of outcomes $x_1 x_2 \ldots x_N$
Key step	Find symbol interval for x_i	Find symbol interval for $x_1 x_2 \ldots x_N$
	Use $[F(x_{i-1}), F(x_i))$	?
Output	Binary string corresponding to chosen interval	Binary string corresponding to chosen interval
	Output first $\ell(x_i)$ bits of mid-point of interval	Output first $\ell(x_1 x_2 \ldots x_N)$ bits of midpoint of interval

Arithmetic Coding: Summary

Arithmetic coding has some important properties:

- We do **not** compute a symbol coding for X and then concatenate
  - Directly work with blocks of size N
- We do not **explicitly** code all length N sequences at once
  - Highly efficient
- We do not assume that each of the x_i's is **independent**
  - Not restricted to extended ensembles
  - Adapts to data distribution

Computing an Interval for Sequences

Say $N = 2$ and we want to code $x_1 x_2$

Ideally, we'd like to compute $p(\mathbf{x})$ for all possible $\mathbf{x}$ of length 2, and then find the interval for $p(x_1 x_2)$

Key ideas:

- we can write $p(x_1 x_2) = p(x_1) p(x_2 | x_1)$
  - decompose joint into conditional probabilities

Computing an Interval for Sequences

Say $N = 2$ and we want to code $x_1 x_2$

Ideally, we'd like to compute $p(\mathbf{x})$ for all possible $\mathbf{x}$ of length 2, and then find the interval for $p(x_1 x_2)$

Key ideas:

- we can write $p(x_1 x_2) = p(x_1) p(x_2 | x_1)$
  - decompose joint into conditional probabilities
- $p(\cdot | x_1)$ is just another probability distribution
  - so we can compute intervals as per SFE

Computing an Interval for Sequences

Say $N = 2$ and we want to code $x_1 x_2$

Ideally, we'd like to compute $p(\mathbf{x})$ for all possible $\mathbf{x}$ of length 2, and then find the interval for $p(x_1 x_2)$

Key ideas:

- we can write $p(x_1 x_2) = p(x_1)p(x_2|x_1)$
  - decompose joint into conditional probabilities
- $p(\cdot|x_1)$ is just another probability distribution
  - so we can compute intervals as per SFE
- we can find an interval for $p(x_2|x_1)$ **within** the interval for x_1
  - normal SFE computes the interval within $[0, 1)$ by default

Computing an Interval for Sequences

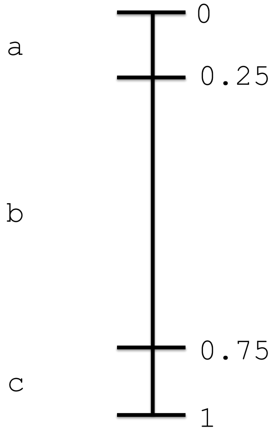
Example: Suppose $\mathcal{A} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and $p(\mathtt{a}) = 0.25, p(\mathtt{b}) = 0.5, p(\mathtt{c}) = 0.25$

Like with SFE coding, we'd begin by slicing up $[0, 1)$ into three subintervals:

Computing an Interval for Sequences

Example: Suppose $\mathcal{A} = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ and $p(\texttt{a}) = 0.25, p(\texttt{b}) = 0.5, p(\texttt{c}) = 0.25$

Like with SFE coding, we'd begin by slicing up $[0, 1)$ into three subintervals:



So e.g. we treat $[0.25, 0.75)$ as the interval for $\texttt{b}$

Computing an Interval for Sequences

Suppose the first symbol is $\texttt{b}$, and $p(\texttt{a}|\texttt{b}) = 0.25, p(\texttt{b}|\texttt{b}) = 0.5, p(\texttt{c}|\texttt{b}) = 0.25$

Computing an Interval for Sequences

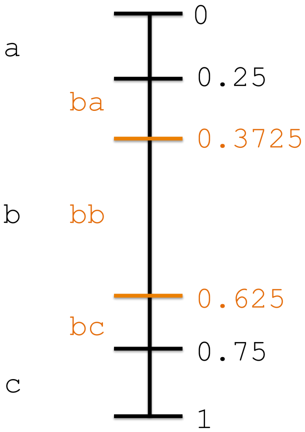
Suppose the first symbol is $\mathtt{b}$, and $p(\mathtt{a}|\mathtt{b}) = 0.25, p(\mathtt{b}|\mathtt{b}) = 0.5, p(\mathtt{c}|\mathtt{b}) = 0.25$

To code $\mathtt{ba}, \mathtt{bb}, \mathtt{bc}$, now slice up $[0.25, 0.75)$, the interval for $\mathtt{b}$ itself:

Computing an Interval for Sequences

Suppose the first symbol is $\texttt{b}$, and $p(\texttt{a}|\texttt{b}) = 0.25$, $p(\texttt{b}|\texttt{b}) = 0.5$, $p(\texttt{c}|\texttt{b}) = 0.25$

To code $\texttt{ba}, \texttt{bb}, \texttt{bc}$, now slice up $[0.25, 0.75)$, the interval for $\texttt{b}$ itself:



For $\texttt{ba}$ we choose the interval of length $p(\texttt{a}|\texttt{b}) = 0.25$ times the length of the enclosing interval $(0.75 - 0.25 = 0.5)$, i.e. $(0.25)(0.5) = 0.125$

Arithmetic Coding: End of Stream Symbol

It is convenient to explicitly have a special “end of stream” symbol, $\square$

We add this symbol to our ensemble, with some suitable probability

- e.g. $p(\square)$ = probability of seeing empty string, $p(\square|\texttt{a})$ = probability of seeing just the string `a`, etc
- Implicitly we think of `ab` as actually being $\texttt{ab}\square$

End of stream is by definition reached when we choose the sub-interval for this special symbol

Arithmetic Coding: End of Stream Example

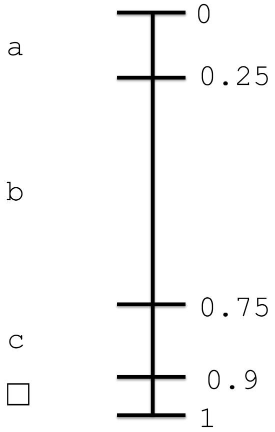
Example: Suppose $\mathcal{A} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \Box\}$ and
$p(\mathtt{a}) = 0.25, p(\mathtt{b}) = 0.5, p(\mathtt{c}) = 0.15, p(\Box) = 0.1$

Like with SFE coding, we'd begin by slicing up $[0, 1)$ into three subintervals:

Arithmetic Coding: End of Stream Example

Example: Suppose $\mathcal{A} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \Box\}$ and
$p(\mathtt{a}) = 0.25, p(\mathtt{b}) = 0.5, p(\mathtt{c}) = 0.15, p(\Box) = 0.1$

Like with SFE coding, we'd begin by slicing up $[0, 1)$ into three subintervals:



Arithmetic Coding: End of Stream Example

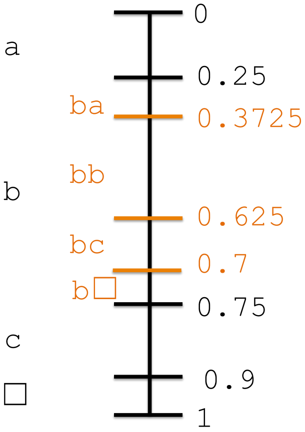
Now suppose that $p(\cdot|\texttt{b})$ stays the same as $p(\cdot)$

If the first symbol is `b`, we carve the interval for `b` into four pieces:

Arithmetic Coding: End of Stream Example

Now suppose that $p(\cdot|\texttt{b})$ stays the same as $p(\cdot)$

If the first symbol is `b`, we carve the interval for `b` into four pieces:



Exact same idea as before, just with special symbol □

Arithmetic Coding for Arbitrary Sequences

These ideas generalise to arbitrary length sequences

- We don't even need to know the sequence length beforehand!

As we see more symbols, we slice the appropriate sub-interval of $[0, 1)$ based on the probabilities

- Terminate whenever we see $\Box$

Arithmetic Coding: Codeword Generation

Once we've seen the entire sequence, we end up with interval $[u, v)$

- How to output a codeword?

Arithmetic Coding: Codeword Generation

Once we've seen the entire sequence, we end up with interval $[u, v)$

- How to output a codeword?

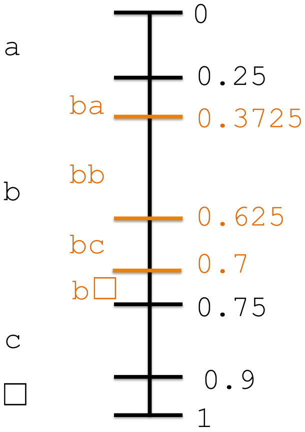
As per SFE coding, we can use the first $\ell(x_1 x_2 \ldots x_N)$ bits of $(u + v)/2$

- Here, $\ell(x_1 x_2 \ldots x_N) = \lceil \log 1/p(x_1 x_2 \ldots x_N) \rceil + 1$
- As before, this guarantees all strings starting with codeword are contained in the codeword interval

Generally, we can output some bits on the fly, rather than wait till we process the entire sequence

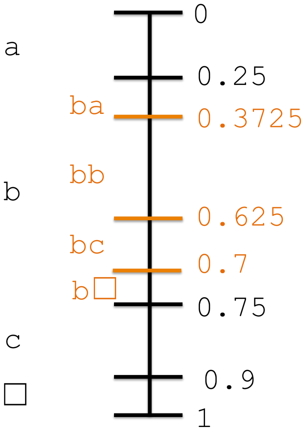
Arithmetic Coding: Codeword Generation Example

In previous example with input `b`, we'd stop in the interval for `b□`, i.e. $[0.7, 0.75)$



Arithmetic Coding: Codeword Generation Example

In previous example with input b, we'd stop in the interval for b□, i.e. $[0.7, 0.75)$



Midpoint is $0.725 = \mathtt{10111\overline{0011}}$, and $p(\mathtt{b}\square) = (1/2) \cdot (0.1) = 0.05$

Output the first $\lceil \log_2 1/0.05 \rceil + 1 = 6$ bits, i.e. 101110

Arithmetic Coding: Formal Encoder

Formally, we compute the interval $[u, v)$ for a generic sequence as follows:

Arithmetic Coding of stream $x_1 x_2 \ldots$

$u \leftarrow 0.0$
$v \leftarrow 1.0$
$p \leftarrow v - u$

for $n = 1, 2, \ldots$

- Compute $L_n(a_i | x_1, ..., x_{n-1}) = \sum_{i'=1}^{i-1} p(x_n = a_{i'} | x_1, \ldots, x_{n-1})$
- Compute $U_n(a_i | x_1, ..., x_{n-1}) = \sum_{i'=1}^{i} p(x_n = a_{i'} | x_1, \ldots, x_{n-1})$
- $v \leftarrow u + p \cdot U_n(x_n | x_1, ..., x_{n-1})$
- $u \leftarrow u + p \cdot L_n(x_n | x_1, ..., x_{n-1})$
- $p \leftarrow v - u$
- **if** $x_n = \Box$, terminate

Output first $\ell(x_1 x_2 \ldots x_N) = \lceil \log 1/p \rceil + 1$ bits of $(u + v)/2$

Here, L_n, U_n just compute the appropriate lower and upper bounds, as per SFE coding

- We rescale these based on the current interval length

Decoding

How do we decode a sequence of bits?

Rough Sketch:

- Carve out $[0, 1)$ based on initial distribution
- Keep reading bits until current code interval in a symbol interval
- Output that symbol
- Carve out appropriate interval based on probabilities
- $\vdots$

We can stop once we have containment in interval for □

Decoding: Example

Suppose $p(\mathtt{a}) = 0.5, p(\mathtt{b}) = 0.125, p(\mathtt{c}) = 0.25, p(\square) = 0.125$ for every outcome in sequence

Decode `0110111`:

Sequence	**Interval (Binary)**	**Interval (Decimal)**	**Comment**
`0`	$[0.0, 0.1)_2$	$[0, 0.5)_{10}$	First symbol `a`

Decoding: Example

Suppose $p(\texttt{a}) = 0.5, p(\texttt{b}) = 0.125, p(\texttt{c}) = 0.25, p(\Box) = 0.125$ for every outcome in sequence

Decode `0110111`:

Sequence	Interval (Binary)	Interval (Decimal)	Comment
`0`	$[0.0, 0.1)_2$	$[0, 0.5)_{10}$	First symbol `a`
`01`	$[0.01, 0.10)_2$	$[0.25, 0.5)_{10}$	

Decoding: Example

Suppose $p(\texttt{a}) = 0.5, p(\texttt{b}) = 0.125, p(\texttt{c}) = 0.25, p(\square) = 0.125$ for every outcome in sequence

Decode `0110111`:

Sequence	**Interval (Binary)**	**Interval (Decimal)**	**Comment**
`0`	$[0.0, 0.1)_2$	$[0, 0.5)_{10}$	First symbol `a`
`01`	$[0.01, 0.10)_2$	$[0.25, 0.5)_{10}$	
`011`	$[0.011, 0.100)_2$	$[0.375, 0.5)_{10}$	Next symbol `c`

Decoding: Example

Suppose $p(\texttt{a}) = 0.5, p(\texttt{b}) = 0.125, p(\texttt{c}) = 0.25, p(\Box) = 0.125$ for every outcome in sequence

Decode `0110111`:

Sequence	Interval (Binary)	Interval (Decimal)	Comment
`0`	$[0.0, 0.1)_2$	$[0, 0.5)_{10}$	First symbol `a`
`01`	$[0.01, 0.10)_2$	$[0.25, 0.5)_{10}$	
`011`	$[0.011, 0.100)_2$	$[0.375, 0.5)_{10}$	Next symbol `c`
`0110`	$[0.0110, 0.0111)_2$	$[0.375, 0.4375)_{10}$	

Decoding: Example

Suppose $p(\mathtt{a}) = 0.5, p(\mathtt{b}) = 0.125, p(\mathtt{c}) = 0.25, p(\square) = 0.125$ for every outcome in sequence

Decode `0110111`:

Sequence	Interval (Binary)	Interval (Decimal)	Comment
`0`	$[0.0, 0.1)_2$	$[0, 0.5)_{10}$	First symbol `a`
`01`	$[0.01, 0.10)_2$	$[0.25, 0.5)_{10}$	
`011`	$[0.011, 0.100)_2$	$[0.375, 0.5)_{10}$	Next symbol `c`
`0110`	$[0.0110, 0.0111)_2$	$[0.375, 0.4375)_{10}$	
`01101`	$[0.01101, 0.01110)_2$	$[0.40625, 0.4375)_{10}$	

Decoding: Example

Suppose $p(\mathtt{a}) = 0.5, p(\mathtt{b}) = 0.125, p(\mathtt{c}) = 0.25, p(\square) = 0.125$ for every outcome in sequence

Decode `0110111`:

Sequence	**Interval (Binary)**	**Interval (Decimal)**	**Comment**
`0`	$[0.0, 0.1)_2$	$[0, 0.5)_{10}$	First symbol `a`
`01`	$[0.01, 0.10)_2$	$[0.25, 0.5)_{10}$	
`011`	$[0.011, 0.100)_2$	$[0.375, 0.5)_{10}$	Next symbol `c`
`0110`	$[0.0110, 0.0111)_2$	$[0.375, 0.4375)_{10}$	
`01101`	$[0.01101, 0.01110)_2$	$[0.40625, 0.4375)_{10}$	
`011011`	$[0.011011, 0.011100)_2$	$[0.421875, 0.4375)_{10}$	Next symbol $\square$

The last bit here is actually redundant (inherited from $+1$ bit in midpoint representation)

Adaptive Probabilities

So far we assume the sequence of probabilities are given in advance

In **Lecture 5**, you saw Bernoulli distribution for two outcomes

Adaptive Probabilities

So far we assume the sequence of probabilities are given in advance

In **Lecture 5**, you saw Bernoulli distribution for two outcomes

- Beta distribution $\text{Beta}(\theta|m_h, m_t)$ as a prior for $\text{Bern}(x|\theta)$
- The posterior after observing n_h heads and n_t tails is just $\text{Beta}(\theta|m_h + n_h, m_t + n_t)$
- The expected value of θ under the posterior is

$$p(x = \mathtt{h}|n_h, n_t, m_h, m_t) = \frac{m_h + n_h}{m_h + n_h + m_t + n_t}$$

Dirichlet Model

A **Dirichlet distribution** is a generalisation of the Beta distribution to more than two outcomes. Its parameter is a vector $\mathbf{m} = (m_1, \ldots, m_K)$ can be viewed as "virtual counts" for each symbol $a_1, \ldots, a_K$:

$$P(x = a_i | x_1 \ldots x_n) = \frac{\sharp(a_i) + m_i}{\sum_{k=1}^{K} \sharp(a_k) + m_k}$$

Can implement an adaptive guesser by just counting symbol occurrences.

Dirichlet Model

A **Dirichlet distribution** is a generalisation of the Beta distribution to more than two outcomes. Its parameter is a vector $\mathbf{m} = (m_1, \ldots, m_K)$ can be viewed as "virtual counts" for each symbol $a_1, \ldots, a_K$:

$$P(x = a_i | x_1 \ldots x_n) = \frac{\sharp(a_i) + m_i}{\sum_{k=1}^{K} \sharp(a_k) + m_k}$$

Can implement an adaptive guesser by just counting symbol occurrences.

Flexible

- e.g., Choose **m** to be frequency of English letters
- $\sum_k m_k$ Large = Stable; Small = Responsive

title

Example: Start with $m_h = m_t = 1$ and observe sequence `hht`.

$$p(\cdot|\epsilon) = (\tfrac{1}{2}, \tfrac{1}{2}),\ p(\cdot|\mathtt{h}) = (\tfrac{2}{3}, \tfrac{1}{3}),\ p(\cdot|\mathtt{hh}) = (\tfrac{3}{4}, \tfrac{1}{4})\ p(\cdot|\mathtt{hht}) = (\tfrac{3}{5}, \tfrac{2}{5})$$

viz. Laplace's Rule, where ϵ means empty string

Why? Because e.g.

$$p(\mathtt{h}|\mathtt{h}) = \frac{1+1}{1+0+1+1} = 2/3$$

$$p(\mathtt{t}|\mathtt{h}) = \frac{0+1}{1+0+1+1} = 1/3$$

We'll assume this learning is only for non $\Box$ symbols

- assume $\Box$ occurs with fixed probability each time

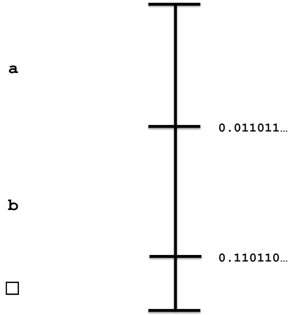
Adaptive Probabilities: Example

Possible outcomes $\texttt{a}, \texttt{b}, \square$

Sequence: ϵ

Probabilities: $p_{|\epsilon} = (0.425, 0.425, 0.15)$

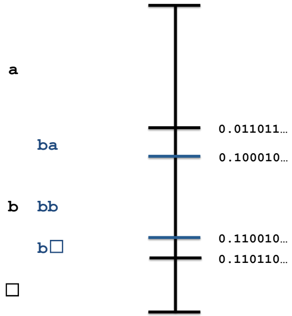
Encoder: ϵ



We start off with virtual counts $m_{\texttt{a}} = m_{\texttt{b}} = 1$

Adaptive Probabilities: Example

Possible outcomes $\mathtt{a}, \mathtt{b}, \square$



Observations: $\mathtt{b}$

Probabilities: $p_{|\mathtt{b}} \approx (0.28, 0.57, 0.15)$

Encoder Output: ϵ

Seeing $\mathtt{b}$ makes us update $p(\mathtt{a}|\mathtt{b}) = (0.85) \cdot (1/3) \approx 0.28$, and $p(\mathtt{b}|\mathtt{b}) = (0.85) \cdot (2/3) \approx 0.57$. We keep $p(\square|\mathtt{b}) = p(\square)$.

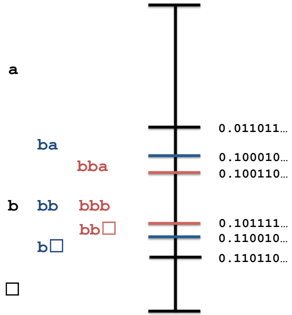
Adaptive Probabilities: Example

Possible outcomes $\mathtt{a}, \mathtt{b}, \square$

Observations: bb

Probabilities: $p_{|\mathtt{bb}} \approx (0.21, 0.64, 0.15)$

Encoder Output: 1



Seeing bb makes us update $p(\mathtt{a}|\mathtt{bb}) = (0.85) \cdot (1/4) \approx 0.21$, and $p(\mathtt{b}|\mathtt{bb}) = (0.85) \cdot (3/4) \approx 0.64$
Now the first bit is unambiguously 1

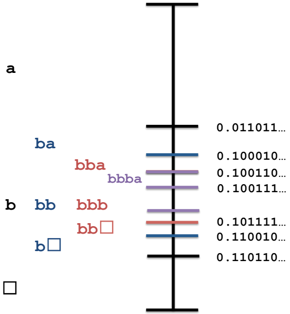
Adaptive Probabilities: Example

Possible outcomes $\texttt{a}, \texttt{b}, \Box$

Observations: bbb

Probabilities: $p_{|\texttt{bbb}} \approx (0.17, 0.68, 0.15)$

Encoder Output: 1



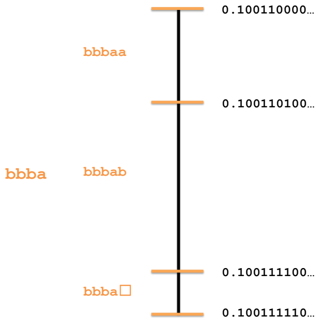
Adaptive Probabilities: Example

Possible outcomes $\mathtt{a}, \mathtt{b}, \square$

Observations: `bbba`

Probabilities: $p_{|\mathtt{bbba}} \approx (0.28, 0.57, 0.15)$

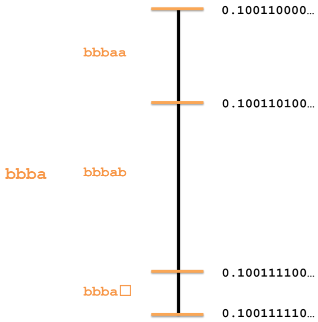
Encoder Output: `1001`



On seeing a, we can fill in three further bits unambiguously

Adaptive Probabilities: Example

Possible outcomes a, b, □



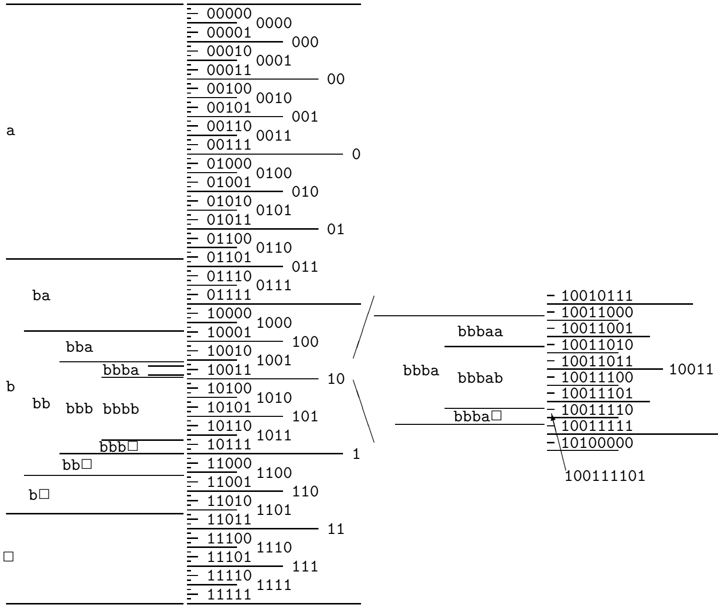
Observations: bbba□

Probabilities: N/A

Encoder Output: 100111101

To terminate, we find midpoint of 0.100111100 . . . and 0.100111110...

Arithmetic Coding: Example (MacKay, Figure 6.4)



Summary and Reading

Main Points

- Arithmetic Coding:
  - Uses interval coding and conditional probability
  - Separates coding and prediction
  - No need to compute every code word
- Predictive distributions:
  - Update distribution after each symbol
  - Beta and Dirichlet priors = virtual counts

Reading

- Section 6.2 of MacKay