Lecture 6

Arithmetic coding

Announcements

- Hw1 due on Wednesday
- Clarifications
  - Q1 - np.~~assert~~_almost_eq $1e^{-7}$
  - Q5 - hints: floats → bytes
- OH → Shubham after lecture today

Quiz Q1 Typical Set Size

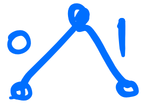
Consider a binary source with $P(0) = P(1) = 0.5$. What is the size of the typical set $A^{(n)}$, in terms of n?

$\mathrm{Ber}(1/2)$

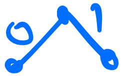
2^n

$H(X) = 1 \rightarrow$ almost every sequence is typical

Quiz Q2 KL Divergence



Consider a source $P = Ber(0.5)$. Zoro knows the distribution of this source and designs a per-symbol Huffman code for $P = Ber(0.5)$ to encode a sequence of symbols obtained using this source. However, Luffy doesn't know the distribution of this source and encodes it using a per-symbol Huffman code assuming that the sequence of symbols came from $Q = Ber(0.25)$.



2.1 How many extra number of bits in expectation (per-symbol) does Luffy need over Zoro to encode a sequence from the above source P?

0

same code

Quiz Q2 KL Divergence

Consider a source $P = Ber(0.5)$. Zoro knows the distribution of this source and designs a per-symbol Huffman code for $P = Ber(0.5)$ to encode a sequence of symbols obtained using this source. However, Luffy doesn't know the distribution of this source and encodes it using a per-symbol Huffman code assuming that the sequence of symbols came from $Q = Ber(0.25)$.

2.2 What is the KL divergence $D(P||Q)$ between distributions P and Q specified above?

0.2075

$0.5 \log_2 \frac{0.5}{0.25} + 0.5 \log_2 \frac{0.5}{0.75}$

Quiz Q2 KL Divergence

Consider a source $P = Ber(0.5)$. Zoro knows the distribution of this source and designs a per-symbol Huffman code for $P = Ber(0.5)$ to encode a sequence of symbols obtained using this source. However, Luffy doesn't know the distribution of this source and encodes it using a per-symbol Huffman code assuming that the sequence of symbols came from $Q = Ber(0.25)$.

2.3 In the class we learnt that KL divergence is an indicator of the excess code-length for mismatched codes. How do you explain that the two answers above do not match?

0/1 → does not achieve entropy for Q.

For a large block size → you would get $D(P\|Q)$.

Slides credit - Kedar Tatwawadi

RECAP

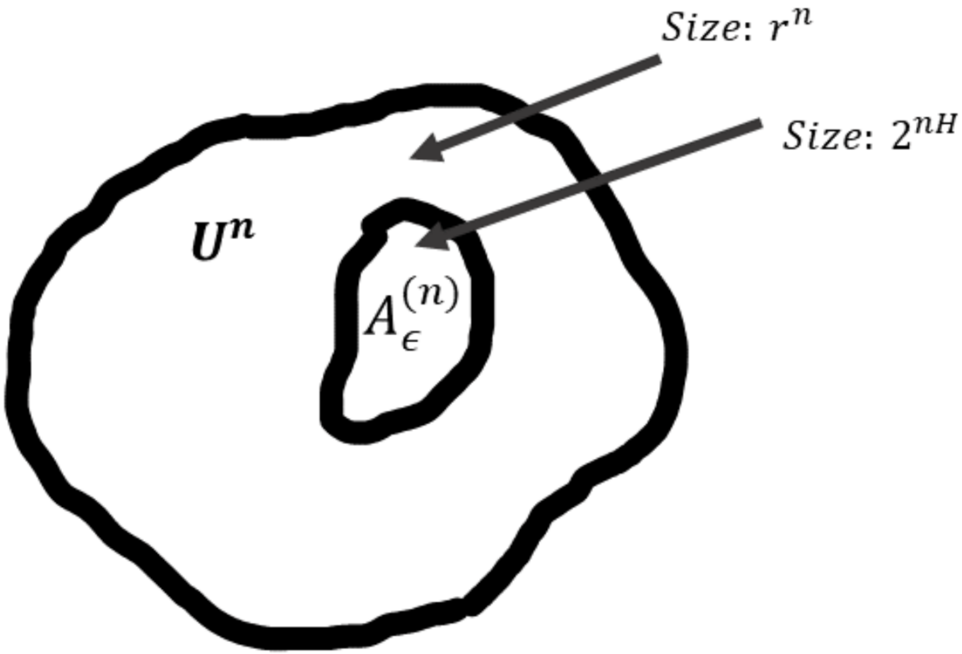
$H(X) \leq \mathbb{E}\, l_{Huff}(X) \leq \mathbb{E}\, l_{shann}(X) < H(X) + 1$

Issues with symbol codes:

1. `P = {A: 0.1, B: 0.9}`, `H(P) = 0.47`
   Huffman code can only compress this to `1 bit`.
2. For any symbol s, ideally we want to use $l(s) = \log_2 \frac{1}{P(s)}$ bits. But, as we are using a symbol code, we cant use fractional bits.

Thus, there is always going to be `~1 bit` overhead per symbol with symbol codes.

RECAP - AEP



U^n = set of all possible n-tuples

$A_\epsilon^{(n)}$ = set of all typical n-tuples

$$P\left(U^n \epsilon\, A_\epsilon^{(n)}\right) \approx 1$$

$$\forall\, u^n \epsilon\, A_\epsilon^{(n)} : \; P(u^n) \approx 2^{-nH} *$$

$*$ *where u^n represents a particular $n-tuple$*

RECAP

Solution -> use block codes

We can do better by considering blocks of size 2: `P = {AA: 0.01, AB: 0.09, BA: 0.09, BB: 0.81}`, this way the overhead is `~1 bit` per 2 symbol!

(or in case of blocks of size `B`, the overhead is `~1 bit` per symbol)

```
## Huffman code for blocks
block_size: 1, entropy: 0.47, avg_codelen: 1.00 bits/symbol
block_size: 2, entropy: 0.47, avg_codelen: 0.65 bits/symbol
block_size: 3, entropy: 0.47, avg_codelen: 0.53 bits/symbol
block_size: 4, entropy: 0.47, avg_codelen: 0.49 bits/symbol
block_size: 5, entropy: 0.47, avg_codelen: 0.48 bits/symbol
```

Huffman codes on blocks

```
block_size: 1, entropy: 0.47, avg_codelen: 1.00 bits/symbol

  |--A
·-|
  |--B
block_size: 2, entropy: 0.47, avg_codelen: 0.65 bits/symbol

       |--BA
  |--·-|
  |    |    |--AA
  |    |--·-|
  |         |--AB
·-|
  |--BB
block_size: 3, entropy: 0.47, avg_codelen: 0.53 bits/symbol

                      |--AAA
                 |--·-|
                 |    |--BAA
            |--·-|
            |    |    |--ABA
            |    |--·-|
            |         |--AAB
       |--·-|
       |    |--ABB
  |--·-|
  |    |    |--BBA
  |    |--·-|
  |         |--BAB
·-|
  |--BBB
```

Huffman codes on blocks

```
block_size: 5, entropy: 0.47, avg_codelen: 0.48 bits/symbol

            |--BABBB
       |--.-|
       |    |              |--BBABA
       |    |         |--.-|
       |    |         |    |--ABBBA
       |    |    |--.-|
       |    |    |    |    |--BABBA
       |    |    |    |--.-|
       |    |    |         |--AABBB
       |    |--.-|
       |         |                        |--BBAAA
       |         |                   |--.-|
       |         |                   |    |--AABAB
       |         |              |--.-|
       |         |              |    |    |--AAABB
       |         |              |    |--.-|
       |         |              |         |--ABBAA
       |         |         |--.-|
       |         |         |    |         |--AABBA
       |         |         |    |    |--.-|
       |         |         |    |    |    |              |--AAAAB
       |         |         |    |    |    |         |--.-|
       |         |         |    |    |    |         |    |--AABAA
       |         |         |    |    |    |    |--.-|
       |         |         |    |    |    |    |    |         |--AAAAA
       |         |         |    |    |    |    |    |    |--.-|
       |         |         |    |    |    |    |    |    |    |--BAAAA
       |         |         |    |    |    |    |    |--.-|
       |         |         |    |    |    |    |         |    |--ABAAA
       |         |         |    |    |    |    |         |--.-|
       |         |         |    |    |    |    |              |--AAABA
       |         |         |    |    |    |--.-|
       |         |         |    |    |         |--BAAAB
       |         |         |    |--.-|
       |         |         |         |         |--BABAA
       |         |         |         |    |--.-|
       |         |         |         |    |    |--ABABA
       |         |         |         |--.-|
       |         |         |              |    |--BAABA
       |         |         |              |--.-|
       |         |         |                   |--ABAAB
       |         |    |--.-|
       |         |    |    |    |--ABBAB
       |         |    |    |--.-|
       |         |    |         |--BABAB
       |         |--.-|
       |              |         |--BBAAB
       |              |    |--.-|
       |              |    |    |--BAABB
       |              |--.-|
       |                   |    |--ABABB
       |                   |--.-|
       |                        |--BBBAA
  |--.-|
  |    |         |--BBBAB
  |    |    |--.-|
  |    |    |    |--BBBBA
  |    |--.-|
  |         |    |--BBABB
  |         |--.-|
  |              |--ABBBB
.-|
  |--BBBBB
```

Huffman codes on blocks

1. Huffman codes

$$H(X) \leq \mathbb{E}[l(X)] \leq H(X) + 1$$

2. Huffman codes on blocks of size B

$$H(X) \leq \frac{\mathbb{E}[l(X_1^B)]}{B} \leq H(X) + \frac{1}{B}$$

Huffman codes on blocks

if $H(X)$ is small

$\frac{1}{B} \gg H(X)$

$H(X) = 0.01$

$B \approx 100$

for $2\times$ entropy

1. Convergence to entropy $H(X)$ is quite fast -> `1/B`
2. But, not very practical, as the codebook size needed is large (exponential):

$$size = |\mathcal{X}|^B$$

3. *Larger codebook* -> difficult to handle,
   block size limited by device memory,
   higher latency, ... → not streaming

boundary conditions (data is not multiple of B)

Arithmetic coding

	Block coding	Arithmetic coding
Upper bound	$H(x) + 1/n$	$H(x) + 2/n$
Complexity	$O(2^n)$	$O(n)$

1. For data x_1^n, the `block_size = n`

   i.e. the entire data is a single block!

2. Codeword is computed on *on the fly*

   No need to pre-compute the codebook beforehand

3. Very Efficient! -> *theoretically* the performance can be shown to be:

$$H(X) \leq \frac{\mathbb{E}[l(X_1^n)]}{n} \leq H(X) + \frac{2}{n}$$

i.e. `~2 bits` of overhead for the *entire* sequence

How does Arithmetic coding work?

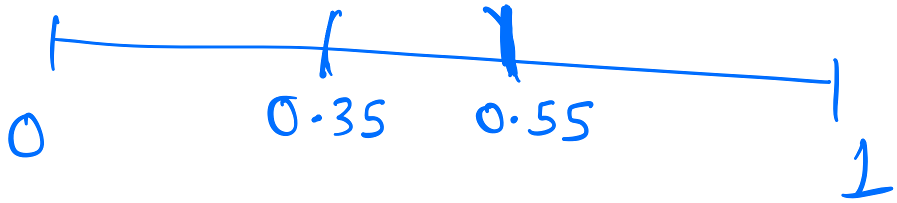
Arithmetic Encoding

1. **STEP I**: Find an *interval* (or a *range*) `[L,H)`, corresponding to the *entire sequence* x_1^n

```
x_input -> |-----------------[.....)------------|
           0                low      high          1
```

2. **STEP II**: Communicate the *interval* `[L,H)` *efficiently*
   (i.e. using less number of bits)

```
x_input -> [L,H) -> 011010
```



0.355555....

0.41786....

$\left.\begin{array}{l}0.5\\0.4\end{array}\right\} \to$ easy to describe

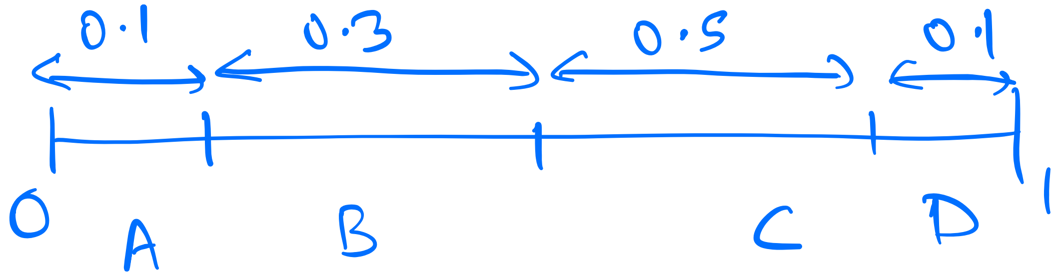
$(0.3567, 0.3568)$

0.35675

1 Shorter intervals

$\Downarrow$

take more bits
to describe



$P = \{A: 0.1, B: 0.3, C: 0.5, D: 0.1\}$

2 → Map sequence to interval with length = P(sequence)

① + ② $\Rightarrow$ More probable sequence

$\Downarrow$

Bigger interval

$\Downarrow$

Shorter bit length

Primer: the number line (in binary)

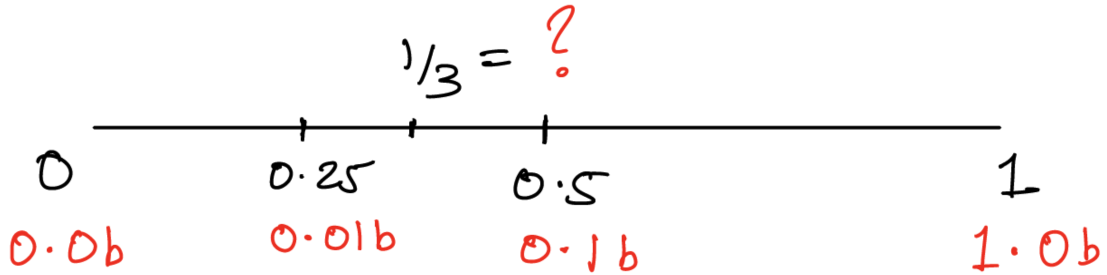
$0.1b = 1/2$

$0.01b = \frac{1}{2^2}$

$\vdots$

$0.11b = \frac{1}{2} + \frac{1}{2^2}$

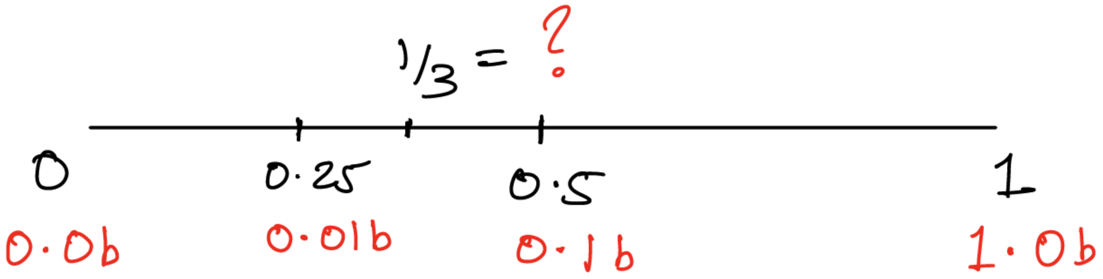
$= 3/4$




```
# floating point values in binary
0.3333 = # b0.... ?
0.6666 = #?
```

$\longrightarrow$ b0.0101...

Primer: the number line (in binary)




```
# floating point values in binary
from utils.bitarray_utils import float_to_bitarrays
_, binary_exp = float_to_bitarrays(0.3333333, 20)

0.3333 = b0.010101...
0.6666 = b0.101010...
```

Arithmetic Encoding

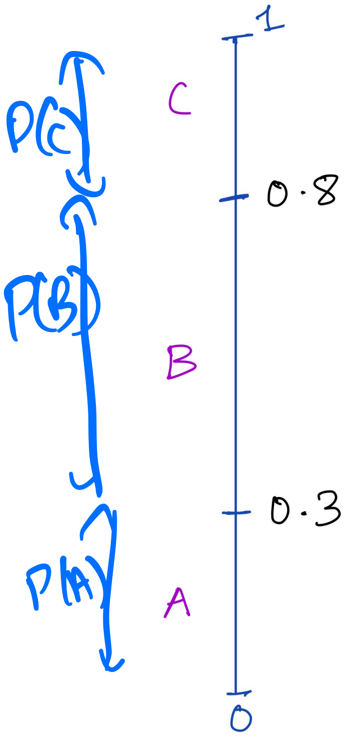
We will consider the following running example:

```
P = ProbabilityDist({A: 0.3, B: 0.5, C: 0.2})
x_input = BACB
```

We want to encode the sequence $x_1^n = BACB$ sampled from the distribution P.

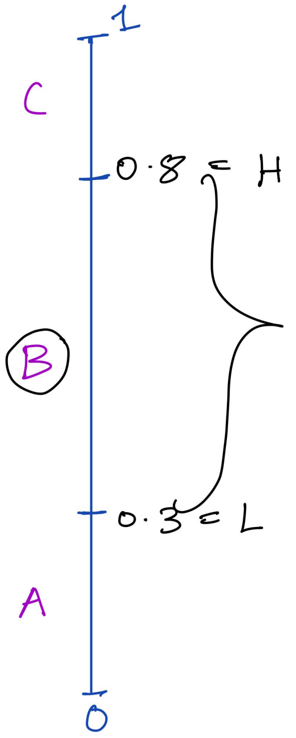
Arithmetic coding example

$P = \{A: 0.3,\ B: 0.5,\ C: 0.2\}$, $X_1^n = BACB$



Arithmetic coding example

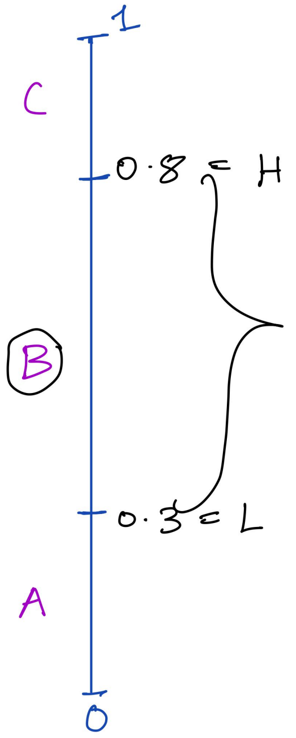
$P = \{A: 0.3,\ B: 0.5,\ C: 0.2\}$, $X_1^n = BACB$



B → Interval $[0.3, 0.8)$

Arithmetic coding example

$P = \{A: 0.3,\ B: 0.5,\ C: 0.2\}$, $X_1^n = BACB$



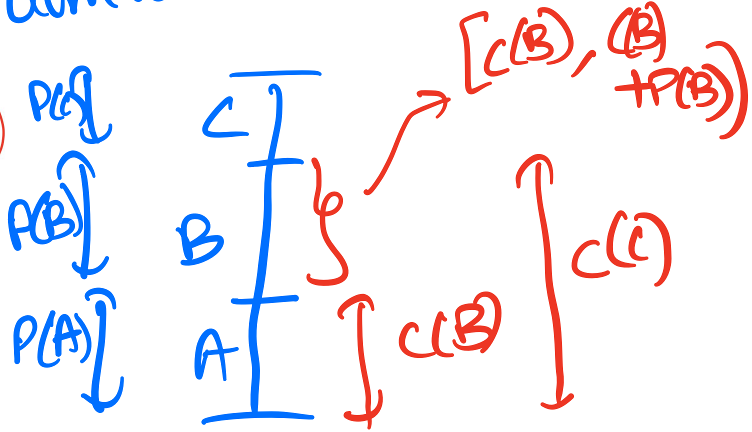
B → Interval $[0.3, 0.8)$

	A	B	C
P :	0.3	0.5	0.2
C :	0.0	0.3	0.8

Probabilities

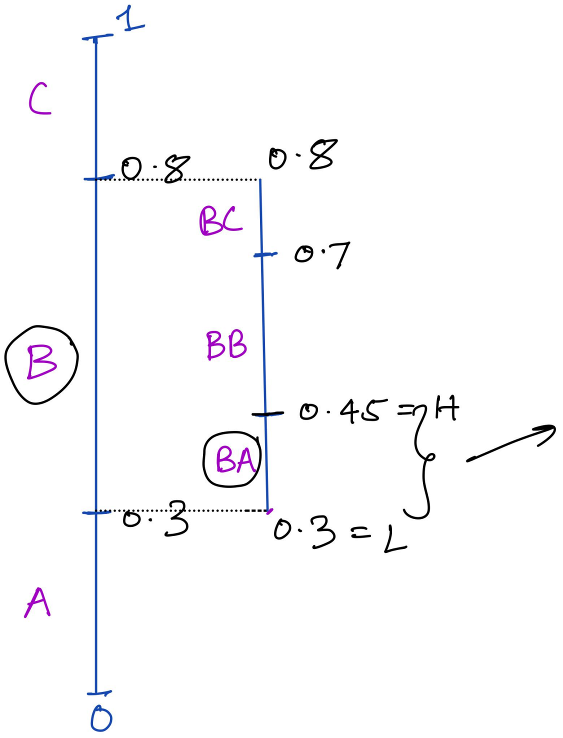
Cumulative Prob.

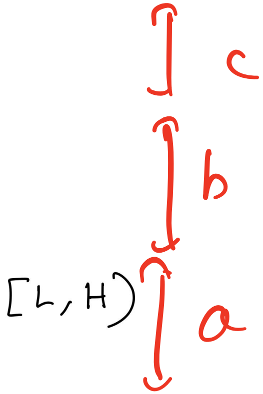
$[L, H) = [C(B),\ C(B) + P(B))$



Arithmetic coding example

$P = \{A : 0.3,\ B : 0.5,\ C : 0.2\}$, $X_1^n = BACB$





$B \rightarrow [0.3, 0.8)$

$BA \rightarrow [0.3, 0.45)$

$a : b : c = 0.3 : 0.5 : 0.2$

B: ~~0.5~~ $[0.3, 0.8)$

$\downarrow$

length $= 0.5$

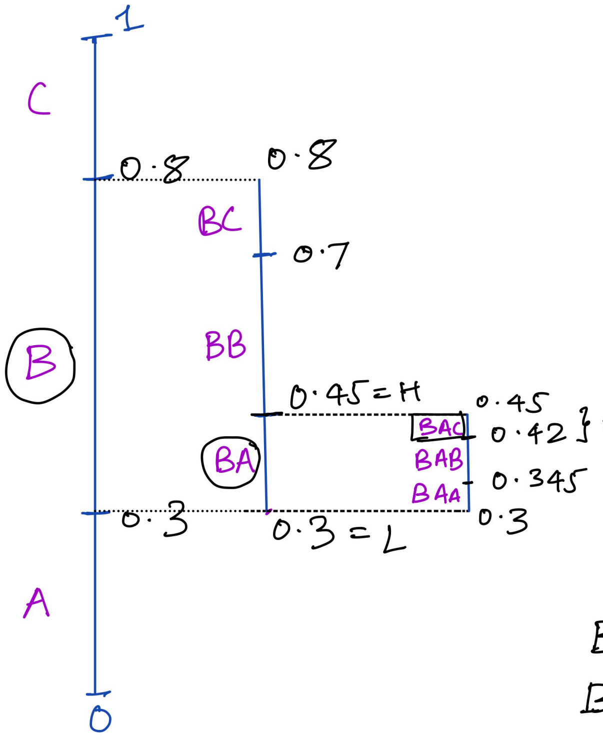
$BA \rightarrow 0.5 \times 0.3$

$= 0.15$

$BA : [0.3, 0.3 + 0.15]$

Arithmetic coding example

$P = \{A: 0.3,\ B: 0.5,\ C: 0.2\}$, $X_1^n = BACB$



$\Longrightarrow [L, H) = [0.42, 0.45)$

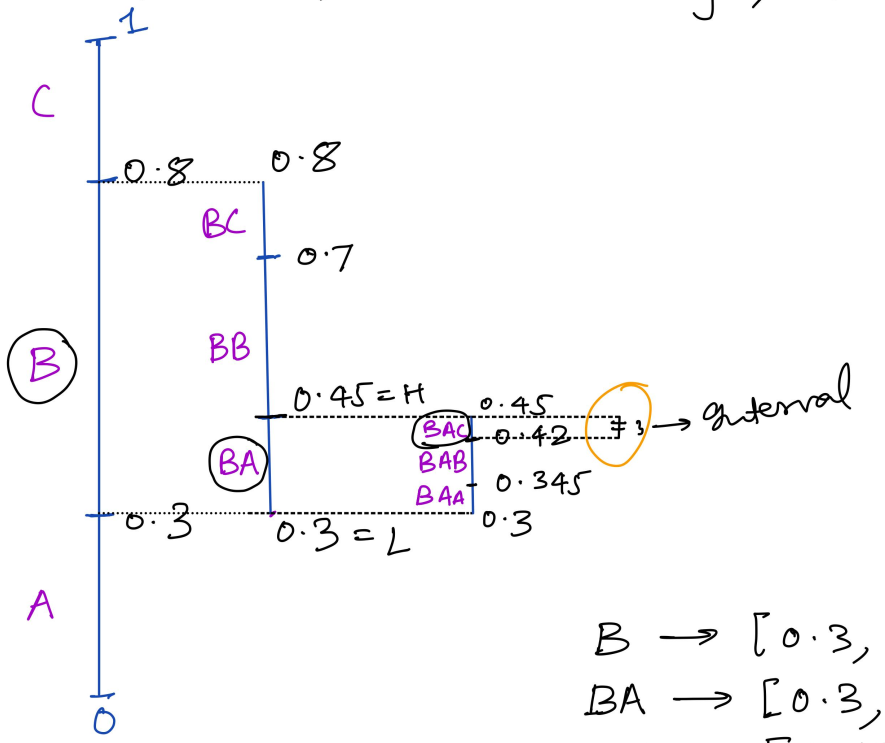
$B \rightarrow [0.3, 0.8)$

$BA \rightarrow [0.3, 0.45)$

$BAC \rightarrow [0.42, 0.45)$

Arithmetic coding example

$P = \{A: 0.3,\ B: 0.5,\ C: 0.2\}$, $X_1^n = BACB$



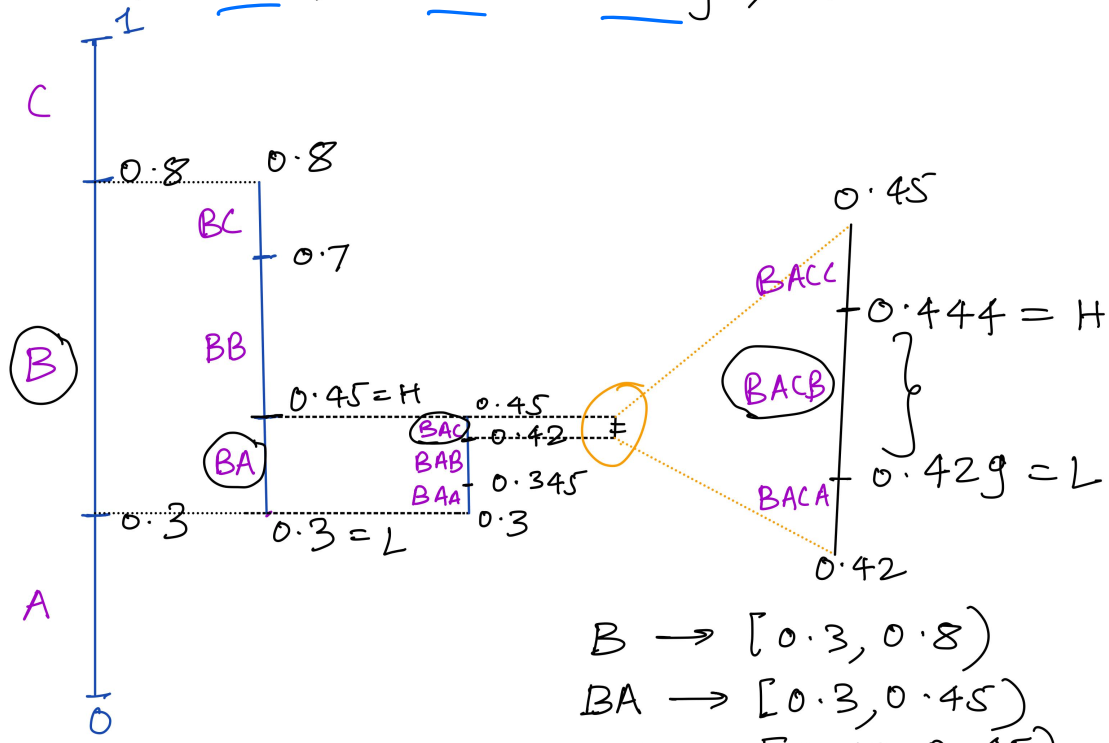
$B \rightarrow [0.3, 0.8)$

$BA \rightarrow [0.3, 0.45)$

$BAC \rightarrow [0.42, 0.45)$

Arithmetic coding example

$P = \{A: 0.3,\ B: 0.5,\ C: 0.2\}$, $X_1^n = BACB$



$B \rightarrow [0.3, 0.8)$

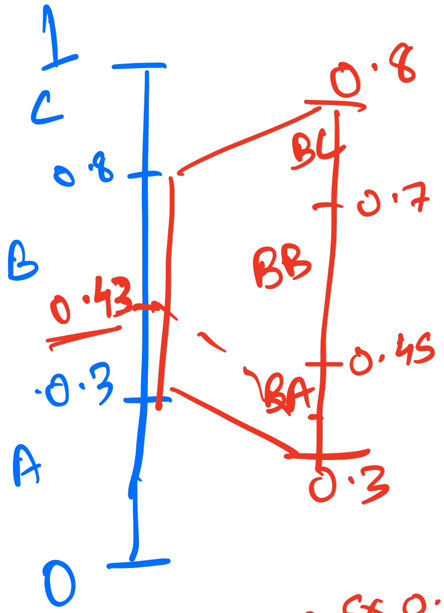
$BA \rightarrow [0.3, 0.45)$

$BAC \rightarrow [0.42, 0.45)$

$BACB \rightarrow [0.429, 0.444)$ $\rightarrow$ 0.43

Decoding

A : 0.3, B: 0.5, C: 0.2



0.5×0.3

B A

When to stop decoding?

→ Prepend the length "n" at the start

→ Create a special symbol called EOF (end of file), assign a probability, encode {sequence <EOF>}

Sequence

↓

Interval → Bits

↓

Interval

↓

Sequence

TODO:

1.

2. Calculate #bits

Arithmetic coding example

1. **STEP I**: Find an *interval* (or a *range*) `[L,H)`, corresponding to the *entire sequence* x_1^n

```
prob = ProbabilityDist({A: 0.3, B: 0.5, C: 0.2})
x_input = BACB

# find interval corresp to BACB
ENCODE: B -> [L,H) = [0.30000,0.80000)
ENCODE: A -> [L,H) = [0.30000,0.45000)
ENCODE: C -> [L,H) = [0.42000,0.45000)
ENCODE: B -> [L,H) = [0.42900,0.44400)
```

$P(B) = 0.5$

$P(B)P(A) = 0.5 \times 0.3$

$0.5 \times 0.3 \times P(C)$

Thus, the final interval is: `x_input -> [0.429, 0.444)`

Arithmetic coding pseudo-code

```
class ArithmeticEncoder:
    ...

    def shrink_range(self, L, H, s):
        rng = H - L
        new_L = L + (rng * self.P.cumul[s])
        new_H = new_L + (rng * self.P.probs(s))
        return new_L, new_H

    def find_interval(self, x_input):
        L,H = 0.0, 1.0
        for s in x_input:
            L,H = self.shrink_range(L,H,s)
        return L,H

    def encode_block(self, x_input):
        # STEP1
        L,H = self.find_interval(x_input)

        # STEP-II
        ...
```

prob. of current symbol

previous range

*

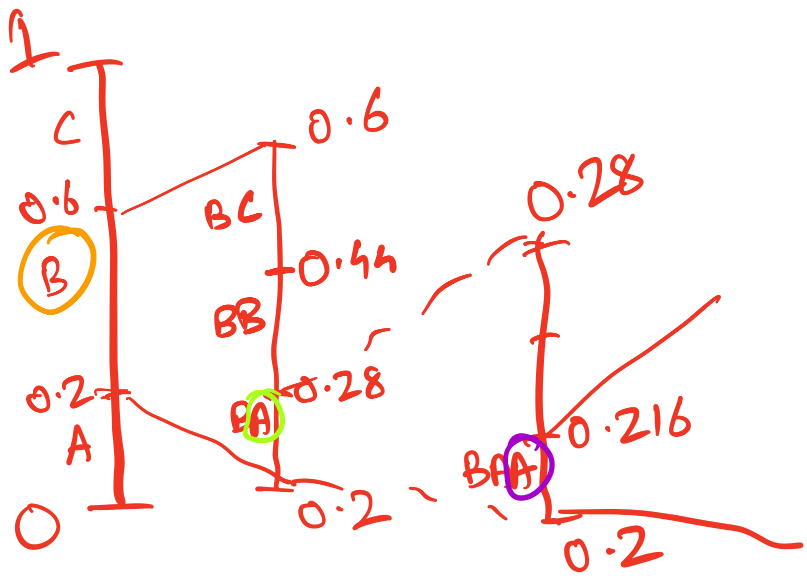
next range

Arithmetic coding example-2

```
P = {A: 0.2, B: 0.4, C: 0.4}
x_input = BAAB
```

$P = \{A : 0.2,\ B = 0.4,\ C = 0.4\}$

seq: BAAB



BA - length
$= 0.4 \times 0.2$
$= 0.08$

$BB = 0.4 \times 0.4$
$= 0.16$

$BC = 0.16$

BAA → length
$= \text{length}(BA) \times P(A)$
$= 0.08 \times 0.2$
$= 0.016$

Arithmetic coding example:2

```
P = {A: 0.4, B: 0.4, C: 0.2}
x_input = BACA
```

Probability distribution changed out of nowhere

```
ENCODE: B -> [L,H) = [0.40000,0.80000)
ENCODE: A -> [L,H) = [0.40000,0.56000)
ENCODE: C -> [L,H) = [0.52800,0.56000)
ENCODE: A -> [L,H) = [0.52800,0.54080)
```

STEP-I: Find the interval [L,H)

```
P = {A: 0.3, B: 0.5, C: 0.2}
x_input = BACB
L = [0.429, 0.444)
```

1. **Observation**: Interval size reduces as we encode more symbols
2. **QUIZ-1**: What is the size of the interval (`H-L`) for the input X_1^n?

$$P(x_1^n) = P(x_1)P(x_2)\cdots P(x_n)$$

STEP-I: Find the interval [L,H)

```
P = {A: 0.3, B: 0.5, C: 0.2}
x_input = BACB
L = [0.429, 0.444)
```

1. **Observation**: Interval size reduces as we encode more symbols
2. **QUIZ-1**: What is the size of the interval (`H-L`) for the input X_1^n?

$$
\begin{aligned}
(H-L) &= p(x_1) * p(x_2) \ldots p(x_n) \\
&= \prod_{i=1}^{n} p(x_i) \\
&= p(x_1^n)
\end{aligned}
$$

Arithmetic Encoding

$H = 0.444$

$Z = \frac{L+H}{2}$

$L = 0.429$

```
P = {A: 0.3, B: 0.5, C: 0.2}
x_input = BACB
L = [0.429, 0.444)
```

1. **STEP-I**: Find an *interval* (or a *range*) `[L,H)` corresponding to the *entire sequence* x_1^n
2. **STEP-II**: Communicate the interval $[L, H)$ using a value $Z \in [L, H)$

For example: $Z = \frac{(L+H)}{2}$, i.e. the midpoint of the range.
(in our example `Z = 0.4365`)

Arithmetic decoding

Quiz-2: If the decoder knows:

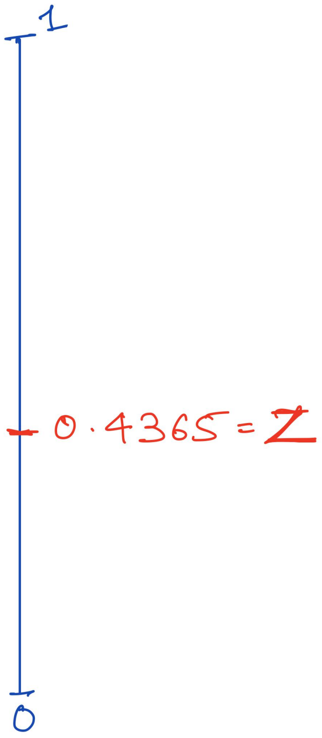
- `n=4`
- `P = {A: 0.3, B: 0.5, C: 0.2}`
- `Z = 0.4365`

How can it decode the entire input sequence? X_1^n.

Arithmetic decoding - example

$P = \{A: 0.3,\ B: 0.5,\ C: 0.2\}$, $Z = 0.4365$

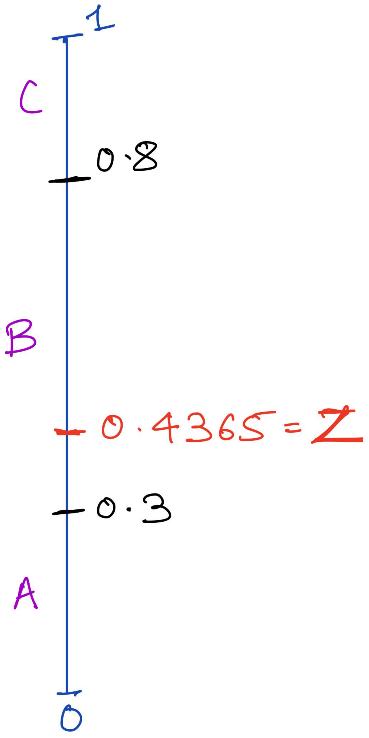
$n = 4$



Arithmetic decoding - example

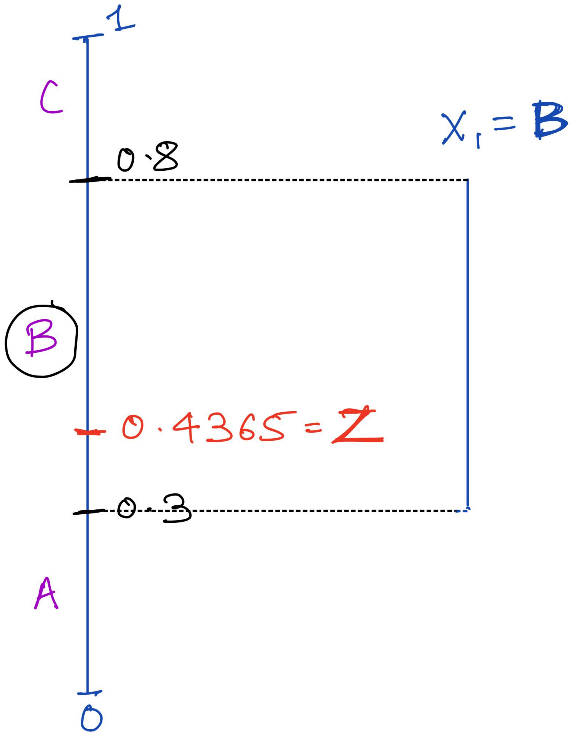
$P = \{A: 0.3,\ B: 0.5,\ C: 0.2\},\ Z = 0.4365$

$n = 4$



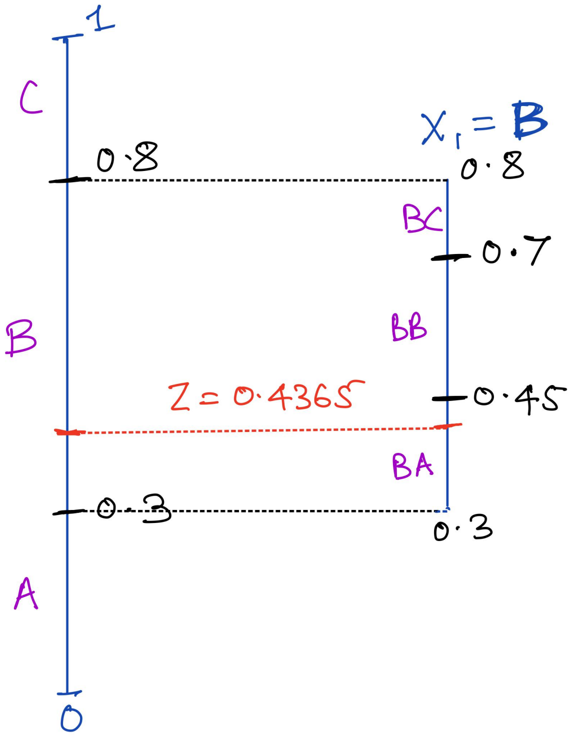
Arithmetic decoding - example

$P = \{A: 0.3, B: 0.5, C: 0.2\}$, $Z = 0.4365$
$n = 4$



Arithmetic decoding - example

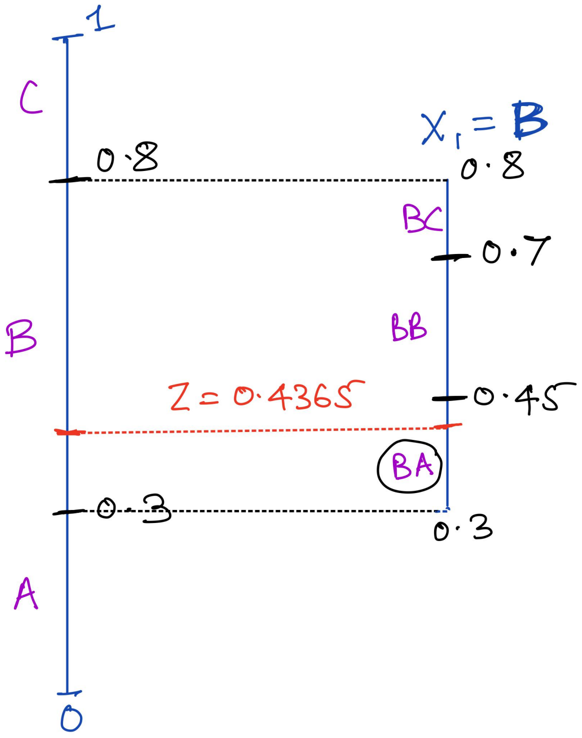
$P = \{A: 0.3,\ B: 0.5,\ C: 0.2\}$, $Z = 0.4365$
$n = 4$



Arithmetic decoding - example

$P = \{A: 0.3,\ B: 0.5,\ C: 0.2\}$, $Z = 0.4365$

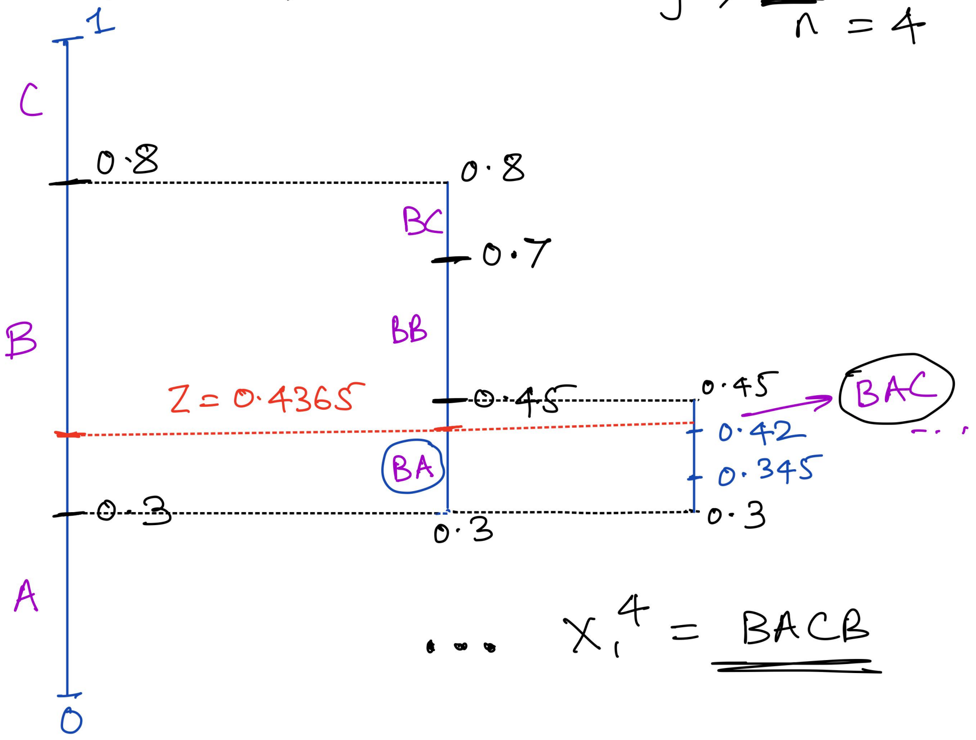
$n = 4$



Arithmetic decoding - example

$P = \{A: 0.3,\ B: 0.5,\ C: 0.2\}$, $Z = 0.4365$

$n = 4$



$\ldots \quad X_1^4 = \underline{BACB}$

Arithmetic decoding

```
Z = 0.4365
ENCODE: B -> [L,H) = [0.30000,0.80000)
ENCODE: A -> [L,H) = [0.30000,0.45000)
ENCODE: C -> [L,H) = [0.42000,0.45000)
ENCODE: B -> [L,H) = [0.42900,0.44400)
------------------------------
DECODE: B -> [L,H) = [0.30000,0.80000)
DECODE: A -> [L,H) = [0.30000,0.45000)
DECODE: C -> [L,H) = [0.42000,0.45000)
DECODE: B -> [L,H) = [0.42900,0.44400)
```

Arithmetic decoding-pseudocode

```python
class ArithmeticDecoder:
    ...
    def shrink_range(self, L, H, s):
        ...
        return new_L, new_H

    def decode_symbol(self, L, H, Z):
        rng = H - L
        search_list = L + (self.P.cumul * rng)
        symbol_ind = np.searchsorted(search_list, Z)
        return self.P.alphabet[symbol_ind]

    def decode_block(self, Z, n):
        L,H = 0.0, 1.0
        for _ in range(n): #main decoding loop
            s = self.decode_symbol(L, H, Z)
            L,H = self.shrink_range(L,H,s)
```

Arithmetic decoding:

Quiz-2: If the decoder knows:

- `n=4`
- `P = {A: 0.3, B: 0.5, C: 0.2}`
- `Z = 0.4365`

Ans -> The decoder creates intervals same as the ones encoder creates, and find which symbol corresponds to the interval in which `Z` lies.

$\frac{1}{3} = 0.3333\ldots$

Arithmetic encoding

1. **STEP-I**: Find an *interval* (or a *range*) `[L,H)` corresponding to the *entire sequence* x_1^n (`[0.429, 0.444]`)
2. **STEP-II**: Find the midpoint of the interval $[L, H)$, $Z = \frac{(L+H)}{2}$ (`Z =0.4365`)
3. **STEP-III:** Write the binary expansion of Z to the bitstream ->

   eg: `Z = 0.4365 = b0.01101111101...`

   then the final `encoded_bitstream = 01101111101...`

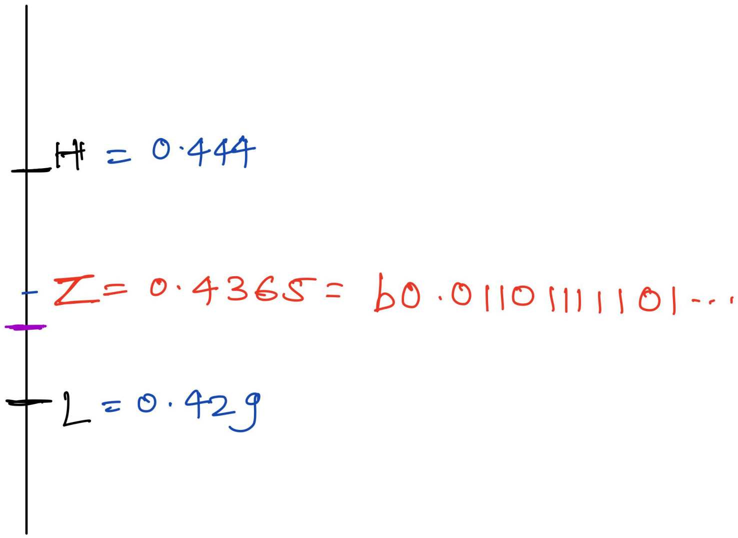
Arithmetic encoding

1. **STEP-I**: Find an *interval* (or a *range*) `[L,H)` corresponding to the *entire sequence* x_1^n (`[0.429, 0.444]`)
2. **STEP-II**: Find the midpoint of the interval $[L, H), Z = \frac{(L+H)}{2}$. (`Z =0.4365`)
3. **STEP-III:** Write the binary expansion of Z to the bitstream ->
   eg: `Z = 0.4365 = b0.01101111101...`
   then the final `encoded_bitstream = 01101111101...`

Quiz-4: Z's binary representation can be long, can also have infinite bits.
How can we fix this?

Communicating the interval `[L,H)`

Communicating Z



Communicating the interval `[L,H)`

Communicating Z

$H = 0.444$

$Z = 0.4365 = b0.01101111101\ldots$

$\hat{Z} = b0.0110111 \approx 0.4296$

$L = 0.429$

encoded bits: $\boxed{0110111}$

We just need a $\hat{Z}$ such the $\hat{Z} \in [L, H)$

$\hat{Z}$ should have a short binary representation.

Arithmetic coding example:

1. **STEP-I**: Find an *interval* (or a *range*) $[L, H)$ corresponding to the *entire sequence* x_1^n (`[0.429, 0.444]`)
2. **STEP-II**: Find the midpoint of the interval $[L, H)$, $Z = \frac{(L+H)}{2}$. (`Z =0.4365`)
3. **STEP-III:** Truncate Z to k bits $(\hat{Z})$

   e.g:

```
L,H = 0.429, 0.444
Z = 0.4365 = b0.01101111101...
Z_hat = b0.011011111 ~ 0.4296
```

Final Encoding = `encoded_bitstream = 011011111`

Communicating the interval `[L,H)`

1. **Cond 1:** Truncate Z to $\hat{Z}$ with k bits, so that $\hat{Z} \in [L, H)$
2. **Cond 2:** If $\hat{Z}$ has binary representation: `Z_hat = b0.011011111` for example, then we also need, any extension of it $Z_{ext} \in [L, H)$.

$\hat{Z} \in [L, H)$

For eg:

```
Z_hat = b0.011011111
Z_ext = b0.011011111011110101..
```

Quiz-5: Why so?

Communicating the interval `[L,H)`

1. **Cond 1:** Truncate Z to $\hat{Z}$ with k bits, so that $ \hat{Z} \in [L,H)$
2. **Cond 2:** If $\hat{Z}$ has binary representation: `Z_hat = b0.011011111` for example, then we also need, any extension of it $Z_{ext} \in [L, H)$.

The two conditions can be written together as:

$$[\hat{Z}, \hat{Z} + 2^{-k}) \in [L, H)$$

Communicating the interval `[L,H)`

Given the interval $[L, H)$, and $Z = \frac{(L+H)}{2}$, truncate Z to k bits so that:

$$[\hat{Z}, \hat{Z} + 2^{-k}) \in [L, H)$$

Quiz-6: What should the k be?

```
## Examples
# Ex1: L=0.429, H=0.444, Z = 0.4365.. how many digits we can truncate from Z?

## Ex2: L=0.552398714,H=0.5524123
Z = 0.5524058..., how many digits we can truncate Z from?
```

Communicating the interval `[L,H)`

Given the interval $[L, H)$, and $Z = \frac{(L+H)}{2}$, truncate Z to k bits so that:

$$[\hat{Z}, \hat{Z} + 2^{-k}) \in [L, H)$$

Quiz-6: What should the k be?

- Shorter the interval, $|H - L|$, the more the number of bits we need to use.

Communicating the interval `[L,H)`

Given the interval $[L, H)$, and $Z = \frac{(L+H)}{2}$, truncate Z to k bits so that:

$$[\hat{Z}, \hat{Z} + 2^{-k}) \in [L, H)$$

Quiz-6: What should the k be?

- Shorter the interval, $|H - L|$, the more the number of bits we need to use.
- the numbers of bits we need to truncate Z by is:

$$k \leq \left\lceil log_2 \frac{1}{(H-L)} \right\rceil + 1$$

$[0.35, 0.45) \to L = 0.1$

$\underline{\underline{0.4}} \in [0.35, 0.45)$

$[0.355, 0.365) \to L = 0.01$

$0.36 \in [0.355, 0.365)$

$[0.3555, 0.3565) \to len = 0.001$

$0.356 \in [0.3 \ldots)$

bits needed $\approx \log_2 \frac{1}{|H-L|}$

Arithmetic Encoding pseudo-code

```python
class ArithmeticEncoder:
    def shrink_range(self, L, H, s):
        ...
    def find_interval(self, x_input):
        L,H = 0.0, 1.0
        for s in x_input:
            L,H = self.shrink_range(L,H,s)
        return L,H

    def encode_block(self, x_input):
        # STEP-1 find interval
        L,H = self.find_interval(x_input)

        # STEP-II,III communicate interval
        Z = (L+H)/2
        num_bits = ceil(log2((H-L))) + 1
        _, code = float_to_bitarray(Z, num_bits)
        return code
```

Arithmetic decoding-pseudocode

```
class ArithmeticDecoder:
    ...
    def shrink_range(self, L, H, s):
        ...

    def decode_symbol(self, L, H, Z):
        ...

    def decode_block(self, code, n):
        Z = bitarray_to_float(code)

        # start decoding
        L,H = 0.0, 1.0
        for _ in range(n): #main decoding loop
            s = self.decode_symbol(L, H, Z)
            L,H = self.shrink_range(L,H,s)

        # add code to remove additional bits read
```

Arithmetic coding compression performance:

- Size of interval $H - L = \log_2 1p(x_1^n)$
- $k \leq \log_2 \frac{1}{H-L} + 2$

Quiz-7: What is the codelength for arithmetic coding?

Arithmetic coding compression performance:

- Size of interval $H - L = \log_2 1p(x_1^n)$
- $k \leq \log_2 \frac{1}{H-L} + 2$

Quiz-7: What is the codelength for arithmetic coding?

$$codelen = k \leq \log_2 \frac{1}{p(x_1^n)} + 2$$

Arithmetic coding compression performance:

- Size of interval $H - L = \log_2 1 p(x_1^n)$
- $k \leq \log_2 \frac{1}{H-L} + 2$

Quiz-7: What is the codelength for arithmetic coding?

$$codelen = k \leq \log_2 \frac{1}{p(x_1^n)} + 2$$

Thus, Arithmetic coding is within `2` bits of the optimal on the *ENTIRE* sequence!

$$\mathbb{E}[codelen] \leq \mathbb{E} \log_2 \frac{1}{P(x^n)} + 2 = n\, \mathbb{E} \log_2 \frac{1}{P(x)} + 2$$

$$= n H(X) + 2$$

Arithmetic coding compression performance:

THEOREM: Arithmetic coding achieves average codelength:

$$H(X) \leq \frac{\mathbb{E}[l(X_1^n)]}{n} \leq H(X) + \frac{2}{n}$$

Arithmetic coding Summary

1. Given *any* distribution P, achieves *optimal* compression. Thus, Arithmetic coding allows for `model` and `entropy coding` separation.
2. Encoding, decoding is linear time and quite efficient!
3. As we are not saving a large codebook, memory requirements are not very high

   Don't even need to compute $P(x^n)$
4. Can work very well with changing distribution P.

   i.e. Adaptive algorithms work well with Arithmetic coding

   lecture 8

Arithmetic coding in practice

Quiz-8: Whar are the practical issues with our Arithmetic encoding/decoding?

Hint ->

```
prob = ProbabilityDist({A: 0.3, B: 0.5, C: 0.2})
x_input = BACBBCCBA

# find interval corresp to BACB
ENCODE: B -> [L,H) = [0.30000,0.80000)
ENCODE: A -> [L,H) = [0.30000,0.45000)
ENCODE: C -> [L,H) = [0.42000,0.45000)
ENCODE: B -> [L,H) = [0.42900,0.44400)
ENCODE: C -> [L,H) = [0.44100,0.44400)
ENCODE: C -> [L,H) = [0.44340,0.44400)
ENCODE: B -> [L,H) = [0.44358,0.44388)
ENCODE: A -> [L,H) = [0.44358,0.44367)
```

Arithmetic coding in practice

Quiz-8: What are the practical issues with our Arithmetic encoding/decoding?

Ans -> The interval becomes too small very quickly and we run out of bits to represent `L,H`.

```
prob = ProbabilityDist({A: 0.3, B: 0.5, C: 0.2})
x_input = BACBBCCBA

# find interval corresp to BACB
ENCODE: B -> [L,H) = [0.30000,0.80000)
ENCODE: A -> [L,H) = [0.30000,0.45000)
ENCODE: C -> [L,H) = [0.42000,0.45000)
ENCODE: B -> [L,H) = [0.42900,0.44400)
...
```

0.444
0.429
already know first digit is 4

Arithmetic coding in practice

Quiz-9: What can we do to avoid the interval `[L,H)` from getting too small?

Hint ->

```
L = 0.429 = b0.0110110...
H = 0.444 = b0.01110001...
```

0.44
0.29

Arithmetic coding in practice

Quiz-9: What can we do to avoid the interval `[L,H)` from getting too small?

Idea: If `L`, `H` start with `011` then any value lying inside the interval `[L,H)` also will start with `011`!

```
L = 0.429 = b0.0110110...
H = 0.444 = b0.01110001...
Z_hat = b0.011...
```

Arithmetic coding in practice

Quiz-9: What can we do to avoid the interval `[L,H)` from getting too small?

Idea: If `L`, `H` start with `011` then any value lying inside the interval `[L,H)` also will start with `011`!

Rescale:: Already output bits `011`, and rescale `L,H`

```
L = 0.429 = b0.0110110...
H = 0.444 = b0.01110001...

Rescaled: L=0.8580, H=0.8880, bitarray='0'
Rescaled: L=0.7160, H=0.7760, bitarray='01'
Rescaled: L=0.4320, H=0.5520, bitarray='011'
ENCODE: B -> [L,H) = [0.42900,0.44400)
```

Arithmetic Encoding with rescaling

```
class ArithmeticEncoder:
    def shrink_range(self, L, H, s):
        ...
    def rescale_range(self, L, H):
        ...
    def find_interval(self, x_input):
        L,H, bitarray = 0.0, 1.0, Bitarray("")
        for s in x_input:
            L,H = self.shrink_range(L,H,s)
            L,H, bits = self.rescale_range(L,H)
            bitarray += bits
        return L,H, bitarray
```

Arithmetic Encoding with rescaling

```python
def rescale_range(self, L, H):
    bitarray = ""
    while (L >= 0.5) or (H < 0.5):
        if (L < 0.5) and (H < 0.5):
            bitarray+= "0"
            L,H = L*2, H*2
        elif ((L >= 0.5) and (H >= 0.5)):
            bitarray += "1"
            L,H = (L - 0.5)*2, (H - 0.5)*2
    return L, H, bitarray
```

Arithmetic Encoding with rescaling

Rescale:: Already output bits which are same between L, H (`011`), and rescale L, H.

```
L = 0.429 = b0.0110110...
H = 0.444 = b0.01110001...

Rescaled: L=0.8580=b0.11011..., H=0.8880, bitarray='0'
Rescaled: L=0.7160=b0.1011... , H=0.7760, bitarray='01'
Rescaled: L=0.4320=b0.011...  , H=0.5520, bitarray='011'
ENCODE: B -> [L,H) = [0.42900,0.44400)
```

Quiz-10: There is one case in which our algorithm can still have L,H being really close. What is that?

Arithmetic Encoding with rescaling

Lots of Variants of Arithmetic coding; mainly come from how they implement the rescaling.

1. **Arithmetic coding:** Bit-based rescaling -> keeping a count of the mid-ranges etc.
   SCL arithmetic coder
2. **Range Coding** Byte (8-bit based rescaling), word-based rescaling
   SCL range coder
3. Variants on the above based on how compressors handle the edge case (L starts with `b0.0` and H starts with `b0.1..`, but the interval is very small)

Arithmetic/Range coders in practice

Used almost everywhere! (either as Range coder or Arithmetic coding)

1. JPEG2000, BPG, H265, H266, VP8 → second half
2. CMIX, tensorflow-compress, NNCP etc. → lec. 9

What are the problems with Arithmetic coding

Although Arithmetic coding algorithms are very fast, they are not fast enough!
(especially when compared with Huffman coding)

Codec	Encode speed	Decode speed	Compression
Huffman coding	252 MB/s	300 MB/s	1.66
Arithmetic coding	120 MB/s	69 MB/s	1.24

NOTE -> Speed numbers from: Charles Bloom's blog

Beyond Arithmetic coding

Codec	Encode speed	Decode speed	Compression
Huffman coding	252 MB/s	300 MB/s	1.66
Arithmetic coding	120 MB/s	69 MB/s	1.24
rANS	76 MB/s	140 MB/s	1.24
tANS	163 MB/s	284 MB/s	1.25

NOTE -> Speed numbers from: Charles Bloom's blog

Next Class -> ANS: Asymmetric Numeral Systems