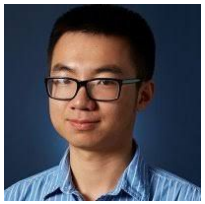
CS294-158 Deep Unsupervised Learning

Lecture 4 (b) Bits-back Coding

Pieter Abbeel, Peter Chen, Jonathan Ho, Aravind Srinivas

UC Berkeley

References for this lecture

- Brendan J. Frey and Geoffrey E. Hinton, *"Efficient Stochastic Source Coding and an Application to a Bayesian Network Source Model,"* The Computer Journal 1997

- James Townsend, Thomas Bird, David Barber, "*Practical lossless compression with latent variables using bits back coding,"* ICLR 2019

- Friso Kingma, Pieter Abbeel, Jonathan Ho, *"Bit-Swap: Practical Bits Back Coding with Hierarchical Latent Variables,"*

- Jarek Duda, *"Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding,"* 2007

- GE Hinton, D Van Camp, *"Keeping Neural Networks Simple by Minimizing the Description Length of the Weights"*, 1993

Refresher on Coding

Entropy = expected encoding length when encoding each symbol i with $\log_2 \frac{1}{p(x_i)}$ bits

$$H(X) = \sum_i p(x_i) \log_2 \frac{1}{p(x_i)}$$

Theorem [Shannon 1948] If data source P(X) is an order 0 Markov model, then any compression scheme must use >= H(X) bits per symbol on average

Theorem [Huffman 1952] If data source P(X) is an order 0 Markov model, then the Huffman code uses <= H(X) + 1 bits per symbol on average

Arithmetic Coding (AC): Pays only +2 for entire batch of symbols.

Key Assumptions

Previous slide ***assumes*** we have a model p(x) for which we can:

- Huffman: tractably enumerate all x
- AC: tractably assign intervals to each x

→ Generically: small number of possible values x can take on

When violated?

- When x continuous --- but easily "fixable" (see next slide)
- When x high-D --- challenge we need to address!
  - Key observation:
    - some high-D distributions still allow for convenient coding
    - leverage that to efficiently code mixture models of easy high-D distributions
      key win: mixture models much more expressive then individual components!

Continuous domain for X

A real number x has infinite information, so we can't expect to send x exactly

→ assume discretization with some precision t

→ can discretize in x domain (or, alternatively, in the cumulative distribution domain)

E.g. X ~ N(0,1):

Differential Entropy <> Entropy of discretized variable

$$
\begin{aligned}
H(x_{\text{disc,t}}) &= -\sum_i tp(x_i)\log(tp(x_i)) \\
&= -\sum_i tp(x_i)\log p(x_i) + -\sum_i tp(x_i)\log t \\
&\approx -\int_x p(x)\log p(x)dx + -\log t \int_x p(x)dx \\
&= h(x) - \log t
\end{aligned}
$$

-> matches intuition: making t smaller results in more buckets hence higher entropy

→ can use Huffman coding and AC again

Some high-dimensional P(x) allow for easy coding

X is high dimensional, but P(X) sometimes still easy to encode

- E.g.
  - X ~ N(0,I)
    - can treat as n independent random variables (n = dimensionality of x)
    - each independent variable can be coded efficiently (per previous slide)
  - X autoregressive
    - (see previous lecture)

Mixture Models

Mixture models allow to represent wider range of distributions than their components

E.g. mixture of Gaussians = NOT a Gaussian

Key Question

If P(X) is a mixture model of easily encodable distributions, can we efficiently encode P(X)?

Reminders:

* We'll make our illustrations in 1-D to get the point across in a way that's easy to draw on slides, but keep in mind we are working on something that generalizes to high-D.

* We won't allow ourselves to rely on domain of X being small, in which case the solution is trivial by simply calculating the marginal for each value X can take on, summed over all mixture components and then using Huffman or AC.

Alright, now let's see if we can handle these harder cases!

Mixture Model Running Example

$$P(x) = \sum_i p(i)p(x|i)$$

$p(x|i)$ assumed easy to code for

→ will cost: $\log \frac{1}{p(x|i)}$

Scheme 1: “Max-Mode” Coding

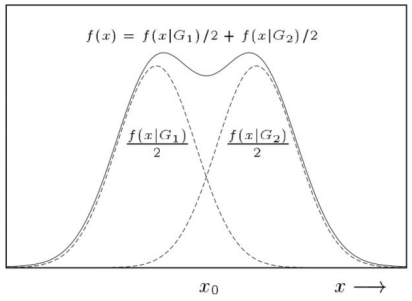
$$P(x) = \sum_i p(i)p(x|i)$$

To code x:

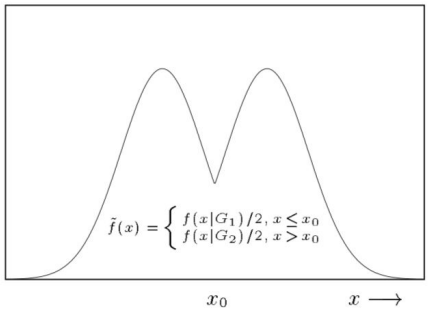
- Find i that maximizes p(i|x)
- Send i --- cost: log 1/p(i)
- Send x --- cost: log 1/p(x|i)

Optimal? NO --- because effectively using different distribution Q(x) ⟶ cost: H(X) + KL(P||Q)

P:



Q:



Scheme 2: “Posterior-Mode-Sampling” Coding

$$P(x) = \sum_i p(i)p(x|i)$$

To code x:

- Sample i from p(i|x)
- Send i --- cost: log 1/p(i) --- why p(i), why not p(i|x)? b/c recipient doesn’t have x
- Send x --- cost: log 1/p(x|i) --- this is probably efficient, but not as efficient as using best i...

Optimal? Yes and No. Yes if we like to send (i, x), b/c we use log 1/p(x,i)

BUT: we are looking to send just x, so overhead of log 1/p(i|x)

Scheme 3: Bits-Back Coding

$$P(x) = \sum_i p(i)p(x|i)$$

To code x:

- Sample i from p(i|x)
- Send i --- cost: log 1/p(i) --- why p(i)? b/c recipient doesn't have x
- Send x --- cost: log 1/p(x|i) --- this is probably efficient, but not as efficient as using best i...

BITS-BACK IDEA (APPROXIMATE INFERENCE):

recipient decodes i, x + knows p(i|x)

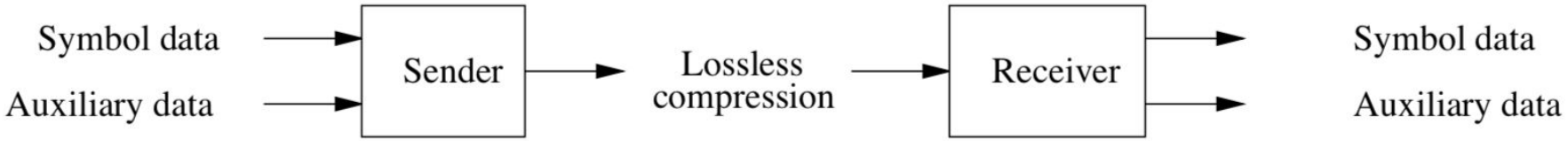
→ can reconstruct the random bits used to sample p(i|x)

→ those random bits were also sent → these are log 1/p(i|x) random bits, which we now don't have to count

→ bits-back coding cost: log 1/p(i) + log 1/p(x|i) - log 1/p(i|x) = log p(i|x) / (p(i) * p(x|i)) = log 1/p(x)

→ Optimal!!!

Bits-Back Coding in a Picture



Assumptions:

(1) We can compute $p(i|x)$

(2) We have auxiliary random data we like to transmit

Bits-Back Coding with Approximate Inference

$$P(x) = \sum_i p(i)p(x|i)$$

To code x:

- Sample i from q(i|x)
- Send i --- cost: log 1/p(i)
- Send x --- cost: log 1/p(x|i)

BITS-BACK IDEA (APPROXIMATE INFERENCE):

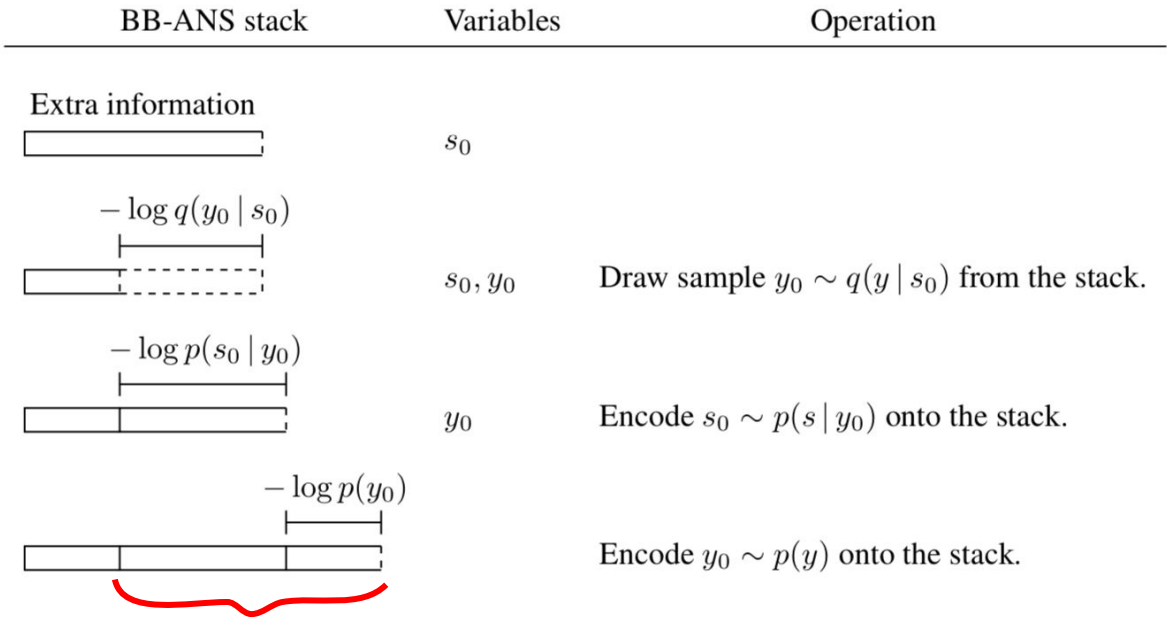
recipient decodes i, x + knows q(i|x)

→ can reconstruct the random bits used to sample q(i|x)

→ those random bits were also sent → these are log 1/q(i|x) random bits, which we now don't have to count

→ bits-back coding cost: log 1/p(i) + log 1/p(x|i) - log 1/q(i|x) = Evidence Lower Bound (ELBO) from VAE

How about that source of random bits we'd like to also send?



this encoding can serve as random bits for the next one

A quick history of bits-back coding

- Wallace, 1990; Hinton & van Camp, 1993
  - Bits-back idea
- Frey & Hinton, 1996
  - Bits-back implementation with arithmetic coding; but not very natural fit
- Duda, 2009
  - Asymmetric Numeral Systems (ANS) coding
- Townsend, Bird & Barber, ICLR 2019
  - Bits Back with ANS (BB-ANS)
- F. Kingma, Abbeel & Ho, 2019
  - Bit Swap

BB-ANS

Published as a conference paper at ICLR 2019

PRACTICAL LOSSLESS COMPRESSION WITH LATENT VARIABLES USING BITS BACK CODING

James Townsend, Thomas Bird & David Barber
Department of Computer Science
University College London
{james.townsend,thomas.bird,david.barber}@cs.ucl.ac.uk

ABSTRACT

Deep latent variable models have seen recent success in many data domains. Lossless compression is an application of these models which, despite having the potential to be highly useful, has yet to be implemented in a practical manner. We present 'Bits Back with ANS' (BB-ANS), a scheme to perform lossless compression with latent variable models at a near optimal rate. We demonstrate this scheme by using it to compress the MNIST dataset with a variational auto-encoder model (VAE), achieving compression rates superior to standard methods with only a simple VAE. Given that the scheme is highly amenable to parallelization, we conclude that with a sufficiently high quality generative model this scheme could be used to achieve substantial improvements in compression rate with acceptable running time. We make our implementation available open source at https://github.com/bits-back/bits-back.

BB-ANS = VAE bits-back coding w/ANS

[James Townsend, Thomas Bird, David Barber, "*Practical lossless compression with latent variables using bits back coding,*" ICLR 2019]

How well does BB-ANS work?

- Assumptions to investigate:
  - Finite precision approximation of log 1/p
    - Always the case with ANS
  - Inefficiency in encoding the first datapoint
    - Hopefully amortized well
  - VAE has continuous latent variables
    - See next slide
  - Need for “clean” bits
    - See next next slide

[James Townsend, Thomas Bird, David Barber, “*Practical lossless compression with latent variables using bits back coding,”* ICLR 2019]

BB-ANS -- VAE has continuous latent variables

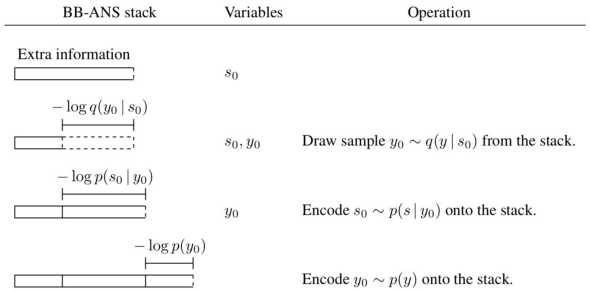
- We can discretize the continuous distribution
- Let's assume we discretize z at level δz
- Expected encoding length with bits-back:

$$-\mathbb{E}_{Q(z_i|x)}\left[\log \frac{p(x|z_i)p(z_i)\delta z}{q(z_i|x)\delta z}\right]$$

$\rightarrow$ δz cancels out, so only cost is in KL between continuous distribution and discrete approximation [Hinton & van Camp, 1993]

[James Townsend, Thomas Bird, David Barber, "*Practical lossless compression with latent variables using bits back coding,*" ICLR 2019]

BB-ANS -- Are the bits clean?



- We assume we sample from $q(z|x)$
- But is our random bit source truly random?
- The bits come from compressing the previous datapoint
- z was encoded with $p(z)$, but came from $q(z|x)$
- What matters is $q(z)$
- And really: $\mathrm{KL}(q(z)||p(z))$
- VAE objective has this as one of the terms, but in practice non-zero

[James Townsend, Thomas Bird, David Barber, "*Practical lossless compression with latent variables using bits back coding,*" ICLR 2019]

How well does VAE bits-back work?

Dataset	Raw data	VAE test ELBO	BB-ANS	bz2	gzip	PNG	WebP
Binarized MNIST	1	0.19	**0.19**	0.25	0.33	0.78	0.44
Full MNIST	8	1.39	**1.41**	1.42	1.64	2.79	2.10

Table 2: Compression rates on the binarized MNIST and full MNIST test sets, using BB-ANS and other benchmark compression schemes, measured in bits per dimension. We also give the negative ELBO value for each trained VAE on the test set.

Dataset	Raw data	BB-ANS with PixelVAE (predicted)	bz2	gzip	PNG	WebP
Binarized MNIST	1	**0.15**	0.25	0.33	0.78	0.44
ImageNet 64×64	8	**3.66**	6.72	6.95	5.71	4.64

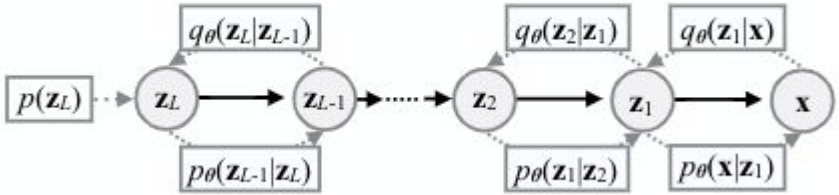
Table 3: Predicted compression of BB-ANS with PixelVAE against other schemes, measured in bits per dimension.

[James Townsend, Thomas Bird, David Barber, "*Practical lossless compression with latent variables using bits back coding,*" ICLR 2019]

Can we do even better?

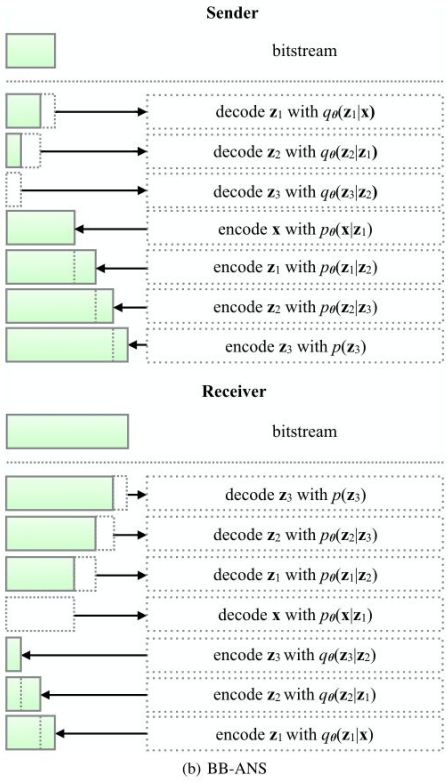
Quality of encoding depends on quality of ELBO

Latent variable models with multiple latent layers tend to achieve better ELBO than with single latent layer



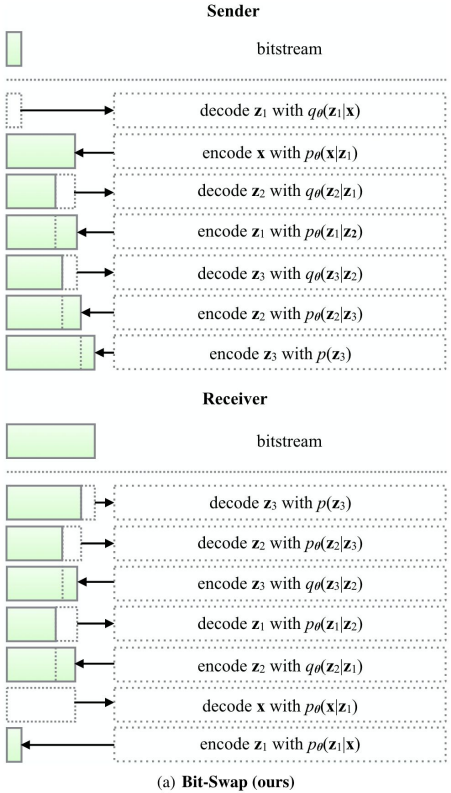
Can we bits-back code for this?

BB-ANS



(b) BB-ANS

Bit-Swap



(a) **Bit-Swap (ours)**

Bit-Swap Results

Table 1: CIFAR compression results over the course of 100 timesteps (datapoints) for Bit-Swap vs BB-ANS. The cumulative moving average in bits/dim is reported at various timesteps. (There is a discrepancy between bits/dim reported here and the ELBO of the models due to the fact that only hundreds of datapoints are processed – a small fraction of the test set.)

Depth (L)	ELBO $-\mathcal{L}(\boldsymbol{\theta})$	Codelength w/o overhead	Scheme	Initial ($n = 1$)	CMA ($n = 50$)	CMA ($n = 100$)
2	4.59	4.90 ± 0.90	BB-ANS	28.40 ± 0.13	5.31 ± 0.03	5.13 ± 0.05
			Bit-Swap	24.73 ± 0.22	5.24 ± 0.03	5.10 ± 0.04
3	4.39	5.19 ± 1.43	BB-ANS	41.57 ± 0.56	5.87 ± 0.13	5.58 ± 0.10
			Bit-Swap	25.75 ± 0.08	5.52 ± 0.14	5.38 ± 0.10
4	4.45	4.80 ± 0.99	BB-ANS	54.29 ± 0.71	5.76 ± 0.09	5.30 ± 0.06
			Bit-Swap	26.21 ± 0.06	5.18 ± 0.09	5.02 ± 0.09

[Kingma, Abbeel, Ho, 2019 -- paper not yet on arxiv as of 2/20, these are preliminary results to be updated in the future]

Bit-Swap Results

Table 2: MNIST compression results over the course of 100 timesteps (datapoints) for Bit-Swap vs BB-ANS. Similar comments apply as Table 1.

Depth (L)	ELBO $-\mathcal{L}(\boldsymbol{\theta})$	Codelength w/o overhead	Scheme	Initial ($n = 1$)	CMA ($n = 50$)	CMA ($n = 100$)
2	1.24	1.25 ± 0.30	BB-ANS	3.31 ± 0.25	1.27 ± 0.05	1.27 ± 0.03
			Bit-Swap	2.33 ± 0.24	1.25 ± 0.05	1.26 ± 0.04
3	1.25	1.26 ± 0.30	BB-ANS	4.38 ± 0.25	1.31 ± 0.05	1.29 ± 0.04
			Bit-Swap	2.38 ± 0.22	1.27 ± 0.05	1.27 ± 0.04
4	1.34	1.34 ± 0.31	BB-ANS	5.57 ± 0.26	1.41 ± 0.06	1.39 ± 0.04
			Bit-Swap	2.49 ± 0.27	1.35 ± 0.06	1.35 ± 0.04

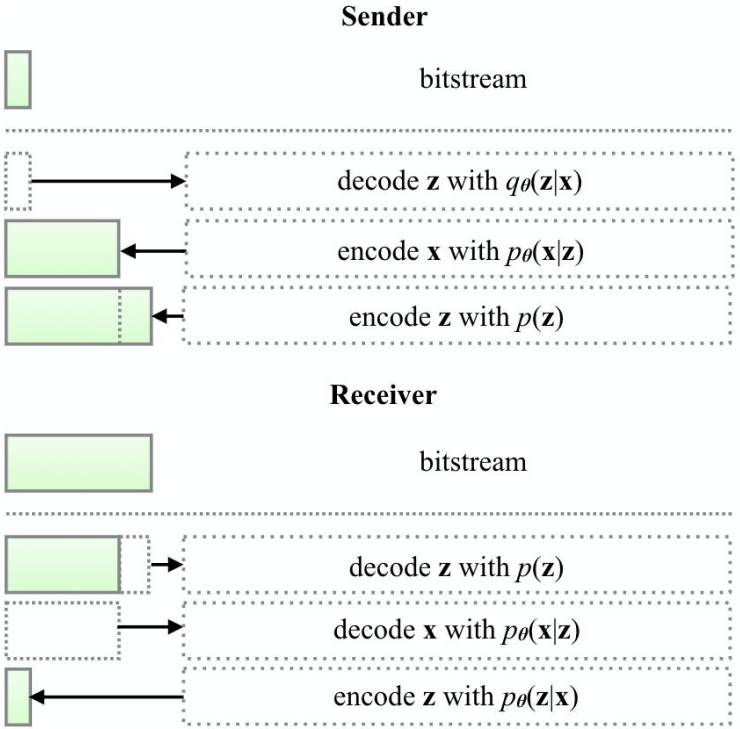
[Kingma, Abbeel, Ho, 2019 -- paper not yet on arxiv as of 2/20, these are preliminary results to be updated in the future]

How do we actually get those bits back?

Recall we have: $i \sim q(i|x)$, and use random bits to generate i

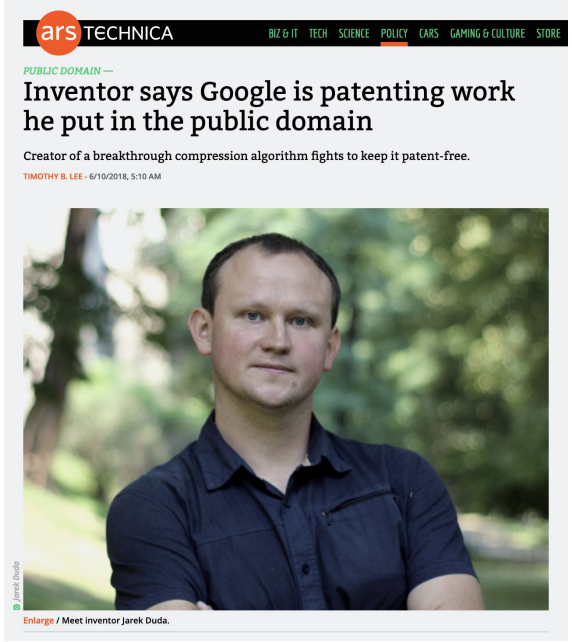
How exactly can this work mechanically, and mechanics of getting them back?

Asymmetric Numeral Systems (ANS)



- Note the stack behavior of the encoding/decoding
- Arithmetic coding aligns with queues/streams
- ANS provides coding scheme compatible with stacks

Asymmetric Numeral Systems (ANS)



Jarek Duda, *"Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding,"* 2007

Asymmetric Numeral Systems

- High level idea: use natural numbers as encodings, the longer the symbol sequence, the higher the natural number becomes

- Example: p(a) = 1/4 ; p(b) = 3/4
  - 1/4 of natural numbers should correspond to a symbol sequence ending in a
  - 3/4 _____ b

$$\mathbb{N} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \cdots\}$$

  - Encoding process: (encoding so far, new symbol) -> new encoding
    - (0,a) -> 0
    - (1,a) -> 4
    - (2,a) -> 8
    - (0,b) -> 1
    - (1,b) -> 2
    - (2,b) -> 3
    - (3,b) -> 5
    - (4,5) -> 6
  - General scheme:
    - (x, s) -> x / p(s) [approximately]

Asymmetric Numeral Systems

- High level idea: use natural numbers as encodings, the longer the symbol sequence, the higher the natural number becomes

- General scheme:
  - (x, s) -> x / p(s) [but made into an integer]

- Hence we end up with $$x_T = \frac{x_0}{p(s_1)p(s_2)p(s_3)\cdots p(s_T)}$$

- And code-length: $$\log x_T = C + \sum_t \log \frac{1}{p_t}$$