Announcements

Announcements

- Wed, usual time/place
- **not** finals period!
- Closed book, but 2 sheets of notes are allowed

- **Open-ended projects due midnight Sun 5/6**

- I fixed that quiz from last week – Tues noon

Deep Neural Networks

Generalizing backprop

e.g.

$$x_7 = x_2 + x_5$$
$$\pi(7) = (2,5)$$
$$f_7 = \text{add}$$

- Starting point: a function of *n* variables
- Step 1: code your function as a series of assignments

Wengert list

- Step 2: back propagate by going thru the list in reverse order, starting with… $\frac{dx_N}{dx_N} \leftarrow 1$
- …and using the chain rule

$$\frac{dx_N}{dx_i} = \sum_{j:i\in\pi(j)} \frac{dx_N}{dx_j}\frac{\partial x_j}{\partial x_i}$$

Step 1: forward

inputs: $x_1, x_2, \ldots, x_n$

for $i = n+1, n+2, \ldots, N$

$\quad x_i \leftarrow f_i(\mathbf{x}_{\pi(i)})$

return x_N

A function

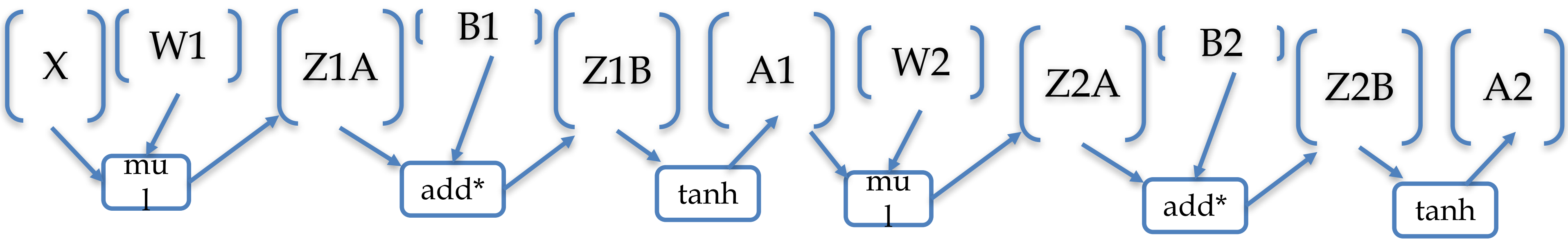
Step 2: backprop

for $i = N-1, N-2, \ldots, 1$

$$\frac{dx_N}{dx_i} \leftarrow \sum_{j:i\in\pi(j)} \frac{dx_N}{dx_j}\frac{\partial f_j}{\partial x_i}$$

Computed in previous step

Example: 2-layer neural network




```
Inputs: X,W1,B1,W2,B2
Z1a = mul(X,W1)          // matrix mult
Z1b = add*(Z1a,B1)       // add bias vec
A1 = tanh(Z1b)           //element-wise
Z2a = mul(A1,W2)
Z2b = add*(Z2a,B2)
A2 = tanh(Z2b)           // element-wise
P = softMax(A2)          // vec to vec
C = crossEnt_Y(P)        // cost function
```

Target Y; *N* examples; *K* outs; *D* feats, *H* hidden

X is *N*D*, W1 is *D*H*, B1 is *1*H*, W2 is *H*K*, …

Z1a is *N*H*

Z1b is *N*H*

A1 is *N*H*

Z2a is *N*K*

Z2b is *N*K*

A2 is *N*K*

P is *N*K*

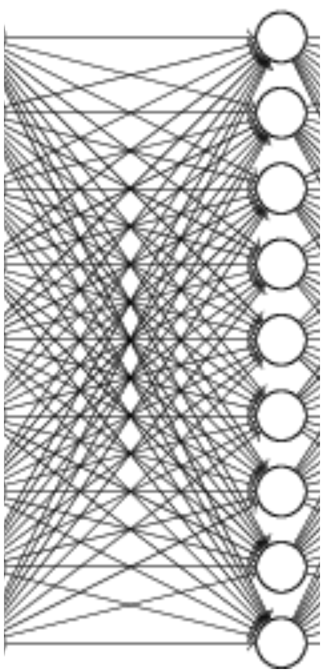
C is a scalar

$$\mathbf{p}_i = \frac{\exp(\mathbf{a_i})}{\sum_j \exp(\mathbf{a}_j)}$$

Minibatch SGD on GPU

Let X be a matrix with *k* examples
Let $\mathbf{w}_i$ be the input weights for the i-th hidden unit
Then A = X W is output for all *m* units
for all *k* examples

w_1	w_2	w_3	…	w_m
0.1	-0.3	…		
-1.7	…			
0.3	…			
1.2				



$\mathbf{x}_1$	1	0	1	1
$\mathbf{x}_2$	…			
…				
$\mathbf{x}_k$				

There's a lot of chances to do this in parallel

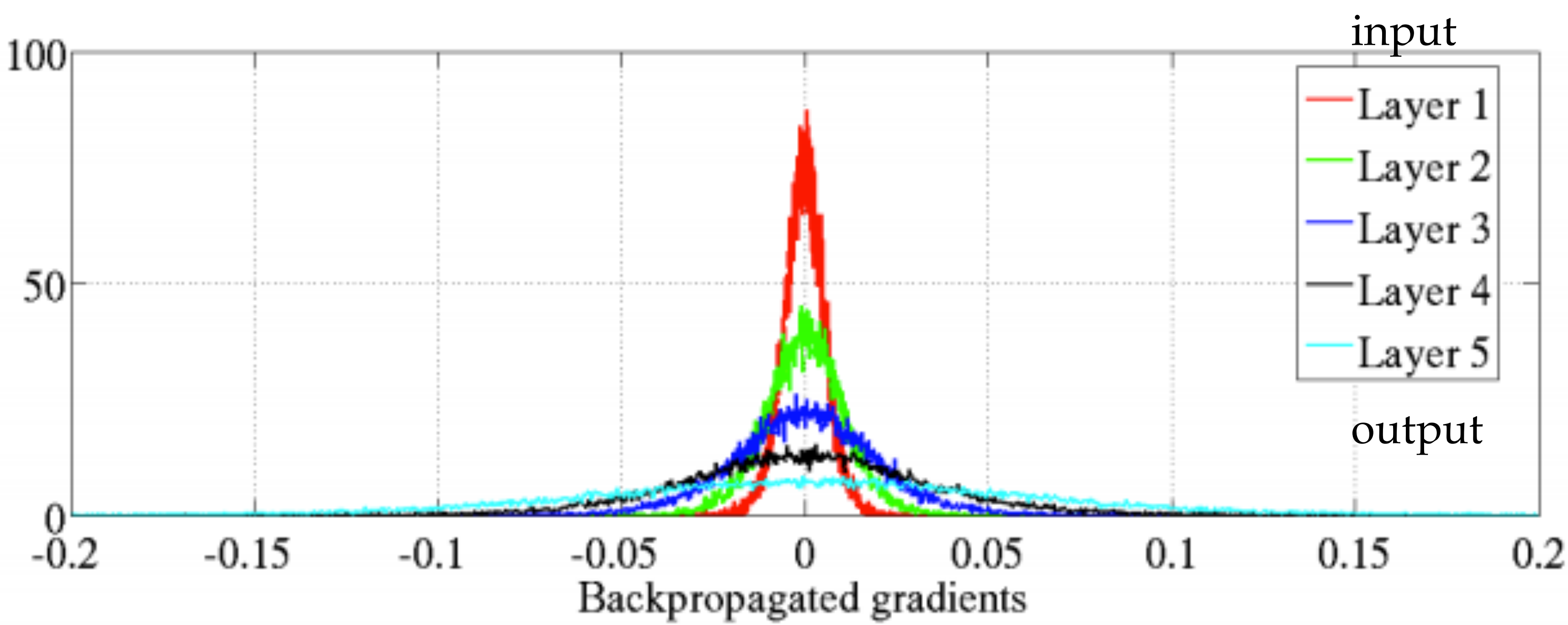
XW =

$\mathbf{x}_1.\mathbf{w}_1$	$\mathbf{x}_1.\mathbf{w}_2$	…	$\mathbf{x}_1.\mathbf{w}_m$
$\mathbf{x}_k.\mathbf{w}_1$	…	…	$\mathbf{x}_k.\mathbf{w}_m$

Understanding the difficulty of training deep feedforward neural networks

AI Stats 2010

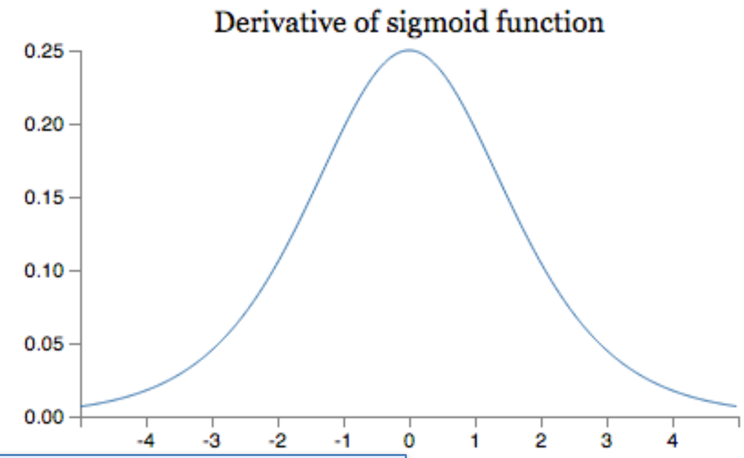
Xavier Glorot **Yoshua Bengio**
DIRO, Université de Montréal, Montréal, Québec, Canada



Histogram of gradients in a 5-layer network for an artificial image recognition task

Gradients are unstable

Max at 1/4

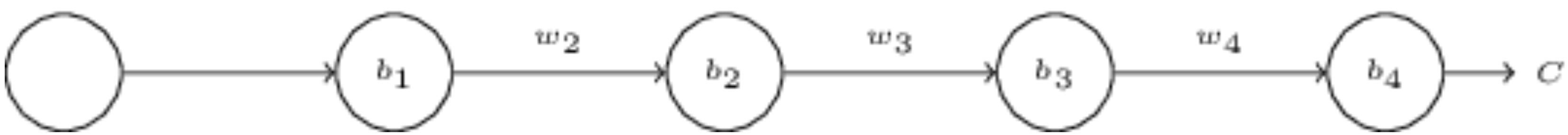


If weights are usually < 1 then we are multiplying by many numbers < 1 so the gradients get very small.

The **vanishing gradient** problem

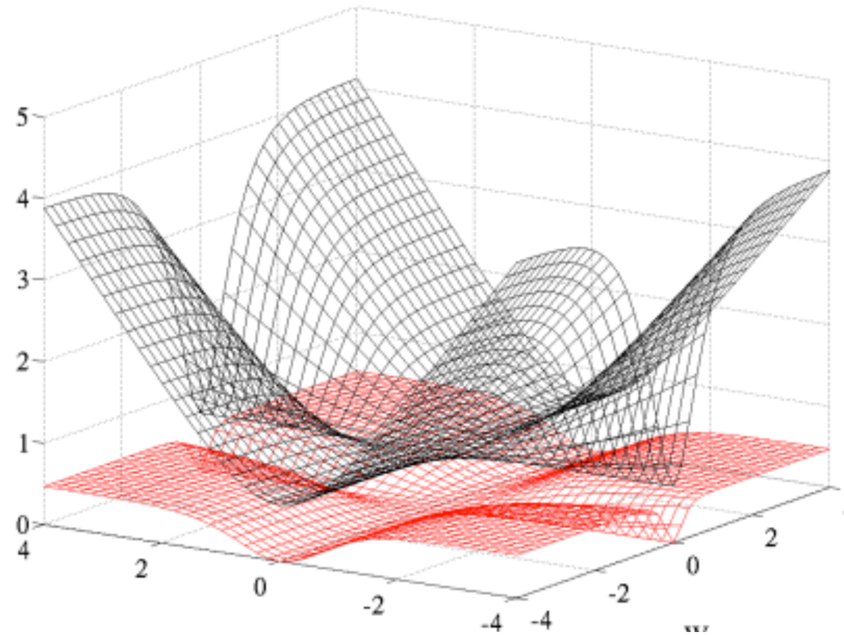
What happens as the layers get further and further from the output layer? E.g., what's gradient for the bias term with several layers after it in a trivial net?

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



Some key differences in modern ANNs

- Use of softmax and entropic loss instead of quadratic loss.
- Use of alternate non-linearities
  - reLU and hyperbolic tangent
- Better understanding of weight initialization
- …



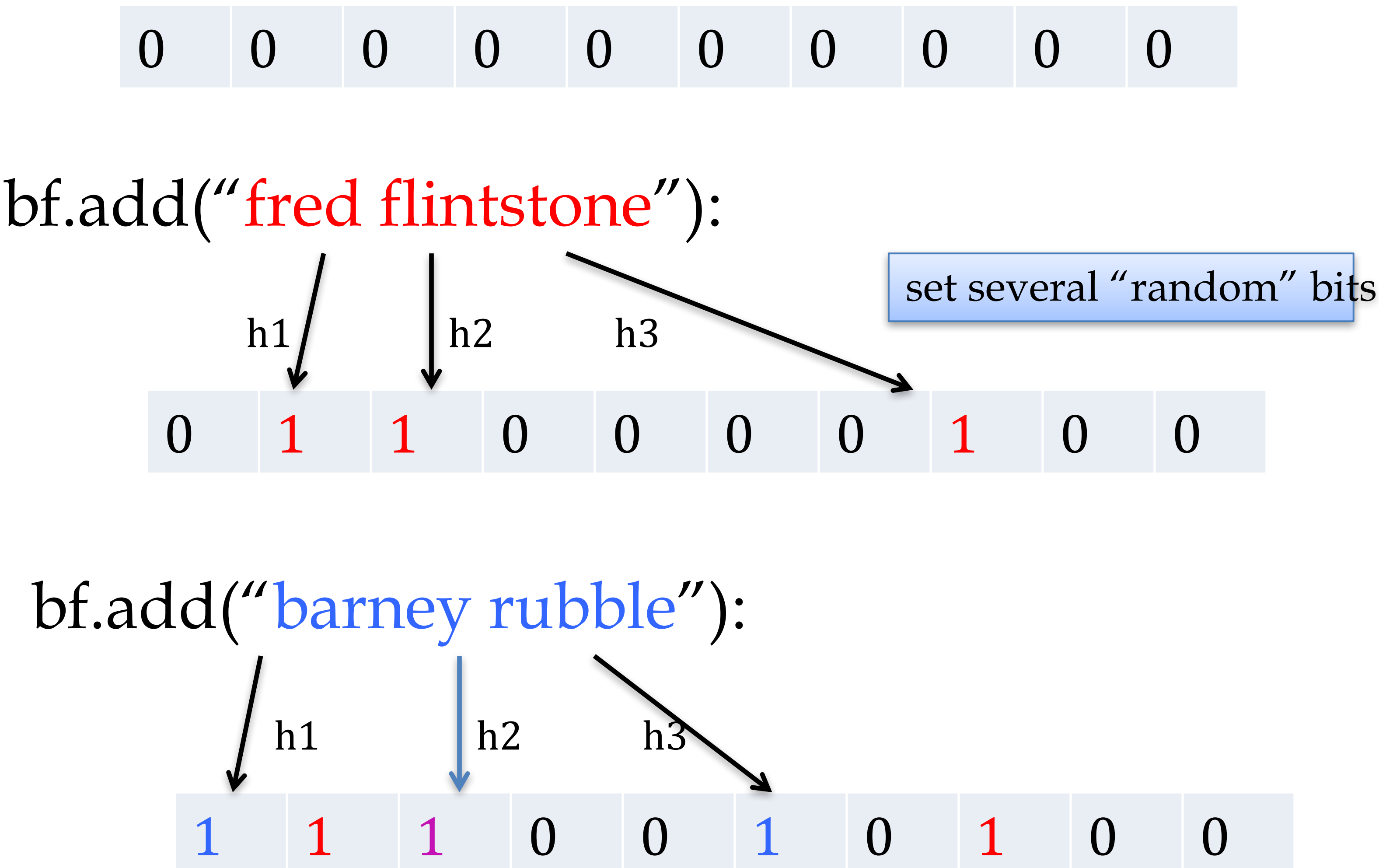
Bloom filters

- Interface to a Bloom filter
  - BloomFilter(int maxSize, double p);
  - void bf.add(String s); // insert s
  - bool bd.contains(String s);
    - // If s was added return true;
    - // else with probability at least *1-p* return false;
    - // else with probability at most *p* return true;

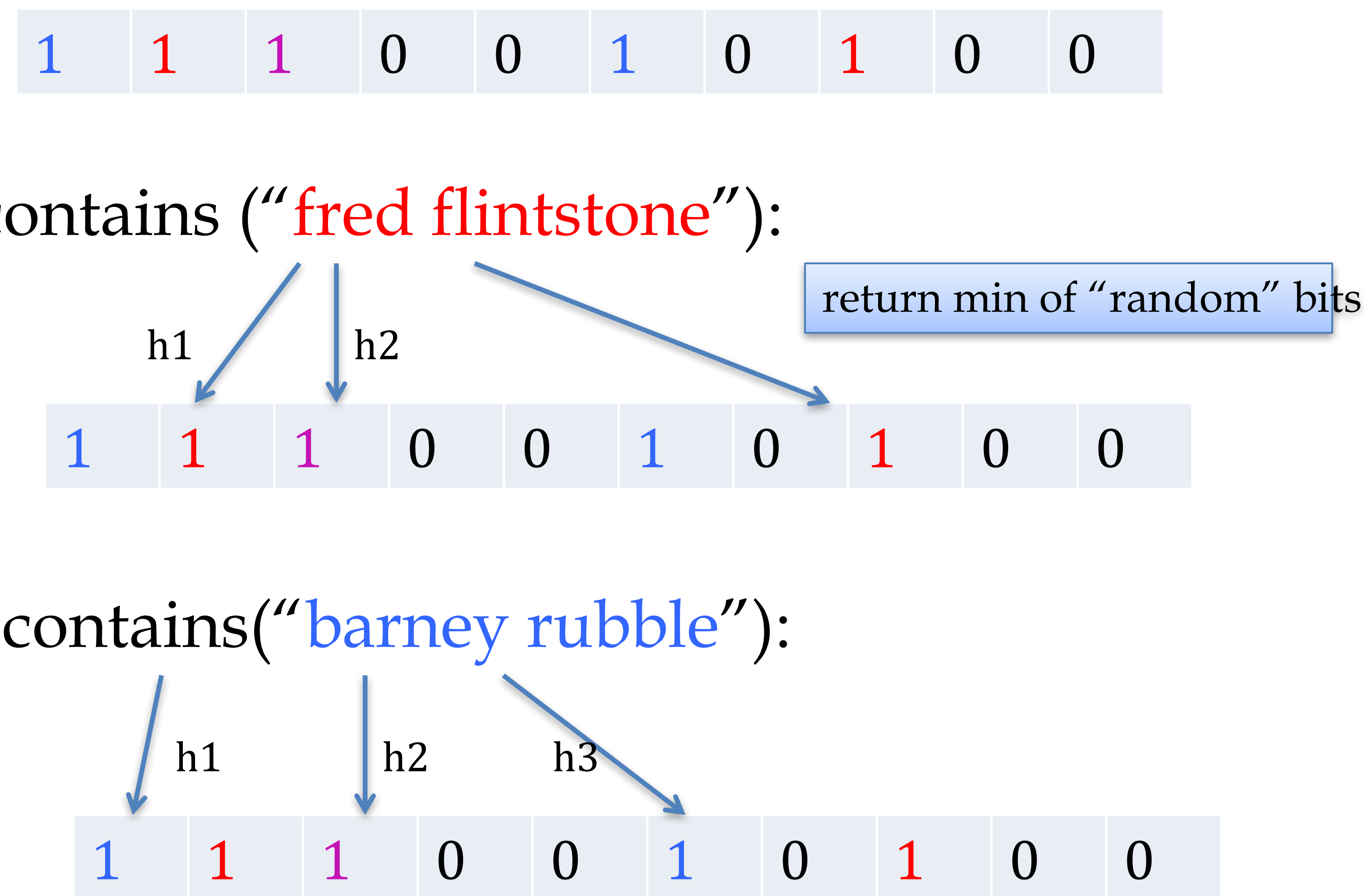
  - I.e., a noisy "set" where you can test membership (and that's it)

Randomized algorithms

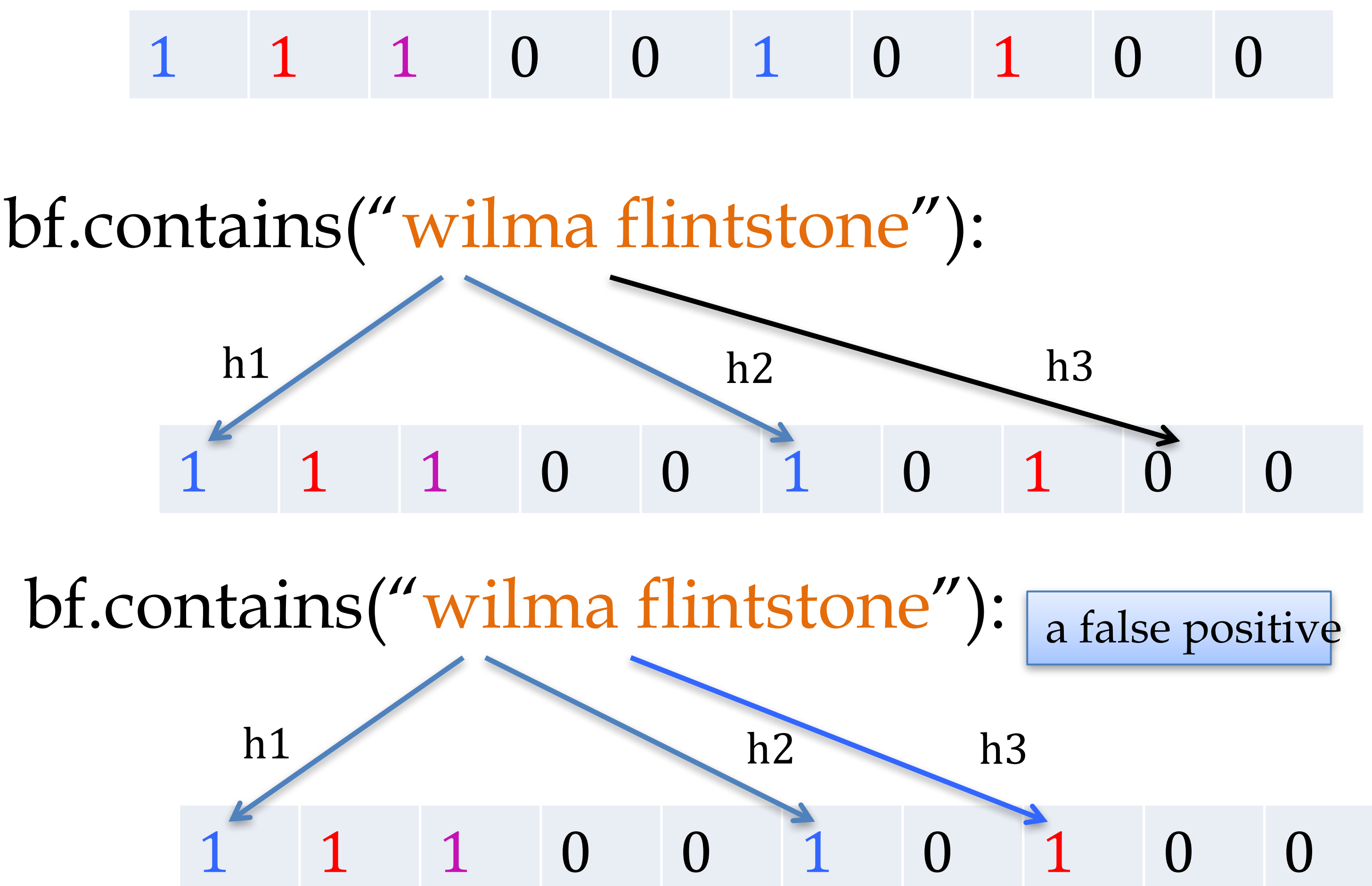
Bloom filters



Bloom filters



Bloom filters



Randomized algorithms

- What is a Bloom filter for (what’s the API)?
- What are the guarantees? What kind of errors do they make?
- How can you build up more complex operations (eg, counting to K) with multiple filters?
- How about countmin sketches?
- How about LSH?
- What are the problems that on-line LSH is trying to fix?

Architectures

Graph architectures

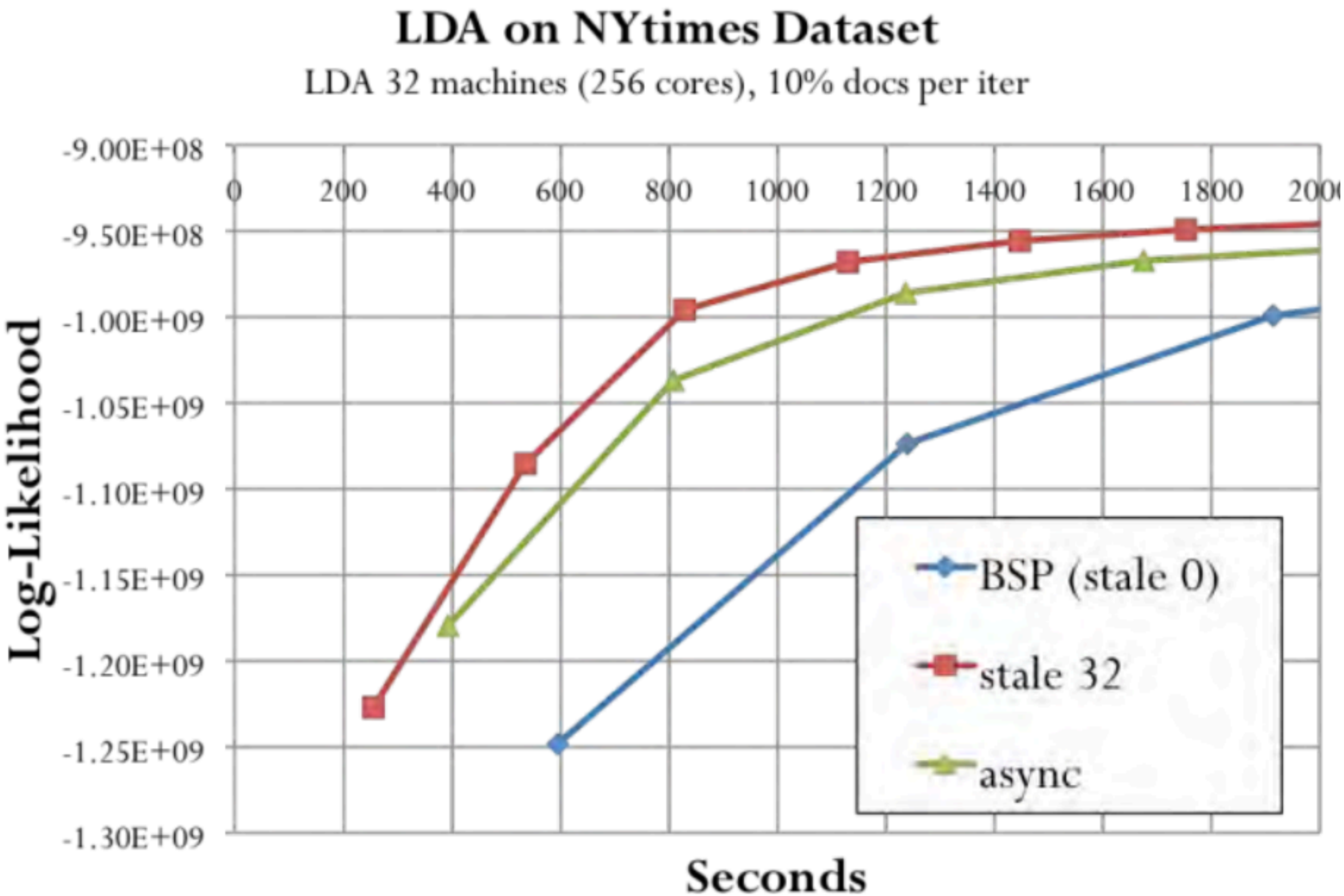
- Differences between
  - Signal/collect
  - GraphX
  - PowerGraph
  - GraphChi
- Can you understand/extend simple programs?

`initialState`	`if (isTrainingData) trainingData else avgProbDist`

`collect()`	`if (isTrainingData)`
	`   return oldState`
	`else`
	`   return signals.sum.normalise`

`signal()`	`return source.state`

Stale Synchronous Parallel (SSP)

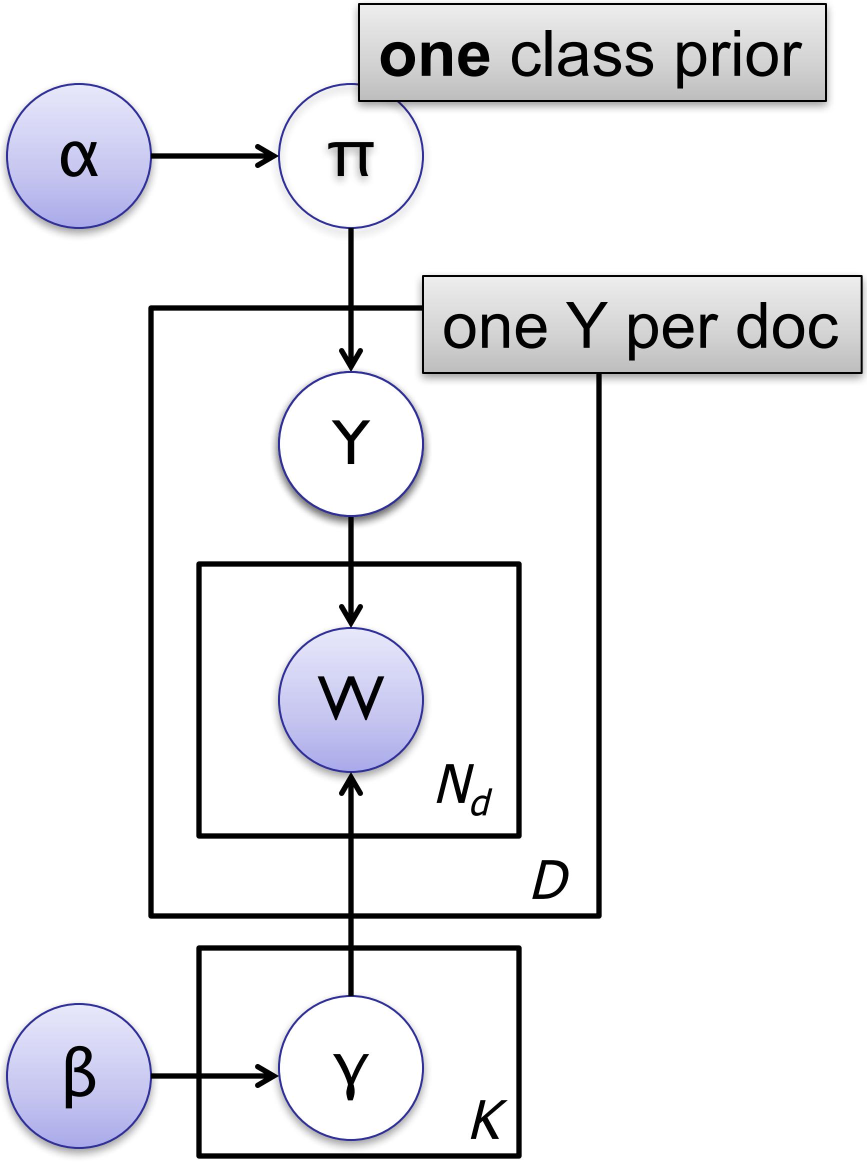


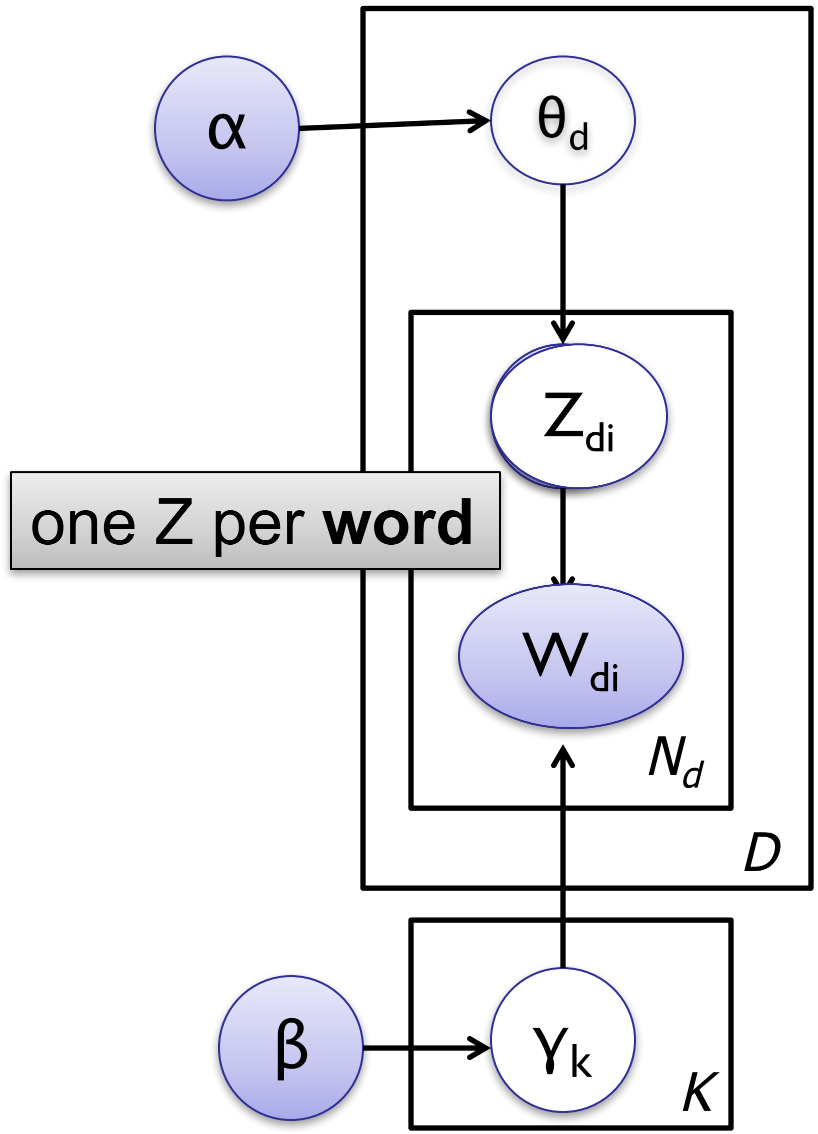
[Ho et al 2013]

LDAs and sampling

Unsupervised NB vs LDA

different class distrib θ for **each** doc





Recap: Collapsed Sampling for LDA

Pr(Z|E+)

Pr(E−|Z)

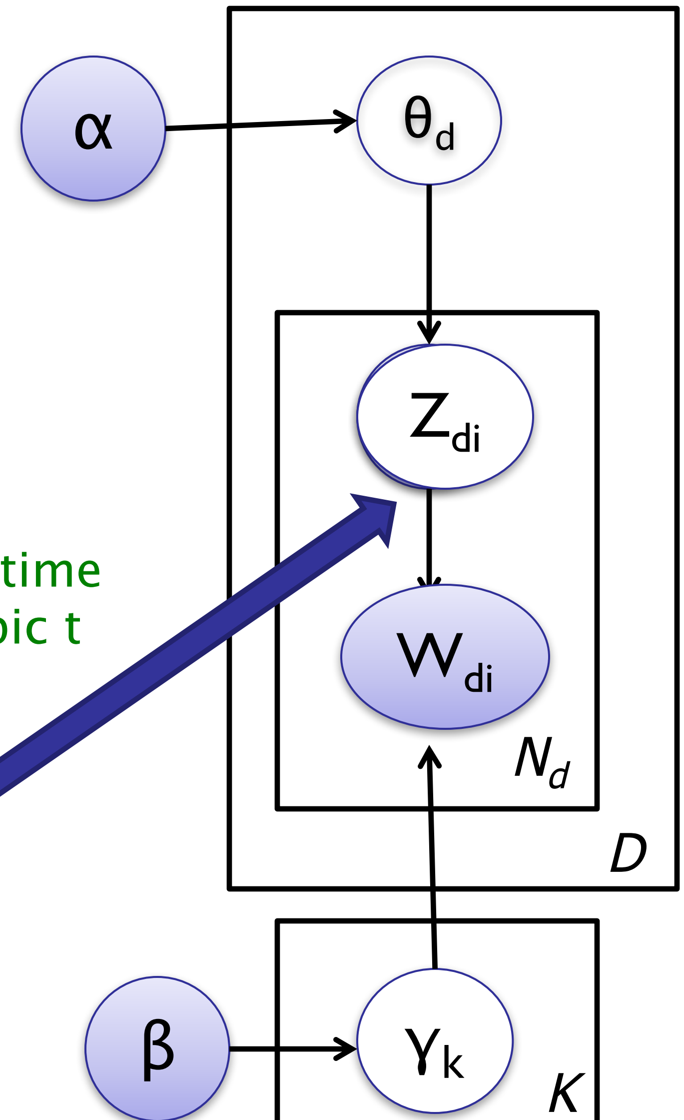
$$P(z = t|w) \propto (\alpha_t + n_{t|d}) \frac{\beta + n_{w|t}}{\beta V + n_{\cdot|t}}.$$

"fraction" of time *Z=t* in doc *d*

fraction of time W=w in topic t

ignores a detail – counts should not include the Z_{di} being sampled

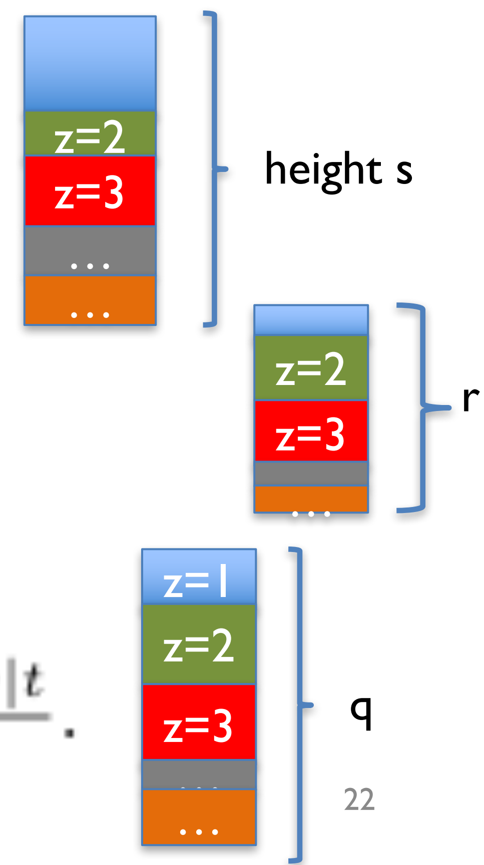
Only sample the Z's



$$P(z=t|w) \quad \propto \quad (\alpha_t + n_{t|d}) \frac{\beta + n_{w|t}}{\beta V + n_{\cdot|t}}.$$

$$P(z=t|w) \propto \frac{\alpha_t \beta}{\beta V + n_{\cdot|t}} + \frac{n_{t|d}\beta}{\beta V + n_{\cdot|t}} + \frac{(\alpha_t + n_{t|d}) n_{w|t}}{\beta V + n_{\cdot|t}}.$$

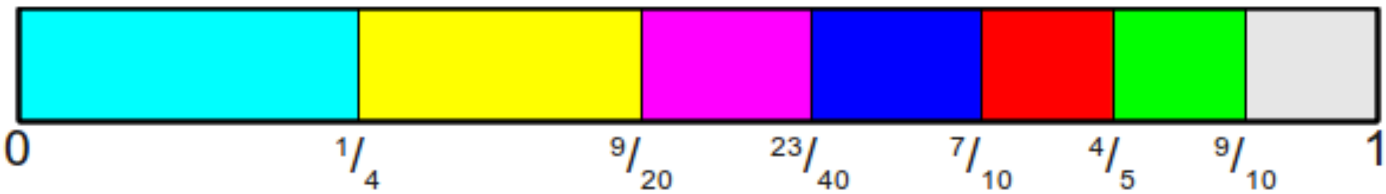
$$z=s+r+q \quad \left\{ \begin{aligned} s &= \sum_t \frac{\alpha_t \beta}{\beta V + n_{\cdot|t}} \\ r &= \sum_t \frac{n_{t|d}\beta}{\beta V + n_{\cdot|t}} \\ q &= \sum_t \frac{(\alpha_t + n_{t|d}) n_{w|t}}{\beta V + n_{\cdot|t}}. \end{aligned} \right.$$



Fenwick Tree Sampler

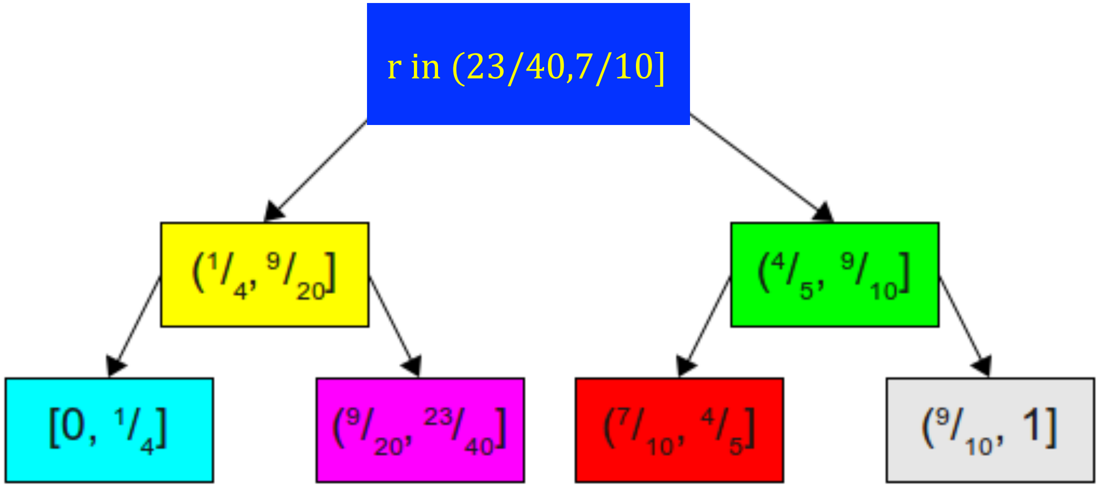
O(K)

Basic problem: how can we sample from a biased die quickly….



…and update quickly? maybe we can use a binary tree….

O(log2K)

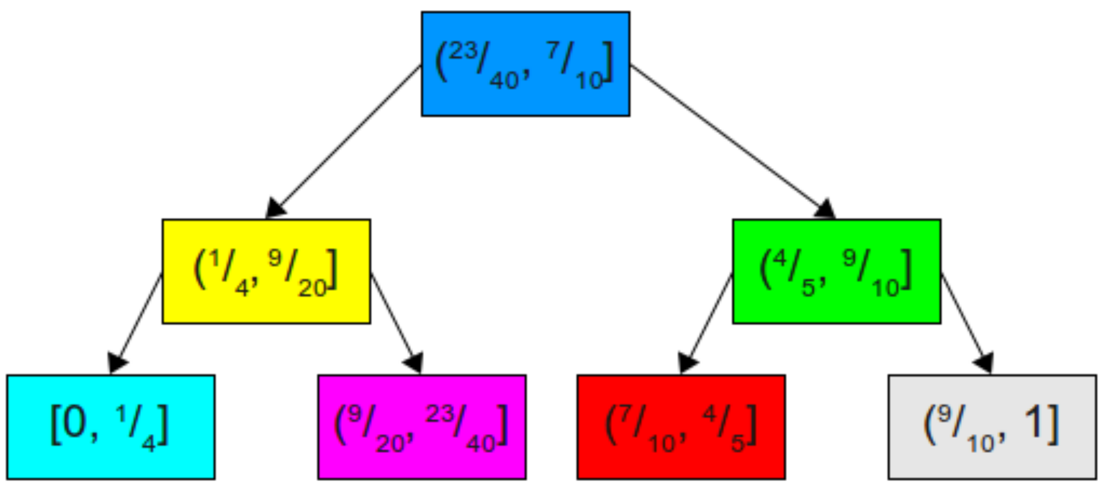


http://www.keithschwarz.com/darts-dice-coins/

Data structures and algorithms

	Data Structure Space	Initialization Time	Initialization Space	Generation Time	Parameter Update Time
LSearch	$c_T = \mathbf{p}^\top \mathbf{1}$: $O(1)$	$O(T)$	$O(1)$	$O(T)$	$O(1)$
BSearch	$\mathbf{c} = \mathtt{cumsum}(\mathbf{p})$: $O(T)$	$O(T)$	$O(1)$	$O(\log T)$	$O(T)$
Alias Method	$\mathit{prob}, \mathit{alias}$: $O(T)$	$O(T)$	$O(T)$	$O(1)$	$O(T)$
F+tree Sampling	$\mathtt{F.initialize}(\mathbf{p})$: $O(T)$	$O(T)$	$O(1)$	$O(\log T)$	$O(\log T)$

F+ tree



Unsupervised/SS Learning on graphs

- What's different between HF, MRW, MAD?
  - Which have hard/soft seeds?
  - How do they scale with #edges, #nodes?
- What are the methods trying to optimize?
- Do they optimize it exactly or approximately?