Path Kernels and Multiplicative Updates

Manfred K. Warmuth
Visiting NICTA Canberra
from UC Santa Cruz, USA

Joint work with Eiji Takimoto

September 27, 2004
Last update June 1, 2007

Kernels

$$\boldsymbol{x} \qquad \rightarrow \qquad \Phi(\boldsymbol{x})$$

instance space
n

feature space
N

$(x_1, x_2, \ldots, x_n)$

$(1, x_i, x_i x_j, x_i x_j x_k, \ldots)$

all subsets kernel
$N = 2^n$

Dot products computed via kernels

$$\underset{O(N)}{\Phi(\boldsymbol{x}) \cdot \Phi(\boldsymbol{z})} \quad = \quad \underset{O(\text{poly}(n))}{k(\boldsymbol{x}, \boldsymbol{z})}$$

efficient computation
implicit access to features

$$(1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1x_2x_3)$$

$$\bullet$$

$$(1, z_1, z_2, z_3, z_1z_2, z_1z_3, z_2z_3, z_1z_2z_3)$$

$$= \prod_{i=1}^{3} (1 + x_i z_i)$$

$O(n)$ versus $O(\underbrace{N}_{2^n})$

Kernel algorithms

- Weight vector linear combination of expanded instances

$$\mathcal{W} = \sum_{t=1}^{T} \alpha_t \Phi(\boldsymbol{x}_t)$$

Kernel algorithms

$$\begin{aligned} \mathcal{W} \cdot \Phi(\boldsymbol{x}) &= \left(\sum_t \alpha_t \Phi(\boldsymbol{x}_t) \right) \cdot \Phi(\boldsymbol{x}) \\ &= \sum_t \alpha_t \underbrace{\Phi(\boldsymbol{x}_t) \cdot \Phi(\boldsymbol{x})}_{k(\boldsymbol{x}_t, \boldsymbol{x})} \end{aligned}$$

- Kernels efficient

- Maintain $\underbrace{\alpha_t}_{T}$ instead of $\underbrace{W_F}_{N}$ parameters

Kernel family

- SVMs
- Perceptron Algorithm
- Ridge Regression
- Rotation invariant algorithms
- $||\mathcal{W}||^2$ regularization
- Representer Theorem

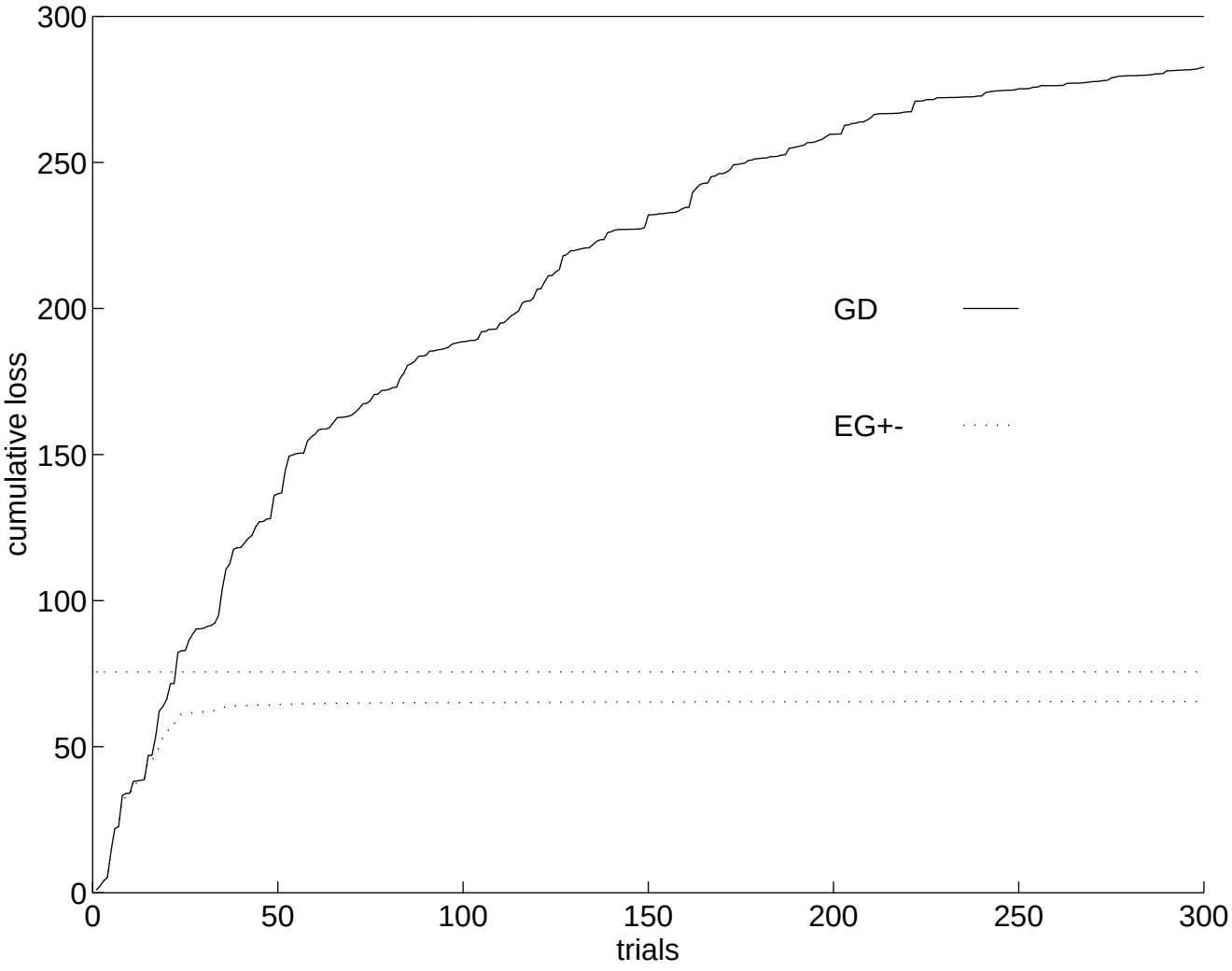
Good: Implicit access to feature
Bad generalization when target $\mathcal{W} \in \mathbf{R}^N$ sparse

Experiments

All subsets kernel ($n = 8$, $N = 2^8 = 256$)

x_i and $\mathcal{X}_F \in \{+1, -1\}$

Target $x_2x_3x_4 + x_2x_3x_6 + x_1x_2x_3x_4x_5x_6x_7x_8$



Cumulative losses for GD and EG$^{\pm}$

GD Kernel alg. - EG$^{\pm 1}$ multiplicative alg.

Multiplicative Updates

$$\widetilde{W}_F := \frac{W_F \text{fact}_F}{\text{normaliz.}}$$

- Motivated by $\sum_F \widetilde{W}_F \ln \frac{\widetilde{W}_F}{W_F}$
- Weight vectors are probability vectors

Family

- Weighted Majority Algorithm
- Expert algorithms
- Exponentiated Gradient algorithm
- Winnow

Good: Bounds grow as $O(\log N)$ when target sparse

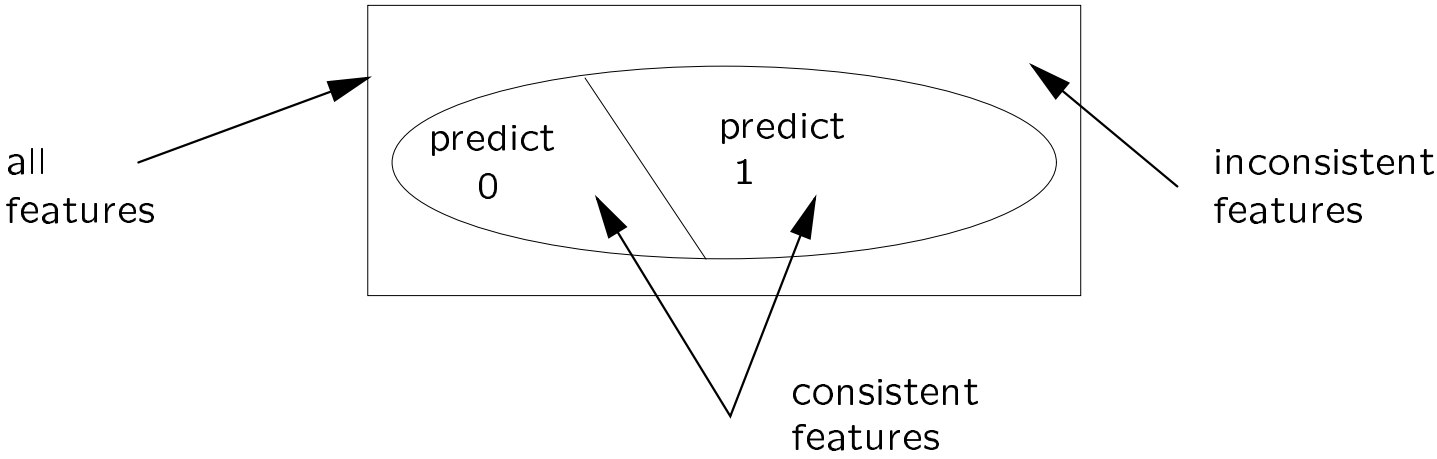
Bad: Require explicit access to features

We use special kernels to avoid above

In scecial cases obtain best of both worlds

Halving Algorithm

$$(\underbrace{\Phi(x_1)}_{\{0,1\}^N}, \underbrace{y_1}_{\{0,1\}}), (\Phi(x_2), y_2), (\Phi(x_3), y_3), \ldots$$



$\leq \lfloor \log N \rfloor$ mistakes

Explicit access to features needed

So far

- Introduced both families
- Good / bad of both

To come

Multiplicative updates with kernels

- Special kernels
- Loss must have a certain form

Use of kernels

Kernel algorithms

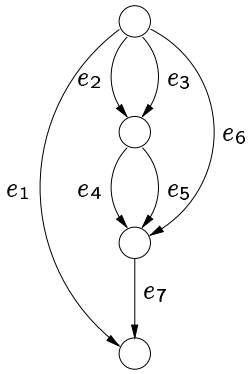
$$\underbrace{\mathcal{W}}_{\sum_t \alpha_t \Phi(\boldsymbol{x}_t)} \cdot \underbrace{\mathcal{X}}_{\Phi(\boldsymbol{x})} = \sum_t \alpha_t \, k(x_t, x)$$

Now

$$\underbrace{\mathcal{W}}_{\Phi(\boldsymbol{w})} \cdot \underbrace{\mathcal{X}}_{\Phi(\boldsymbol{x})}$$

- $\mathcal{W} = \Phi(\boldsymbol{w})$ implicitly represented by $\boldsymbol{w}$
- Update $\underbrace{\mathcal{W}}_{N}$ by updating $\underbrace{\boldsymbol{w}}_{n}$
- Features must be products

Defining kernels in terms of digraphs



- Edge e receives input x_e
- Features are source sink paths
- Value of the feature is product along path

$$\mathcal{X}_P = \prod_{e \in P} x_e$$

$$\Phi(\boldsymbol{x}) = (x_1, x_2x_4x_7, x_3x_4x_7, x_2x_5x_7, x_3x_5x_7, x_6x_7)$$

Dot product

$$\phi(\boldsymbol{w}) \cdot \phi(\boldsymbol{x}) = \sum_{P} \prod_{e \in P} w_e \prod_{e \in P} x_e$$

Computation when digraph is acyclic:

Sort vertices so that all edges go downward

Edge e receives input $w_e x_e$

$val(\text{sink}) = 1$

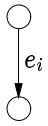
In bottom up order

$$val(u) = \sum_{(u,v)} w_{(u,v)} x_{(u,v)} \ \ val(v)$$

Cyclic Digraphs

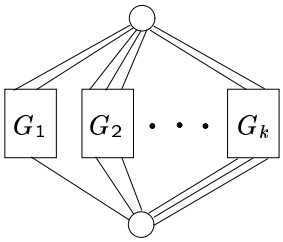
- Number of paths is infinite
- Edge e receives $w_e x_e$
- Dot product via dynamic programming
  (as Forward Backward Algorithm for HMMs)

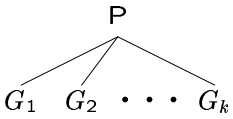
Series Parallel Digraphs



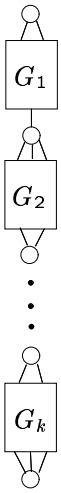
e_i

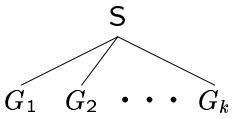
Base graph and its composition tree





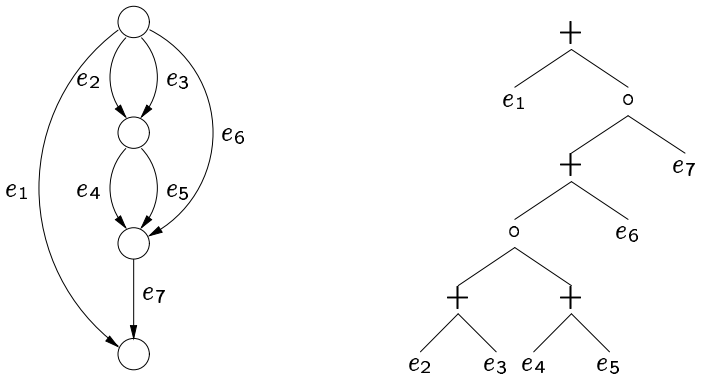
Parallel composition



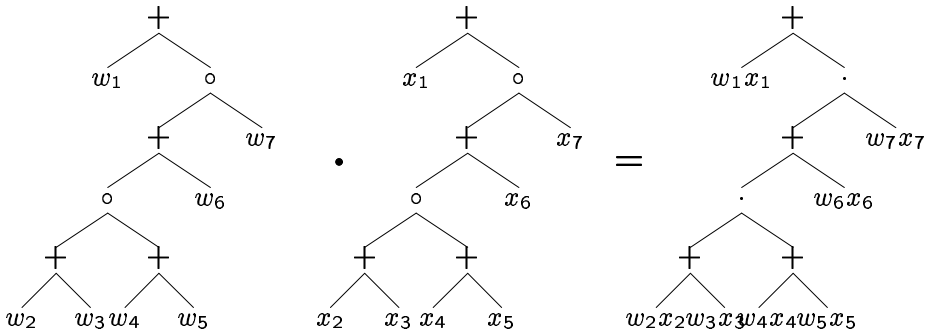


Series composition

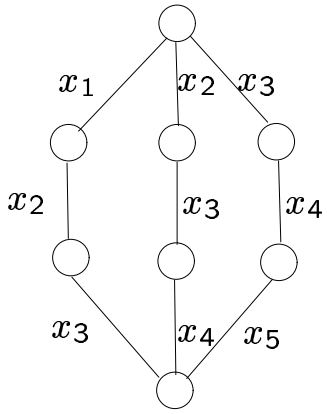
SP digraph and its composition tree

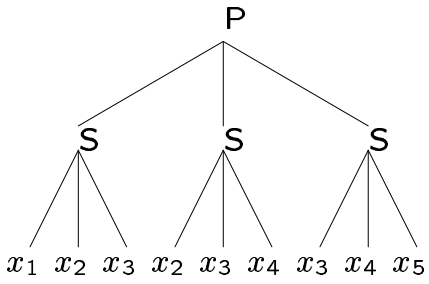


Dot product $\Phi(w) \cdot \Phi(x)$



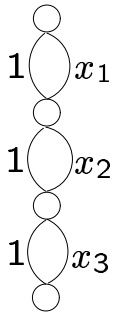
Some inputs may be the same

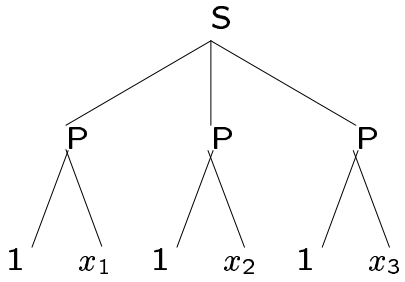




$$\Phi(x) = (x_1x_2x_3, x_2x_3x_4, x_3x_4x_5)$$

One feature per subset [KW97]



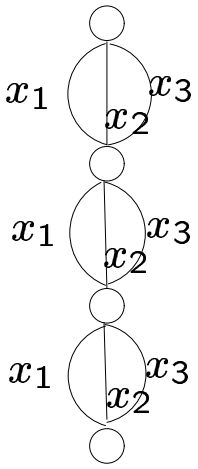


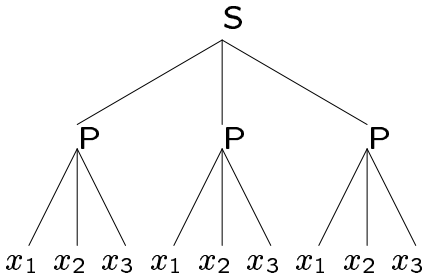
$$\Phi(\boldsymbol{x}) = (1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1x_2x_3)$$

$$\begin{aligned}\Phi(\boldsymbol{w}) \cdot \Phi(\boldsymbol{x}) &= \sum_{I \subseteq 1..n} \prod_{i \in I} w_i \prod_{i \in I} x_i \\ &= \prod_{i=1}^{n} (1 + w_i x_i)\end{aligned}$$

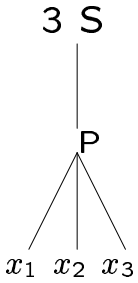
Polynomial kernel

$$\Phi(\boldsymbol{x}) \cdot \Phi(\boldsymbol{z}) = (\boldsymbol{x} \cdot \boldsymbol{z})^3 = (x_1 z_1 + x_2 z_2 + x_3 z_3)^3$$





Shorthand for identical subgraphs



P	$+$	plus
S	$\cdot$	times
n S	$\wedge n$	power n
n P	$\cdot n$	times n

So far

- Digraphs define kernels
- SP digraphs most efficient
- Localize inefficient computation:

  Cyclic digraphs as subgraphs of SP digraphs

What is next

Application of kernels to multiplicative updates

Probabilistic weights

- $$W_P = \prod_{e \in P} w_e$$

- For any vertex u

  $$\sum_{(u,v)} w_{(u,v)} = 1$$

  i.e. outflow one

- $$\sum_P W_P = 1$$

  where P any source to sink path

Decomposable Multiplicative Updates

Each edge e multiplied by factor b_e
Total weight renormalized

- Start with $\mathcal{W} = \phi(\boldsymbol{w})$, i.e. $W_P = \prod_{e \in P} w_e$

- Update

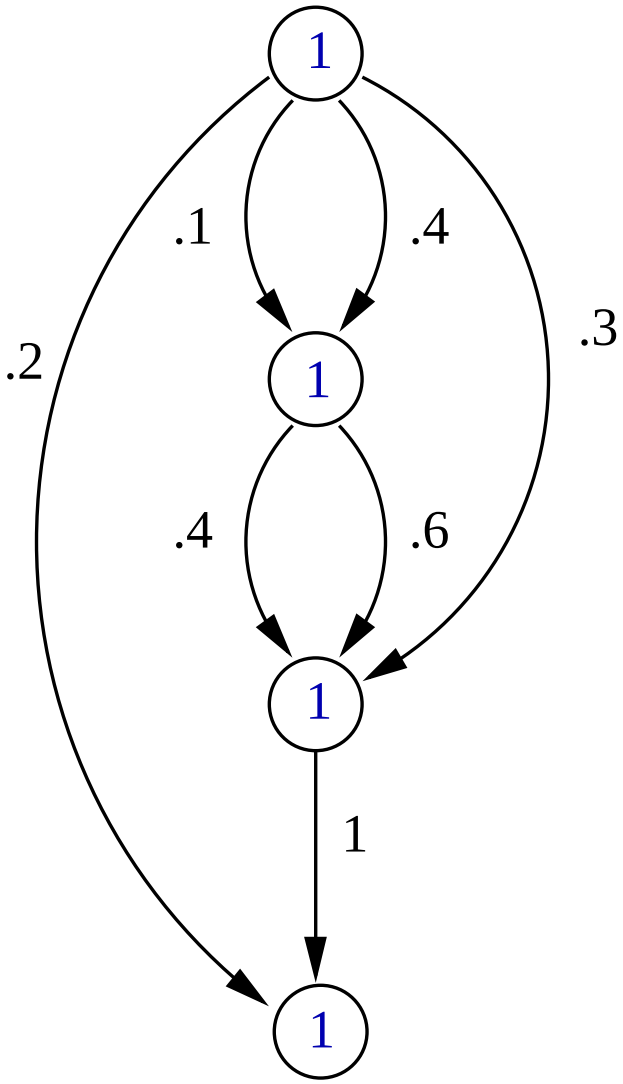
$$\widetilde{W}_P = \frac{W_P \prod_{e \in P} b_e}{\sum_P W_P \prod_{e \in P} b_e}$$

  Normalization $= \Phi(\boldsymbol{w}) \cdot \Phi(\boldsymbol{b}) = k(\boldsymbol{w}, \boldsymbol{b})$

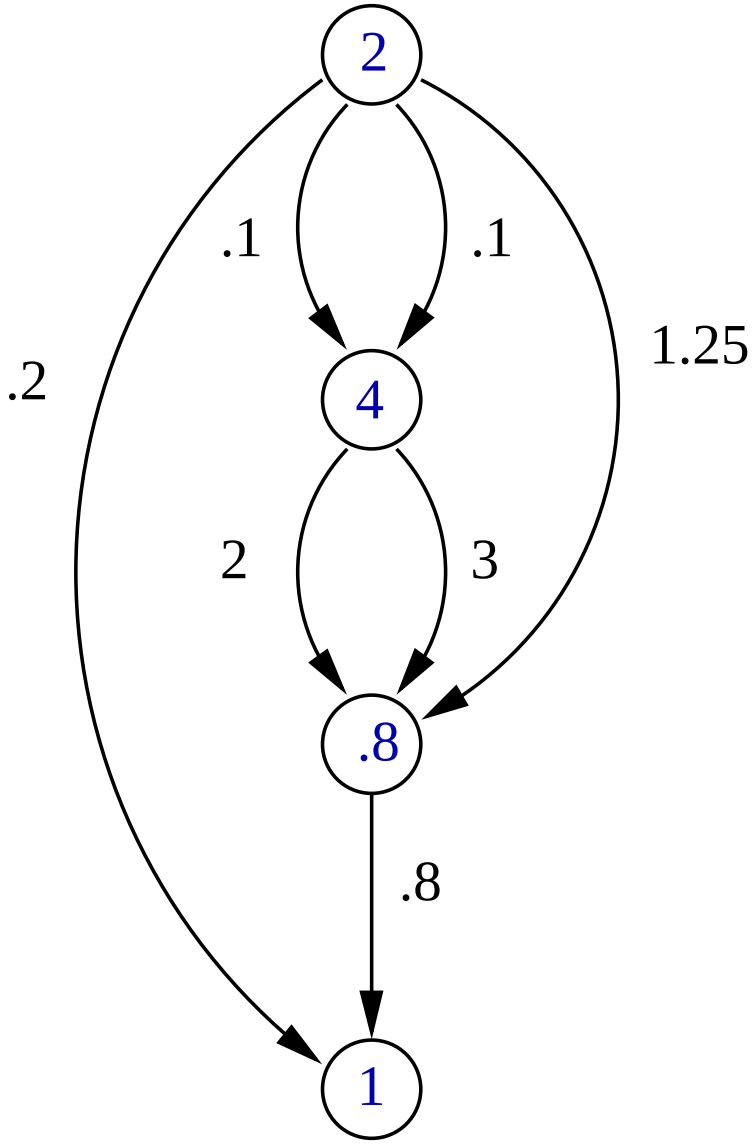
- Reestablish product form:

  Find $\widetilde{\boldsymbol{w}}$ s.t. $\widetilde{\mathcal{W}} = \Phi(\widetilde{\boldsymbol{w}})$
  I.e. find $\tilde{w}_e$ s.t. $\widetilde{W}_P = \prod_{e \in P} \tilde{w}_e$

In product form



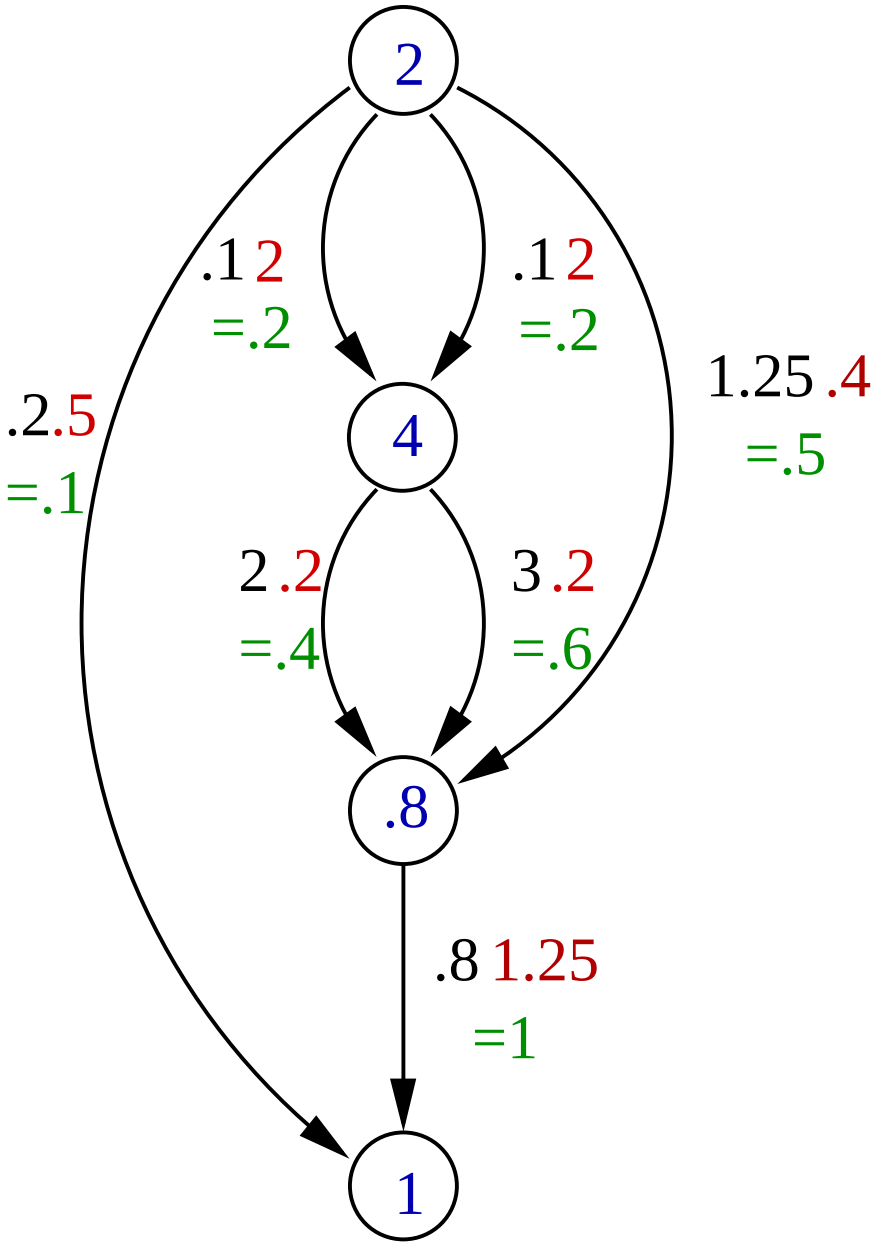
Factors destroy product form



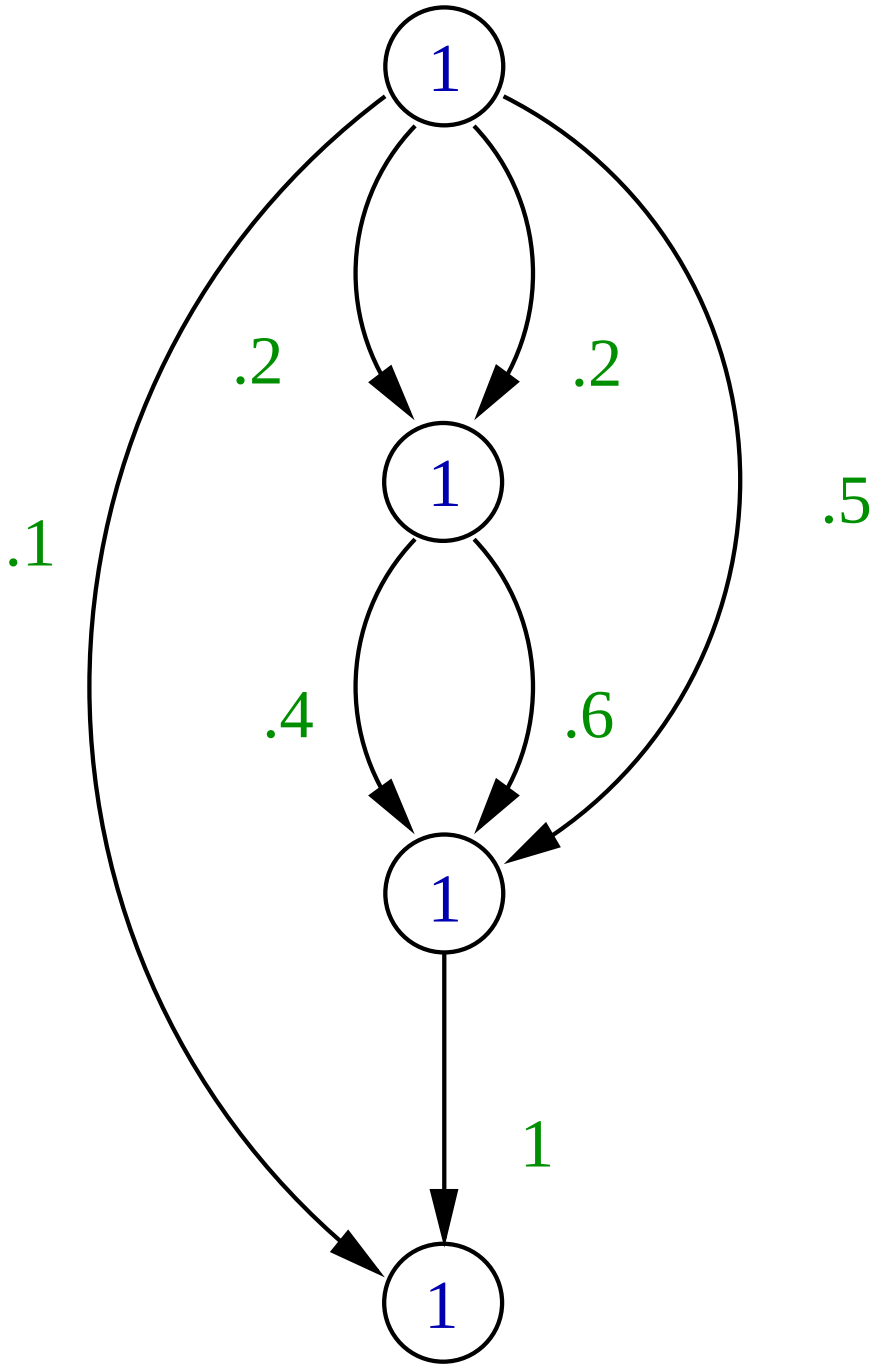
Normalization: $k(\boldsymbol{w}, \boldsymbol{b}) = 2$

Local normalization does not work

Reestablish product form



Outflow one again



Weight Pushing Algorithm [M]

$$K_u(\boldsymbol{w}, \boldsymbol{b}) = \sum_{P: u \to \text{sink}} \prod_{e \in P} w_e b_e$$

$$\widetilde{W}_P = \left(\prod_{e \in P} w_e b_e \right) \frac{1}{k(\boldsymbol{w}, \boldsymbol{b})}$$

$$\frac{1}{k(\boldsymbol{w}, \boldsymbol{b})} = \frac{K_{\texttt{sink}}(\boldsymbol{w}, \boldsymbol{b})}{K_{\texttt{source}}(\boldsymbol{w}, \boldsymbol{b})} = \prod_{i=1}^{k} \frac{K_{u_i}(\boldsymbol{w}, \boldsymbol{b})}{K_{u_{i-1}}(\boldsymbol{w}, \boldsymbol{b})}$$

where $\langle u_i \rangle$ is any *source* to *sink* path

Update:

$$\widetilde{w}_{(u,v)} = w_{(u,v)} b_{(u,v)} \frac{K_v(\boldsymbol{w}, \boldsymbol{b})}{K_u(\boldsymbol{w}, \boldsymbol{b})}$$

Another way to see it

$$P(u \to v) = \frac{\text{Total}(s \to u) w(u,v) b(u,v) \text{Total}(v \to t)}{\text{Total}(s \to u) \text{Total}(u \to t)},$$

where Total$(p \to q)$ is total of all products of factors along all $p \to q$ paths

Note that Total$(u \to t) = K_u(\boldsymbol{w}, \boldsymbol{b})$

Decomposable Multiplicative Updates?

Want $\text{fact}_P = \prod_{e \in P} b_e$

- $$\widetilde{W}_P = \frac{W_P \exp(-\eta \text{loss}_P)}{\sum_P W_P \exp(-\eta \text{loss}_P)}$$

  $$\text{loss}_P = \sum_{e \in P} \text{loss}_e \text{ and } b_e = \exp(-\eta \text{loss}_e)$$

  Loss must decompose

- $$\widetilde{W}_P = W_P \exp\left(-\eta \frac{\partial \text{loss}}{\partial W_P}\right) / Z$$

  Or gradient of loss must decompose

On-line Shortest Path Problem

- Features/Experts are the paths
- Predict with random path
- Each edge incurs a loss/distance $\in [0, 1]$
- Loss of path is sum of loss of its edges
- Multiplicative update give a factor to each edge
- Weight Pushing Algorithm establishes weights in product form

Learning Conjunctions

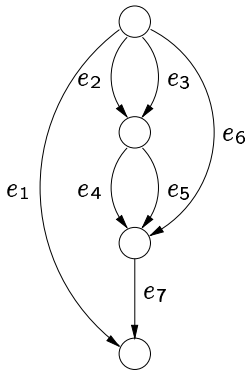
- All subsets kernel - $x_i \in \{0, 1\}$
- Products corresponding to subsets are conjunctions
- Discrete loss $\text{loss}_C = |y - X_C|$ does not decompose
- Winnow algorithm based on this loss would learn DNF formulas
- Computing predictions is $\#P$-hard [KRS]

- Upper bound discrete loss by decomposable hinge loss

- Now our method leads to Winnow algorithm for learning conjunction and disjunction

Pruning

Minimal set of edges that
interrupts all source-sink paths



- Each pruning R is a feature

- If input to edge e is x_e, then $x_R = \prod_{e \in R} x_e$

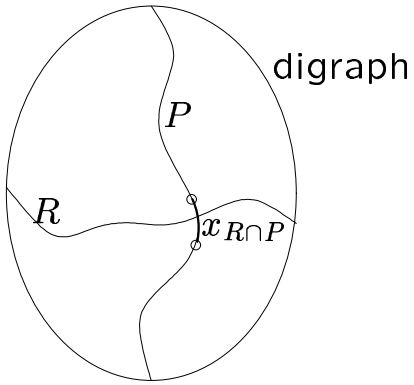
$$\Phi^d(\boldsymbol{x}) = (x_1x_2x_3x_6, x_1x_4x_5x_6, x_1x_7)$$

- Prunings a paths in dual SP digraph where S and P are swapped

Predict as well as best pruning [HW97, TW00]

- Instance is a source-sink path P
- $$\text{Input to edge } e = \begin{cases} x_e & \text{if } e \in P \\ 1 & \text{if } e \notin P \end{cases}$$
- Pruning R predicts as edge that cuts P

$$X_R = x_{R \cap P} = \prod_{e \in R} x_e$$



- Expanded input vector is $\mathcal{X} = \Phi^d(\boldsymbol{x})$

Decomposable multiplicative update?

- Loss is sum over edges

$$\text{loss}_R = \sum_{e \in R} \text{loss}_e$$

- Update factors $b_e = exp(-\eta \text{loss}_e)$

- Maintain implit weights $\boldsymbol{w}$ using Weight Pushing Algorithm

- Predict $\mathcal{W} \cdot \mathcal{X} = \Phi^d(\boldsymbol{w}) \cdot \Phi^d(\boldsymbol{x}) = k(\boldsymbol{w}, \boldsymbol{x})$

Future

- Connection to Graphical Models?
- Weight Pushing versus
  Sum Product Algorithm?
- Kernels with p-norm algorithms