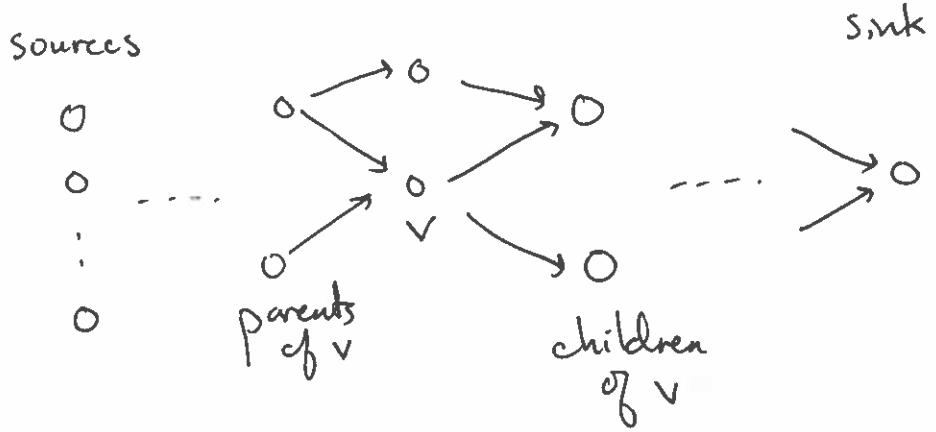
Backpropagation for training

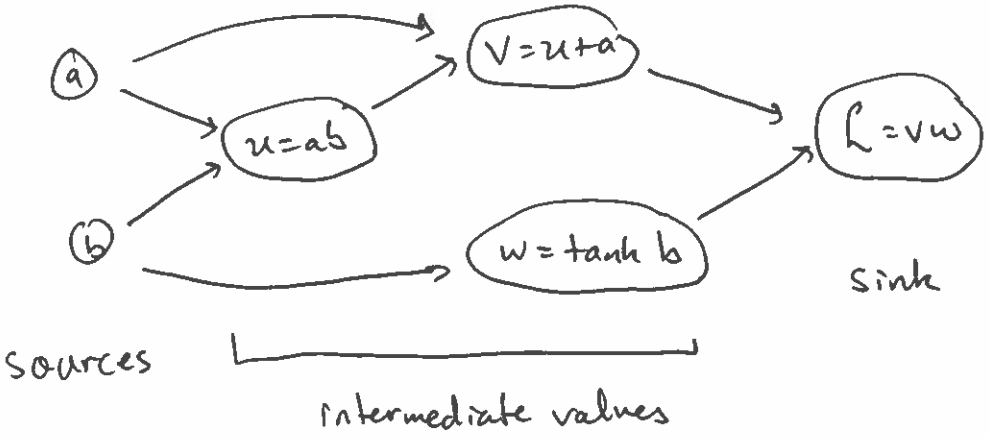
Goal: Gradient descent: $\theta_{t+1} = \theta_t - \eta \nabla \mathcal{L}(\theta_t)$

Problem: efficiency to compute $\nabla \mathcal{L}(\theta_t)$

Computational Graphs



Running Example



DAG = directed acyclic graph

Def' A "topological ordering" on vertices of DAG is an ordering s.t. parents occur before children.

Def' A scalar computational graph is a DAG w/ topological ordering

$$v_1, \ldots, v_m$$

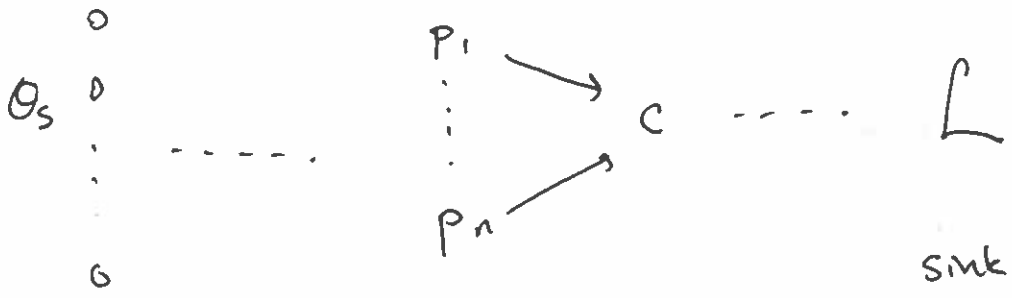
and nodes labelled by differentiable functions of their parents:

$$v_i = \varphi_i\,(v_j : j \in \text{parents}(i))$$

If sources are initialized to #s, this gives values at all vertices.

The "forward pass" computes these values left to right using φ_i.

Forward-Mode Automatic Differentiation : to compute $\frac{\partial L}{\partial \theta_s}$



left-to-right

$$\dot{c} := \frac{\partial c}{\partial \theta_s} = \sum_{\text{parents}} \frac{\partial c}{\partial p} \cdot \frac{\partial p}{\partial \theta_s}$$

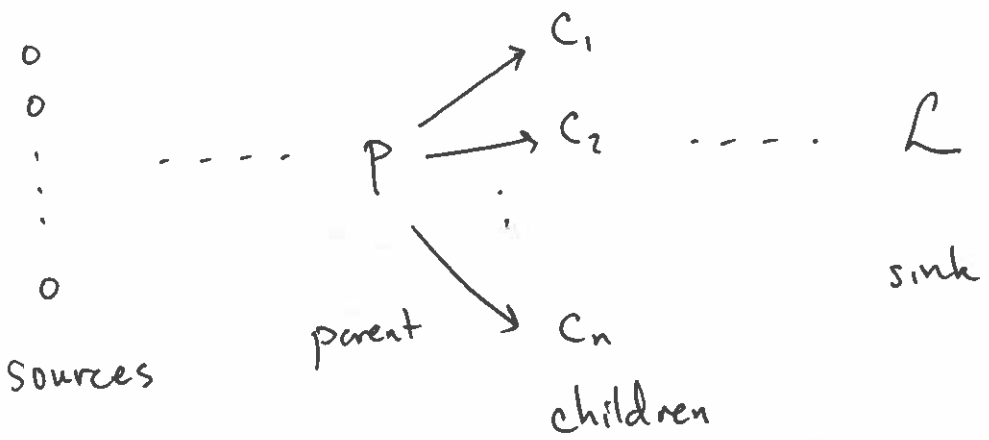
"tangent"

compute now

known by previous computation

- begin w/ $\frac{\partial \theta_s}{\partial \theta_s} = 1$
- values known by forward pass
- computing numbers not functions
- one pass per source

Backpropagation: to compute $\frac{\partial \mathcal{L}}{\partial \theta_s}$



right-to-left

$$\bar{p} := \frac{\partial \mathcal{L}}{\partial p} = \sum_{\text{children}} \frac{\partial \mathcal{L}}{\partial c} \cdot \frac{\partial c}{\partial p}$$

"adjoint"
"gradient"

compute now

known by previous steps

- begin w/ $\frac{\partial \mathcal{L}}{\partial \mathcal{L}} = 1$
- values known by forward pass
- computing #s
- one pass per sink (more efficient)

Def' Feedforward neural language model consists of:

① encoder $e_{\theta_1} : V^* \longrightarrow \mathbb{R}^{n_0}$
context $\longmapsto$ feature vec

② feed f. net. $f_{\theta_2} : \mathbb{R}^{n_0} \to \mathbb{R}^V$

giving $P(\theta, c) = p_\theta(\cdot \mid c) = \text{softmax}(f_{\theta_2}(e_{\theta_1}(c)))$

Example "Bengio-style" "neural n-gram"

encoder: $C \in \mathbb{R}^{V \times d}$ embedding matrix gives $\{ V \to \mathbb{R}^d \;\; c \mapsto r_c \; (c^{th} \text{ row}) \}$

So tokens $a_1 \cdots a_{n-1} \longmapsto \vec{x} = r_{a_1} \cdots r_{a_{n-1}} \in \mathbb{R}^{(n-1)d}$ concatenated into vector

ff. net: one hidden layer:

$$h_{ctx} = \tanh(W_1 \vec{x} + b_1) \qquad z = W_2 h_{ctx} + b_2$$

$$\theta = C, W_1, b_1, W_2, b_2$$

$$V \times d,\; h \times (n-1)d,\; h,\; V \times h,\; V$$

Training

Batch size B

Learning Rate Schedule $\eta_0, \eta_1, \eta_2, \ldots$

For $t = 0, 1, 2, \ldots$

① Sample B examples (x_i, y_i) "minibatch"

② Forward Pass on θ_t to compute loss ("minibatch loss")
& record computational graph.

③ Backprop on graph to compute $\nabla \mathcal{L}(\theta)$

④ Update $\theta_{t+1} = \theta_t - \eta_t \nabla \mathcal{L}(\theta)$

Diamond Classifier Demo

Playground Training Demo

Bengio?