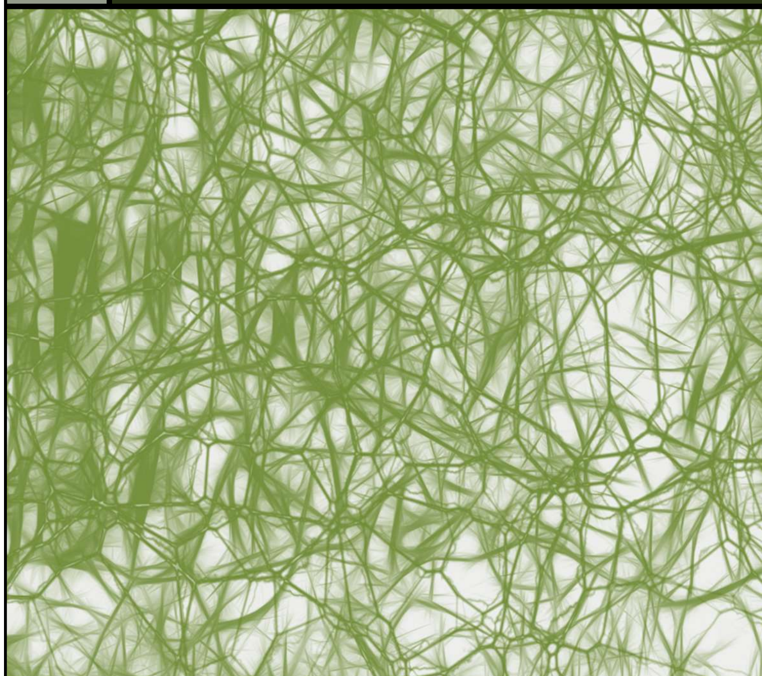
Department of Computer Science
University of Bristol

COMSM0045 – Applied Deep Learning 2020/21

comsm0045-applied-deep-learning.github.io



Lecture 2

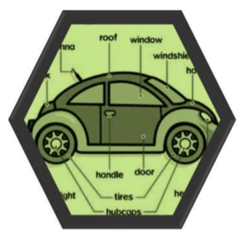
TOWARDS TRAINING DEEP FORWARD NETWORKS

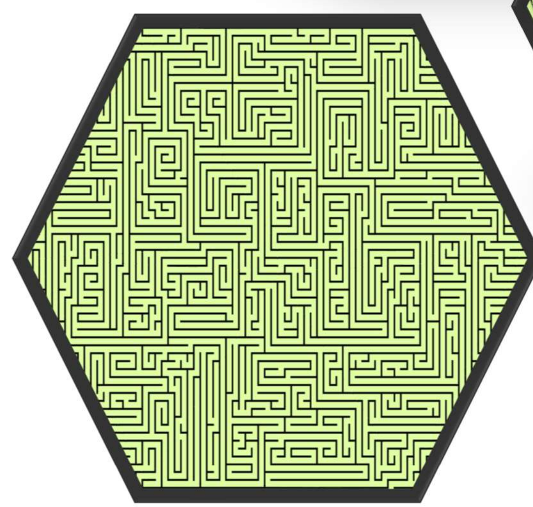
Tilo Burghardt | tilo@cs.bris.ac.uk

18 Slides

Agenda for Lecture 2

- Recap Gradient Descent
- Computational Graphs
- Reverse Auto-Differentiation

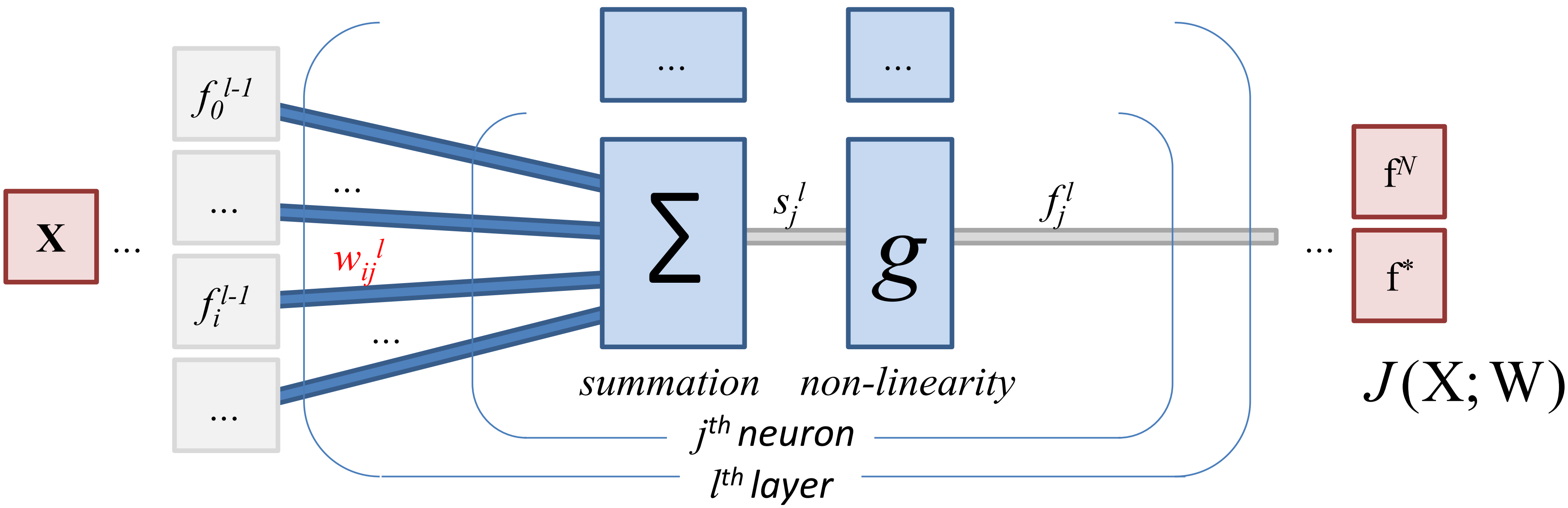




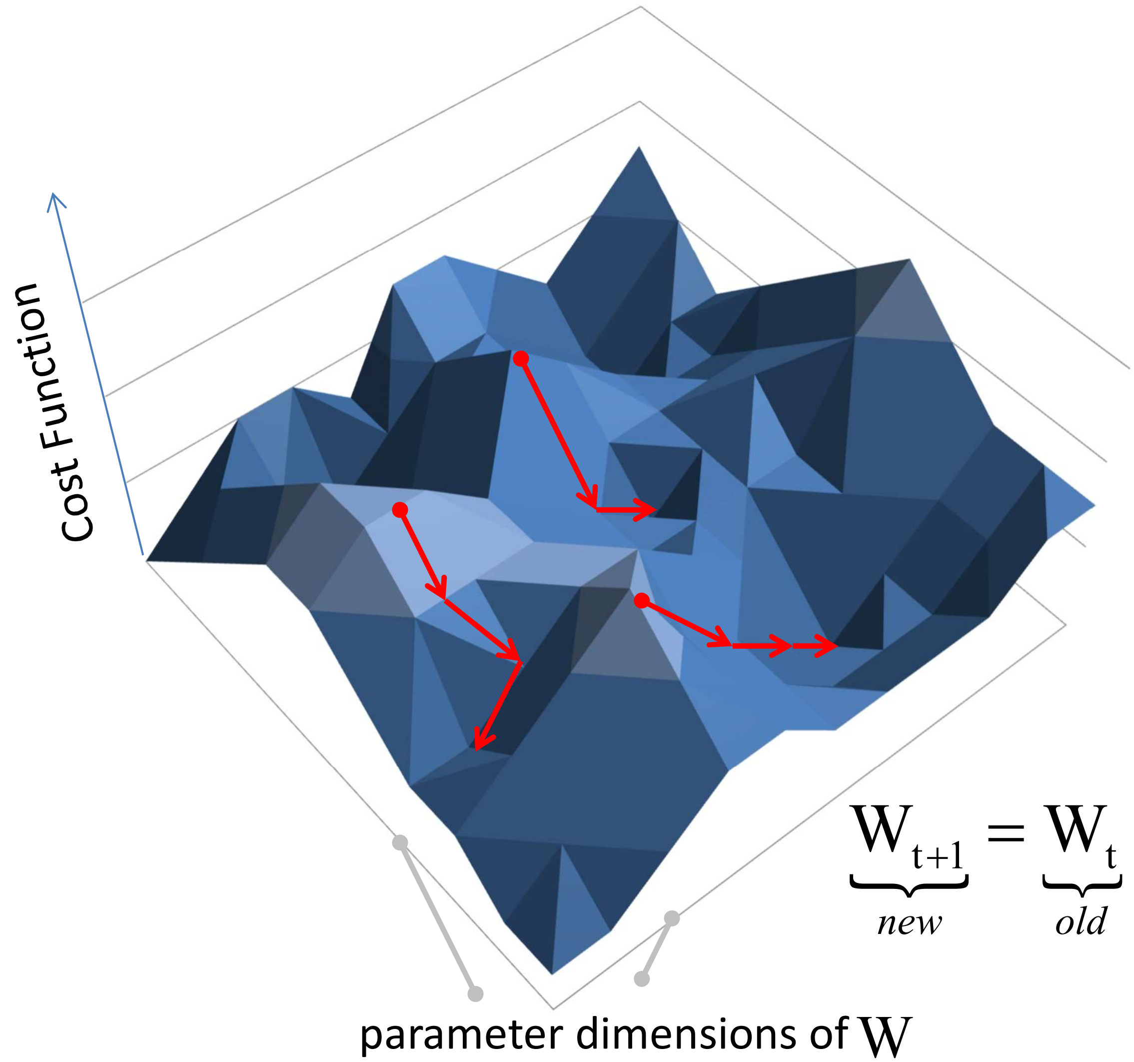
Recap: Gradient Descent

The Global Training Problem

- **Training Problem:** We have a (highly) non-linear function, the cost function $\boldsymbol{J}$ of a network, and we want to find a parameterization $\mathbf{W}$ across **all** weights w_{ij}^{l} that minimizes it...

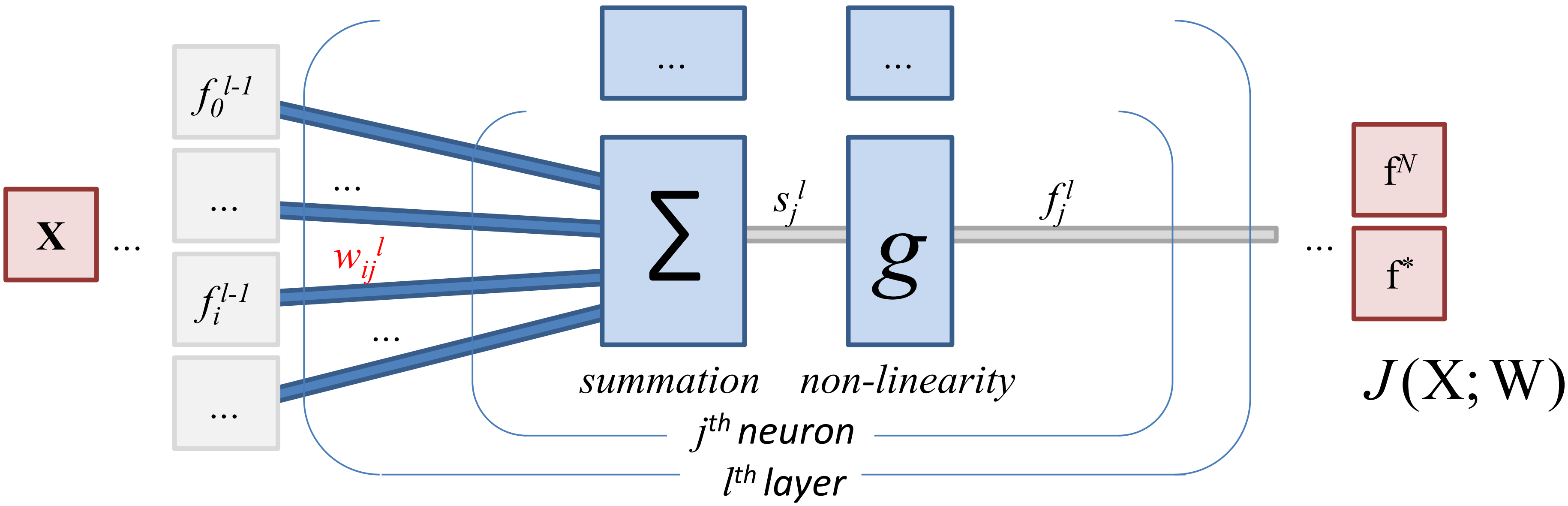


Recap: Idea of ‘Steepest’ Gradient Descent



$$\underbrace{W_{t+1}}_{new} = \underbrace{W_t}_{old} - \underbrace{\eta}_{learning\ rate} \underbrace{\nabla J(X; W_t)}_{steepest\ gradient}$$

Partial Derivatives of Interest



We require: $\nabla J(X;W) = \nabla J_X(W)$ as given by **all** $\dfrac{\partial J_X(W)}{\partial w_{ij}^l}$,

where w_{ij}^l is the i^{th} weight to the j^{th} neuron of the l^{th} layer. Thus, we need to compute partial derivatives of J w.r.t. **all** weights.

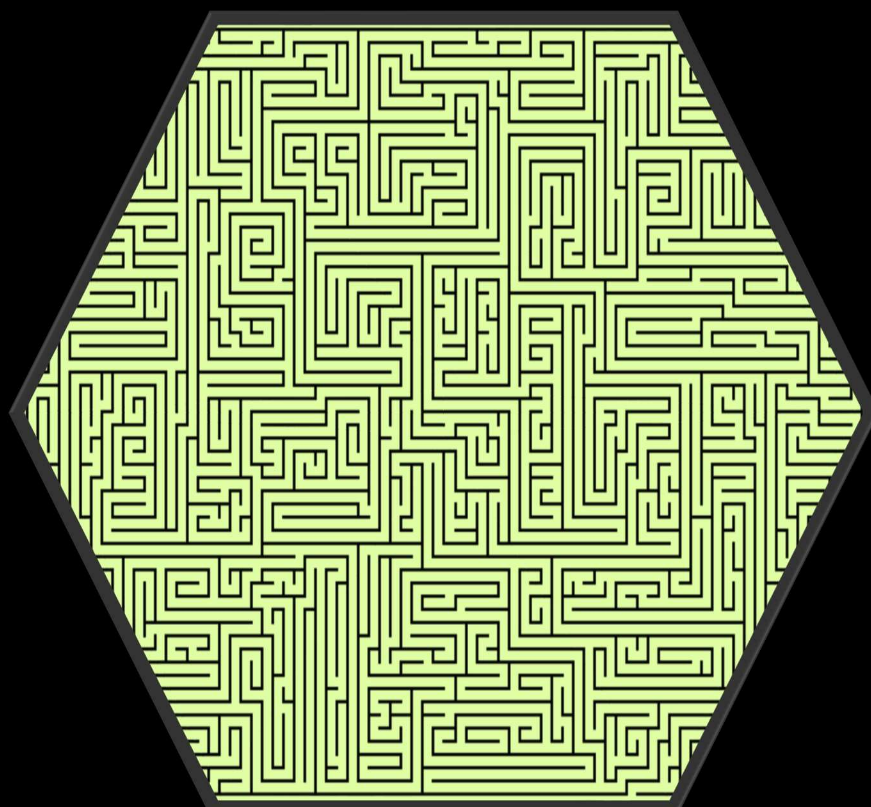
Potential Options for Calculating Error Derivatives

- **Symbolic Differentiation?**
  - not supporting arbitrary setups
  - solution structure may not resemble network structure at all

- **Numerical Differentiation?**
  - trivial to implement
  - low accuracy
  - potentially high computational cost

→ **Automatic Differentiation of the Network's Computational Graph** (as used by Tensorflow)

Reverse Auto-Differentiation in Computational Graphs



Relationships in Feedforward Computational Graphs

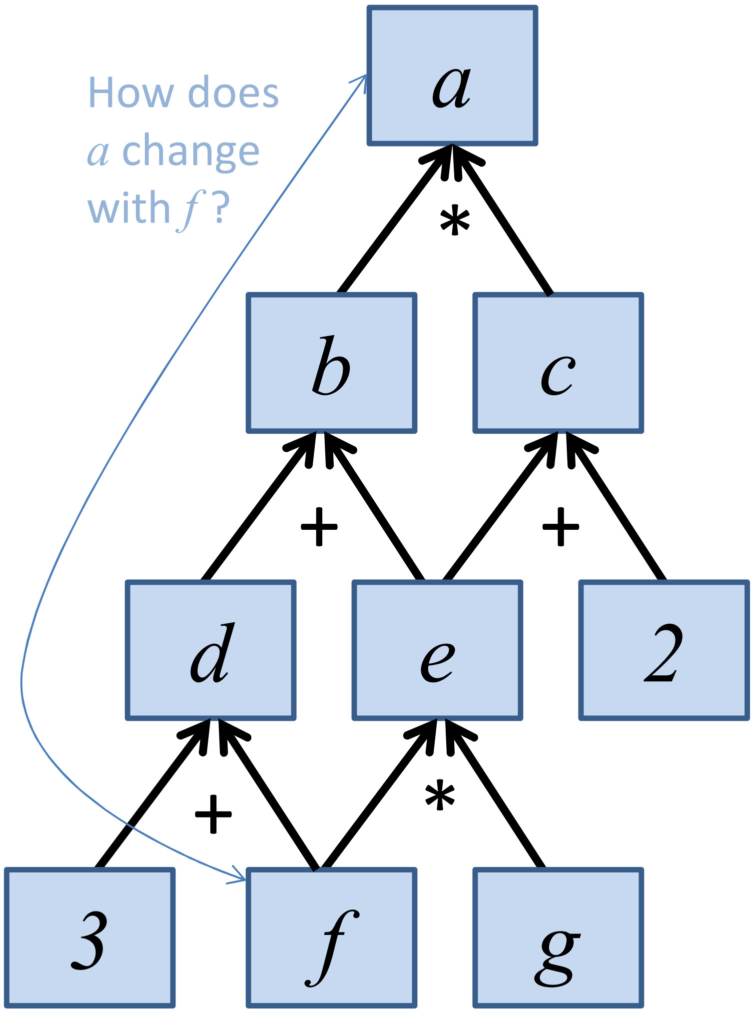
$a = b * c$

$b = d + e$

$c = e + 2$

$d = 3 + f$

$e = f * g$



$$\frac{\partial a}{\partial f} = ?$$

Analytical Solution:

$$\begin{aligned} a &= (d+e)(e+2) = de + 2d + e^2 + 2e \\ &= (3+f)fg + 2(3+f) + (fg)^2 + 2fg \\ &= 3fg + f^2 g + 6 + 2f + f^2 g^2 + 2fg \\ &= f^2(g^2+g) + 5fg + 2f + 6 \end{aligned}$$

$$\frac{\partial a}{\partial f} = 2fg^2 + 2fg + 5g + 2$$

A General Strategy: Chain Rule and Summing over all Paths

$a = b * c$

$b = d + e$

$c = e + 2$

$d = 3 + f$

$e = f * g$

Analytical Solution:

$$a = (d+e)(e+2) = de + 2d + e^2 + 2e$$
$$= (3+f)fg + 2(3+f) + (fg)^2 + 2fg$$
$$= 3fg + f^2 g + 6 + 2f + f^2 g^2 + 2fg$$
$$= f^2(g^2+g) + 5fg + 2f + 6$$
$$\frac{\partial a}{\partial f} = 2fg^2 + 2fg + 5g + 2$$

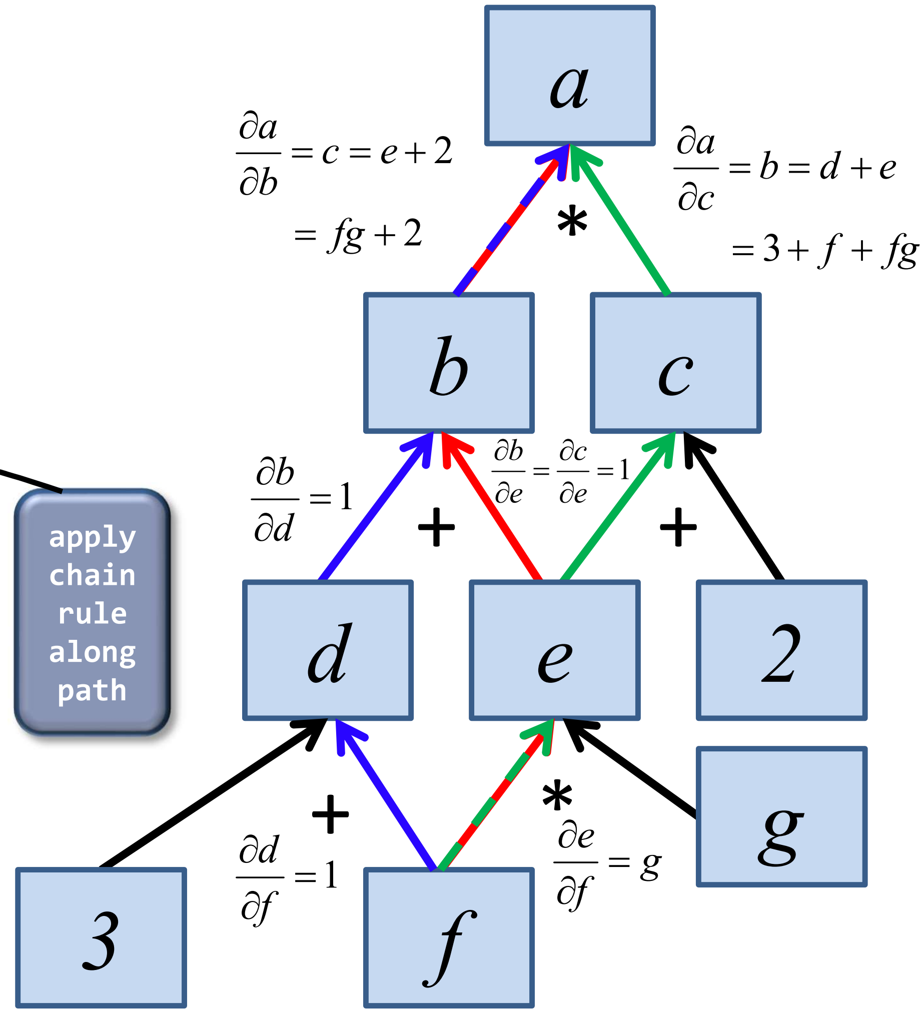
General Approach based on Network Layout:

$$\frac{\partial a}{\partial f} = \underbrace{\frac{\partial a}{\partial b}\frac{\partial b}{\partial e}\frac{\partial e}{\partial f}}_{path1} + \underbrace{\frac{\partial a}{\partial c}\frac{\partial c}{\partial e}\frac{\partial e}{\partial f}}_{path2} + \underbrace{\frac{\partial a}{\partial b}\frac{\partial b}{\partial d}\frac{\partial d}{\partial f}}_{path3}$$

apply chain rule along path

sum over all paths that connect f to a

$$= (fg+2) + (fg+2)g + (3+f+fg)g$$
$$= fg + 2 + fg^2 + 2g + 3g + fg + fg^2$$
$$= 2fg^2 + 2fg + 5g + 2$$



Observation of Hierarchical Dependency

Global Structure used so far:

$$\frac{\partial a}{\partial f}=\underbrace{\frac{\partial a}{\partial b}\frac{\partial b}{\partial e}\frac{\partial e}{\partial f}}_{path1}+\underbrace{\frac{\partial a}{\partial c}\frac{\partial c}{\partial e}\frac{\partial e}{\partial f}}_{path2}+\underbrace{\frac{\partial a}{\partial b}\frac{\partial b}{\partial d}\frac{\partial d}{\partial f}}_{path3}$$

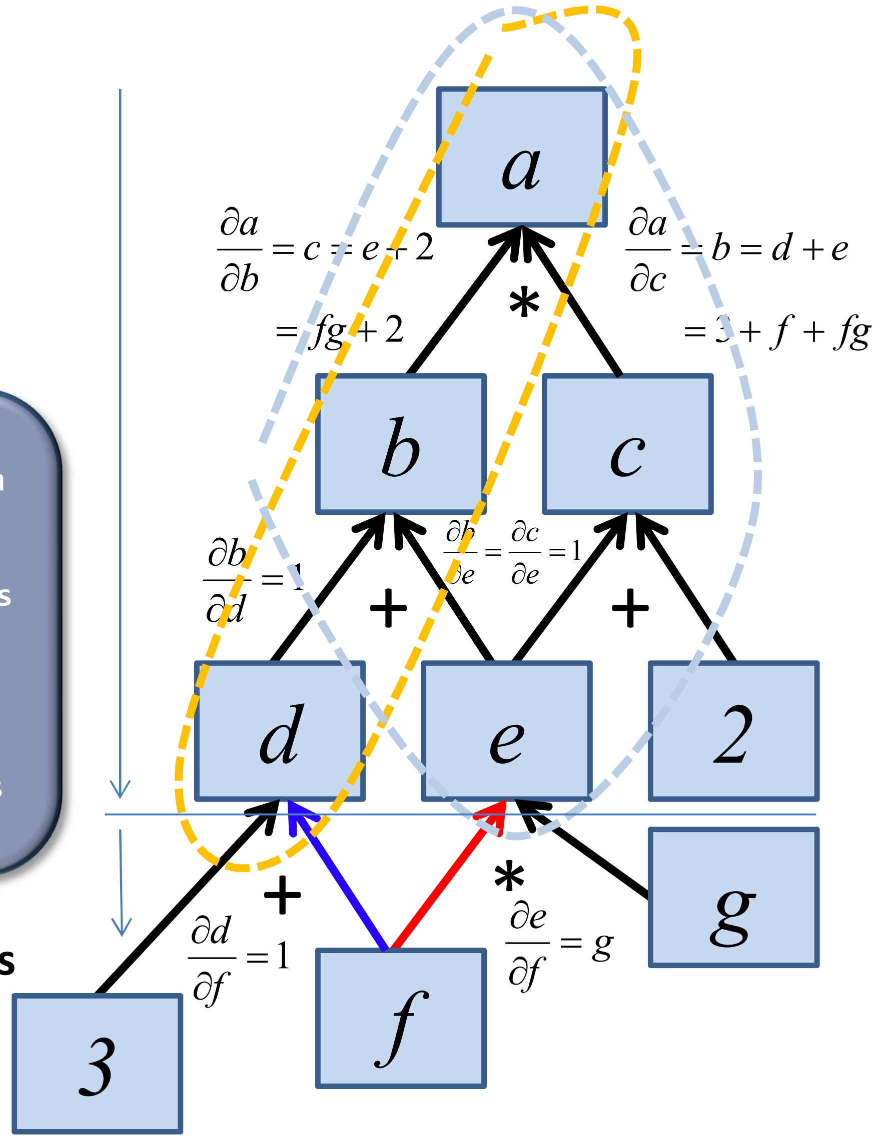
Hierarchical Structure:

$$\frac{\partial a}{\partial f}=\frac{\partial a}{\partial e}\frac{\partial e}{\partial f}+\frac{\partial a}{\partial d}\frac{\partial d}{\partial f}$$

$$\frac{\partial a}{\partial e}=\frac{\partial a}{\partial b}\frac{\partial b}{\partial e}+\frac{\partial a}{\partial c}\frac{\partial c}{\partial e}$$

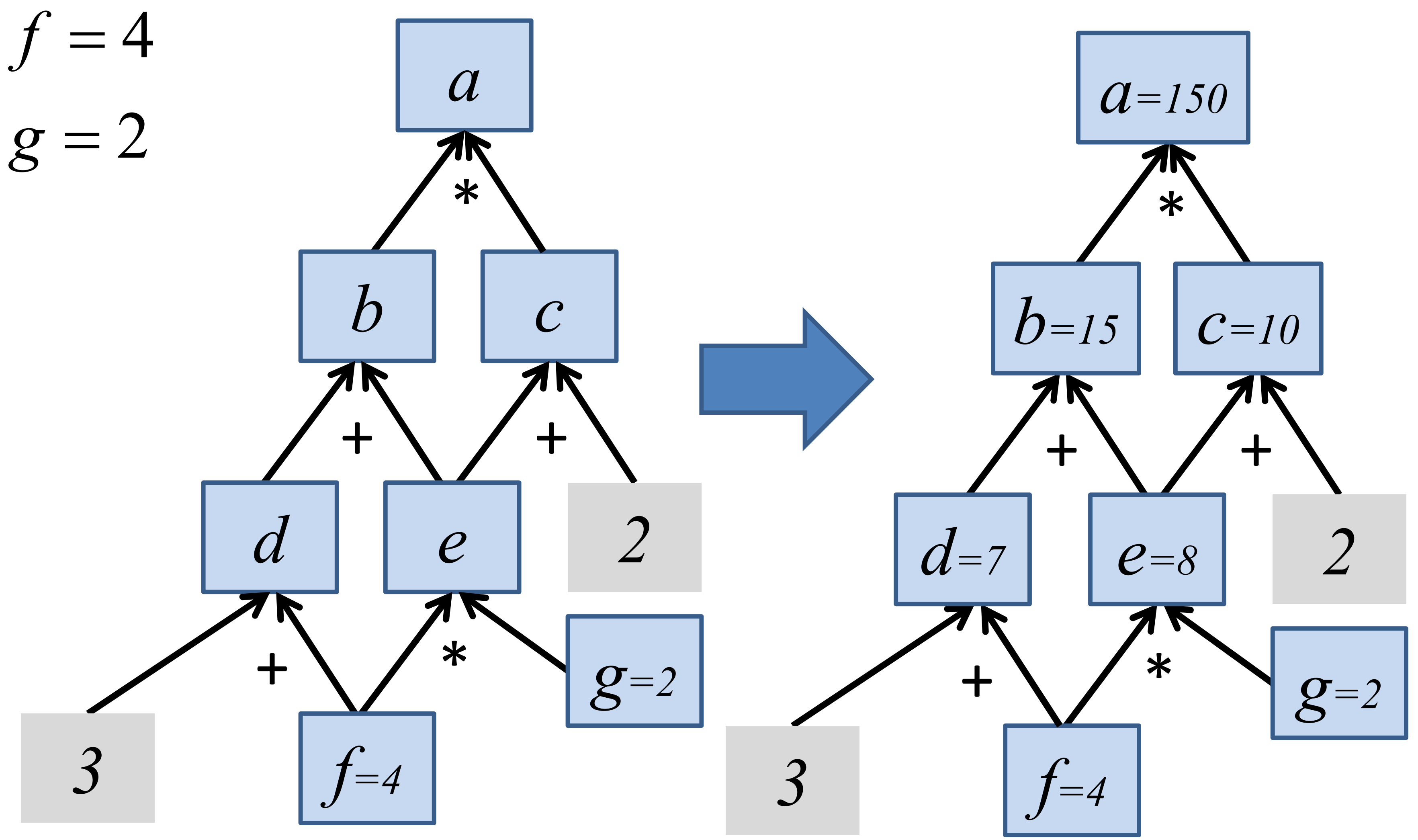
$$\frac{\partial a}{\partial d}=\frac{\partial a}{\partial b}\frac{\partial b}{\partial d}\quad\ldots$$

We observe that, to calculate results from the layer above, for each node we can sum over all incoming edges from the layer above and multiply each by the result we have obtained in the node that the edge connects to in the layer above.

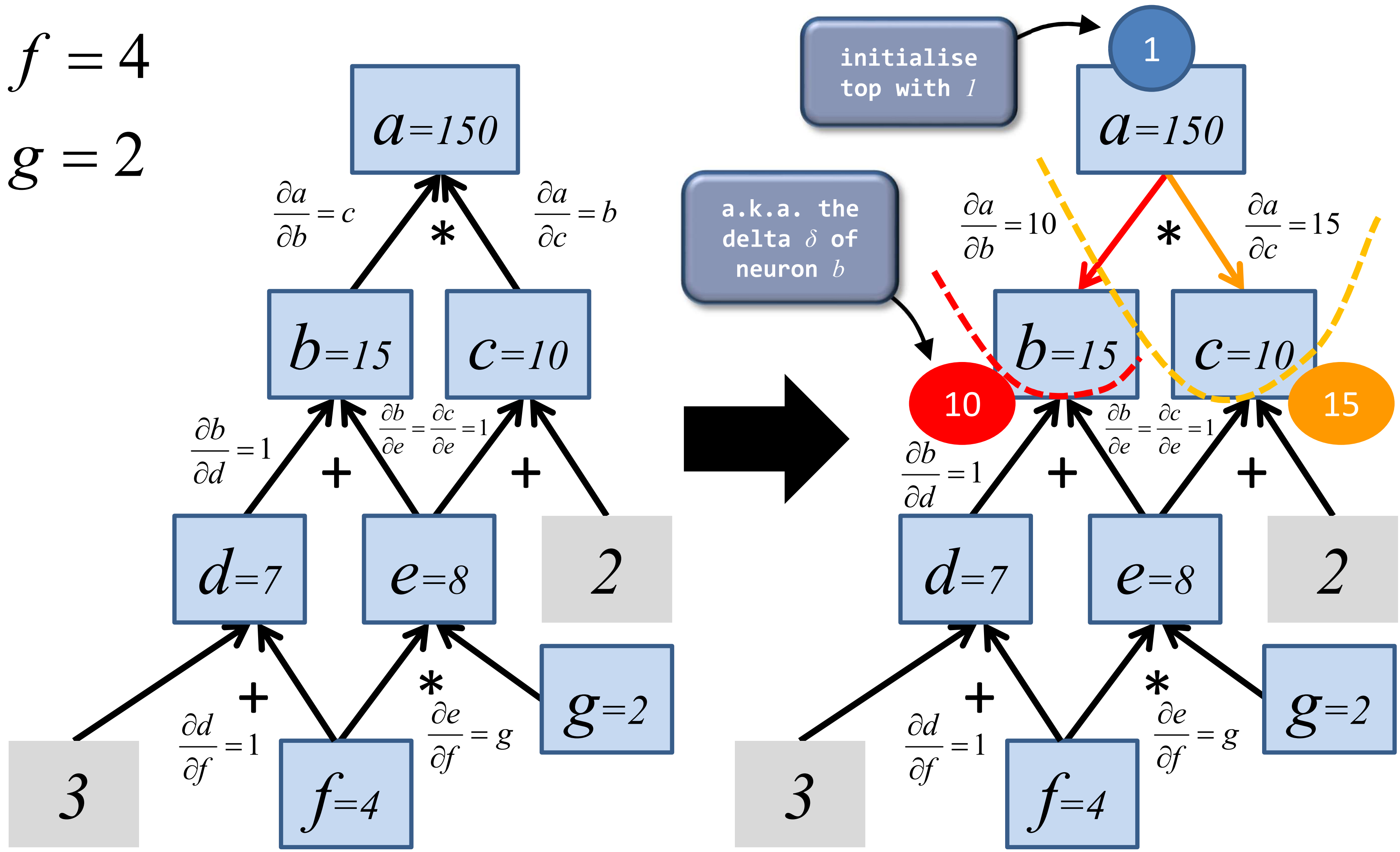


→ **once you know all (part-evaluated) derivatives associated to a layer above, summation of them from connected nodes times local derivatives is sufficient to get the next layer of derivatives**

Example Calculation: Complete Forward Pass



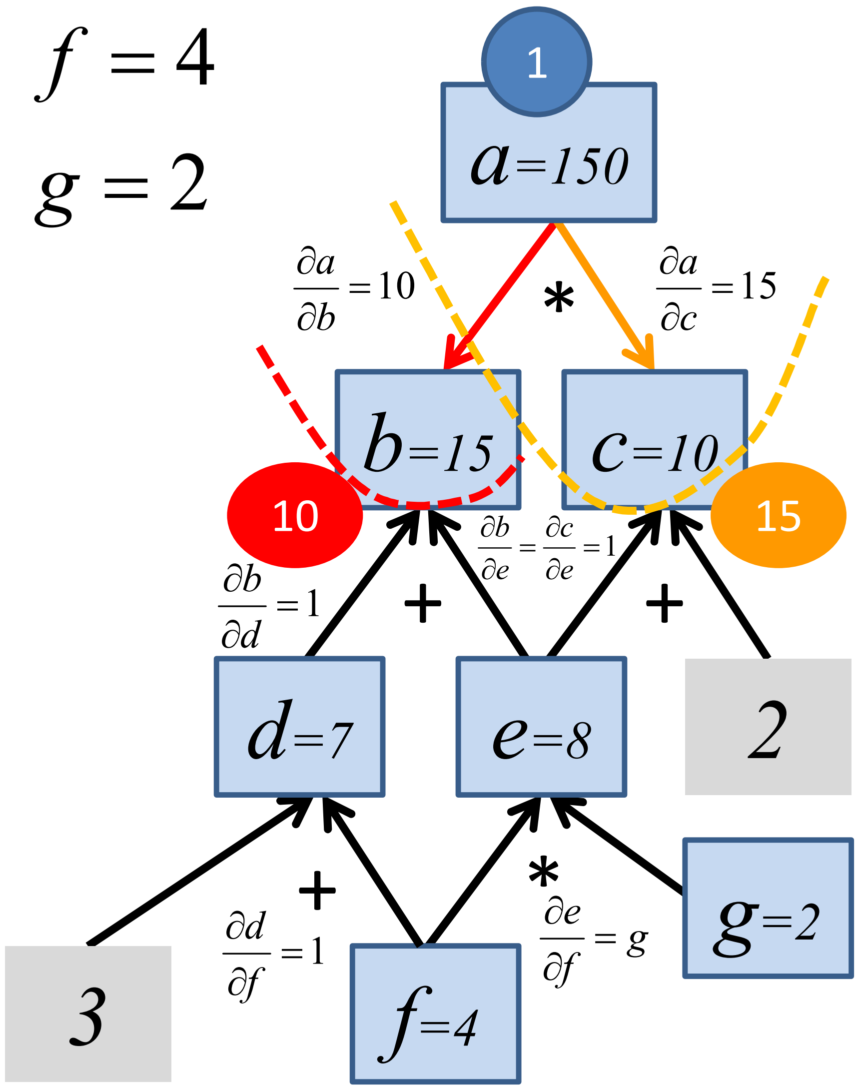
Example Calculation: Backward Pass Layer 1

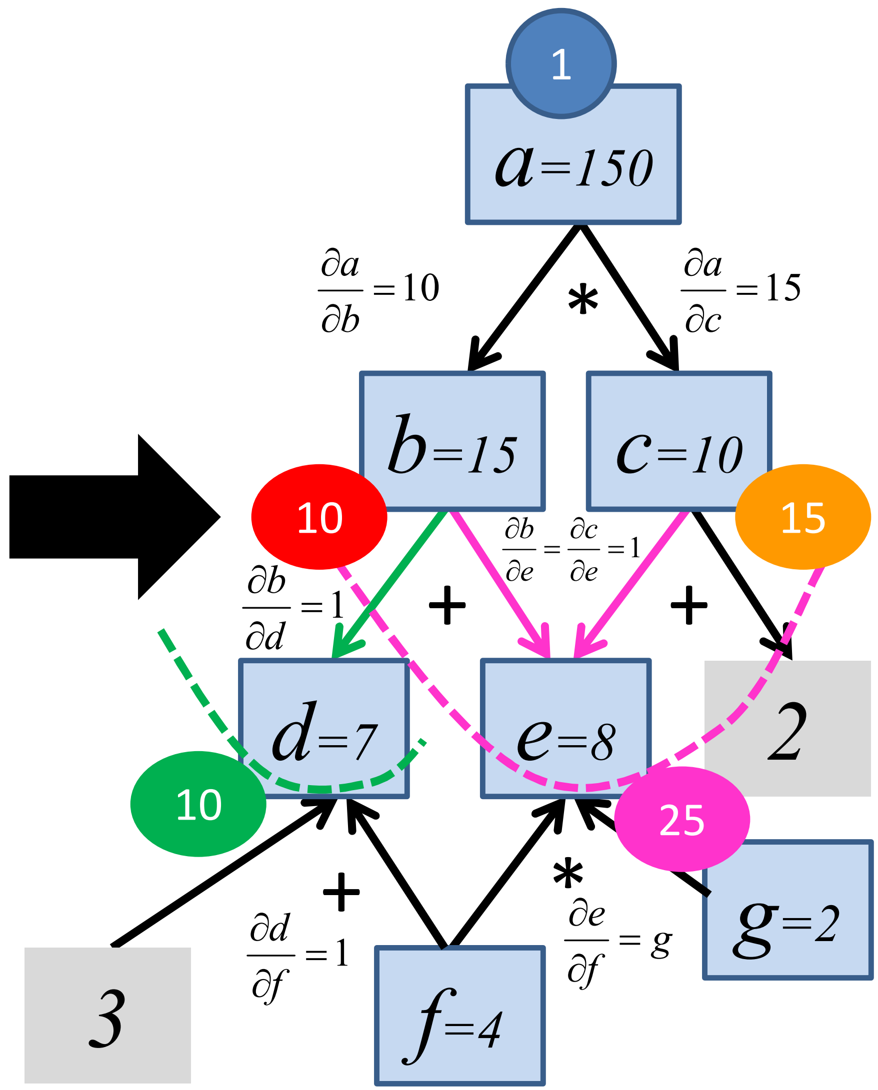


Example Calculation: Backward Pass Layer 2

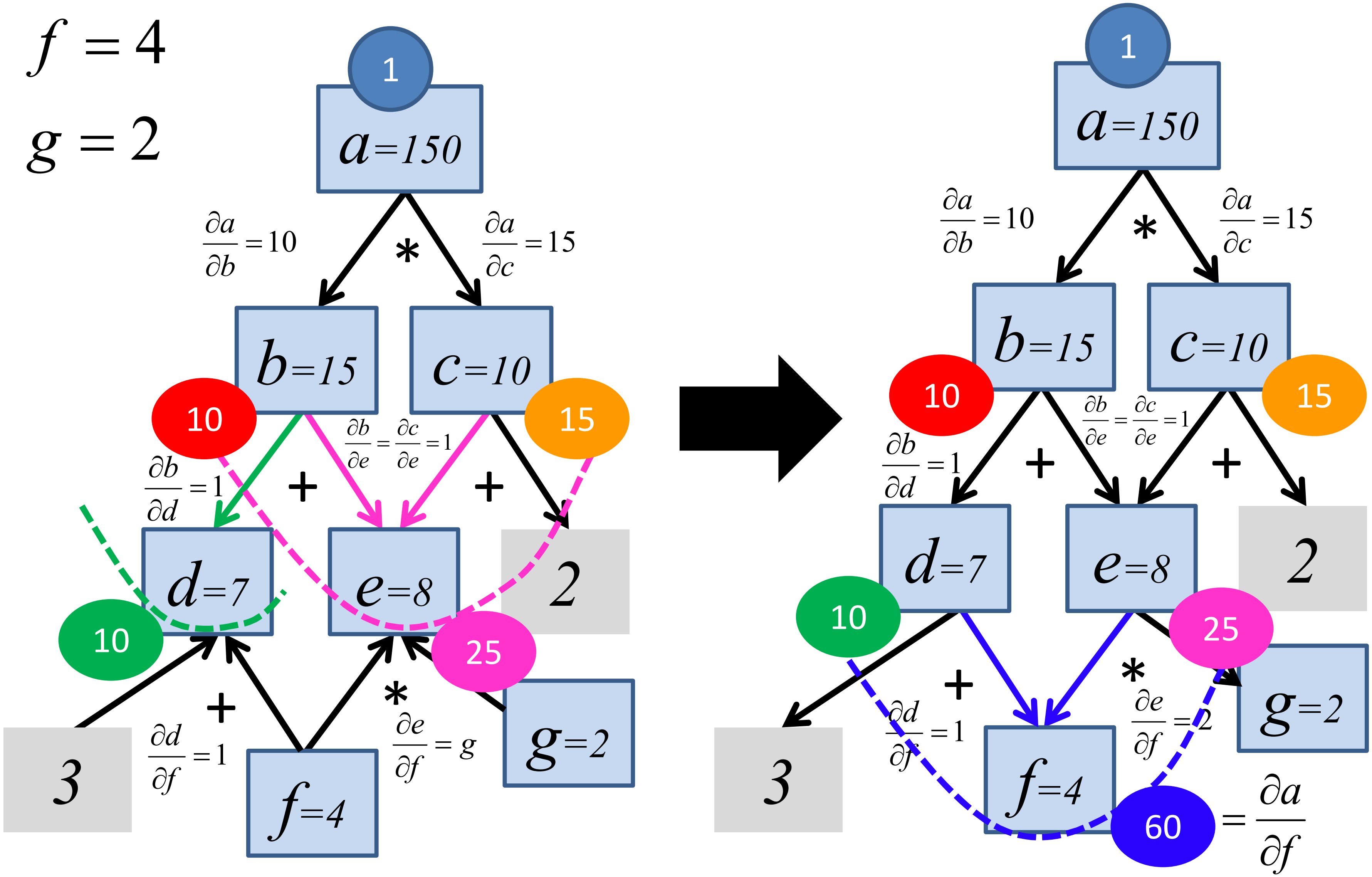
$f = 4$

$g = 2$



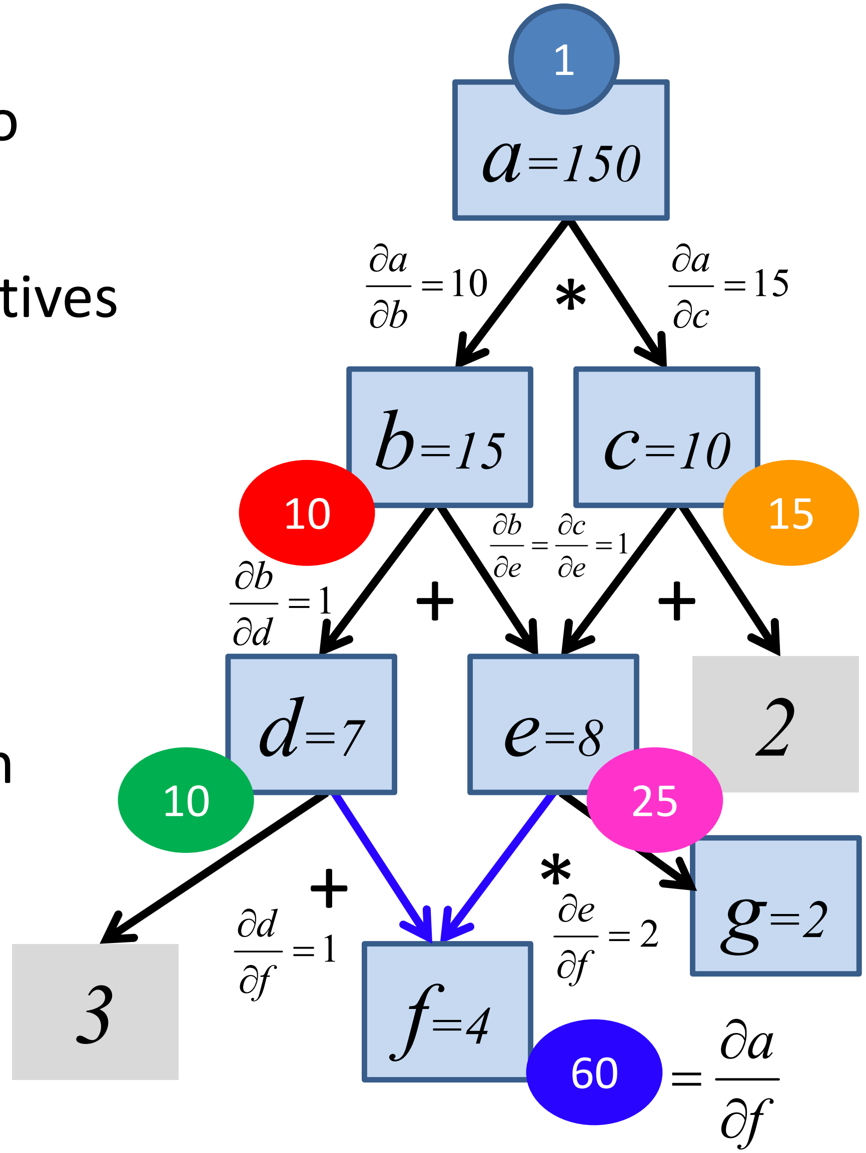


Example Calculation: Backward Pass Layer 3

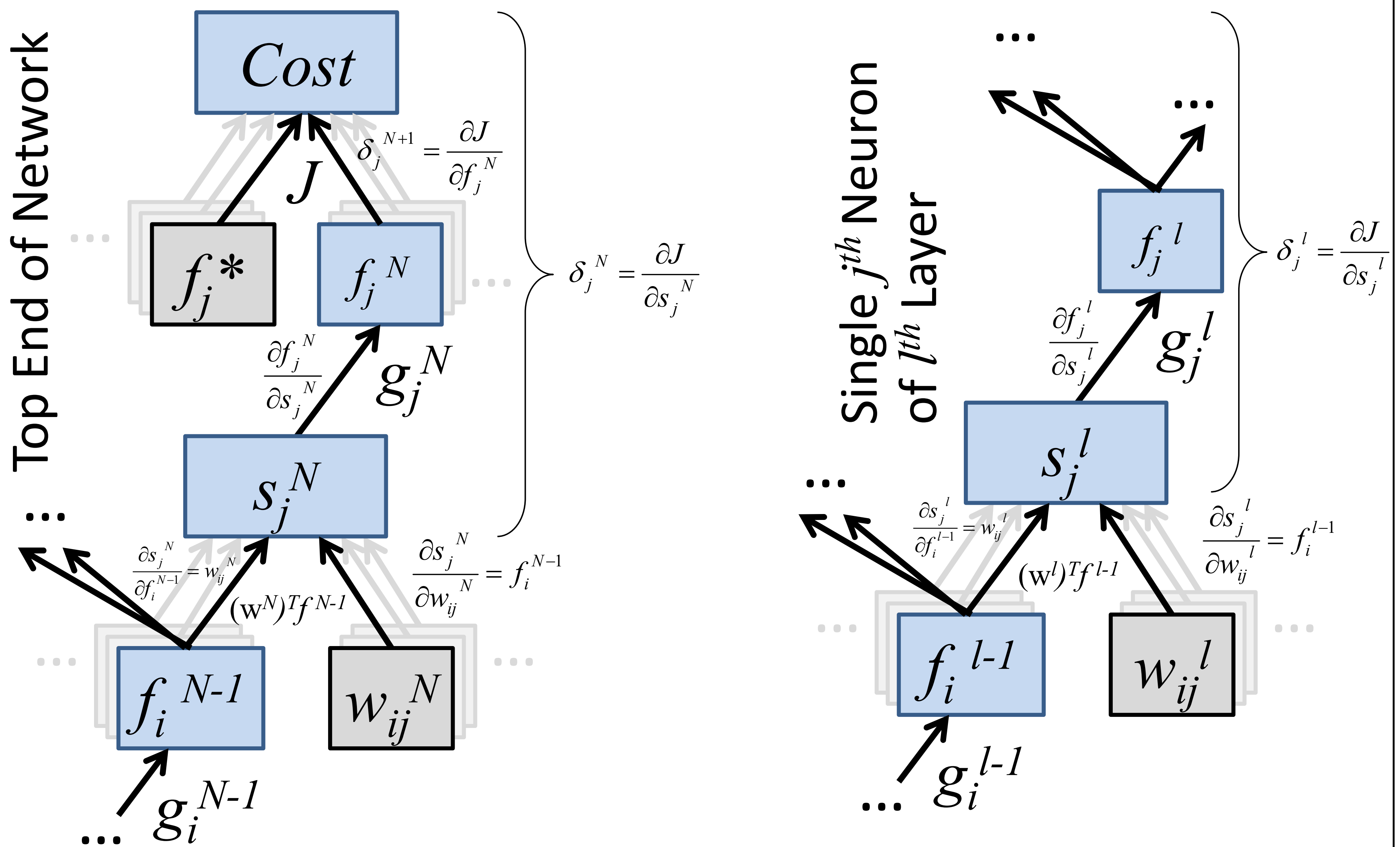


Summary of Reverse Auto-Differentiation

- **Two-pass Strategy**
  - forward pass to give values to nodes and output
  - backward pass to establish deltas δ, i.e. **<u>all</u>** partial derivatives
- **Requirements**
  - feed-forward network
  - local per-edge derivatives must be known
- **Solution Tactic**
  - instead of explicit summation over all paths, layer-by-layer evaluation via summation over all incoming local derivatives times their associated deltas



Deep Neural Networks as Special Computational Graphs



Next Time: Training

- The Backpropagation Algorithm in Full Detail
- Activation Functions