Deep Learning

4DV661

Deep Feedforward Networks

Welf Löwe

Course structure

1. Introduction
2. Applied Math Basics
3. Deep Feedforward Networks
4. Regularization for Deep Learning
5. Optimization for Training Deep Models
6. Convolutional Networks
7. Sequence Modeling: Recurrent and Recursive Nets
8. Practical Methodology
9. Applications

Agenda for today

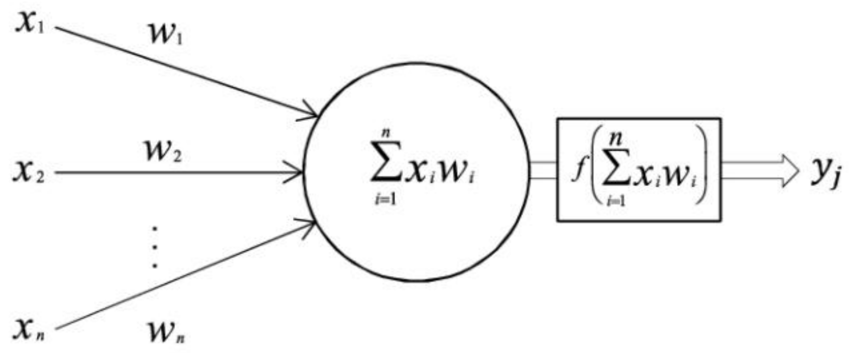
- Architecture
- Hidden Layers
- Example: Learning XOR (1/3)
- Gradient-Based Learning
- Example: Learning XOR (2/3)
- Back propagation
- Example: Learning XOR (3/3)

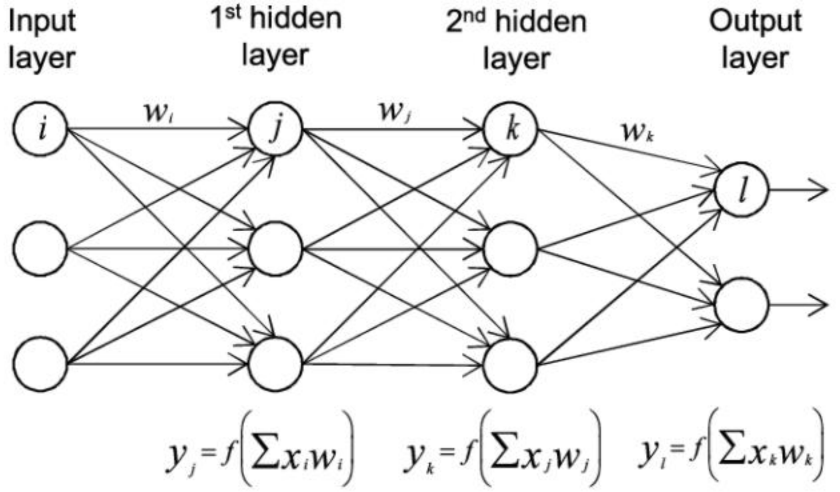
Neural Network Architecture

- A layered graph of computational nodes (or neurons)
- Layered graph
  - directed connected acyclic graph
  - layers L_0 ... L_k partition the nodes
  - each edge connects only nodes in successive layers
    - To all or a subset thereof
  - the depth is the number of layers
  - the width the greatest number of nodes in any layer.
- Layers that are neither input nor output are called hidden

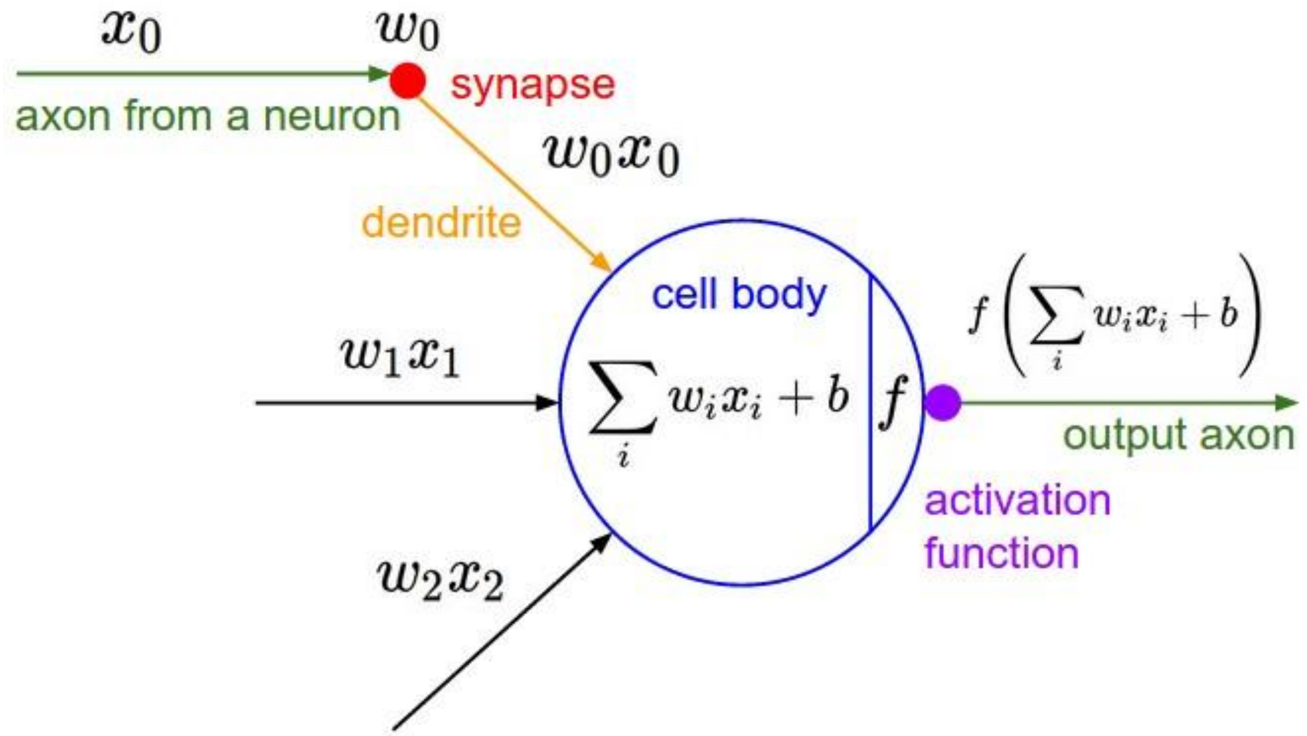
Neural Network and Neurons

- Core function: weighted sum Σ of input (+bias)
  - Weights (+bias) to be learned
- Activation f
  - Parameterized filter function
  - Parameters set, not learned
- The whole architecture describes a composed function





Neurons with cell and activation functions



Activation functions

- Known activation functions:
  - Identity (linear, no activation)
  - ReLU
  - Softplus
  - Logistic, Sigmoid
  - Tanh
  - Softmax

S-shaped

- they are monotonically increasing $x_1 \leq x_2 \Rightarrow f(x_1) \leq f(x_2)$
- they are all differentiable (with ReLU a theoretical exception at x=0)

Why several layers

- Neural network composition consisting of input and output layers
- If f is a linear (activation) function, the network computes just linear functions

$$y = f\left(\sum_{i=1}^{n}(w_i x_i)\right)$$

$$f(z) = wz$$

$$\begin{aligned} y &= w\left(\sum_{i=1}^{n}(w_i x_i)\right) \\ &= \sum_{i=1}^{n}(w w_i x_i) \\ &= \sum_{i=1}^{n}(w'_i x_i) \end{aligned}$$

Why several layers (cont’d)

- Neural network composition consisting of input and output layers

$$y = f\left(\sum_{i=1}^{n}(w_i x_i)\right)$$

- If f is a monotonically increasing function,
  - the network computes a function (result y) that is monotonic in x_i
  - increasing if w_i is positive and decreasing, otherwise
- Hence, with linear core and monotonic activation functions, we need hidden layers to compose non-monotonic functions
- Actually, we can approximate any function this way (strong learner)

Agenda for today

- Architecture
- Hidden Layers
- Example: Learning XOR (1/3): using a Neural Network with Tensorflow, Python
- Gradient-Based Learning
- Example: Learning XOR (2/3)
- Back propagation
- Example: Learning XOR (3/3)

Loss function

- Let $m(X)$, $X = x_1, \dots., x_n$ be the composite function of the neural network model, n the arity of the function
- Let $loss(m, X_1, \dots, X_k, Y_1, \dots, Y_k)$ be the function measuring the error of m, for the training data set $X_1, \dots, X_k, Y_1, \dots, Y_k$, k the size of the training data set
- Examples
  - Residual sum of squares, RSS
  - Mean absolute error, MAE
  - Mean squared error, MSE
  - Cross-entropy, H
  - Kullback-Leibler divergence, KL
  - …
- The $loss$ functions are composed functions with the model function m as a component
- They are differentiable too
- Other names: optimization goal or goal or target function

Gradient-Based Learning

- $m(X, \mathbf{ws})$, $\mathbf{ws} = w_1, \dots., w_l, b_1, \dots., b_o$ is the model function parameterized with weights w_i and biases b_j
- The $loss$ function is parameterized too, e.g., aggregated/averaged differences of observation Y_i and prediction $m(X_i, \mathbf{ws})$:
$$loss(m(X, \mathbf{ws}), X_1, \dots, X_k, Y_1, \dots, Y_k)$$
- Learning is an optimization problem:
$$\widehat{\mathbf{ws}} = \arg\min loss(m(X, \mathbf{ws}), X_1, \dots, X_k, Y_1, \dots, Y_k)$$
- DL uses gradient-based optimization, e.g., gradient descent: starting with initial parameters $\mathbf{ws}_0$ iterate over
$$\mathbf{ws}_{iter+1} = \mathbf{ws}_{iter} - \varepsilon \nabla loss(m(X, \mathbf{ws}_{iter}), X_1, \dots, X_k, Y_1, \dots, Y_k)$$

Gradient-Based Learning (cont'd)

- For computing the gradient $\nabla loss(m(X, \mathbf{ws}_{iter}), X_1, \dots, X_k, Y_1, \dots, Y_k)$ we could
  - compute the derivatives of the $loss$ function with respect to the parameters (weights w_i and biases b_j)
  - In each iteration, set in the current (initial) weights and biases $\mathbf{ws}_{iter}$
- As $loss$ is composed from the neural network function $m(X, \mathbf{ws})$, calculus (the chain rule and others) shows how to compute the derivatives of m with respect to the parameters
- As m is composed from the core functions and a few known (differentiable) activation functions, the chain rule requires to compute the derivatives of the core and activation function with respect to the parameters
- Follow the definition of derivatives of composite functions (chain, product, sum, … rules)
- The derivatives are composed functions following the composition of the model function.

Calculus: derivatives of functions

Constant	a	0
Power	x^n	nx^{n-1}
	x^2	$2x$
	$\sqrt{x}$	$(½)x^{-½}$
Exponential	e^x	e^x
	a^x	$\ln(a)\ a^x$
Logarithms	$\ln(x)$	$1/x$
	$\log_a(x)$	$1 / (x \ln(a))$
Trigonometry (x is in rad)	$\sin(x)$	$\cos(x)$
	$\cos(x)$	$-\sin(x)$
	$\tan(x)$	$\sec^2(x)$
Inverse Trigonometry	$\sin^{-1}(x)$	$1/\sqrt{(1-x^2)}$
	$\cos^{-1}(x)$	$-1/\sqrt{(1-x^2)}$
	$\tan^{-1}(x)$	$1/(1+x^2)$

Calculus: derivatives of composite functions

Multiplication by constant	$a\ f$	$a\ f'$
Sum Rule	$f + g$	$f' + g'$
Difference Rule	$f - g$	$f' - g'$
Product Rule	$f\ g$	$f\ g' + f'\ g$
Quotient Rule	f/g	$(f'\ g - g'\ f\ )/g^2$
Reciprocal Rule	$1/f$	$-f'/f^2$
Chain Rule	$f(g(x))$	$f'(g(x))g'(x)$

Agenda for today

- Architecture
- Hidden Layers
- Example: Learning XOR (1/3)
- Gradient-Based Learning
- Example: Learning XOR (2/3): using gradient-based optimization with Matlab
- Back propagation
- Example: Learning XOR (3/3)

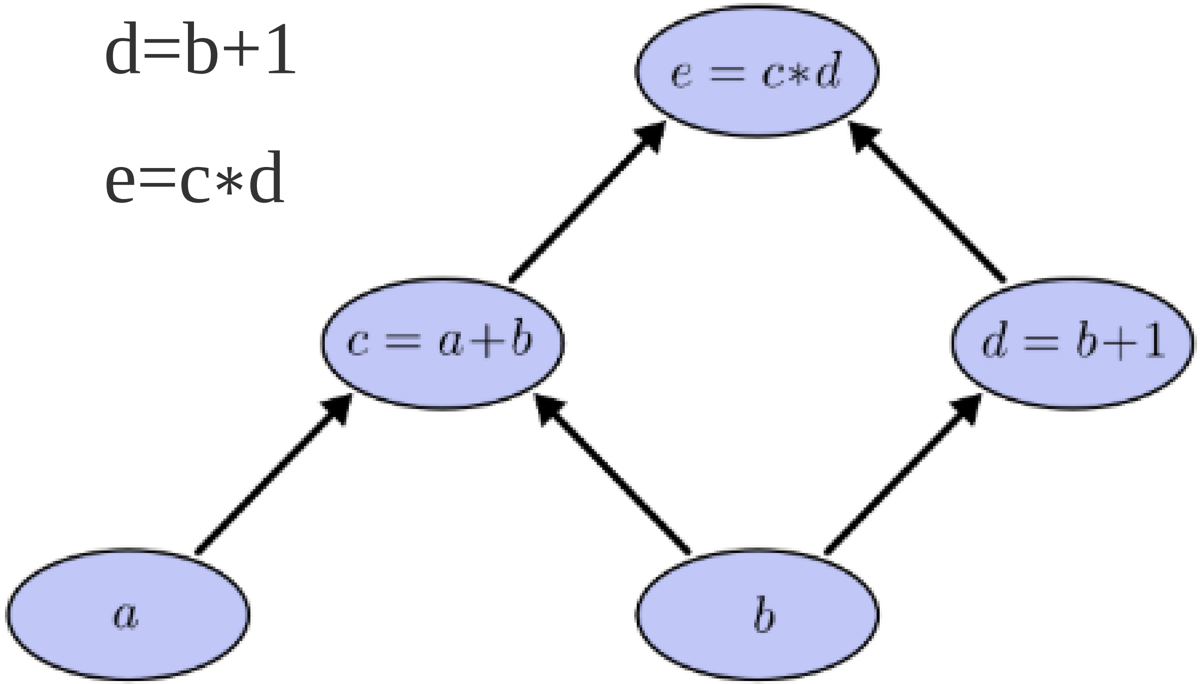
Froward propagation in computational graphs

https://colah.github.io/posts/2015-08-Backprop/

c=a+b

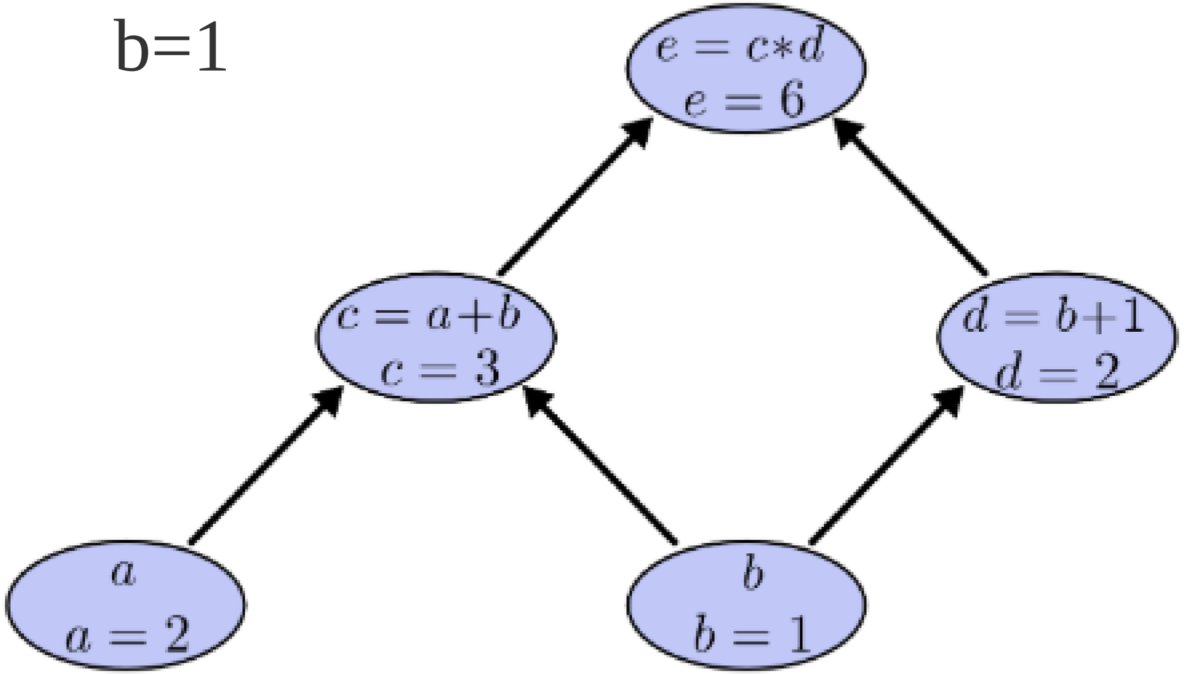
d=b+1

e=c∗d



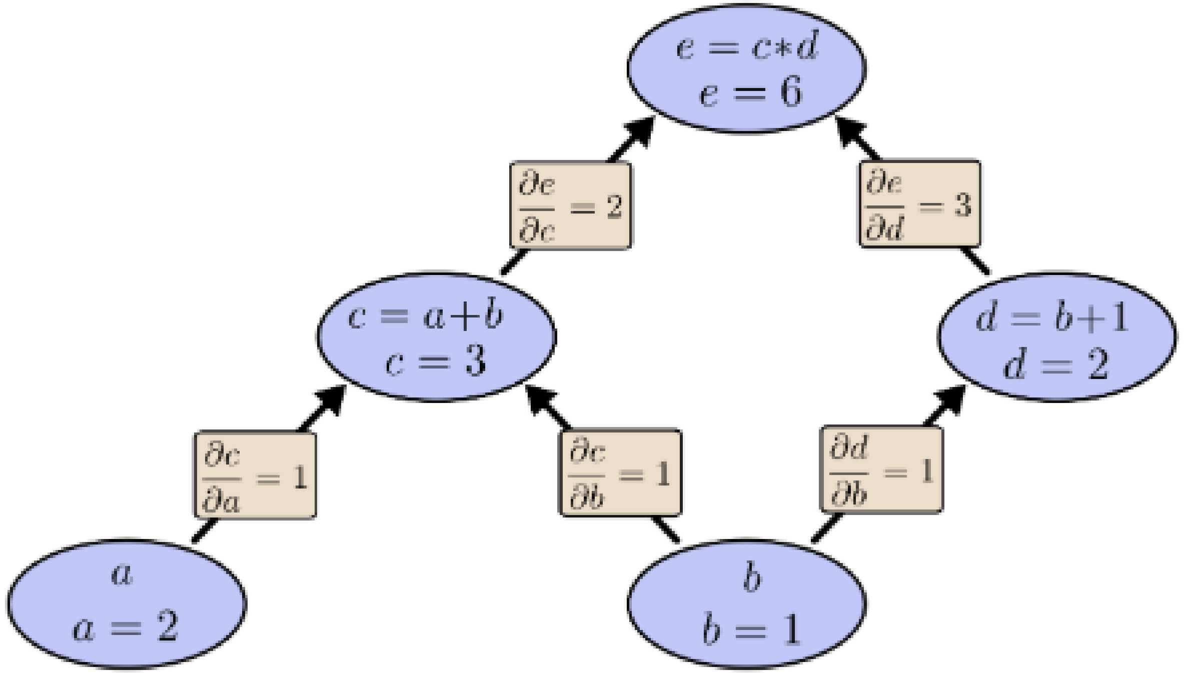
a=2

b=1



Derivatives on Computational Graphs

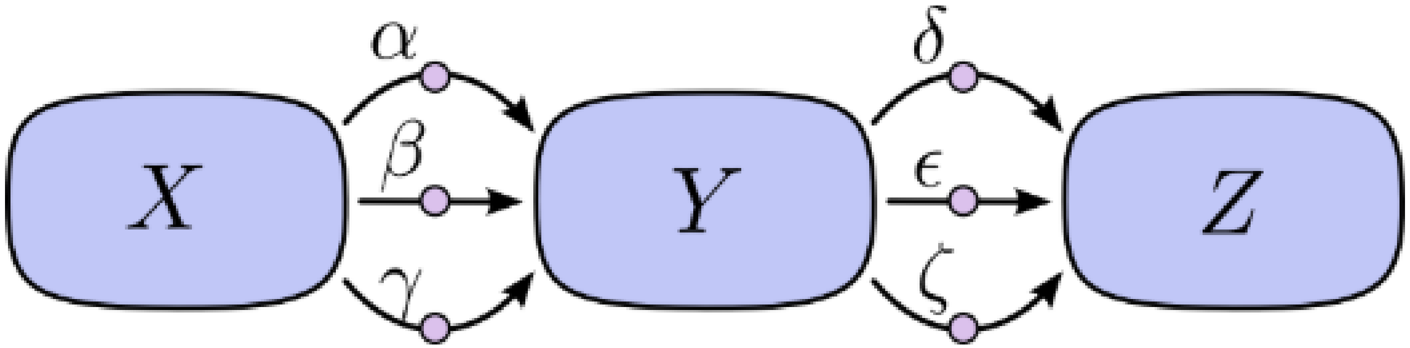
- Changes of a lead to changes of c, quantified as the partial derivative of c with respect to a



- Changing a by a tiny bit, c changes by the same amount causing e to change by a factor 2 with respect to the tiny changes of a.
- General rule: sum over all paths from one node to the other, multiplying the derivatives on each edge of the path together gives the partial derivatives

Problem of exponential growth of all paths

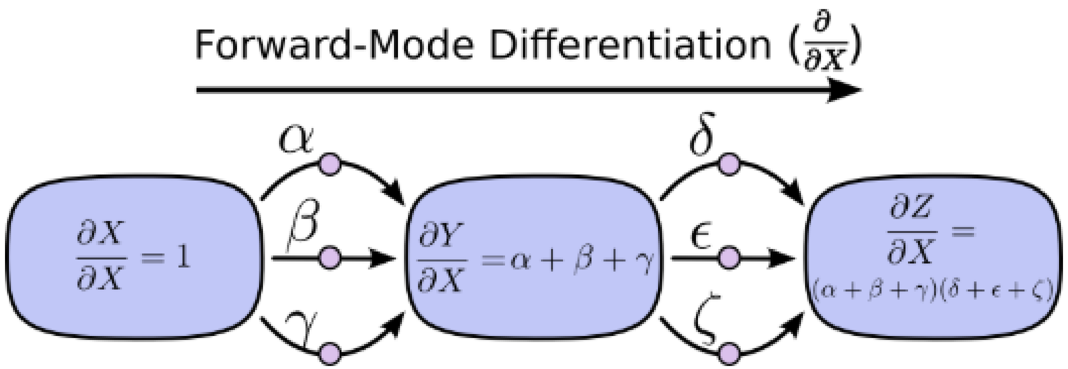
- “sum over all paths” leads to a combinatorial explosion in the number of possible paths.

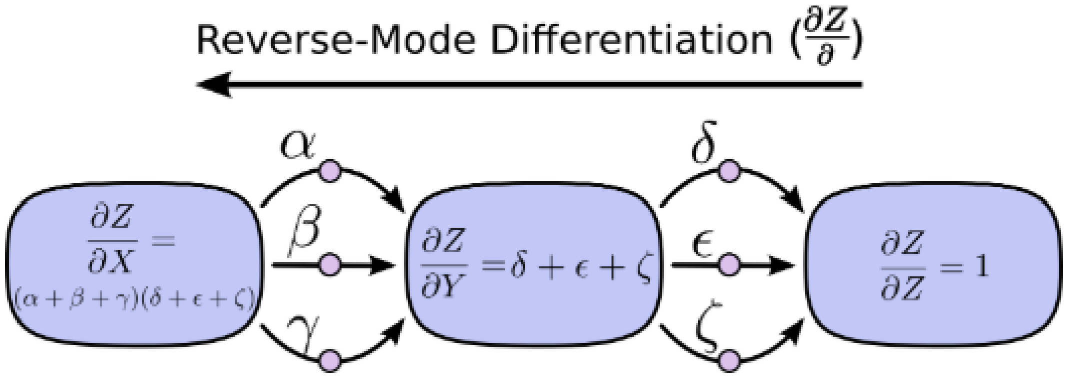


- 3^2 path

$$\frac{\partial Z}{\partial X} = \alpha\delta + \alpha\epsilon + \alpha\zeta + \beta\delta + \beta\epsilon + \beta\zeta + \gamma\delta + \gamma\epsilon + \gamma\zeta$$

Forward and backward factorization of paths

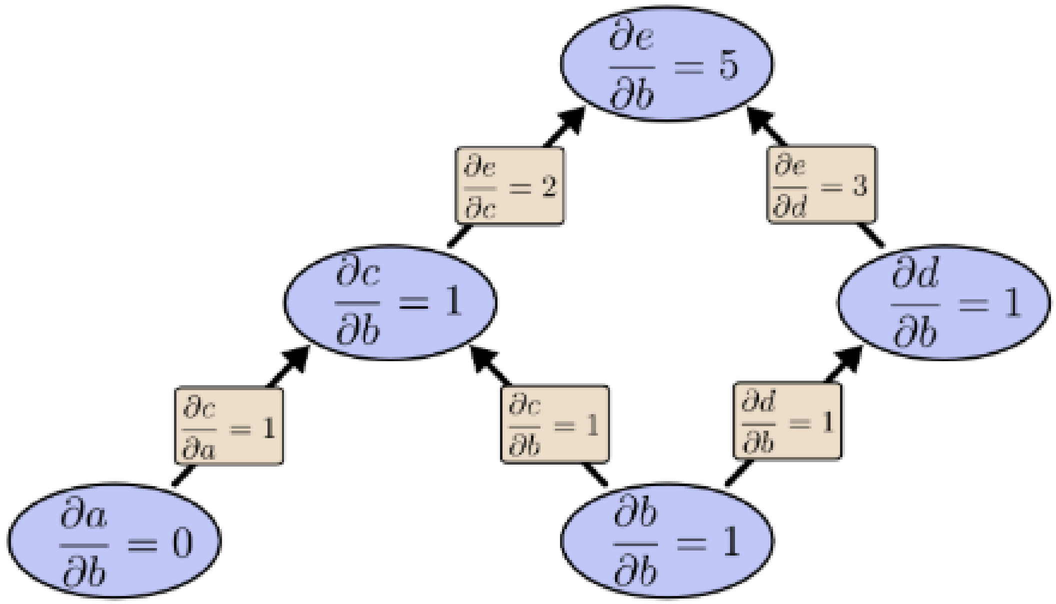




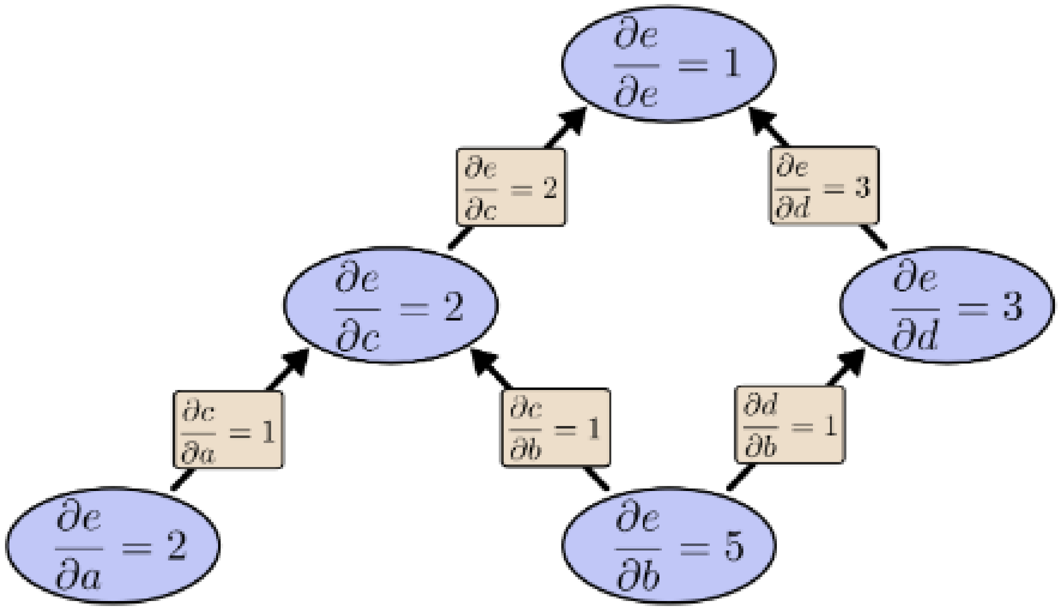
Forward-mode differentiation: how one input affects every node

Reverse-mode differentiation: how every node affects one output.

Forward and backward factorization of paths

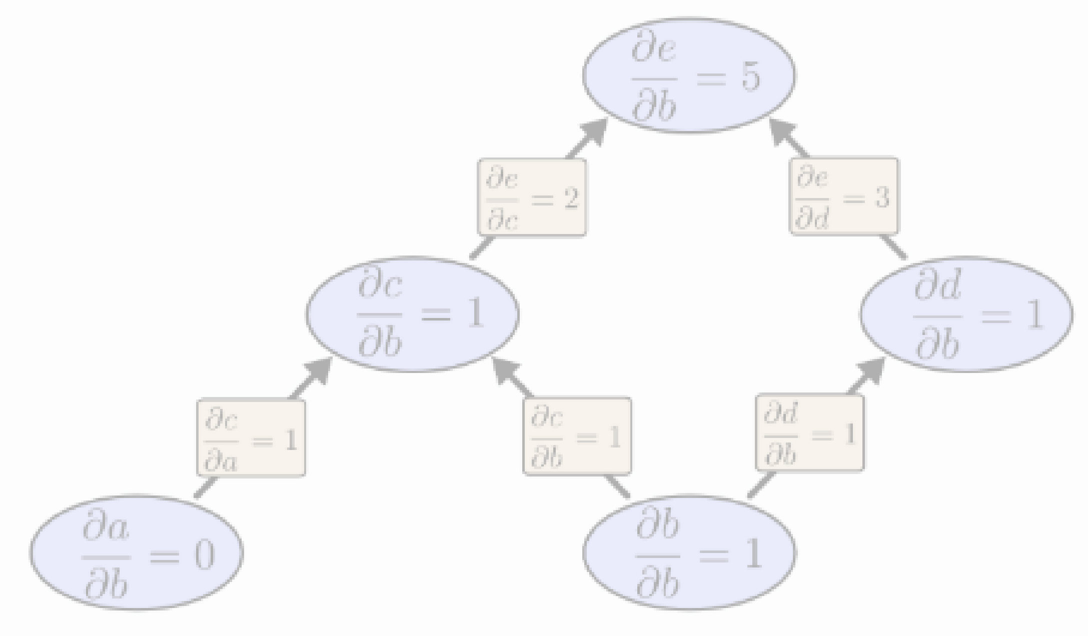


Forward-mode differentiation from b up gives the derivative of every node with respect to b.

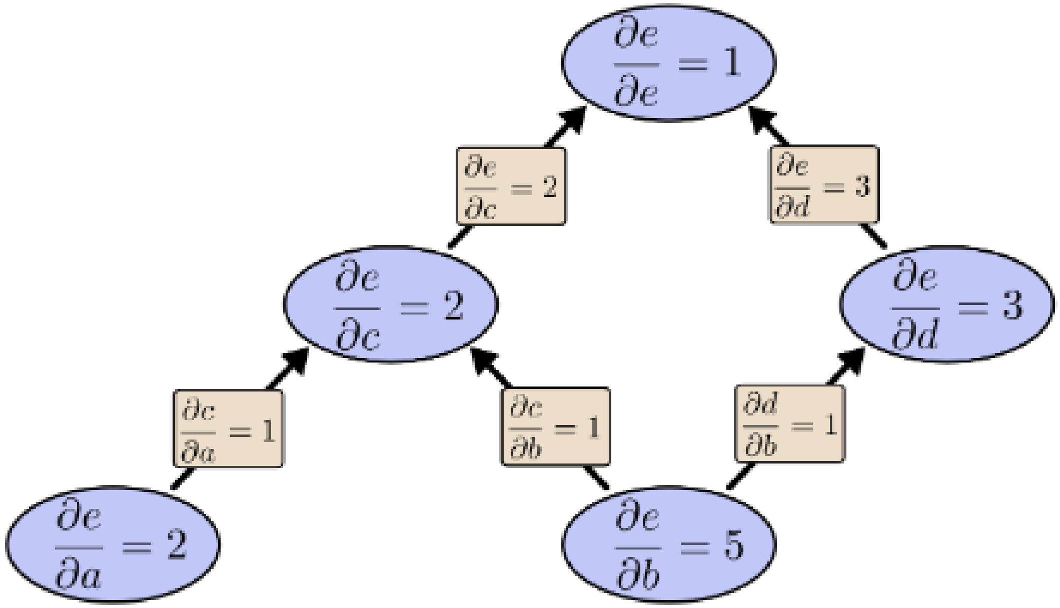


Reverse-mode differentiation from e down gives the derivative of e with respect to every node.

Backward factorization of paths is back propagation



Forward-mode differentiation from b up gives the derivative of every node with respect to b.



Reverse-mode differentiation from e down gives the derivative of e with respect to every node.

Imagine a loss function with a million inputs weights and one output loss.

Agenda for today

- Architecture
- Hidden Layers
- Example: Learning XOR (1/3)
- Gradient-Based Learning
- Example: Learning XOR (2/3)
- Back propagation
- Example: Learning XOR (3/3): using back propagation with Python

Assignment 3

- Chapter 7: Regularization for Deep Learning
  - Read 46 pages
- Create a new Jupyter notebook with text and Python code implementing learning XOR using back propagation
  - Notebook (3/3) that I showed earlier is not uploaded
- Deadline: 2025-04-22 (before the next lecture)