

Backpropagation

CS 4804: Introduction to AI

Fall 2025

<https://tuvllms.github.io/ai-fall-2025/>

Tu Vu



Logistics

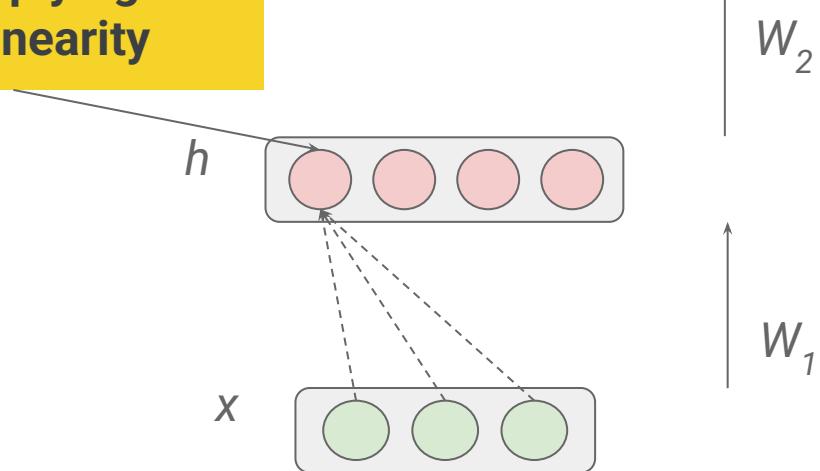
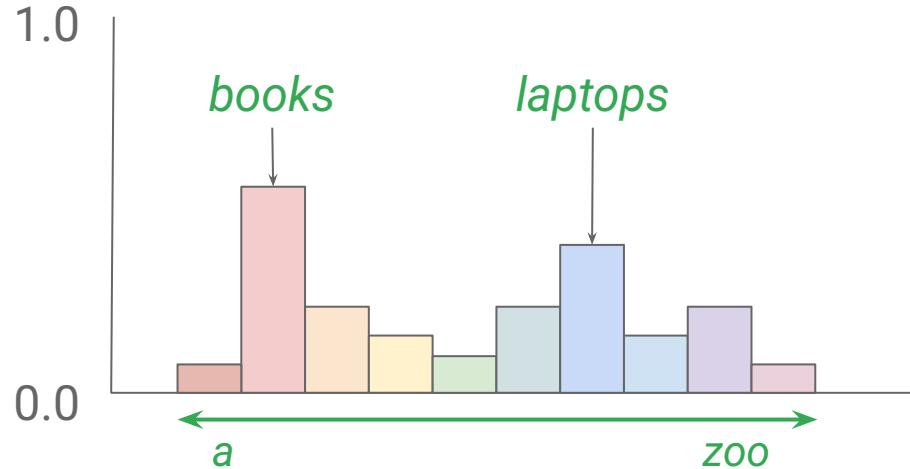
- Google form for submitting group information available on Piazza (**due tomorrow**)
- Quiz 0 will be released on Piazza tomorrow (**due September 12th**)
 - graded for genuine attempt, not correctness

A recap on neural networks

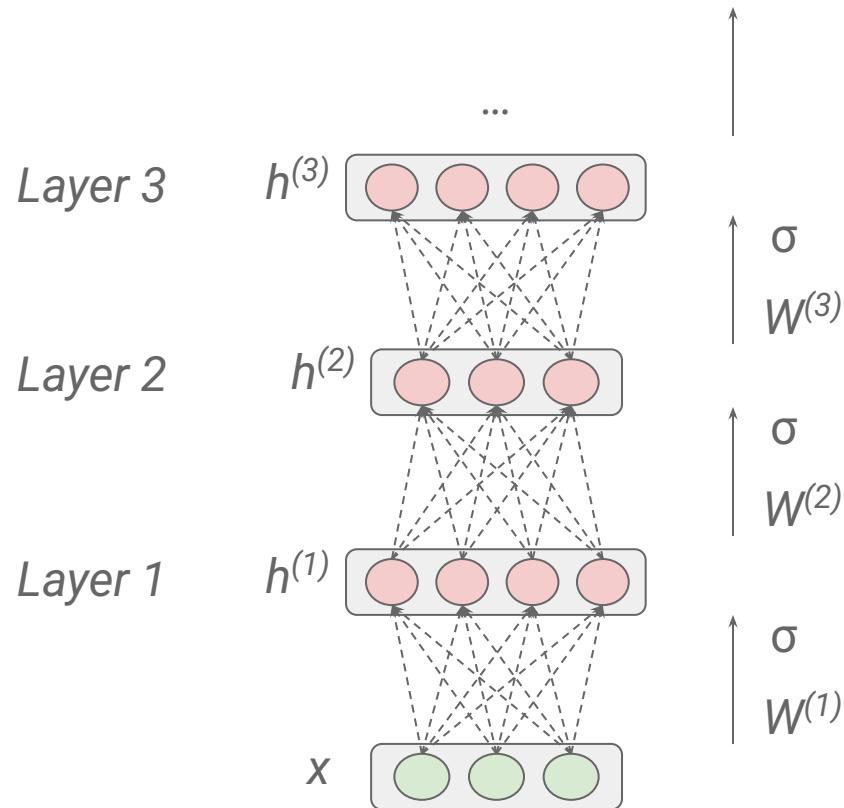
hidden layer

$$h = f(W_1 x)$$

hidden unit (neuron):
taking a weighted
sum of its inputs and
then applying a
non-linearity



Deep neural networks



hierarchical representations, where each layer builds upon the previous one

$$\text{Let } W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \text{ (dimensions } 4 \times 3\text{)} \text{ and } x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \text{ (dimensions } 3 \times 1\text{).}$$

Then, the multiplication yields the output vector h as:

$$h = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + w_{13}x_3 \\ w_{21}x_1 + w_{22}x_2 + w_{23}x_3 \\ w_{31}x_1 + w_{32}x_2 + w_{33}x_3 \\ w_{41}x_1 + w_{42}x_2 + w_{43}x_3 \end{bmatrix} \text{ (dimensions } 4 \times 1\text{).}$$

Let $W = [W_1 \quad W_2 \quad W_3]$, where:

$$W_1 = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \\ w_{41} \end{bmatrix}, \quad W_2 = \begin{bmatrix} w_{12} \\ w_{22} \\ w_{32} \\ w_{42} \end{bmatrix}, \quad W_3 = \begin{bmatrix} w_{13} \\ w_{23} \\ w_{33} \\ w_{43} \end{bmatrix} \quad (\text{dimensions } 4 \times 1)$$

and $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ (dimensions 3×1).

Then, the multiplication yields the output vector h as:

$$h = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} = W_1x_1 + W_2x_2 + W_3x_3 = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + w_{13}x_3 \\ w_{21}x_1 + w_{22}x_2 + w_{23}x_3 \\ w_{31}x_1 + w_{32}x_2 + w_{33}x_3 \\ w_{41}x_1 + w_{42}x_2 + w_{43}x_3 \end{bmatrix}$$

Bias values

$$h = \sigma(Wx + b)$$

bias values

Logits

Logits: the vector of raw scores right before the final softmax

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_V \end{bmatrix}$$

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}, \quad \text{for } i = 1, 2, \dots, V$$

Geoffrey Hinton - “The Godfather of AI”

Played a central role in reviving neural network research after the AI winters of the mid-1970s and late 1980s to early 1990s, when funding and interest in AI declined because earlier methods failed to deliver on their promises.



2018 Turing Award



Yoshua Bengio



Geoffrey Hinton



Yann LeCun

We want AI to be our mothers!



https://x.com/slow_developer/status/1962719631631696299

The partial derivative of the loss function

The partial derivative of the loss function L with respect to the parameter w represents how much the loss changes as w changes.

$$\frac{dL}{dw}$$

The gradient

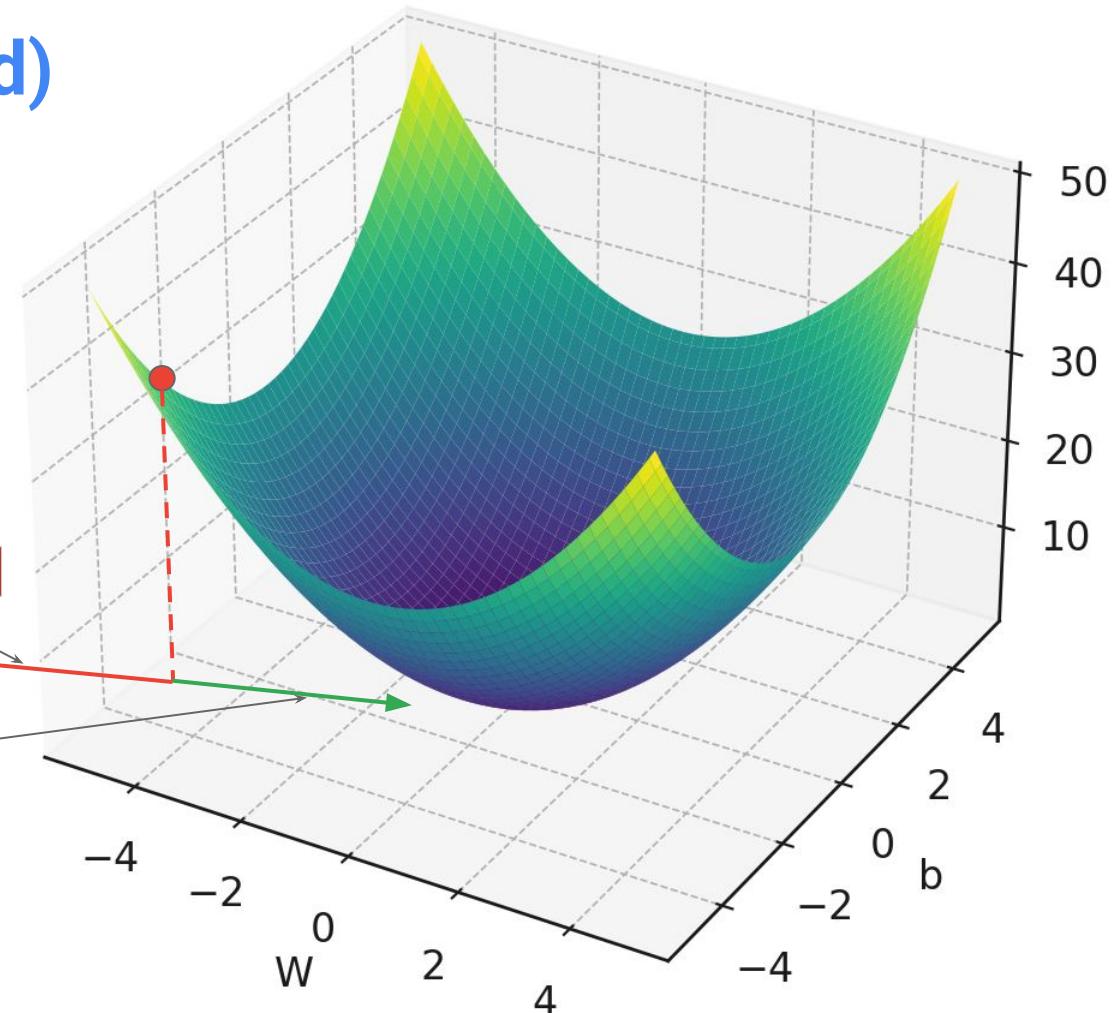
A **derivative** applies to a function with a single variable. It measures the rate of change of the function with respect to that variable. For example, if $f(x) = x^2$, then the derivative $f'(x) = 2x$ shows how fast $f(x)$ changes as x changes.

A **gradient** applies to a function with multiple variables. It is a vector that contains all of the partial derivatives of the function with respect to each variable. For example, if $f(x, y) = x^2 + y^2$, then the gradient is

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) = (2x, 2y).$$

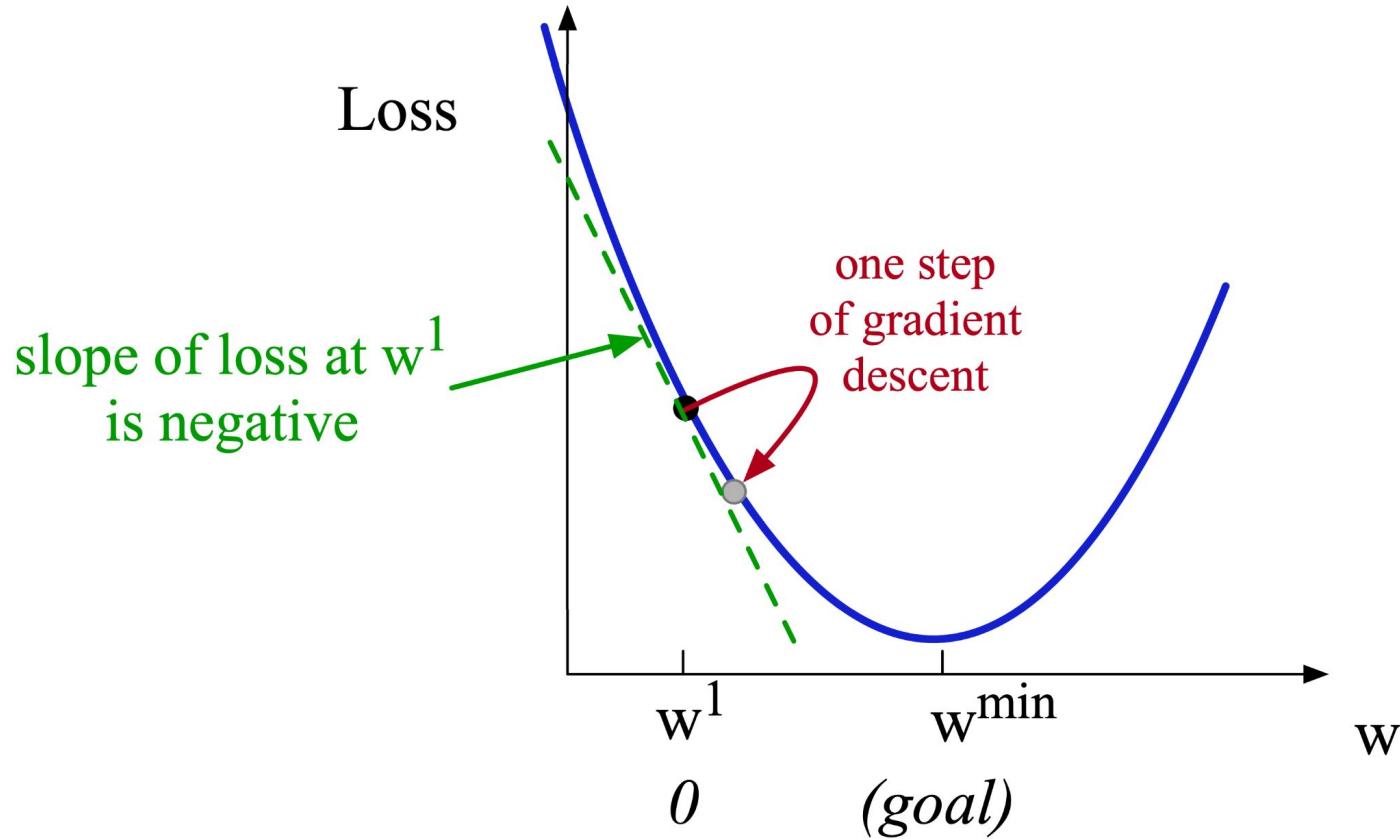
The gradient (cont'd)

the gradient points in the direction of the steepest increase in the loss



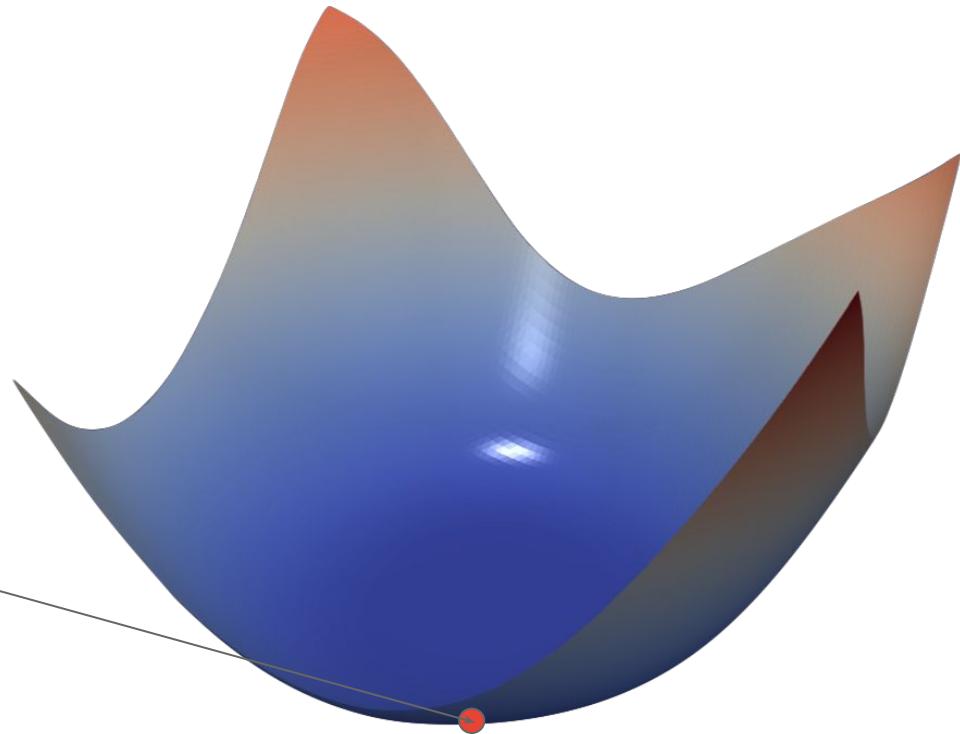
negative gradient

Gradient descent

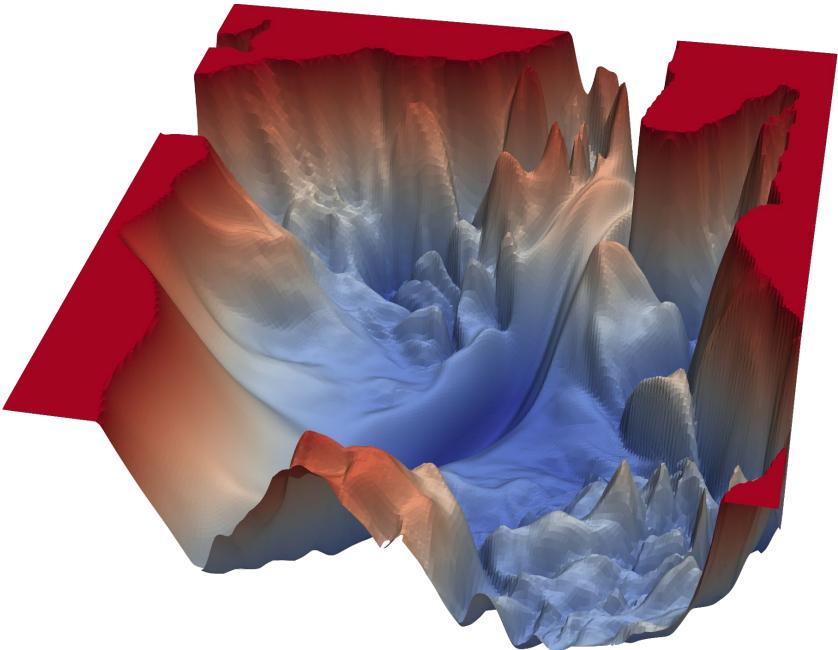
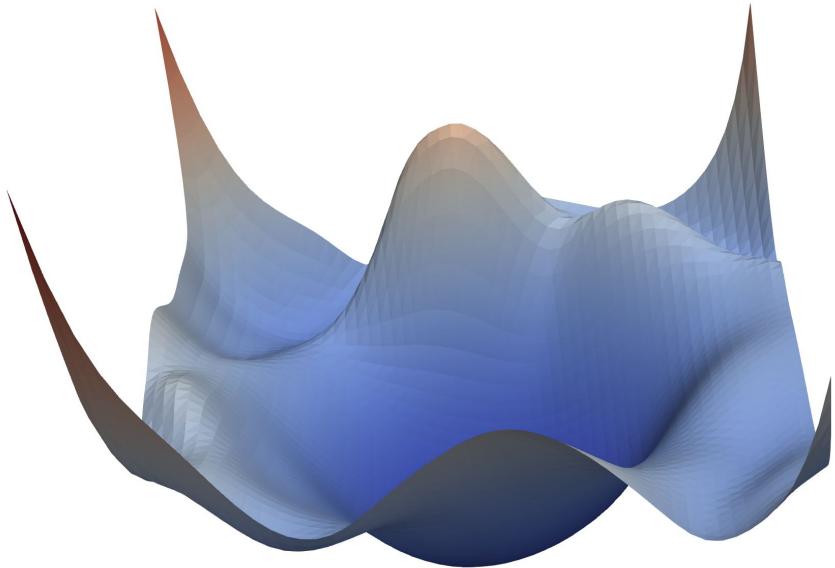


The loss landscape

A convex function has at most one minimum; there are no local minima to get stuck in.



The loss landscape of neural nets (cont'd)



The loss for multi-layer neural networks is non-convex, and gradient descent may get stuck in local minima and never find the global optimum

<https://www.cs.umd.edu/~tomg/project/landscapes/>

Updating parameters

$$w_{t+1} = w_t - \eta \cdot \frac{\partial L}{\partial w_t}$$

Updating parameters (cont'd)

$$w_{t+1} = w_t - \eta \cdot \frac{\partial L}{\partial w_t}$$

Where:

- w_t is the parameter at the current time step.
- w_{t+1} is the updated parameter after applying the gradient.
- η is the learning rate, which controls the step size.
- $\frac{\partial L}{\partial w_t}$ is the gradient of the loss function L with respect to the parameter w_t , representing how the loss changes as the parameter changes.

Hyperparameters of gradient descent

- Learning rate
- Batch size

Optimizer

- SGD (Stochastic gradient descent)
- Adam (Adaptive moment estimation)
- Muon



 **Dzmitry Bahdanau**
@DBahdanau



Adam deserves the award, but in Singapore everyone still uses SGD

9:32 PM · Apr 27, 2025 · 106.1K Views

Backpropagation

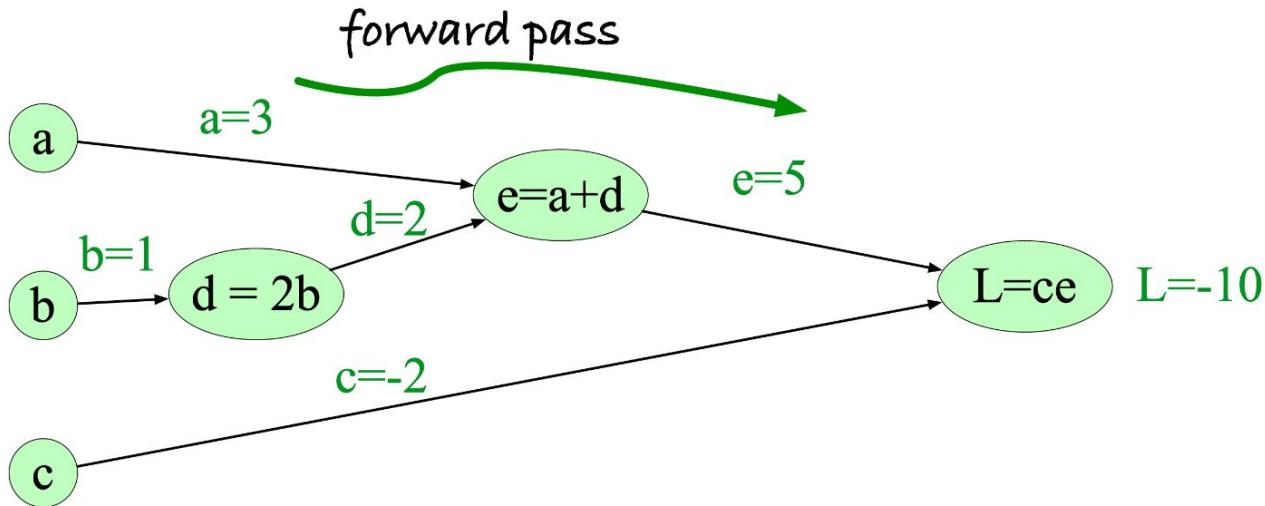


Figure 7.12 Computation graph for the function $L(a, b, c) = c(a + 2b)$, with values for input nodes $a = 3, b = 1, c = -2$, showing the forward pass computation of L .

Backpropagation (cont'd)

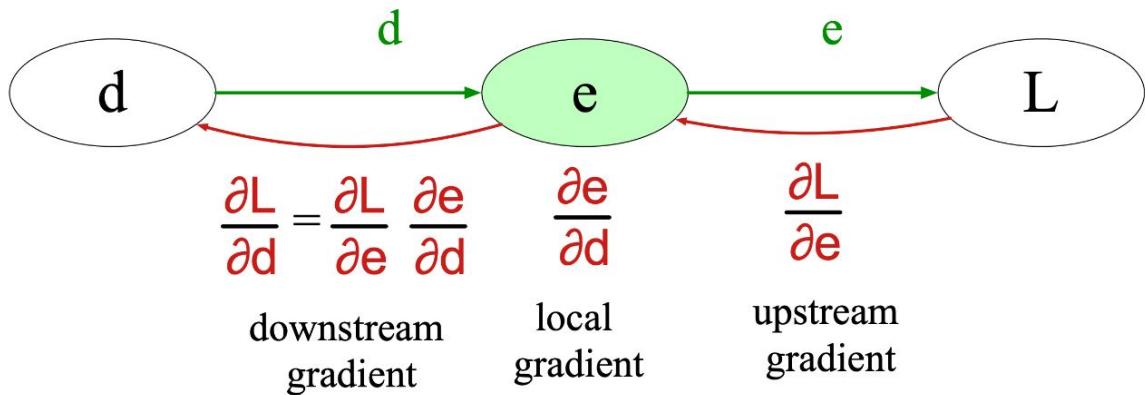


Figure 7.13 Each node (like e here) takes an upstream gradient, multiplies it by the local gradient (the gradient of its output with respect to its input), and uses the chain rule to compute a downstream gradient to be passed on to a prior node. A node may have multiple local gradients if it has multiple inputs.

Backpropagation (cont'd)

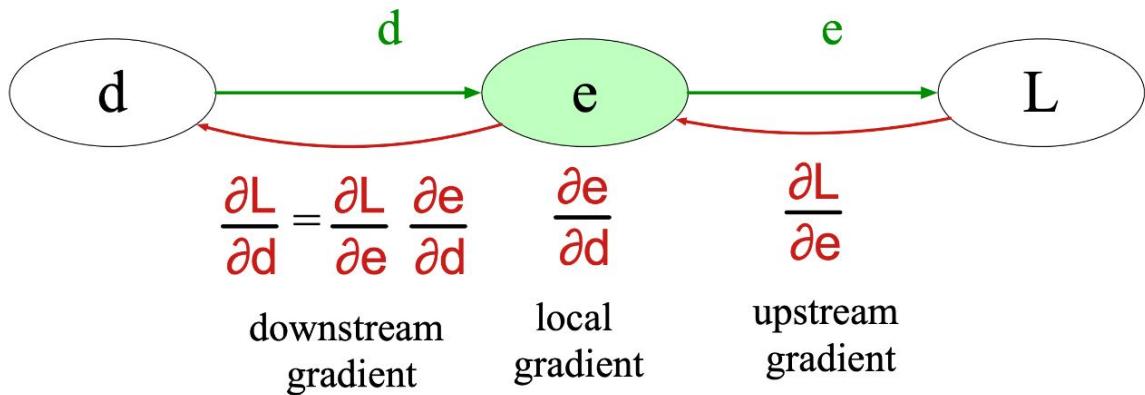


Figure 7.13 Each node (like e here) takes an upstream gradient, multiplies it by the local gradient (the gradient of its output with respect to its input), and uses the chain rule to compute a downstream gradient to be passed on to a prior node. A node may have multiple local gradients if it has multiple inputs.

Cross-entropy loss

The predicted probabilities

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_V \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_V \end{bmatrix}$$

The ground truth label

$$y_i = \begin{cases} 1, & \text{if } i = c \text{ (correct class index)} \\ 0, & \text{otherwise} \end{cases} \quad \hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}, \quad \text{for } i = 1, 2, \dots, V$$

Cross-entropy loss (cont'd)

$$L_{CE}(\hat{y}, y) = - \sum_{i=1}^V y_i \log \hat{y}_i$$

$$L_{CE}(\hat{y}, y) = - (y_1 \log \hat{y}_1 + y_2 \log \hat{y}_2 + \cdots + y_V \log \hat{y}_V)$$

Since the true label y is one-hot encoded, only one term in the sum is nonzero, corresponding to the correct class c , where $y_c = 1$ and $y_i = 0$ for all $i \neq c$. This simplifies the sum to:

$$L_{CE}(\hat{y}, y) = -y_c \log \hat{y}_c$$

Since $y_c = 1$, this further reduces to:

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_c$$

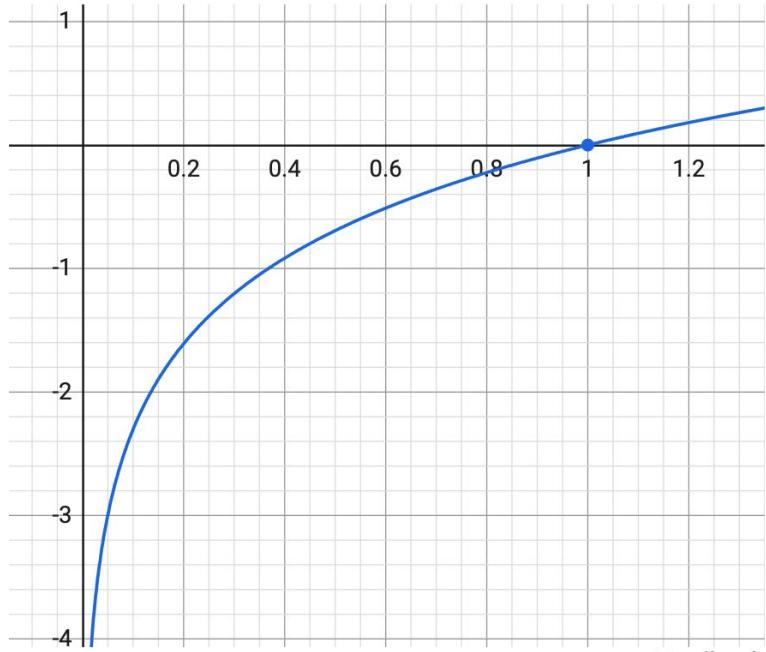
The loss models the distance between the system output and the gold output
—lower is better

Cross-entropy loss (cont'd)

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_c$$

$$\hat{y}_c \rightarrow 0, \log \hat{y}_c \rightarrow -\infty$$

- If $\hat{y}_c = 0.9$, then $\log(0.9) \approx -0.105$, and the loss will be small.
- If $\hat{y}_c = 0.1$, then $\log(0.1) \approx -2.302$, and the loss will be much larger.



Thank you!