Backpropagation

CS 5624: Natural Language Processing

Spring 2025

https://tuvllms.github.io/nlp-spring-2025

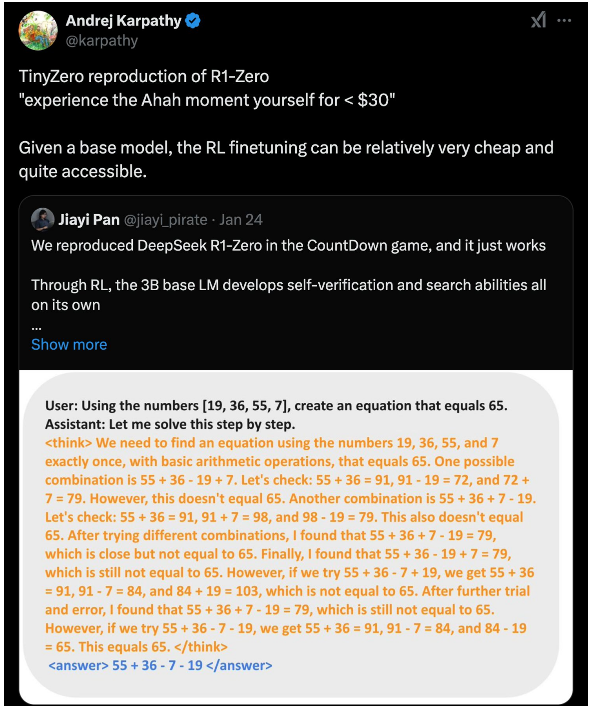
Tu Vu

VIRGINIA TECH.

Logistics

- 🚨 Final project group information due tomorrow 🚨
- Quiz0 will be released on Piazza tomorrow (due February 7th)
- HW0 was released on Piazza (due February 7th)

Efforts to replicate DeepSeek-R1



Andrej Karpathy @karpathy

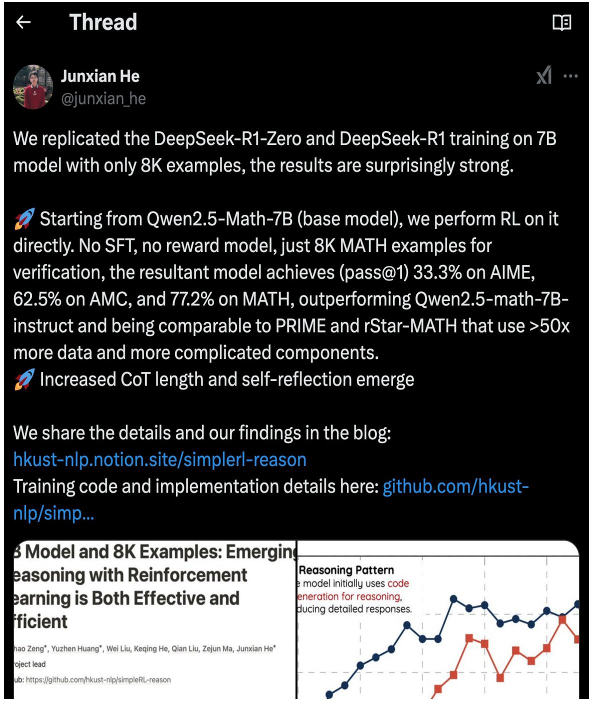
TinyZero reproduction of R1-Zero
"experience the Ahah moment yourself for < $30"

Given a base model, the RL finetuning can be relatively very cheap and quite accessible.

Jiayi Pan @jiayi_pirate · Jan 24
We reproduced DeepSeek R1-Zero in the CountDown game, and it just works

Through RL, the 3B base LM develops self-verification and search abilities all on its own
...
Show more

User: Using the numbers [19, 36, 55, 7], create an equation that equals 65.
Assistant: Let me solve this step by step.
<think> We need to find an equation using the numbers 19, 36, 55, and 7 exactly once, with basic arithmetic operations, that equals 65. One possible combination is 55 + 36 - 19 + 7. Let's check: 55 + 36 = 91, 91 - 19 = 72, and 72 + 7 = 79. However, this doesn't equal 65. Another combination is 55 + 36 + 7 - 19. Let's check: 55 + 36 = 91, 91 + 7 = 98, and 98 - 19 = 79. This also doesn't equal 65. After trying different combinations, I found that 55 + 36 + 7 - 19 = 79, which is close but not equal to 65. Finally, I found that 55 + 36 - 19 + 7 = 79, which is still not equal to 65. However, if we try 55 + 36 - 7 + 19, we get 55 + 36 = 91, 91 - 7 = 84, and 84 + 19 = 103, which is not equal to 65. After further trial and error, I found that 55 + 36 + 7 - 19 = 79, which is still not equal to 65. However, if we try 55 + 36 - 7 - 19, we get 55 + 36 = 91, 91 - 7 = 84, and 84 - 19 = 65. This equals 65. </think>
<answer> 55 + 36 - 7 - 19 </answer>



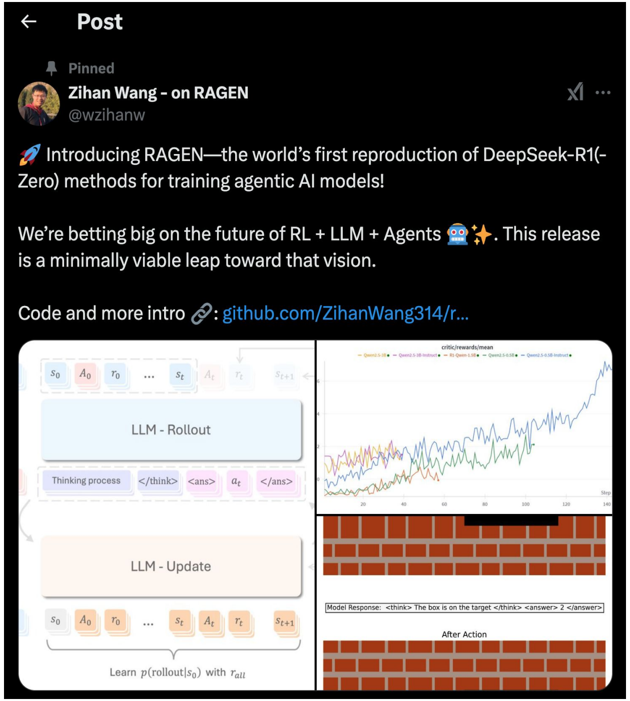
Thread

Junxian He @junxian_he

We replicated the DeepSeek-R1-Zero and DeepSeek-R1 training on 7B model with only 8K examples, the results are surprisingly strong.

🚀 Starting from Qwen2.5-Math-7B (base model), we perform RL on it directly. No SFT, no reward model, just 8K MATH examples for verification, the resultant model achieves (pass@1) 33.3% on AIME, 62.5% on AMC, and 77.2% on MATH, outperforming Qwen2.5-math-7B-instruct and being comparable to PRIME and rStar-MATH that use >50x more data and more complicated components.
🚀 Increased CoT length and self-reflection emerge

We share the details and our findings in the blog: hkust-nlp.notion.site/simplerl-reason
Training code and implementation details here: github.com/hkust-nlp/simp...



Post

Pinned
Zihan Wang - on RAGEN @wzihanw

🚀 Introducing RAGEN—the world's first reproduction of DeepSeek-R1(-Zero) methods for training agentic AI models!

We're betting big on the future of RL + LLM + Agents 🤖✨. This release is a minimally viable leap toward that vision.

Code and more intro 🔗: github.com/ZihanWang314/r...

CountDown game
https://x.com/jiayi_pirate/status/1882839370505621655

Math
https://x.com/junxian_he/status/1883183099787571519

LLM Agents
https://x.com/wzihanw/status/1884092805598826609

SFT vs. RL

SFT Memorizes, RL Generalizes: A Comparative Study of Foundation Model Post-training

Tianzhe Chu ♠ * Yuexiang Zhai ♥ ♣ * Jihan Yang ♦ Shengbang Tong ♦
Saining Xie ♣ ♦ Dale Schuurmans ♣ Quoc V. Le ♣ Sergey Levine ♥ Yi Ma ♠ ♥

AI] 28 Jan 2025

Abstract

Supervised fine-tuning (SFT) and reinforcement learning (RL) are widely used post-training techniques for foundation models. However, their respective role in enhancing model generalization remains unclear. This paper studies the comparative effect of SFT and RL on generalization and memorization, focusing on text-based and visual environments. We introduce `GeneralPoints`, an arithmetic reasoning card game, and also consider `V-IRL`, a real-world navigation environment, to assess how models trained with SFT

generalization (Bousquet & Elisseeff, 2000; Zhang et al., 2021) remain unclear, which makes it challenging to build reliable and robust AI systems. A key challenge in analyzing the generalization ability of foundation models (Bommasani et al., 2021; Brown et al., 2020) is separating data memorization[1] from the acquisition of transferable principles. We therefore investigate the key question of whether SFT or RL primarily memorize the training data (Allen-Zhu & Li, 2023a; Ye et al., 2024; Kang et al., 2024), or whether they learn generalizable principles that can adapt to novel task variants.

To address this question, we focus on two aspects of gener-

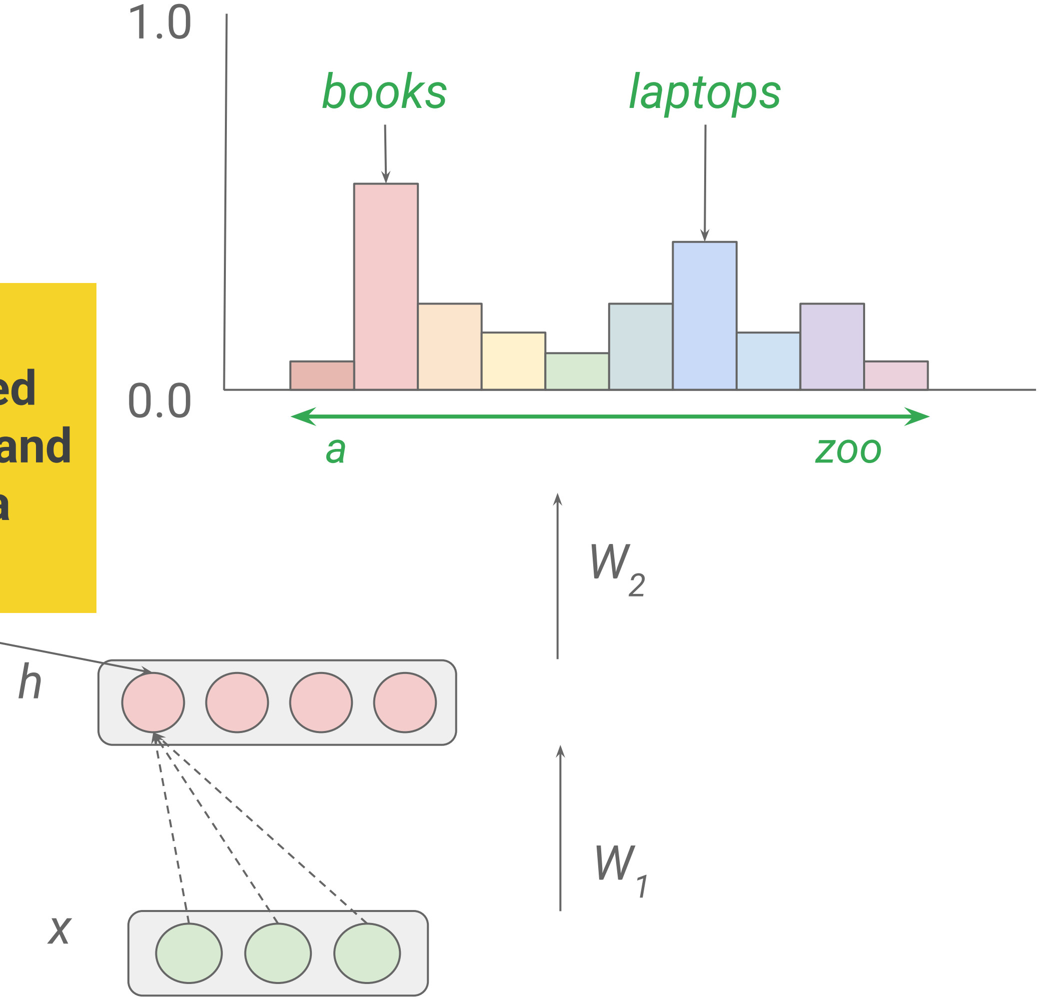
https://arxiv.org/pdf/2501.17161v1

A recap on neural language models

hidden layer

$h = f(W_1 x)$

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



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

Then, the multiplication yields the output vector h as:

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

Let $W = \begin{bmatrix} W_1 & W_2 & W_3 \end{bmatrix}$, where:

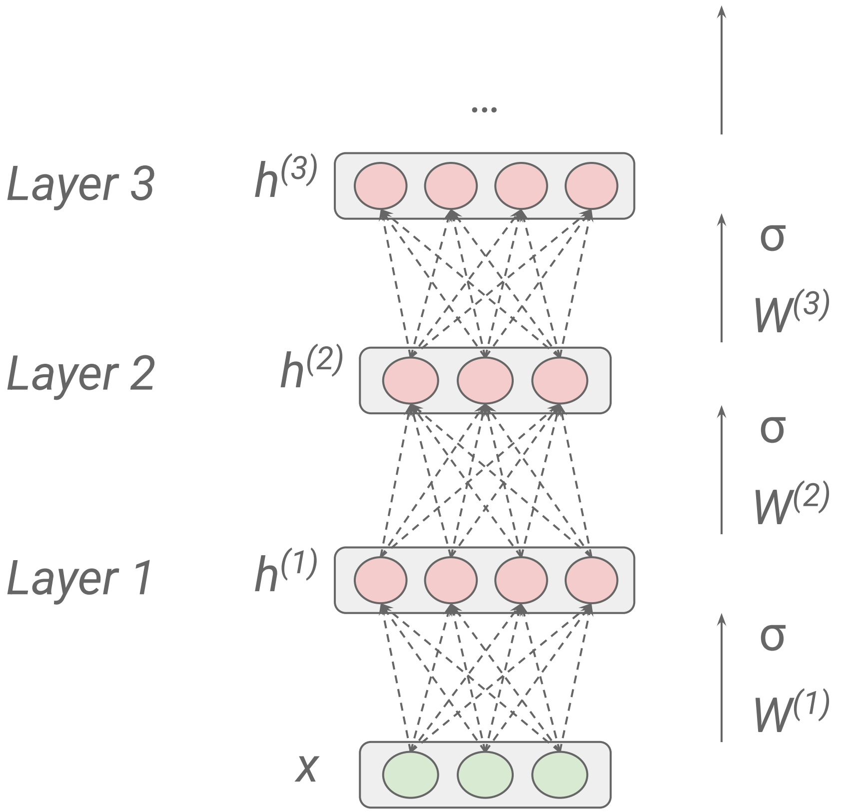
$$W_1 = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \\ w_{41} \end{bmatrix}, \quad W_2 = \begin{bmatrix} w_{12} \\ w_{22} \\ w_{32} \\ w_{42} \end{bmatrix}, \quad W_3 = \begin{bmatrix} w_{13} \\ w_{23} \\ w_{33} \\ w_{43} \end{bmatrix} \quad \text{(dimensions } 4 \times 1)$$

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

Then, the multiplication yields the output vector h as:

$$h = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} = W_1 x_1 + W_2 x_2 + W_3 x_3 = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + w_{13}x_3 \\ w_{21}x_1 + w_{22}x_2 + w_{23}x_3 \\ w_{31}x_1 + w_{32}x_2 + w_{33}x_3 \\ w_{41}x_1 + w_{42}x_2 + w_{43}x_3 \end{bmatrix}$$

Deep neural networks



hierarchical representations, where each layer builds upon the previous one

Bias values

$$h = \sigma(\mathbf{W}x + b)$$

bias values

Logits

Logits: the vector of scores right before the final softmax

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

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

The partial derivative of the loss function

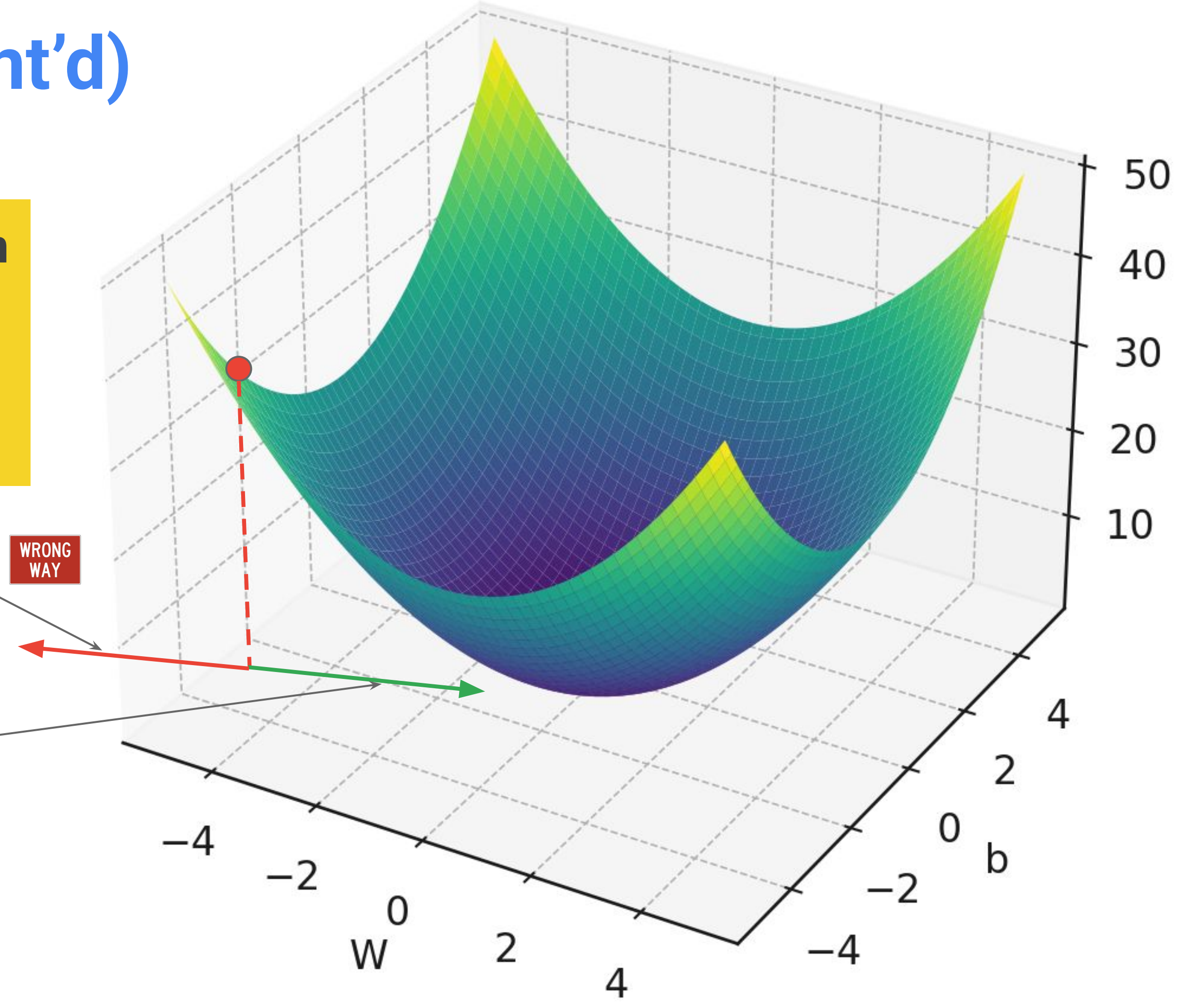
The partial derivative of the loss function L L with respect to the parameter w represents how much the loss changes as w w changes.

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

The gradient (cont'd)

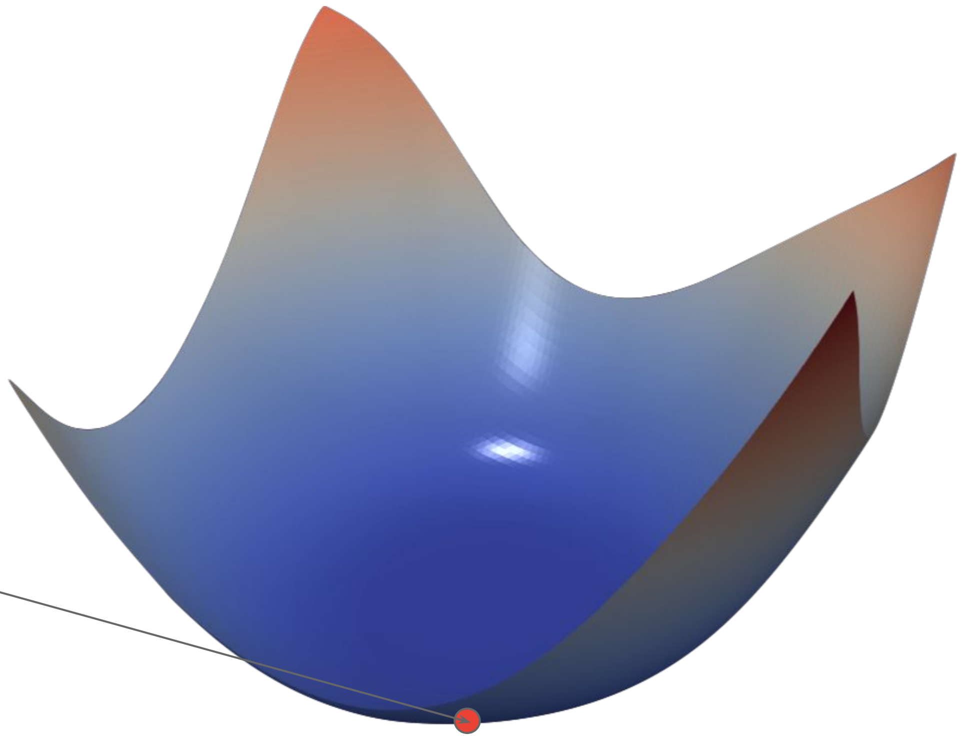
the gradient points in the direction of the steepest increase in the loss

negative gradient



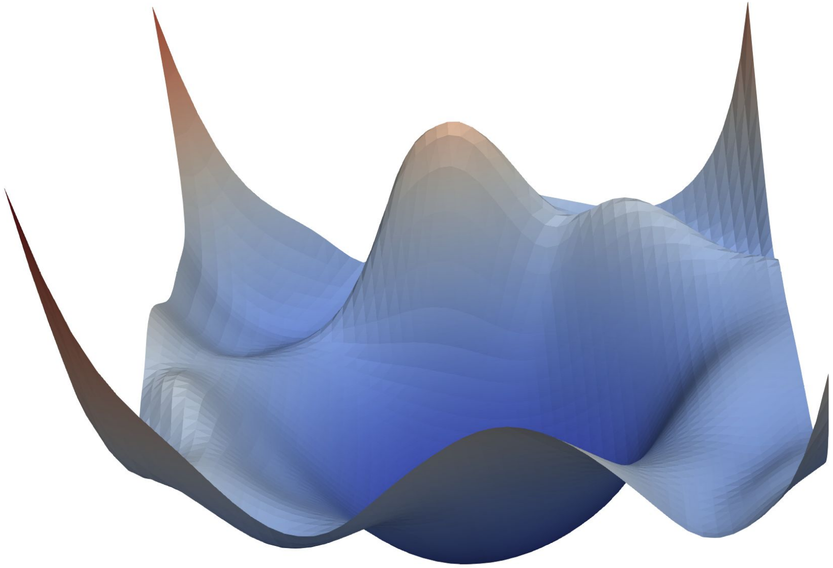
The loss landscape of neural nets

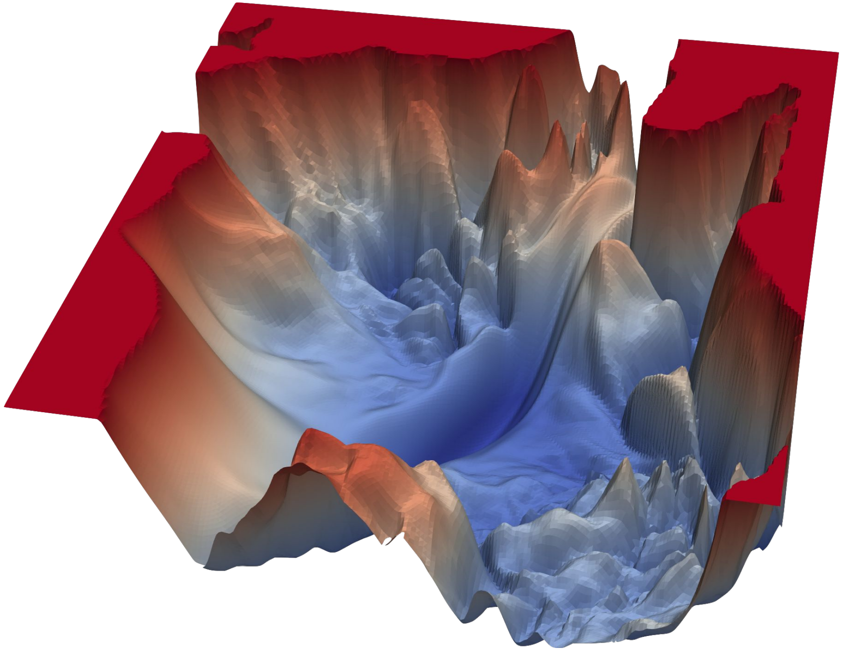
A convex function has at most one minimum; there are no local minima to get stuck in.



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

The loss landscape of neural nets (cont’d)

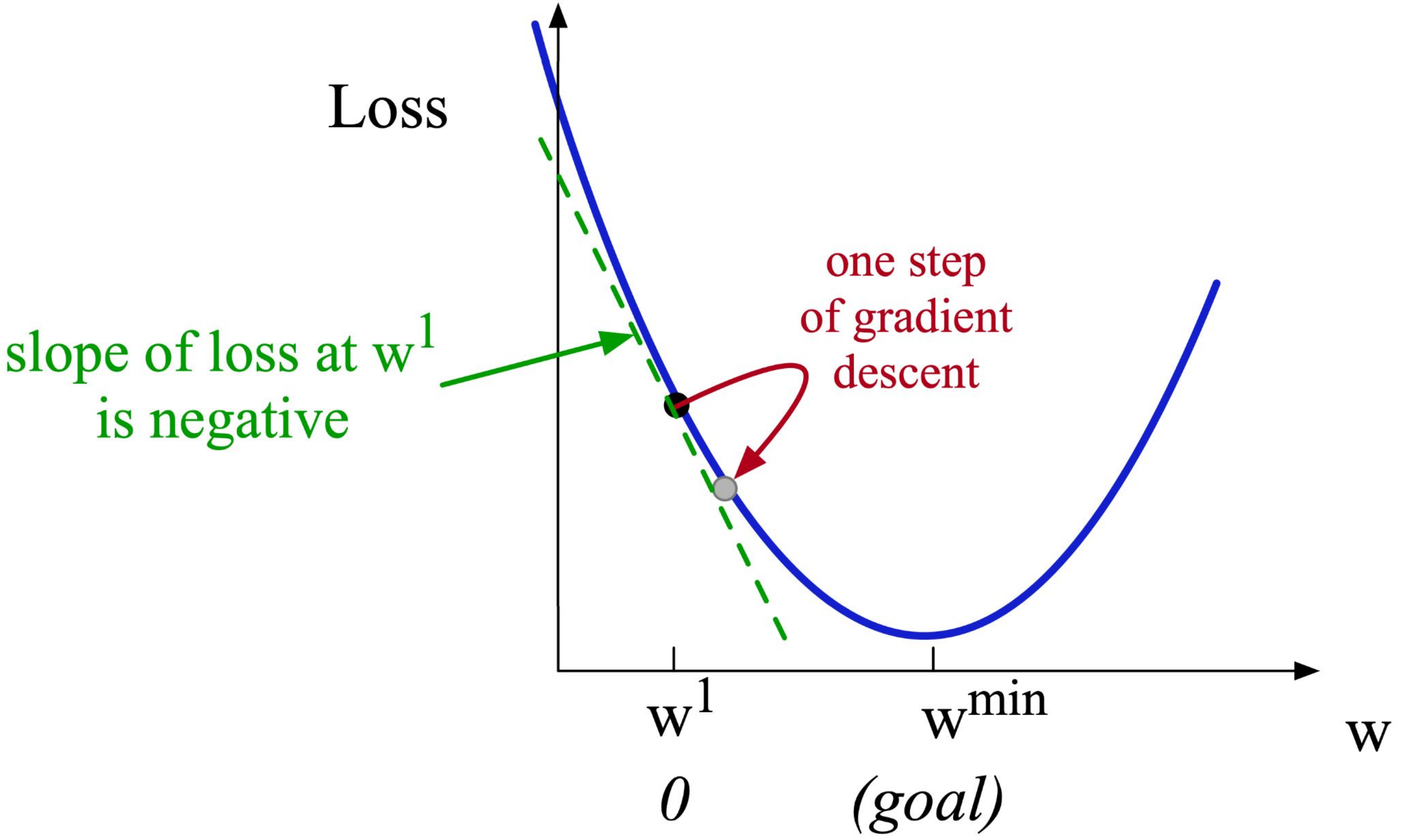




The loss for multi-layer neural networks is non-convex, and gradient descent may get stuck in local minima and never find the global optimum

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

Gradient descent



Gradient descent (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.

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

$$y_i = \begin{cases} 1, & \text{if } i = c \text{ (correct class index)} \\ 0, & \text{otherwise} \end{cases}$$

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

Cross-entropy loss (cont'd)

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

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

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:

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

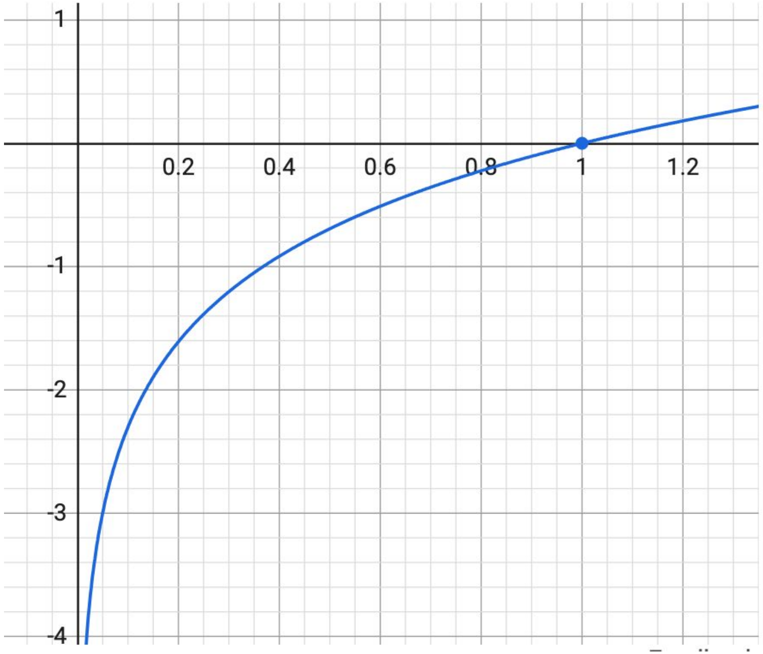
Since $y_c = 1$, this further reduces to:

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

Cross-entropy loss (cont'd)

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

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



- 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.

Backpropagation

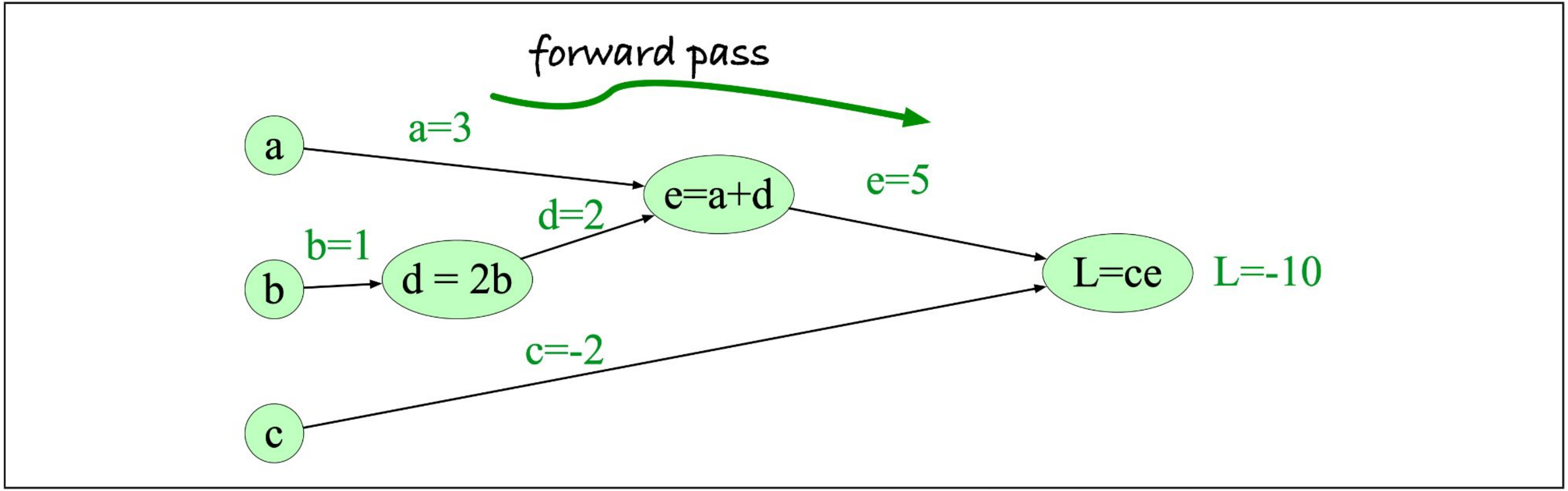


Figure 7.12 Computation graph for the function $L(a,b,c) = c(a+2b)$, with values for input nodes $a = 3$, $b = 1$, $c = -2$, showing the forward pass computation of L.

Backpropagation (cont'd)

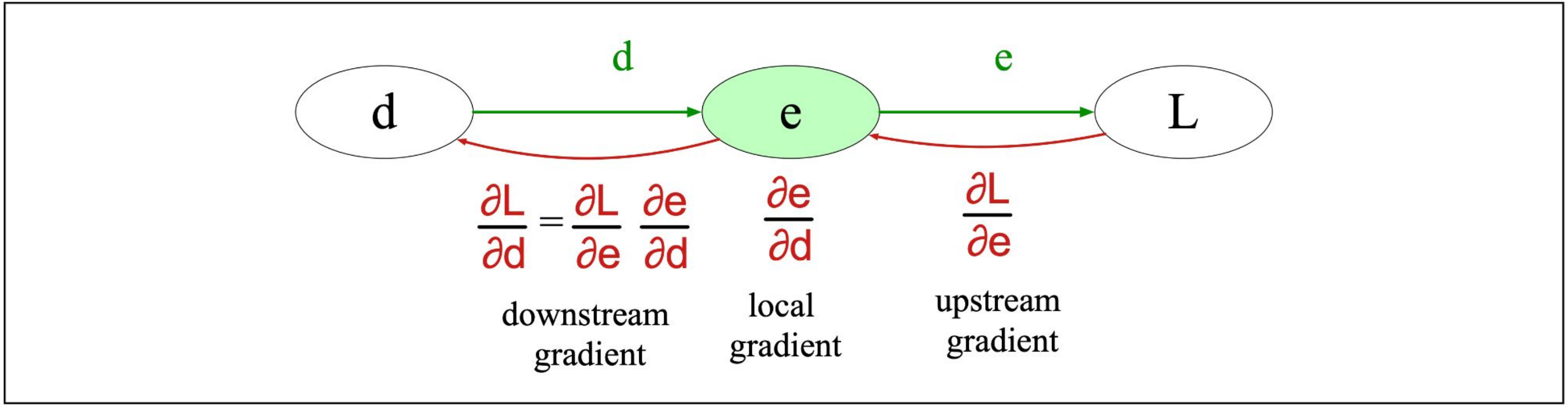


Figure 7.13 Each node (like *e* here) takes an upstream gradient, multiplies it by the local gradient (the gradient of its output with respect to its input), and uses the chain rule to compute a downstream gradient to be passed on to a prior node. A node may have multiple local gradients if it has multiple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

Backpropagation (cont'd)

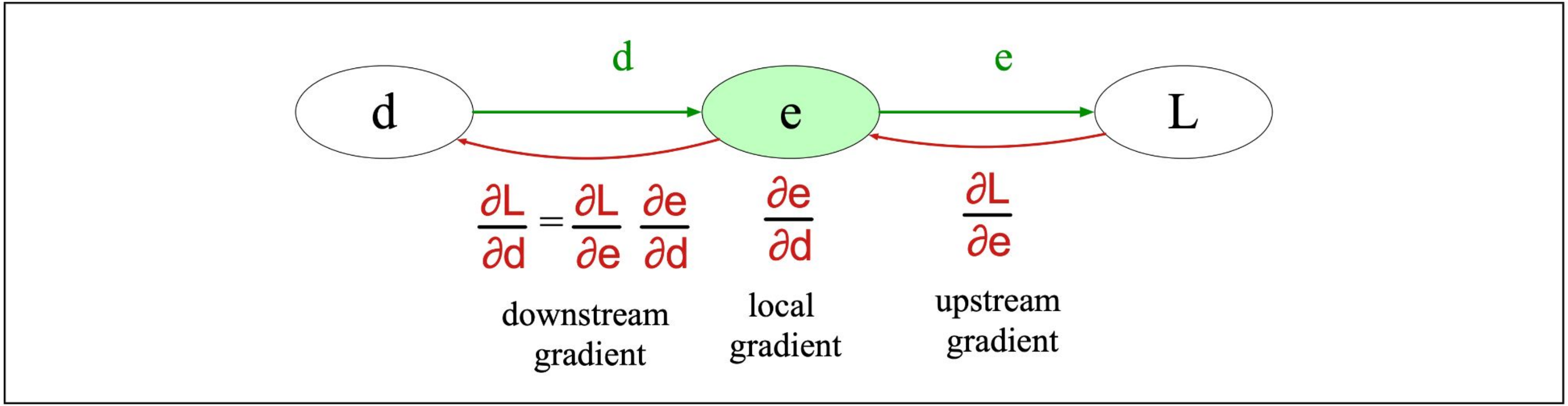


Figure 7.13 Each node (like *e* here) takes an upstream gradient, multiplies it by the local gradient (the gradient of its output with respect to its input), and uses the chain rule to compute a downstream gradient to be passed on to a prior node. A node may have multiple local gradients if it has multiple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

Updating parameters

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

Thank you!