CS 668 Applied Large Language Models
Spring Semester, 2026
Doc 10 NN Review 2
Feb 12, 2026

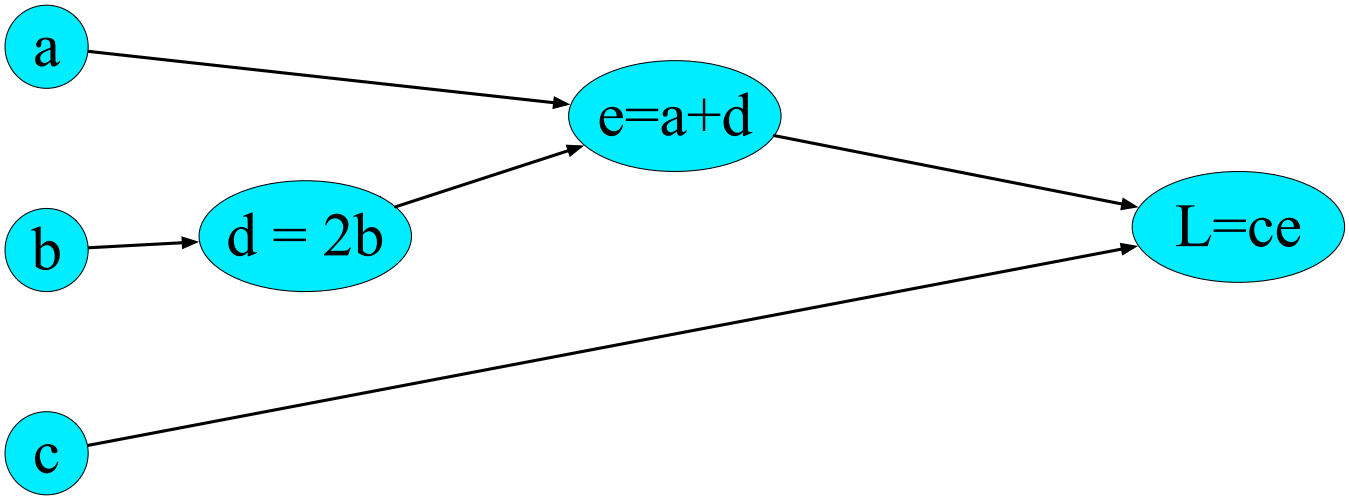
Computation Graphs

A computation graph represents the process of computing a mathematical expression

Example: $L(a,b,c) = c(a+2b)$

Computations:

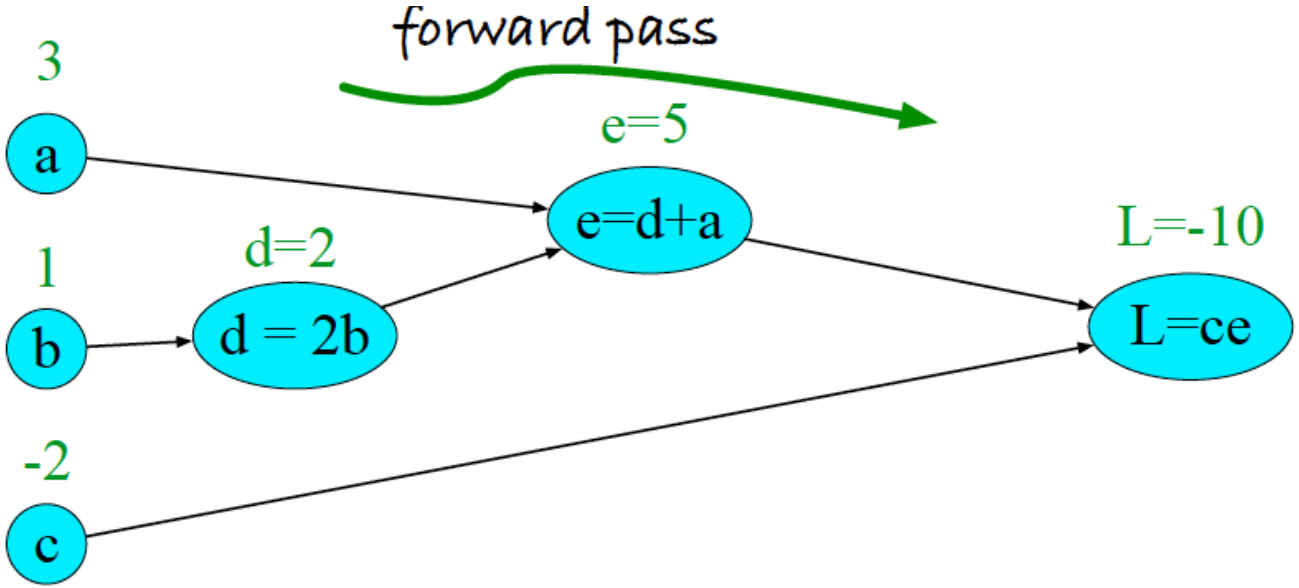
$$d = 2*b$$
$$e = a+d$$
$$L = c*e$$



Example: $L(a,b,c) = c(a+2b)$

Computations:

$$d = 2*b$$
$$e = a+d$$
$$L = c*e$$



Intuition: Training a 2-layer network

For every training tuple (x, y)

- Run *forward* computation to find our estimate $\hat{y}$
- Run *backward* computation to update weights:
  - For every output node
    - Compute loss L between true y and the estimated $\hat{y}$
    - For every weight w from hidden layer to the output layer
      - Update the weight
  - For every hidden node
    - Assess how much blame it deserves for the current answer
    - For every weight w from input layer to the hidden layer
      - Update the weight

How can I find that gradient for every weight in the network?

These derivatives on the prior slide only give the updates for one weight layer: the last one!

What about deeper networks?

- Lots of layers, different activation functions?

Solution in the next lecture:

- Even more use of the chain rule!!
- Computation graphs and backward differentiation!

Example $L(a,b,c) = c(a+2b)$

$$
\begin{aligned}
d &= 2*b \\
e &= a+d \\
L &= c*e
\end{aligned}
$$

We want: $\frac{\partial L}{\partial a}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial c}$

The derivative $\frac{\partial L}{\partial a}$, tells us how much a small change in *a* affects *L*.

The chain rule

Computing the derivative of a composite function:

$$f(x) = u(v(x)) \qquad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$

$$f(x) = u(v(w(x))) \qquad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx}$$

Example $L(a,b,c) = c(a+2b)$

$$
\begin{aligned}
d &= 2*b \\
e &= a+d \\
L &= c*e
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial L}{\partial c} &= e \\
\frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \\
\frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}
\end{aligned}
$$

Example

$$L(a,b,c) = c(a+2b)$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

$$\begin{aligned} L = ce &: \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e \\ e = a+d &: \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1 \\ d = 2b &: \quad \frac{\partial d}{\partial b} = 2 \end{aligned}$$

Example

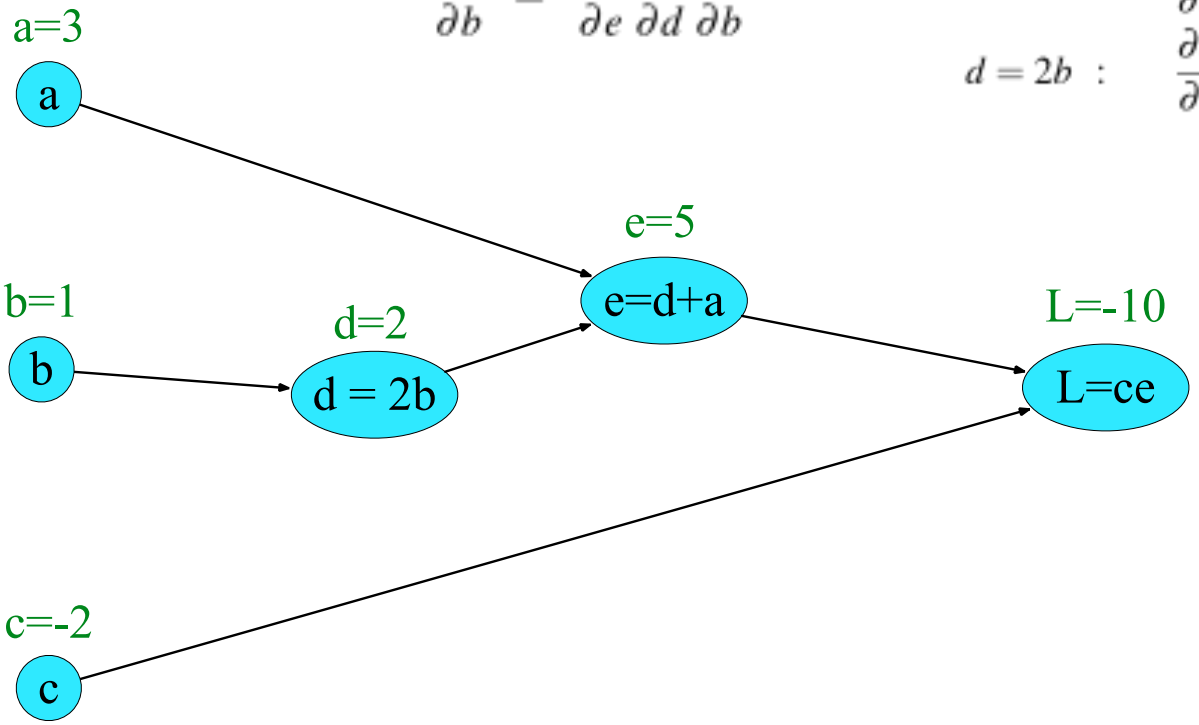
$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

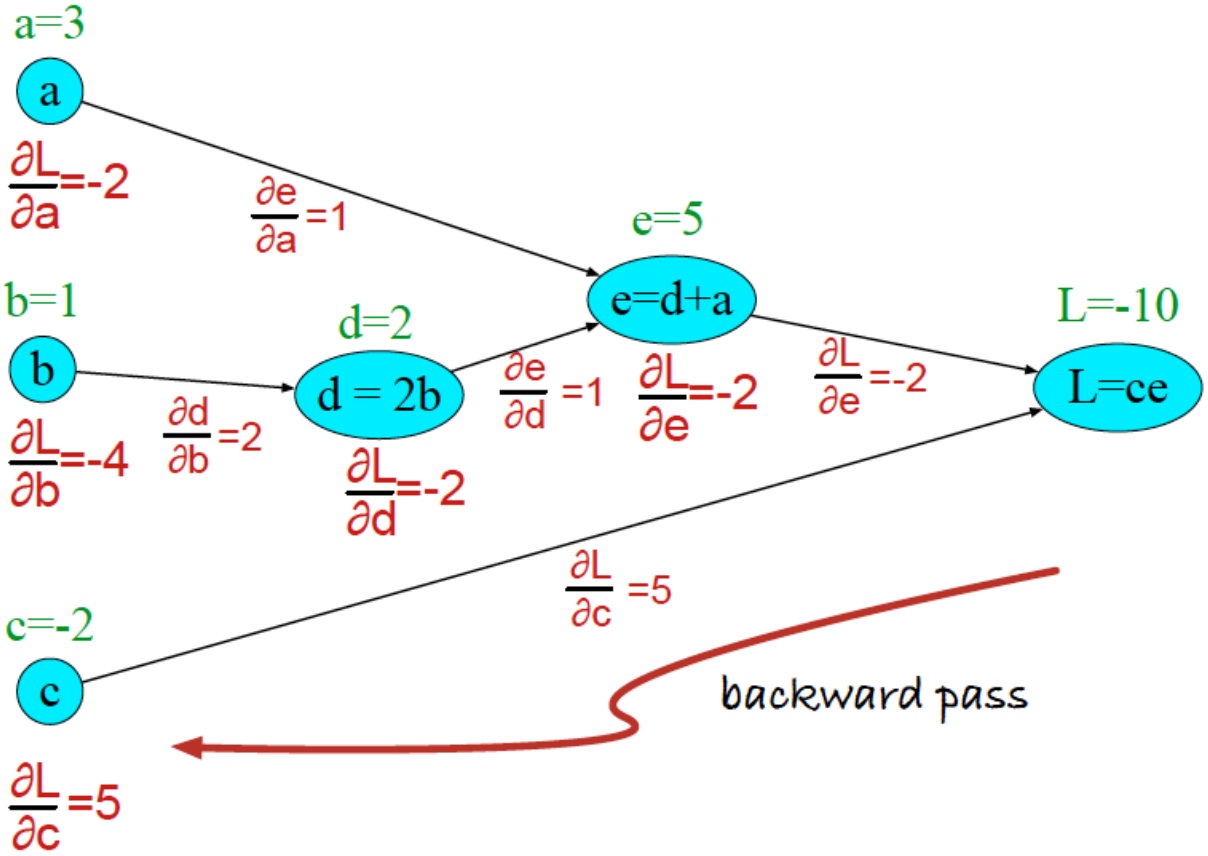
$$L = ce : \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

$$e = a + d : \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$

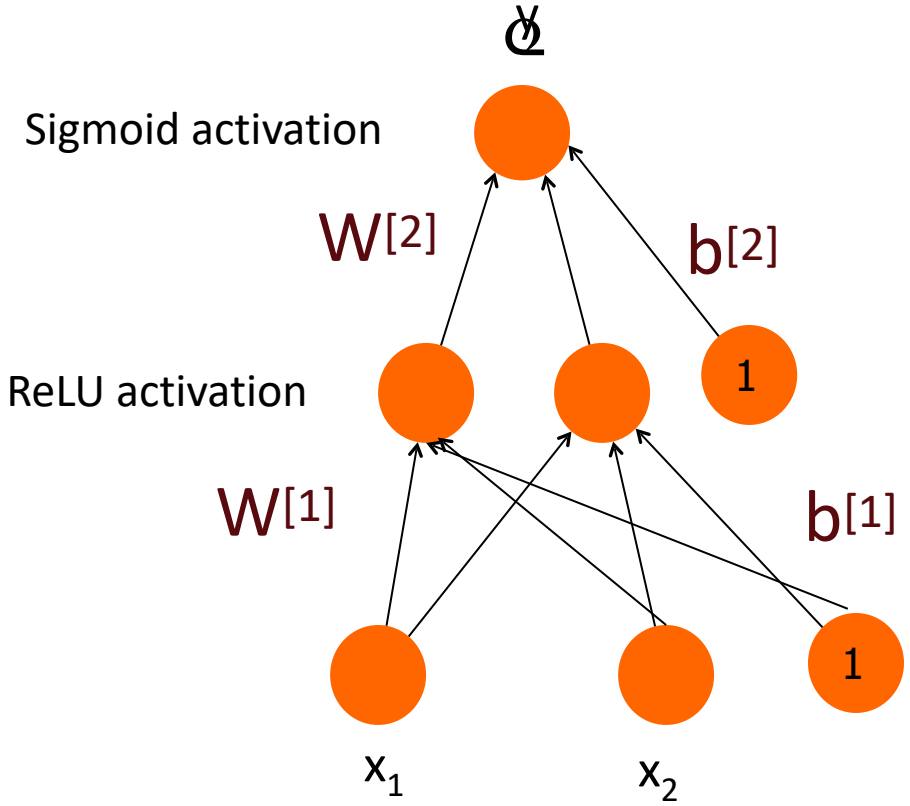
$$d = 2b : \quad \frac{\partial d}{\partial b} = 2$$



Example


a=3
a
∂L/∂a=-2
∂e/∂a=1
e=5
e=d+a
b=1
b
d=2
d = 2b
∂d/∂b=2
∂L/∂b=-4
∂L/∂d=-2
∂e/∂d=1
∂L/∂e=-2
∂L/∂e=-2
L=-10
L=ce
∂L/∂c=5
c=-2
c
backward pass
∂L/∂c=5

Backward differentiation on a two layer network



$$
\begin{aligned}
z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
a^{[1]} &= \mathrm{ReLU}(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$

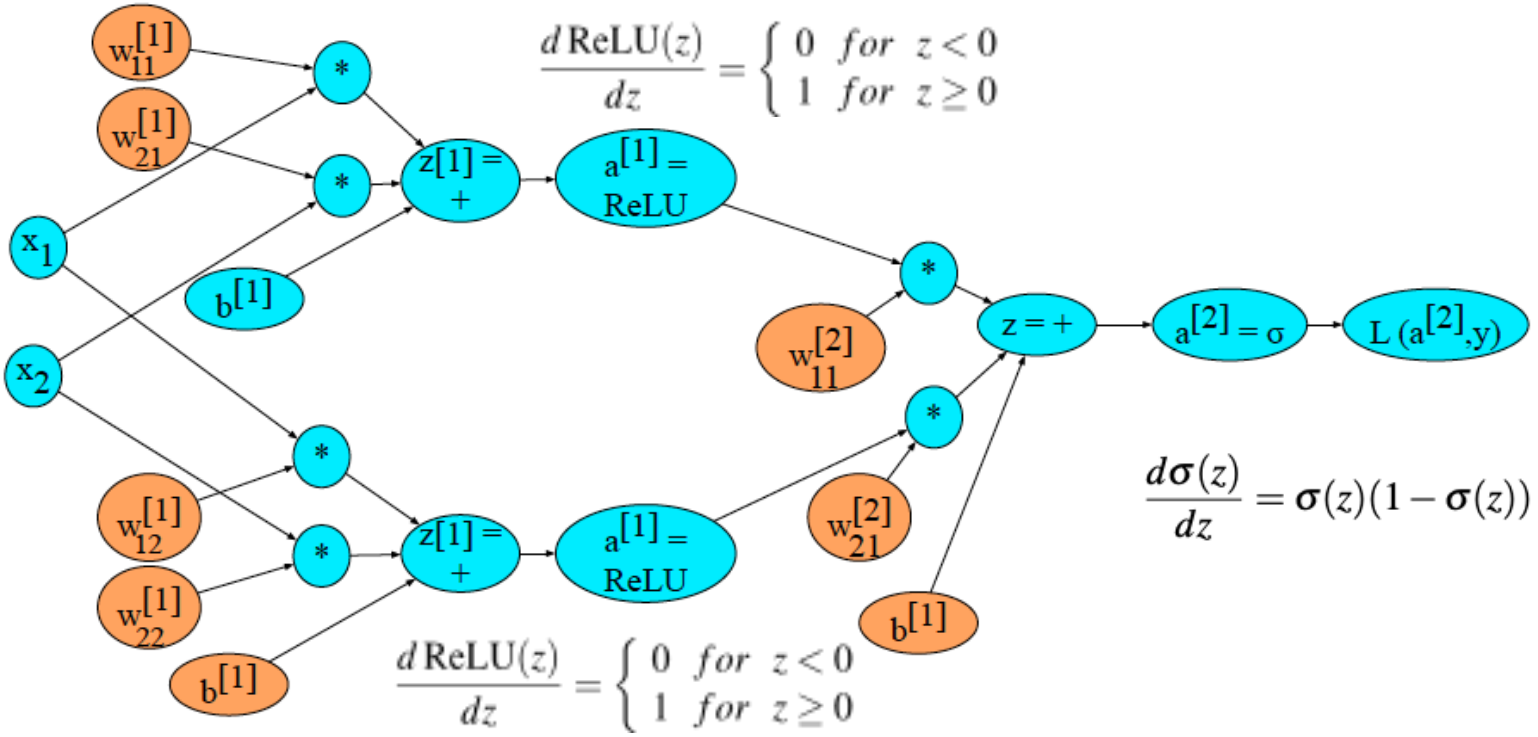
Backward differentiation on a two layer network

$$
\begin{aligned}
z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
a^{[1]} &= \mathbf{ReLU}(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$

$$
\frac{d\,\mathrm{ReLU}(z)}{dz} = \begin{cases} 0 & for \;\; z < 0 \\ 1 & for \;\; z \geq 0 \end{cases}
$$

$$
\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))
$$

Backward differentiation on a 2-layer network



$$
\begin{aligned}
z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
a^{[1]} &= \mathrm{ReLU}(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$

Starting off the backward pass: $\frac{\partial L}{\partial z}$

(I'll write a for $a^{[2]}$ and z for $z^{[2]}$)

$$L\left(\hat{y}, y\right) = -\left(\mathrm{y}\log(\hat{y}) + \left(1-y\right)\log(1-\hat{y})\right)$$

$$L\left(a, y\right) = -\left(\mathrm{y}\log a + \left(1-y\right)\log(1-a)\right)$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}$$

$$\frac{\partial L}{\partial a} = -\left(\left(y\frac{\partial \log(a)}{\partial a}\right) + (1-\mathrm{y})\frac{\partial \log(1-a)}{\partial a}\right)$$

$$= -\left(\left(y\frac{1}{a}\right) + \left(1-\mathrm{y}\right)\frac{1}{1-a}(-1)\right) = -\left(\frac{y}{a} + \frac{y-1}{1-a}\right)$$

$$\frac{\partial a}{\partial z} = a(1-a) \qquad \frac{\partial L}{\partial z} = -\left(\frac{y}{a} + \frac{y-1}{1-a}\right)a(1-a) = \mathrm{a} - \mathrm{y}$$

Summary

For training, we need the derivative of the loss with respect to weights in early layers of the network

- But loss is computed only at the very end of the network!

Solution: **backward differentiation**

Given a computation graph and the derivatives of all the functions in it we can automatically compute the derivative of the loss with respect to these early weights.

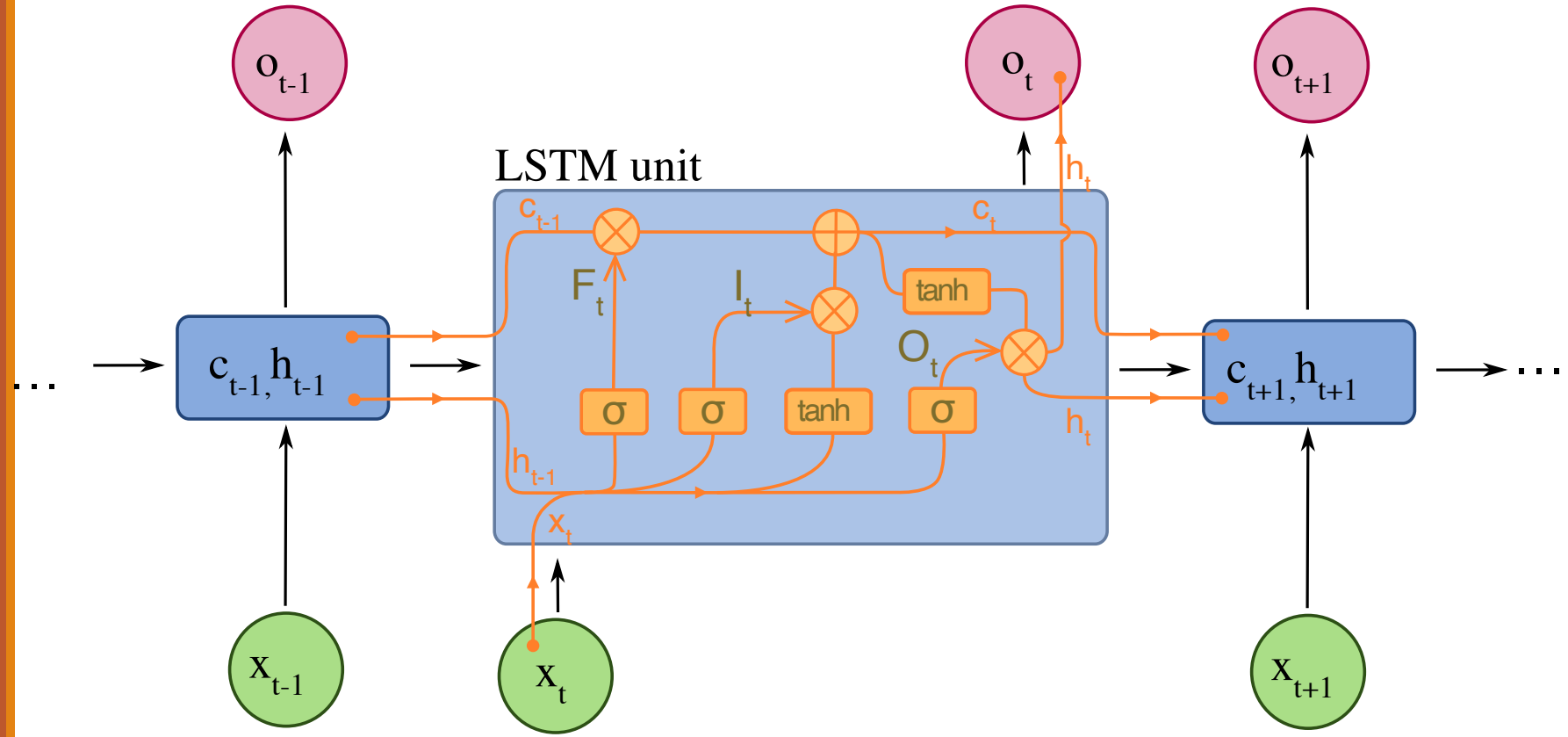
Why doesn’t this work Well?

Vanishing Gradient Problem

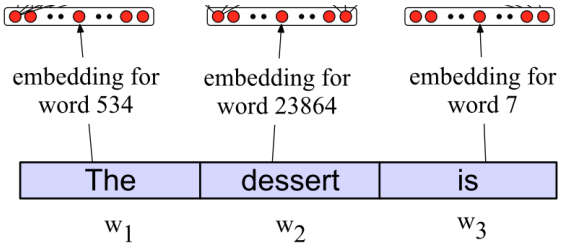
As add more layers

Gradient magnitude decreases or grows uncontrollably

Long Short-term Memory Cell



Issue: texts come in different sizes



This assumes a fixed size length (3)!

Kind of unrealistic.

Some simple solutions (more sophisticated solutions later)

1. Make the input the length of the longest review
   - If shorter then pad with zero embeddings
   - Truncate if you get longer reviews at test time
2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
   - Take the mean of all the word embeddings
   - Take the element-wise max of all the word embeddings
     - For each dimension, pick the max value from all words

Neural Language Models (LMs)

Language Modeling: Calculating the probability of the next word in a sequence given some history.

- We've seen N-gram based LMs
- But neural network LMs far outperform n-gram language models

State-of-the-art neural LMs are based on more powerful neural network technology like Transformers

But **simple feedforward LMs** can do almost as well!

Simple feedforward Neural Language Models

Task: predict next word w_t

given prior words w_{t-1}, w_{t-2}, w_{t-3}, …

Problem: Now we're dealing with sequences of arbitrary length.

Solution: Sliding windows (of fixed length)

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$$

Neural Language Model