Recitation 2: Computing Derivatives

(A story of influences)
Today

• **Goal:** Conceptual understanding of the math behind backprop/autograd

• This will be helpful for **hw1p1** and DL in general
  
  • hw1p1 writeup should be enough to complete assignment
  
  • But this recitation will provide context, breadth, and depth
  
  • Concepts here will be useful throughout the course

• **We’ll try to minimize overlap with the writeup to keep this helpful**
Agenda

1. Motivation: Training and Loss
2. Backprop: Derivatives, Gradients, and the Chain Rule
3. Intro to Autograd (WIP; will cover in actual recitation)
4. Helpful code/math/tips
   - Depth-First Search and recursion (Autograd backward)
   - Derivatives on matrix operations (WIP; will cover in actual recitation)
5. Autograd example
Motivation: Training and Loss
Why Calculus?

• Training a NN is essentially an optimization problem.

Goal: Minimize the loss by adjusting network parameters

• To see how an NN does this, let’s look at a single training loop iteration
Training a Neural Network

1. Forward Propagation

a. Provide observation to network, network tries to guess label
Training a Neural Network

1. Forward Propagation
   a. Provide observation to network, network tries to guess label

b. Network makes guess

c. “Score” performance by generating a loss value.

Actual: 1

Loss Function

Loss = 2.324
Training a Neural Network

1. Forward Propagation
   a. Provide observation to network, network tries to guess label
   b. Network makes guess

   "Score" performance by generating a loss value.

   Actual: 1

   Loss = 2.324

2. Backpropagation
   Starting from the loss and moving backward through the network, calculate gradient of loss w.r.t. each param \( \frac{\partial \text{loss}}{\partial w_i} \)

   Goal is to understand how adjusting each param would affect the loss.

\[
\frac{\partial \text{Loss}}{\partial w_i} = \frac{\partial \text{Loss}}{\partial \text{LossFunc}} \cdot \frac{\partial \text{LossFunc}}{\partial \text{Guess}} \cdot \frac{\partial \text{Guess}}{\partial w_j} \cdot \frac{\partial w_j}{\partial w_i}
\]

(For each \( w_i \))
Training a Neural Network

1. Forward Propagation
   a. Provide observation to network, network tries to guess label
   b. Network makes guess

   Observation

   (Repeat)

2. Backpropagation
   Starting from the loss and moving backward through the network, calculate gradient of loss w.r.t. each param \( \frac{\partial \text{loss}}{\partial w_i} \)
   Goal is to understand how adjusting each param would affect the loss.

3. Step
   Update weights using optimizer.
   The optimizer, based on the gradients, determines how to update weights in order to minimize loss.
Loss Values
Loss Function & Value

- Really important in ML and optimization
- General metric for evaluating performance
- Minimizing an (appropriate) loss metric should cause improved performance

Actual: 1
Loss Function
Loss = 2.324
Example: CrossEntropyLoss

• Task: classifying dogs and cats

Observations (batch of 3)
Example: CrossEntropyLoss

• Task: classifying dogs and cats

Observations (batch of 3)

Logits (label confidence)

<table>
<thead>
<tr>
<th></th>
<th>Cat</th>
<th>Dog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Img_1</td>
<td>1.01</td>
<td>1.2</td>
</tr>
<tr>
<td>Img_2</td>
<td>12.5</td>
<td>0.02</td>
</tr>
<tr>
<td>Img_3</td>
<td>0.23</td>
<td>1.35</td>
</tr>
</tbody>
</table>

Size: (batch_size, num_possible_labels)
Example: CrossEntropyLoss

- Task: classifying dogs and cats

Observations (batch of 3)

Logits (label confidence)

<table>
<thead>
<tr>
<th></th>
<th>Cat</th>
<th>Dog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Img_1</td>
<td>1.01</td>
<td>1.2</td>
</tr>
<tr>
<td>Img_2</td>
<td>12.5</td>
<td>0.02</td>
</tr>
<tr>
<td>Img_3</td>
<td>0.23</td>
<td>1.35</td>
</tr>
</tbody>
</table>

Size: (batch_size, num_possible_labels)

Notice: incorrect guess
Example: CrossEntropyLoss

Logits (label confidence)

<table>
<thead>
<tr>
<th></th>
<th>Cat</th>
<th>Dog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Img_1</td>
<td>1.01</td>
<td>1.2</td>
</tr>
<tr>
<td>Img_2</td>
<td>12.5</td>
<td>0.02</td>
</tr>
<tr>
<td>Img_3</td>
<td>0.23</td>
<td>1.35</td>
</tr>
</tbody>
</table>

True labels

<table>
<thead>
<tr>
<th></th>
<th>Cat</th>
<th>Dog</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Cat</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Cat</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Dog</td>
<td></td>
</tr>
</tbody>
</table>

Size: (batch_size, )
Example: CrossEntropyLoss

<table>
<thead>
<tr>
<th>Logits (label confidence)</th>
<th>True labels</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cat</strong></td>
<td><strong>0</strong></td>
</tr>
<tr>
<td><strong>Dog</strong></td>
<td><strong>1</strong></td>
</tr>
<tr>
<td><strong>Img_1</strong></td>
<td></td>
</tr>
<tr>
<td>1.01</td>
<td></td>
</tr>
<tr>
<td><strong>1.2</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Img_2</strong></td>
<td></td>
</tr>
<tr>
<td>12.5</td>
<td><strong>0</strong></td>
</tr>
<tr>
<td>0.02</td>
<td><strong>Cat</strong></td>
</tr>
<tr>
<td><strong>Img_3</strong></td>
<td></td>
</tr>
<tr>
<td>0.23</td>
<td><strong>1</strong></td>
</tr>
<tr>
<td><strong>1.35</strong></td>
<td><strong>Dog</strong></td>
</tr>
</tbody>
</table>

CrossEntropyLoss
Example: CrossEntropyLoss

<table>
<thead>
<tr>
<th></th>
<th>Cat</th>
<th>Dog</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Img_1</strong></td>
<td>1.01</td>
<td>1.2</td>
</tr>
<tr>
<td><strong>Img_2</strong></td>
<td>12.5</td>
<td>0.02</td>
</tr>
<tr>
<td><strong>Img_3</strong></td>
<td>0.23</td>
<td>1.35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>True labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

CrossEntropyLoss

Average Loss Value for Batch: 0.3583
Loss Value - Notes

• Details of CrossEntropyLoss calculation in hw1p1 writeup
• There are many other possible ways to define loss, and each incentivize/punish different aspects of network training
• In general:
  • Loss value is **one** float for the entire batch
    • Aggregate loss of each observation using summing or averaging
      • (Usually averaging; we’ll do averaging in hw1p1)
Why loss instead of accuracy?

• Loss vs. accuracy (correct guesses / total)?
  • Loss is hard to interpret, which is bad
    • $0 \leq \text{Loss} \leq \ln(\text{num\_classes})$
  • BUT it captures more detail
    • Accuracy only cares about the final answer
    • Loss looks at the confidence in all labels
  • ALSO it’s ‘smoother’
    • In loss, partially correct answers are better than very incorrect
    • In accuracy, partially correct == very incorrect

• Compromise: train on loss, validate on accuracy
  • Makes validation results interpretable
Summary

• Loss value evaluates network performance
• The lower the loss, the better the performance

• This means:
  • Our goal is to modify network params to lower loss
Backprop:
Derivatives, Gradients, and the Chain Rule
1. Forward Propagation
   a. Provide observation to network, network tries to guess label

   b. Network makes guess

   c. “Score” performance by generating a loss value.

   So far:

   
   [Diagram showing a neural network with input, hidden layers, and output with actual 1 and loss 2.324]
So far:

1. **Forward Propagation**
   a. Provide observation to network, network tries to guess label
   b. Network makes guess

   ![Forward Propagation Diagram]

   **Observation**

2. **Backpropagation**
   Determine how each weight affects the loss by calculating partial derivative

3. **Step**
   Adjust weights using those gradients

   **Actual: 1**
   **Loss Function**
   **Loss = 2.324**
Backprop Interlude:
(Re)defining the Derivative
(Re)defining the Derivative

• You probably have experience with scalar derivatives and a bit of multivariable calc

• But how does that extend to matrix derivatives?

• **Now: intuition and context of scalar and matrix derivatives**
  • This should help you understand what derivatives actually do, how this applies to matrices, and what the **shapes** of the input/output/derivative matrices are.
  • This is better than memorizing properties.
Scalar Derivatives ($\alpha$ definition)

$$f(x) = y$$

$x$ and $y$ are scalars
Scalar Derivatives (α definition)

\[ f(x) = y \]

\( x \) and \( y \) are scalars
Scalar Derivatives ($\alpha$ definition)

$f(x) = y$

$x$ and $y$ are scalars

Goal: determine how changing the input affects the output
Scalar Derivatives (α definition)

\[ f(x) = y \]
\[ x \text{ and } y \text{ are scalars} \]

Goal: Find \( \Delta y \) given \( \Delta x \)
Scalar Derivatives (\(\alpha\) definition)

We define relationship between \(\Delta x\) and \(\Delta y\) as \(\alpha\).

\[
\Delta y = \alpha \Delta x
\]

- \(\alpha\) is some factor multiplied to \(\Delta x\) that gives \(\Delta y\)
Scalar Derivatives (α definition)

We define relationship between $\Delta x$ and $\Delta y$ as $\alpha$.

\[ \Delta y = \alpha \Delta x \]

• $\alpha$ is some factor multiplied to $\Delta x$ that gives $\Delta y$.

• **Plot twist: $\alpha$ is the derivative $f'(x)$**
Derivatives (scalar in, scalar out)

\[ \Delta y = f'(x) \Delta x \]

- Key idea: the derivative is not just a value (i.e. ‘the slope’)
- The derivative is a **linear transformation**, mapping \( \Delta x \) onto \( \Delta y \).

\[ f'(x): \Delta x \mapsto \Delta y \]

\[ \mathbb{R}^1 \mapsto \mathbb{R}^1 \]
Derivatives (vector in, scalar out)

Let’s go to *higher dimensions*. **Multiple arguments and scalar output.**

\[ f(x_1, \ldots, x_D) = y \]

Vector-scalar derivatives use the same general form as scalar-scalar derivatives. To do this, group the input variables into a 1-D vector \( \mathbf{x} \).

\[ \Delta y = \alpha \cdot \mathbf{x} \]

\[ = \begin{bmatrix} a_1 & \ldots & a_D \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_D \end{bmatrix} \]

Note: vectors are notated in bold and unitalicized font.
Derivatives (vector in, scalar out)

\[ \Delta y = \alpha \cdot x \]

\[ = [a_1 \ldots a_D] \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_D \end{bmatrix} \]

Same thing below, but in more familiar notation:

\[ \Delta y = \frac{\partial y}{\partial x} \cdot x \]

\[ = \begin{bmatrix} \frac{\partial y}{\partial x_1} \ldots \frac{\partial y}{\partial x_D} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_D \end{bmatrix} \]
Derivatives (vector in, scalar out)

\[ \Delta y = \mathbf{a} \cdot \mathbf{x} \]

\[ = \begin{bmatrix} a_1 & \ldots & a_D \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_D \end{bmatrix} \]

Same thing below, but in more familiar notation:

\[ \Delta y = \frac{\partial y}{\partial \mathbf{x}} \cdot \mathbf{x} \]

\[ = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \ldots & \frac{\partial y}{\partial x_D} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_D \end{bmatrix} \]
In summary, for a function of multiple arguments \( \mathbf{x} \) and scalar output \( y \)

\[
f(\mathbf{x}) = y
\]

Its derivative is this:

\[
\frac{\partial y}{\partial \mathbf{x}} = \left[ \frac{\partial y}{\partial x_1} \ldots \frac{\partial y}{\partial x_D} \right]
\]

Note: the derivative’s shape will always be transposed from the input shape.

This will be true for ALL matrix derivatives
(See next slide for why)
Derivatives are Dot Products

Recall:

\[ \Delta y = \nabla_x y \cdot x \]

\[ = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \cdots & \frac{\partial y}{\partial x_D} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_D \end{bmatrix} \]

By notational convention:

\[ a \cdot b = ab^T \]
Derivatives (vector in, vector out)

• For a function that inputs and outputs vectors, $\nabla_x y$ is the "Jacobian".

\[
\begin{align*}
\text{Input} & \quad \text{Output} \\
x &= \begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_K \end{bmatrix} \\
\frac{\partial y}{\partial x} &= \begin{bmatrix}
\frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_D} \\
\vdots & \ddots & \vdots \\
\frac{\partial y_K}{\partial x_1} & \cdots & \frac{\partial y_K}{\partial x_D}
\end{bmatrix}
\end{align*}
\]

$D \times 1 \quad K \times 1 \quad K \times D$
Derivatives (vector in, vector out)

• For a function that inputs and outputs vectors, $\nabla_x y$ is the “Jacobian”.

\[
x = \begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_K \end{bmatrix}
\]

\[
\frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_K}{\partial x_1} & \cdots & \frac{\partial y_K}{\partial x_D} \end{bmatrix}
\]

$D \times 1$  $K \times 1$  $K \times D$

Note: each row of the derivative matrix is essentially a vector-scalar matrix from the previous slide.
Summary

Covered 3 cases:

1. Scalar/scalar function derivative $f'(x)$

2. Vector/scalar derivative $\frac{\partial y}{\partial x} = \left[ \frac{\partial y}{\partial x_1} \ldots \frac{\partial y}{\partial x_D} \right]$

3. Vector/vector derivative $\left( \frac{dy}{dx} \right)$

Key Ideas

- The derivative is the **best linear approximation** of $f$ at a point
- The derivative describes **the effect of each input on the output**
But what is the gradient?

‘Gradients’ are the transpose of a vector-scalar derivative

\[ \nabla f = \left( \frac{\partial y}{\partial x} \right)^T = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \vdots \\ \frac{\partial y}{\partial x_D} \end{bmatrix} \]

They’re technically different from normal derivatives, but have many similar properties. So in conversation, people will often interchange the two.

One difference: interpretation

While the derivative projects change in output onto change in input, the gradient is that change in input interpreted as a vector. Also, as it’s a tangent vector to the input space at a point, you can interpret it in the context of the input space. Derivative would be cotangent.

(\(^ you don’t need to fully understand this for class, don’t worry (see here for more)\))
But what is the gradient?

• One nice property: Great for **optimization** (finding max/min)
  • The gradient is a vector that points towards the ‘**direction**’ of steepest increase.

  ![img source](image-source)

• If **maximizing**, follow the gradient.
• If **minimizing**, go in the opposite direction (gradient descent)
Partial vs. Total Derivatives

\[
\frac{dy}{dx} \quad \text{vs} \quad \frac{\partial y}{\partial x_i}
\]

- The total influence of \( x \) on \( y \)
- (Was today’s topic, same as \( \alpha \) or \( \nabla \))

- The influence of just \( x_i \) on \( y \)
- Assumes other variables are held constant

Remember before:

\[
\frac{\partial y}{\partial x} = \begin{bmatrix}
\frac{\partial y}{\partial x_1} \\
\vdots \\
\frac{\partial y}{\partial x_D}
\end{bmatrix}
\]

But this is pretty idealized; if variables influence each other, it gets messy
Things get messy

Find $\frac{dy}{dx}$ for $f(x, z) = y$, where $z = g(x, w)$
Things get messy

Find $\frac{dy}{dx}$ for $f(x, z) = y$, where $z = g(x, w)$

$x$ affects $y$ twice; directly in $f$, and indirectly through $z$. 
Things get messy

Find $\frac{dy}{dx}$ for $f(x, z) = y$, where $z = g(x, w)$

$x$ affects $y$ twice; directly in $f$, and indirectly through $z$. 
Things get messy

Find $\frac{dy}{dx}$ for $f(x, z) = y$, where $z = g(x, w)$
Things get messy

Find \( \frac{dy}{dx} \) for \( f(x, z) = y \), where \( z = g(x, w) \)
Things get messy

Find $\frac{dy}{dx}$ for $f(x, z) = y$, where $z = g(x, w)$

Goal: get only $x$’s influence on $y$
Things get messy

Find $\frac{dy}{dx}$ for $f(x, z) = y$, where $z = g(x, w)$

If we just said $\frac{dy}{dx} = \frac{\partial y}{\partial x} + \frac{\partial y}{\partial z}$, we’d end up including $w$’s influence on $y$. 
Things get messy

Find \( \frac{dy}{dx} \) for \( f(x, z) = y \), where \( z = g(x, w) \)

It’s time for... “the chain rule”
The Chain Rule

• The chain rule is used to properly **account for influences in nested functions**
  
  • Recursively calculates derivatives on nested functions w.r.t. target
The Chain Rule

• The chain rule is used to properly account for influences in nested functions
  • Recursively calculates derivatives on nested functions w.r.t. target

\[
\frac{dy}{dx} = \frac{\partial y}{\partial x} + \frac{\partial y}{\partial z}
\]
The Chain Rule

- The chain rule is used to properly **account for influences in nested functions**
  - Recursively calculates derivatives on nested functions w.r.t. target

\[
\frac{dy}{dx} = \frac{\partial y}{\partial x} + \frac{\partial y}{\partial z}?
\]
The Chain Rule

- The chain rule is used to properly **account for influences in nested functions**
  - Recursively calculates derivatives on nested functions w.r.t. target

\[
\frac{dy}{dx} = \frac{\partial y}{\partial x} + \frac{\partial y}{\partial z} \frac{dz}{dx}
\]
The Chain Rule

- The chain rule is used to properly **account for influences in nested functions**
  - Recursively calculates derivatives on nested functions w.r.t. target

\[
\frac{dy}{dx} = \frac{\partial y}{\partial x} + \frac{\partial y}{\partial z} \frac{dz}{dx}
\]
HW1P1 Help & Tips
Depth-First Search (DFS)

- We’ll briefly cover DFS, as it’s needed for autograd
- **Algorithm used to traverse nodes in trees/graph**
  - Anything with vertices/edges; directed or undirected

Example of a graph
Depth-First Search (DFS)

Goal: To visit every node in the graph, starting from some node

i.e. Start from here
Depth-First Search (DFS)

Goal: To visit every node in the graph, starting from some node

i.e. Start from here
Depth-First Search (DFS)

- There’s multiple ways to implement DFS, but our implementation of autograd uses **recursion**
- Recursion
  - When a function calls itself, leading to ‘nested’ calls
Depth-First Search (DFS)

- There’s multiple ways to implement DFS, but our implementation of autograd uses **recursion**
- Recursion
  - When a function calls itself, leading to ‘nested’ calls
Recursion

• Useful for tasks where you’re repeating instructions/checks
  • For example, traversing graphs

• Essentially performs ‘iterative’ tasks (just like while loops)
  • In fact, iteration and recursion are equally expressive

• Similar to while loops, you generally need one or more base case(s) that tell the function when to stop recursing
  • Otherwise it recurses infinitely and crashes your computer 😞
Recursion (Simple Example)

def greater_than_three(x):
    print("Recursive call, x=" + str(x))
    if x < 3:
        result = greater_than_three(x + 1)
        print("Received: x=" + str(result) + " and returning upward.")
        return result
    else:
        print("Hit base case. x=" + str(x))
        return x

• This method will continually make recursive calls until the base case
  • Base case: input value is >=3

• After hitting the base case, repeatedly close the nested iterations
Recursion (Simple Example)

def greater_than_three(x):
    print("Recursive call, x=" + str(x))
    if x < 3:
        result = greater_than_three(x + 1)
        print("Received: x=" + str(result) + " and returning upward.")
        return result
    else:
        print("Hit base case. x=" + str(x))
        return x

>>> result = greater_than_three(0)
Recursive call, x=0
Recursive call, x=1
Recursive call, x=2
Recursive call, x=3
Hit base case (>=3). x=3
Received: x=3 and returning upward.
Received: x=3 and returning upward.
Received: x=3 and returning upward.
>>> print("Final result: x=" + str(result))
Final result: 3
def greater_than_three(x):
    print("Recursive call, x=" + str(x))
    if x < 3:
        result = greater_than_three(x + 1)
        print("Received: x=" + str(result) + " and returning upward.")
        return result
    else:
        print("Hit base case. x=" + str(x))
        return x

>>> result = greater_than_three(0)
Recursive call, x=0
Recursive call, x=1
Recursive call, x=2
Recursive call, x=3
Hit base case (>=3). x=3
Received: x=3 and returning upward.
Received: x=3 and returning upward.
Received: x=3 and returning upward.
>>> print("Final result: x=" + str(result))
Final result: 3
Recursion (Simple Example)

def greater_than_three(x):
    print("Recursive call, x=\" + str(x))
    if x < 3:
        result = greater_than_three(x + 1)
        print("Received: x=\" + str(result) + \" and returning upward.\")
        return result
    else:
        print("Hit base case. x=\" + str(x))
        return x

# Here’s an example where
# the base case is already met

>>> result = greater_than_three(4)
Recursive call, x=4
Hit base case (>=3). x=4
>>> print("Final result: x=\" + str(result))
Final result: 4

# No nested calls were made.
Recursion

• You can modify the previous example to achieve different things

• For example, you don’t always need to return an output

• You can also ‘branch’
  • Calling the function multiple times on the same ‘level’
Recursion (Branching Example)

def branching_recursion(x):
    print("Recursive call, x=" + str(x))
    if isinstance(x, list):
        for item in x:
            branching_recursion(item)
    else:
        print("Hit base case (No more nested lists). x=" + str(x))

>>> branching_recursion([[1, 2], [[3], 4], 5])
Recursive call, x=[[1, 2], [[3], 4], 5]
Recursive call, x=[[1, 2], [[3], 4], 5]
Recursive call, x=[1, 2]
Recursive call, x=1
Hit base case (No more nested lists). x=1
Recursive call, x=2
Hit base case (No more nested lists). x=2
Recursive call, x=[[3], 4]
Recursive call, x=[[3], 4]
Recursive call, x=[3]
Recursive call, x=3
Hit base case (No more nested lists). x=3
Recursive call, x=4
Hit base case (No more nested lists). x=4
Recursive call, x=5
Hit base case (No more nested lists). x=5

Interpret this yourself for now, will discuss in detail on Friday
Extra Resources
# Scalar Deriv. Cheat Sheet

<table>
<thead>
<tr>
<th>Rule</th>
<th>$f(x)$</th>
<th>Scalar derivative notation with respect to $x$</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>$c$</td>
<td>$0$</td>
<td>$\frac{d}{dx} 99 = 0$</td>
</tr>
<tr>
<td>Multiplication by constant</td>
<td>$cf$</td>
<td>$c \frac{df}{dx}$</td>
<td>$\frac{d}{dx} 3x = 3$</td>
</tr>
<tr>
<td>Power Rule</td>
<td>$x^n$</td>
<td>$nx^{n-1}$</td>
<td>$\frac{d}{dx} x^3 = 3x^2$</td>
</tr>
<tr>
<td>Sum Rule</td>
<td>$f + g$</td>
<td>$\frac{df}{dx} + \frac{dg}{dx}$</td>
<td>$\frac{d}{dx} (x^2 + 3x) = 2x + 3$</td>
</tr>
<tr>
<td>Difference Rule</td>
<td>$f - g$</td>
<td>$\frac{df}{dx} - \frac{dg}{dx}$</td>
<td>$\frac{d}{dx} (x^2 - 3x) = 2x - 3$</td>
</tr>
<tr>
<td>Product Rule</td>
<td>$fg$</td>
<td>$f \frac{dg}{dx} + \frac{df}{dx} g$</td>
<td>$\frac{d}{dx} x^2 x = x^2 + x 2x = 3x^2$</td>
</tr>
<tr>
<td>Chain Rule</td>
<td>$f(g(x))$</td>
<td>$\frac{df}{du} \frac{du}{dx}$, let $u = g(x)$</td>
<td>$\frac{d}{dx} ln(x^2) = \frac{1}{x^2} 2x = \frac{2}{x}$</td>
</tr>
</tbody>
</table>

Table Source
The Matrix Calculus You Need For Deep Learning

Matrix Calculus Reference

Gradients and Jacobians

The gradient of a function of two variables is a horizontal 2-vector:

\[ \nabla f(x, y) = \left[ \frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y} \right] \]

The Jacobian of a vector-valued function that is a function of a vector is an \( m \times \) possible scalar partial derivatives:

Nice reference, with DL-specific examples and explanations
Clear rules and examples of how to take matrix derivatives.
Good Resources

- [https://en.wikipedia.org/wiki/Matrix_calculus](https://en.wikipedia.org/wiki/Matrix_calculus)
  - Another excellent reference; just be careful about notation
- [Khan Academy’s article on gradients](https://en.wikipedia.org/wiki/Backpropagation)
  - *Simple/intuitive* visualizations and explanation
  - NumPy’s matrix operations documentation
Proof by Examples: Computing Derivatives Can be Trivial
Influence Diagrams
A New (Made Up) Activation in Town

\[ y_i = \cos\left( e^{x_i} \frac{\sum_j x_j}{\sum_j \ln(x_j)} \right) \]
A New (Made Up) Activation in Town

\[ y_i = \cos\left(\frac{e^{x_i} \sum_j x_j}{\sum_j \ln(x_j)}\right) \]
A New (Made Up) Activation in Town

\[ y_i = \cos \left( e^{x_i} \frac{\sum_j x_j}{\sum_j \ln(x_j)} \right) \]

This is a **vector activation**, as inputs affect multiple outputs.
A New (Made Up) Activation in Town

\[ y_i = \cos\left(\frac{e^{x_i} \sum_j x_j}{\sum_j \ln(x_j)}\right) \]

Let's calculate derivatives

Goal: \( \nabla_x L \)
A New (Made Up) Activation in Town

\[ y_i = \cos \left( e^{x_i} \frac{\sum_j x_j}{\sum_j \ln(x_j)} \right) \]

First we’ll break things up so they’re manageable...
A New (Made Up) Activation in Town

\[
y_i = \cos\left(\frac{e^{x_i} \sum_j x_j}{\sum_j \ln(x_j)}\right)
\]

\[
a = \sum_j x_j
\]
\[
b = \sum_j \ln(x_j)
\]
\[
y_i = \cos\left(\frac{e^{x_i} a}{b}\right)
\]

First we’ll break things up so they’re manageable...
A New (Made Up) Activation in Town

\[ a = \sum_{j} x_j \]
\[ b = \sum_{j} \ln(x_j) \]
\[ y_i = \cos\left(\frac{e^{x_i}}{b} \cdot a\right) \]

First we’ll break things up so they’re manageable...
A New (Made Up) Activation in Town

\[ a = \sum_j x_j \]
\[ b = \sum_j \ln(x_j) \]
\[ y_i = \cos\left(\frac{e^{x_i} a}{b}\right) \]

Now we’ll draw the influence diagram...
A New (Made Up) Activation in Town

\[ a = \sum_{j} x_j \]
\[ b = \sum_{j} \ln(x_j) \]
\[ y_i = \cos\left(\frac{e^{x_i a}}{b}\right) \]

Now we’ll draw the influence diagram...
A New (Made Up) Activation in Town

\[ a = \sum_j x_j \]
\[ b = \sum_j \ln(x_j) \]
\[ y_i = \cos\left(\frac{e^{x_i}a}{b}\right) \]
A New (Made Up) Activation in Town

\[ a = \sum_j x_j \]
\[ b = \sum_j \ln(x_j) \]
\[ y_i = \cos\left(\frac{e^{x_i} a}{b}\right) \]
A New (Made Up) Activation in Town

\[ a = \sum x_j \]
\[ b = \sum \ln(x_j) \]
\[ y_i = \cos\left(\frac{e^{x_i a}}{b}\right) \]
A New (Made Up) Activation in Town

\[ a = \sum_j x_j \]
\[ b = \sum_j \ln(x_j) \]
\[ y_i = \cos\left(\frac{e^{x_i} a}{b}\right) \]
A New (Made Up) Activation in Town
A New (Made Up) Activation in Town

Notice $x_1$’s different paths of influence
A New (Made Up) Activation in Town

Notice $x_1$’s different paths of influence

L (loss) is calculated later on...
The derivative $\nabla_{x_1} L$ is the sum of derivatives along these paths.
A New (Made Up) Activation in Town

We will apply the chain rule at each node.
A New (Made Up) Activation in Town

\[
\nabla_{x_1} L =
\]

\[
\begin{align*}
\n\n\end{align*}
\]
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \]
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} \]
A New (Made Up) Activation in Town

\[
\nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1}
\]
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \]
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \]
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} \]
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} \]
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \]
A New (Made Up) Activation in Town

\[
\nabla x_1 L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db}
\]
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} \]
A New (Made Up) Activation in Town

\[
\nabla_{x_1} L = \frac{d y_1}{d x_1} \frac{d L}{d y_1} + \frac{d a}{d x_1} \frac{d y_1}{d a} \frac{d L}{d y_1} + \frac{d b}{d x_1} \frac{d y_1}{d b} \frac{d L}{d y_1}
\]
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \cdots \]

We can do this for the rest of the paths...
A New (Made Up) Activation in Town

\[
\nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_2}{da} \frac{dL}{dy_2} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_2} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_3} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_3}
\]

Seven terms, seven paths
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_2}{da} \frac{dL}{dy_2} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_2} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_3} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_3} \]

Seven terms, seven paths

Questions? What isn’t clear?
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{dy_2}{dy_1} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_1} \]

Now we’re done with the influence diagram
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_2}{da} \frac{dL}{dy_2} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_2} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_3} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_3} \]

Time to calculate the necessary derivatives...
A New (Made Up) Activation in Town

\[\nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_2}{da} \frac{dL}{dy_2} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_2} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_3} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_3}\]

Time to calculate the necessary derivatives...

\[a = \Sigma_j x_j\quad b = \Sigma_j \ln(x_j)\quad y_i = \cos\left(\frac{e^{x_i}}{b}\right)\]
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_2}{da} \frac{dL}{dy_2} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_2} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_3} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_3} \]

Time to calculate the necessary derivatives...

\[ a = \sum_j x_j \quad b = \sum_j \ln(x_j) \quad y_i = \cos\left(\frac{e^{x_i} a}{b}\right) \]

(don’t focus on the following math, just the process)
A New (Made Up) Activation in Town

\[
\nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_2}{da} \frac{dL}{dy_2} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_2} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_3} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_3}
\]

Time to calculate the necessary derivatives...

\[
a = \sum_j x_j \quad b = \sum_j \ln(x_j) \quad y_i = \cos\left(\frac{e^{x_i} a}{b}\right)
\]

\[
\frac{dy_1}{dx_1} = -\sin\left(\frac{e^{x_1} a}{b}\right) \frac{ae^{x_1} a}{b}
\]
A New (Made Up) Activation in Town

$$\nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_2}{da} \frac{dL}{dy_2} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_2} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_3} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_3}$$

Time to calculate the necessary derivatives...

$$a = \Sigma_j x_j \quad b = \Sigma_j \ln(x_j) \quad y_i = \cos(\frac{e^{x_i}}{b})$$

$$\frac{dy_1}{dx_1} = -\sin(\frac{e^{x_1}}{b}) \frac{ae^{x_1}}{b}$$
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_2}{da} \frac{dL}{dy_2} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_2} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_3} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_3} \]

Time to calculate the necessary derivatives...

\[ a = \sum_j x_j \quad b = \sum_j \ln(x_j) \quad y_i = \cos\left(\frac{e^{x_i} a}{b}\right) \]

\[ \frac{dy_1}{dx_1} = -\sin\left(\frac{e^{x_1} a}{b}\right) \frac{a e^{x_1} a}{b} \]

\[ \frac{da}{dx_1} = 1 \]
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_2}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_1} \]

Time to calculate the necessary derivatives...

\[ a = \sum_j x_j \quad b = \sum_j \ln(x_j) \quad y_i = \cos\left(\frac{e^{x_i} \cdot a}{b}\right) \]

\[ \frac{dy_1}{dx_1} = -\sin\left(\frac{e^{x_1} \cdot a}{b}\right) \frac{ae^{x_1} \cdot a}{b} \]

\[ \frac{da}{dx_1} = 1 \]

\[ \frac{dy_1}{da} = -\sin\left(\frac{e^{x_1} \cdot a}{b}\right) \frac{x_1e^{x_1} \cdot a}{b} \]
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_2}{da} \frac{dL}{dy_2} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_2} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_3} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_3} \]

Time to calculate the necessary derivatives...

\[ a = \sum_j x_j \quad b = \sum_j \ln(x_j) \quad y_i = \cos(e^{x_i \frac{a}{b}}) \]

\[ \frac{dy_1}{dx_1} = -\sin\left(\frac{e^{x_1 \frac{a}{b}}}{b}\right) \frac{ae^{x_1 \frac{a}{b}}}{b} \]

\[ \frac{da}{dx_1} = 1 \]

\[ \frac{dy_1}{da} = -\sin\left(\frac{e^{x_1 \frac{a}{b}}}{b}\right) \frac{x_1 e^{x_1 \frac{a}{b}}}{b} \]
A New (Made Up) Activation in Town

$$\nabla_{x_1}L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_2}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_1}$$

Time to calculate the necessary derivatives...

$$a = \sum_j x_j \quad b = \sum_j \ln(x_j) \quad y_i = \cos\left(\frac{e^{x_i}}{b}\right)$$

$$\frac{dy_1}{dx_1} = -\sin\left(\frac{e^{x_1}}{b}\right) \frac{ae^{x_1}}{b}$$

$$\frac{da}{dx_1} = 1$$

$$\frac{dy_1}{da} = -\sin\left(\frac{e^{x_1}}{b}\right) \frac{x_1e^{x_1}}{b}$$
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_2}{da} \frac{dL}{dy_2} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_2} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_3} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_3} \]

Time to calculate the necessary derivatives...

\[ a = \sum_j x_j \quad b = \sum_j \ln(x_j) \quad y_i = \cos\left(\frac{e^{x_i} a}{b}\right) \]

\[ \frac{dy_1}{dx_1} = -\sin\left(\frac{e^{x_1} a}{b}\right) \frac{x_1 e^{x_1} a}{b} \]

\[ \frac{da}{dx_1} = 1 \]

\[ \frac{db}{dx_1} = \frac{1}{x_1} \]

\[ \frac{dy_1}{db} = \sin\left(\frac{e^{x_1} a}{b}\right) \frac{x_1 e^{x_1} a}{b^2} \]

\[ \frac{dy_2}{da} = -\sin\left(\frac{e^{x_2} a}{b}\right) \frac{x_2 e^{x_2} a}{b} \]

\[ \frac{dy_2}{db} = \sin\left(\frac{e^{x_2} a}{b}\right) \frac{x_2 e^{x_2} a}{b^2} \]

\[ \frac{dy_3}{da} = -\sin\left(\frac{e^{x_3} a}{b}\right) \frac{x_3 e^{x_3} a}{b} \]

\[ \frac{dy_3}{db} = \sin\left(\frac{e^{x_3} a}{b}\right) \frac{x_3 e^{x_3} a}{b^2} \]
A New (Made Up) Activation in Town

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_2}{da} \frac{dL}{dy_2} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_2} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_3} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_3} \]

\[
\begin{align*}
\frac{dy_1}{dx_1} &= -\sin\left(\frac{e^{x_1} a}{b}\right) \frac{a e^{x_1} a}{b} \\
\frac{da}{dx_1} &= 1 \\
\frac{dy_1}{da} &= -\sin\left(\frac{e^{x_1} a}{b}\right) \frac{x_1 e^{x_1} a}{b} \\
\frac{db}{dx_1} &= \frac{1}{x_1} \\
\frac{dy_3}{db} &= \sin\left(\frac{e^{x_1} a}{b}\right) \frac{x_1 e^{x_1} a}{b^2}
\end{align*}
\]

Now we plug things in / simplify
A New (Made Up) Activation in Town

\[
\nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} \frac{dL}{dy_1} + \frac{db}{dx_1} \frac{dy_1}{db} \frac{dL}{dy_1} + \frac{dy_2}{dx_1} \frac{dL}{dy_2} + \frac{db}{dx_1} \frac{dy_2}{db} \frac{dL}{dy_2} + \frac{da}{dx_1} \frac{dy_3}{da} \frac{dL}{dy_3} + \frac{db}{dx_1} \frac{dy_3}{db} \frac{dL}{dy_3}
\]

\[
\begin{align*}
\frac{dy_1}{dx_1} &= -\sin\left(\frac{e^{x_1} a}{b}\right) \frac{ae^{x_1} a}{b} \\
\frac{da}{dx_1} &= 1 \\
\frac{db}{dx_1} &= \frac{1}{x_1} \\
\frac{dy_1}{da} &= -\sin\left(\frac{e^{x_1} a}{b}\right) \frac{x_1 e^{x_1} a}{b} \\
\frac{dy_2}{da} &= -\sin\left(\frac{e^{x_2} a}{b}\right) \frac{x_2 e^{x_2} a}{b} \\
\frac{dy_2}{db} &= \sin\left(\frac{e^{x_2} a}{b}\right) \frac{x_2 e^{x_2} a}{b^2} \\
\frac{dy_3}{da} &= -\sin\left(\frac{e^{x_3} a}{b}\right) \frac{x_3 e^{x_3} a}{b} \\
\frac{dy_3}{db} &= \sin\left(\frac{e^{x_3} a}{b}\right) \frac{x_3 e^{x_3} a}{b^2}
\end{align*}
\]

Now we plug things in / simplify

“The simplification is left as an exercise to the reader”
Influence Diagrams

A little painful, but algorithmic:

**Break things up**

\[ a = \sum_j x_j \quad b = \sum_j \ln(x_j) \quad y_i = \cos\left(\frac{e^{x_i} a}{b}\right) \]
Influence Diagrams

A little painful, but algorithmic:

Break things up

\[ a = \sum_j x_j \quad b = \sum_j \ln(x_j) \quad y_i = \cos\left(\frac{e^{x_i} a}{b}\right) \]

Draw the influence diagram

[Diagram of influence diagram]
Influence Diagrams

A little painful, but algorithmic:

**Break things up**

\[ a = \sum_j x_j \quad b = \sum_j \ln(x_j) \quad y_i = \cos\left(\frac{e^{x_i} a}{b}\right) \]

**Draw the influence diagram**

![Influence Diagram](image)

**Write out paths using the diagram / chain rule**

\[
\nabla_{x_i} L = \frac{dy_i}{dx_i} \frac{dL}{dy_i} + \frac{da}{dx_i} \frac{dy_i}{dL} + \frac{db}{dx_i} \frac{dy_i}{dL} + \frac{dL}{dx_i} \frac{da}{dy_i} + \frac{dL}{dx_i} \frac{db}{dy_i} + \frac{dL}{dx_i} \frac{dy_2}{dy_i} + \frac{dL}{dx_i} \frac{dy_3}{dy_i} + \frac{dL}{dx_i} \frac{dy_2}{dy_3} + \frac{dL}{dx_i} \frac{dy_3}{dy_3}
\]
Influence Diagrams

A little painful, but algorithmic:

**Break things up**

\[ a = \sum_j x_j \quad b = \sum_j \ln(x_j) \quad y_i = \cos\left(\frac{e^{x_i}}{b}\right) \]

**Draw the influence diagram**

![Influence Diagram](image)

**Write out paths using the diagram / chain rule**

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dy_1}{da} + \frac{db}{dx_1} \frac{dy_1}{db} + \frac{da}{dx_1} \frac{dy_2}{da} + \frac{db}{dx_1} \frac{dy_2}{db} + \frac{da}{dx_1} \frac{dy_3}{da} + \frac{db}{dx_1} \frac{dy_3}{db} \]

**Calculate necessary derivatives**

\[ \frac{dy_1}{dx_1} = -\sin\left(\frac{e^{x_1}}{b}\right) \quad \frac{dy_2}{dx_1} = -\sin\left(\frac{e^{x_1}}{b}\right) \frac{x_1 e^{x_1}}{b} \quad \frac{db}{dx_1} = \frac{1}{x_1} \quad \frac{dy_3}{db} = \sin\left(\frac{e^{x_1}}{b}\right) \frac{x_1 e^{x_1}}{b^2} \quad \text{etc...} \]
Influence Diagrams

A little painful, but algorithmic:

**Break things up**

\[ a = \sum_j x_j \quad b = \sum_j \ln(x_j) \quad y_i = \cos\left(\frac{e^{x_i} a}{b}\right) \]

**Draw the influence diagram**

![Influence Diagram](image)

**Write out paths using the diagram / chain rule**

\[ \nabla_{x_1 L} = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{dy_1}{dx_1} \frac{da}{dy_1} + \frac{dy_1}{dx_1} \frac{db}{dy_1} + \frac{dy_2}{dx_1} \frac{dL}{dy_2} + \frac{dy_2}{dx_1} \frac{da}{dy_2} + \frac{dy_2}{dx_1} \frac{db}{dy_2} + \frac{dy_3}{dx_1} \frac{dL}{dy_3} + \frac{dy_3}{dx_1} \frac{da}{dy_3} + \frac{dy_3}{dx_1} \frac{db}{dy_3} \]

**Calculate necessary derivatives**

\[ \frac{dy_1}{dx_1} = -\sin\left(\frac{e^{x_1} a}{b}\right) \frac{dx_1}{e^{x_1} a} \quad \frac{da}{dx_1} = 1 \quad \frac{dy_1}{dx_1} = -\sin\left(\frac{e^{x_1} a}{b}\right) \frac{x_1 e^{x_1} a}{b} \quad \frac{db}{dx_1} = \frac{1}{x_1} \quad \frac{dy_1}{db} = \sin\left(\frac{e^{x_1} a}{b}\right) \frac{x_1 e^{x_1} a}{b} \quad \text{etc...} \]

**Plug things in / simplify**

A mess 😊
Influence Diagrams

A little painful, but algorithmic:

Break things up

\[ a = \sum_j x_j \quad b = \sum_j \ln(x_j) \quad y_i = \cos\left(\frac{e^{x_i} a}{b}\right) \]

Draw the influence diagram

Write out paths using the diagram / chain rule

\[ \nabla_{x_1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1} + \frac{da}{dx_1} \frac{dL}{da} + \frac{db}{dx_1} \frac{dL}{db} + \frac{dy_2}{dx_1} \frac{dL}{dy_2} + \frac{da}{dx_1} \frac{dL}{da} + \frac{dy_3}{dx_1} \frac{dL}{dy_3} + \frac{db}{dx_1} \frac{dL}{db} \]

Calculate necessary derivatives

\[ \frac{dy_1}{dx_1} = -\sin\left(\frac{e^{x_1} a}{b}\right) \quad \frac{da}{dx_1} = 1 \quad \frac{dy_2}{dx_1} = \sin\left(\frac{e^{x_1} a}{b}\right) \quad \frac{db}{dx_1} = \frac{1}{x_1} \quad \frac{dy_3}{dx_1} = \sin\left(\frac{e^{x_1} a}{b}\right) \frac{x_1 e^{x_1} a}{b^2} \quad \text{etc...} \]

Plug things in / simplify

A mess 😊

Questions?

What isn’t clear?
Computational Graphs

Feel free to follow the backprop part on paper.
Simple MLP

An MLP with one tanh activated hidden layer
We want to easily compute the derivative with respect to the weights $W_i$ and biases $b_i$. 

Simple MLP
Simple MLP

<table>
<thead>
<tr>
<th>Linear</th>
<th>( z = W_1 x + b_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation</td>
<td>( \text{out} = \tanh(z) )</td>
</tr>
</tbody>
</table>
Simple MLP

\[ z = W_1 x + b_1 \]
\[ \text{out} = \tanh(z) \]

Let’s unravel these equations into **unary** and **binary** operations (one or two arguments only)
Simple MLP

<table>
<thead>
<tr>
<th>Linear</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_1 = W_1 x$</td>
<td></td>
</tr>
<tr>
<td>$z_2 = z_1 + b_1$</td>
<td></td>
</tr>
<tr>
<td><strong>Activation</strong></td>
<td></td>
</tr>
<tr>
<td>out = tanh($z_2$)</td>
<td></td>
</tr>
</tbody>
</table>
This allows us to reuse rules for propagating derivatives through simple functions like +, *
Simple MLP

\[ z_1 = W_1 x \]
\[ z_2 = z_1 + b_1 \]
\[ \text{out} = \tanh(z_2) \]

Now let’s step through this to create a **computational graph** (forward pass)
Simple MLP

\[ z_1 = W_1 x \]
\[ z_2 = z_1 + b_1 \]
\[ \text{out} = \tanh(z_2) \]
Simple MLP

\[ z_1 = W_1 x \]
\[ z_2 = z_1 + b_1 \]
\[ \text{out} = \tanh(z_2) \]
Simple MLP

\[ z_1 = W_1 x \]
\[ z_2 = z_1 + b_1 \]
\[ \text{out} = \tanh(z_2) \]
Simple MLP

\[ z_1 = W_1 x \]
\[ z_2 = z_1 + b_1 \]
\[ \text{out} = \tanh(z_2) \]
Simple MLP

\[ z_1 = W_1 x \]
\[ z_2 = z_1 + b_1 \]
\[ \text{out} = \tanh(z_2) \]
Simple MLP

\[ z_1 = W_1 x \]
\[ z_2 = z_1 + b_1 \]
\[ \text{out} = \tanh(z_2) \]

Derivative \( \frac{dL}{da} \)

Questions?
What isn’t clear?
Simple MLP

Derivative $dL/da$  Variable $a$  Operation

$W_1$  $b_1$  $\text{tanh}$  $\text{out}$

$x$  $\text{mmult}$  $z_1$  $\text{add}$  $z_2$
Simple MLP

Linear (just for reference)

- $x$ (Input)
- $W_1$ (Weight)
- $b_1$ (Bias)
- $z_1$ (Activation of Input Layer)
- $z_2$ (Activation of Hidden Layer)
- $a$ (Output of Hidden Layer)
- $\text{tanh}$ (Tanh activation function)

Derivative $\frac{dL}{da}$

Variables:
- $a$: Output variable

Operations:
- Linear (just for reference)
- Add
- Mult
- Tanh
Simple MLP

\[
\begin{align*}
\text{Derivative } dL/da & \quad \text{Variable } a \\
\text{Operation} & \quad \nabla_a L
\end{align*}
\]
Simple MLP

\[ \frac{\partial L}{\partial a} \]

is the derivative of the loss function with respect to the output.
Simple MLP

\[ a \text{ is calculated from our chosen loss function.} \]

For simplicity, this example does not have the loss computation in the graph.

\[ \nabla_{out} L \]
Simple MLP
Aside: Backward Functions

\[ \nabla_a L \quad \text{add} \quad \nabla_b L \quad \nabla_c L \]

Derivative \( dL/da \)  
Variable \( a \)  
Operation
Aside: Backward Functions

With the chain rule, every operation has a **backward function** to calculate its parent’s gradients.
Aside: Backward Functions

Given $\nabla_c L$, what are $\nabla_a L$ and $\nabla_a L$ in terms of the chain rule?
Aside: Backward Functions

Given $\nabla_c L$, what are $\nabla_a L$ and $\nabla_a L$ in terms of the chain rule?

$$\frac{dL}{da} = \frac{dL}{dc} \cdot \frac{dc}{da}$$
Aside: Backward Functions

Given \( \nabla cL \), what are \( \nabla aL \) and \( \nabla aL \) in terms of the chain rule?

\[
\frac{dL}{da} = \frac{dL}{dc} \cdot \frac{dc}{da} \quad \text{and} \quad \frac{dL}{db} = \frac{dL}{dc} \cdot \frac{dc}{db}
\]
Aside: Backward Functions

What do these simplify to?  
Hint: $c = a + b$, what's $dc/da$

\[
\frac{dL}{da} = \frac{dL}{dc} \times \frac{dc}{da} \quad \frac{dL}{db} = \frac{dL}{dc} \times \frac{dc}{db}
\]
Aside: Backward Functions

What do these simplify to?  
Hint: c = a + b, what's dc/da

\[
\frac{dL}{da} = \frac{dL}{dc} \times \frac{dc}{da} = \frac{dL}{dc} \quad \frac{dL}{db} = \frac{dL}{dc} \times \frac{dc}{db} = \frac{dL}{dc}
\]
Add’s **backward function** is to pass the gradient back unchanged
Aside: Backward Functions

Add’s **backward function** is to pass the gradient back unchanged

**Questions?**
**What isn’t clear?**
Simple MLP

\[ \begin{align*}
\text{Derivative } dL/da & \quad \text{Variable } a \\
\text{Operation } \odot & \quad \text{Derivative } dL/da \\
\end{align*} \]
We will perform a graph search from the end, updating derivatives as we go. DFS is easiest.
Simple MLP

Feel free to follow along on paper.

Diagram:
- Derivative \( \frac{dL}{da} \)
- Variable \( a \)
- Operation
- \( W1 \)
- \( b1 \)
- \( z1 \)
- \( z2 \)
- \( \text{mmult} \)
- \( \text{add} \)
- \( \text{tanh} \)
- \( \text{out} \)
- \( \nabla_{\text{out}} L \)
Simple MLP

If the output, $\mathbf{tanh} \mathbf{out}$, is $N \times 1$, what is the shape of $\nabla_{out} L$?
Simple MLP

If the output, $\text{tanh} \ out$ is $Nx1$, what is the shape of $\nabla_{out} L$?

If $A$ has shape $BxC$, gradients w.r.t. $A$ have the **transpose shape** $CxB$

* shape isn’t transpose for hw1p1, pytorch
In terms of the chain rule, what is the backward function of \( \tanh \text{out} \)? I.e., in terms of the chain rule, what is \( \nabla_{z_2} L \)?
In terms of the chain rule, what is the backward function of $\tanh\ out$?

I.e., in terms of the chain rule, what is $\nabla_{z_2}L$?

$$\frac{dL}{dz_2} = \frac{dL}{dout} \cdot \frac{dout}{dz_2}$$
In terms of the chain rule, what is the backward function of $\text{tanh}_{\text{out}}$? I.e., in terms of the chain rule, what is $\nabla_{z_2}L$?

\[
\frac{dL}{dz_2} = \frac{dL}{d\text{out}} \cdot \frac{d\text{out}}{dz_2} = \frac{dL}{d\text{out}} \cdot \text{tanh}'(z_2)
\]
Simple MLP

What is the shape of $\nabla_{z_2} L$?
Simple MLP

What is the shape of $\nabla_{z_2} L$?

The transpose shape

*except in pytorch, hw1p1
Simple MLP

Derivative $dL/da$

Variable $a$

Operation $a \text{op} \nabla a L$

$\text{mmult} z1$

$\text{add} z2$

$tanh out$

$\nabla_{out} L$

$\nabla_{out} L \cdot tanh'(z2) 1xN$

$\nabla_{out} L \cdot 1xN$

$\nabla_{a} L$
Simple MLP
Simple MLP

We will continue the graph search by visiting add z2.
Simple MLP

What is the backward function of \( \text{add} \) \( z_2 \)?
I.e., what are \( \nabla_{b_1 L} \) and \( \nabla_{z_1 L} \)?

Hint: \( c = a + b \), \[ \frac{dL}{da} = \frac{dL}{dc} \frac{dc}{da} \]
Simple MLP

What is the backward function of $\text{add}$? I.e., what are $\nabla_{b1}L$ and $\nabla_{z1}L$?

Hint: $c = a + b$, $\frac{dl}{da} = \frac{dl}{dc} \frac{dc}{da}$
Simple MLP

What is the backward function of \(\text{add}\) ?
I.e., what are \(\nabla_{b_1} L\) and \(\nabla_{z_1} L\) ?

Hint: \(c = a + b\), \(\frac{dL}{da} = \frac{dL}{dc} \frac{dc}{da}\)

\[
\begin{align*}
\frac{dL}{db_1} &= \frac{dL}{dz_2}, \\
\frac{dL}{dz_1} &= \frac{dL}{dz_2}
\end{align*}
\]
What are the shapes of $\nabla_{b1} L$ and $\nabla_{z1} L$?
Simple MLP

What are the shapes of $\nabla_{b_1} L$ and $\nabla_{z_1} L$?
Simple MLP

Derivative $dL/da$

Variable $a$

Operation $a_{op}$

$\nabla \alpha L$

$\nabla z_2 L$

$\nabla \alpha L \cdot \tanh'(z_2)$

$1 \times N$

$x$

$W_1$

$z_1$

$add z_2$

$\nabla _{z_2} L$

$\nabla _{z_2} L$

$\nabla _{z_2} L$

$b_1$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$

$1 \times N$
Simple MLP

We will continue the graph search by visiting $b_1$. 
Simple MLP

\[ b1 \] has no gradient-enabled parents, and we want its gradient, so its backward function is to accumulate (i.e. save) the gradient passed to it.
Simple MLP

We will continue the graph search by visiting \texttt{mmult} \texttt{z1}.
What is the backward function of $\text{mmult} z1$? I.e., what are $\nabla_{W1} L$ and $\nabla_x L$?
Simple MLP

What is the backward function of $z_1$? I.e., what are $\nabla_{w_1} L$ and $\nabla_x L$?

Given matrix $\nabla_{AB} L$:

$$\nabla_A L = B \nabla_{AB} L$$
$$\nabla_B L = \nabla_{AB} L A$$

confirm for yourself!
Simple MLP

What is the backward function of \( z_1 \)?

I.e., what are \( \nabla_{w_1} L \) and \( \nabla_x L \)?

Given matrix \( \nabla_{AB} L \):

\[
\nabla_A L = B \nabla_{AB} L
\]
\[
\nabla_B L = \nabla_{AB} L A
\]

confirm for yourself!
What are the shapes of $\nabla_{\text{w1}} L$ and $\nabla_{\text{x}} L$?
Simple MLP

What are the shapes of $\nabla_{w_1} L$ and $\nabla_x L$?

... transpose except in hw1p1, pytorch ...

\[ \begin{align*}
W_1 & \quad \text{NxM} \\
x & \quad \text{Mx1} \\
\text{mmult} & \quad \text{z1} \\
\nabla_{z1} L & \quad \text{1xM} \\
\nabla_{z2} L & \quad \text{1xN} \\
\text{add} & \quad \text{z2}
\end{align*} \]
Simple MLP

We will continue the graph search by visiting $W_1$. 
Simple MLP

$W_1$ has no gradient-enabled parents, and we want its gradient, so its backward function is to **accumulate** (i.e. save) the gradient passed to it.
We will continue the graph search by visiting $x$.
Simple MLP

$x$ has no gradient-enabled parents, and we don’t care about its gradient, so we do nothing.
Simple MLP

\[
\begin{align*}
\text{Derivative } \frac{dL}{da} & = \nabla_{a} L \\
\text{Variable } a & = \text{Operation } a \text{ op } \nabla a L \\
\text{mmult } z1 & = \text{add } z2 \\
\text{tanh out} & = \nabla_{\text{out } L} \times \text{tanh}'(z2) \\
\end{align*}
\]
Simple MLP

\[ \nabla_{\text{out}} L \]  

1xN  

from loss function

\[ \nabla_{z_2} L \]  

1xM  

\[ = \nabla_{z_3} L \tanh'(z_2) \]

\[ \nabla_{b_1} L \]  

1xM  

\[ = \nabla_{z_2} L \]

\[ \nabla_{z_1} L \]  

1xM  

\[ = \nabla_{z_2} L \]

\[ \nabla_{W_1} L \]  

PxM  

\[ = x \nabla_{z_1} L \]

\[ \nabla_{x} L \]  

1xP  

\[ = \nabla_{z_1} L W_1 \]
Simple MLP

We have gradients for nodes that accumulated (i.e. saved) them: x, W1, b1, W2, b2
What about reusing parameters or intermediate variables?
Shared Parameter Networks (Scanning MLP)

The scanning MLP “scans” across some input
Shared Parameter Networks (Scanning MLP)

The scanning MLP “scans” across some input
Shared Parameter Networks (Scanning MLP)

The scanning MLP “scans” across some input
The scanning MLP “scans” across some input
Shared Parameter Networks (Scanning MLP)

The scanning MLP “scans” across some input
Shared Parameter Networks (Scanning MLP)

One big network with shared parameters
Shared Parameter Networks (Scanning MLP)

One big network with shared parameters

Let’s create the graph...
Shared Parameter Networks (Scanning MLP)

for position in input:
    MLP(position)
Shared Parameter Networks (Scanning MLP)

for position in input:
MLP(position)

Our initial variables
Shared Parameter Networks (Scanning MLP)

for position in input:
   MLP(position)
Shared Parameter Networks (Scanning MLP)

for position in input:
    MLP(position)
for position in input: MLP(position)
Shared Parameter Networks (Scanning MLP)

for position in input:
MLP(position)
for position in input:
MLP(position)
for position in input:
    MLP(position)
Shared Parameter Networks (Scanning MLP)

for position in input: MLP(position)
for position in input:
    MLP(position)
Shared Parameter Networks (Scanning MLP)
Shared Parameter Networks (Scanning MLP)

Nodes can have multiple avenues of influence
Shared Parameter Networks (Scanning MLP)

Nodes can have multiple avenues of influence

Gradient accumulation is especially important...
Shared Parameter Networks (Scanning MLP)

Let’s DFS...

- $x_1$ to $\text{mmult } z_1$ to $\text{add } z_2$ to $\text{tanh } z_3$
- $x_2$ to $\text{mmult } z_4$ to $\text{add } z_5$ to $\text{tanh } z_6$
- $W_1$, $b_1$
Shared Parameter Networks (Scanning MLP)

Let’s DFS...

\[
\nabla z_3 L
\]

\[
W_1
\]

\[
b_1
\]
Let’s DFS...

\[ \nabla z_2 L \]

\[ \nabla z_3 L \]
Let’s DFS...

Not quite...
Let’s DFS...

Not quite...
Let’s DFS...

Not quite...
Let’s DFS...

\[
\nabla b_1 L = \text{sum}( \nabla b_1 L )
\]
Let’s DFS...

\[ \nabla b_1 = \text{sum}(\nabla b_1) \\
\nabla z_{13} = \text{add} z_2 \\
\nabla z_{13} = \text{tanh} z_3 \\
\nabla z_{13} = \text{add} z_5 \\
\nabla z_{13} = \text{tanh} z_6 \\
\]

When gradients are **accumulated**, we use “+=” to save them!
Accumulating Derivatives

• Derivatives are initialized to 0 or None
• When we visit a node, we always use “+=“ to update the derivative
Accumulating Derivatives

- Derivatives are initialized to 0 or None
- When we visit a node, we always use “+=“ to update the derivative

The rest of the scanning MLP example is nothing new
We can apply this process to any function made up of smaller differentiable functions.
What is this called?

- We create a graph of operations
- We graph search from known gradients
- We accumulate gradients
- We utilize reusable, differentiable operations
What is this called?

• We create a graph of operations
• We graph search from known gradients
• We accumulate gradients
• We utilize reusable, differentiable operations

Autograd
Autograd

- Pytorch builds an implicit graph when you perform operations (also hw1p1)
  - +, -, *, /
  - Batchnorm, Softmax...

- You can also build this graph on paper to calculate derivatives
As an example, we’ll show the graph for a ray tracer for 4x3 images
As an example, we’ll show the graph for a ray tracer for 4x3 images

Note that it has no learnable parameters
If we wanted, we could optimize over the inputs of the ray tracer
If we wanted, we could optimize over the inputs of the ray tracer.

We could also calculate derivatives by hand! The functions are simple...

.1 pts extra credit!