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.

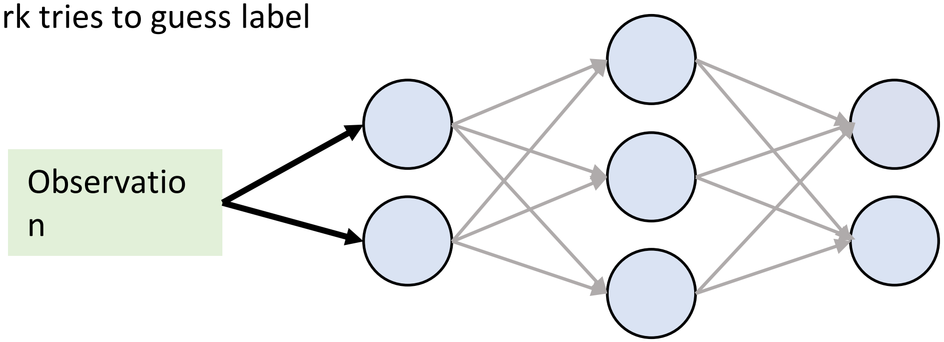
Goal: Minimize the loss by adjusting network parameters

- 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



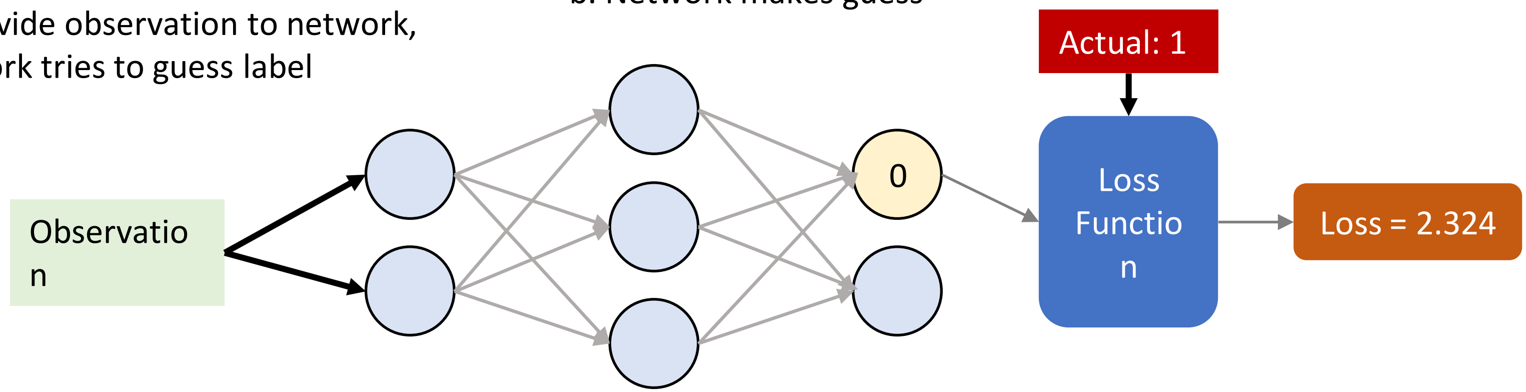
Training a Neural Network

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.



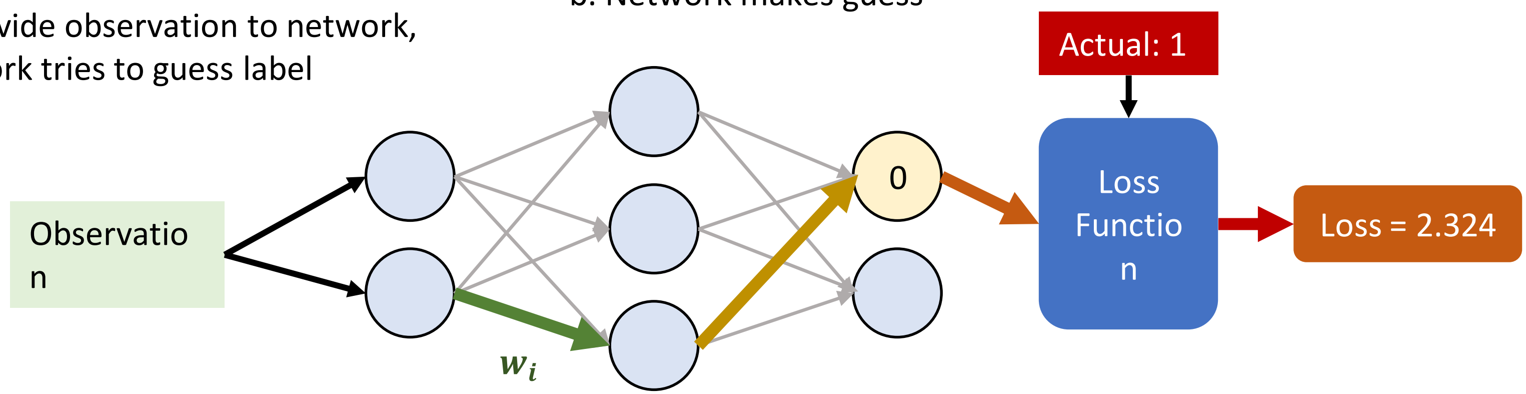
Training a Neural Network

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.



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

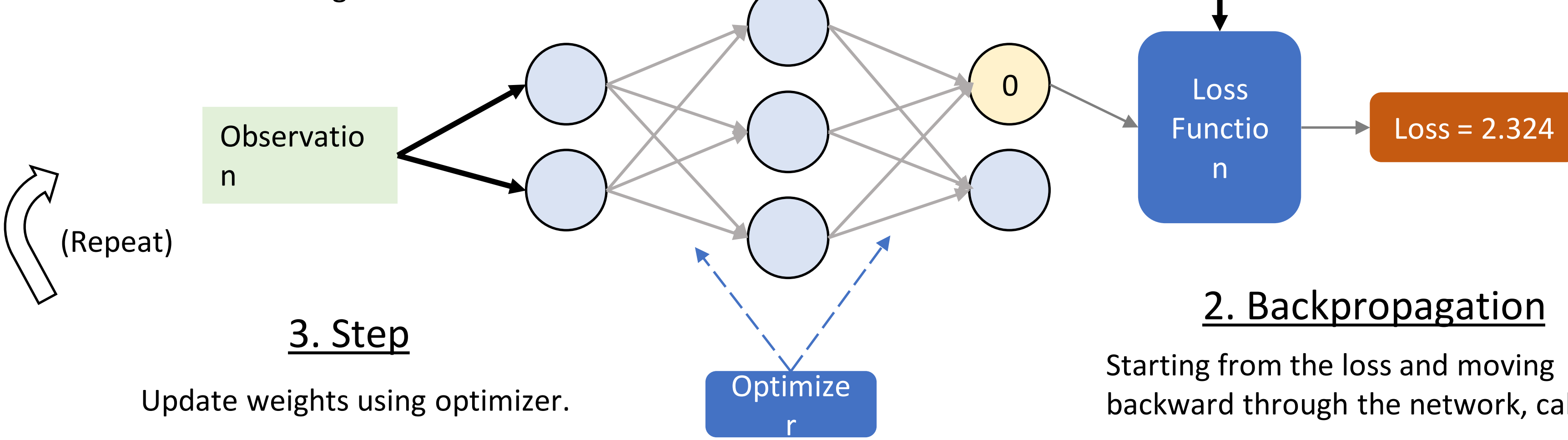
1. Forward Propagation

a. Provide observation to network, network tries to guess label

b. Network makes guess

c. "Score" performance by generating a loss value.

Actual: 1



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.

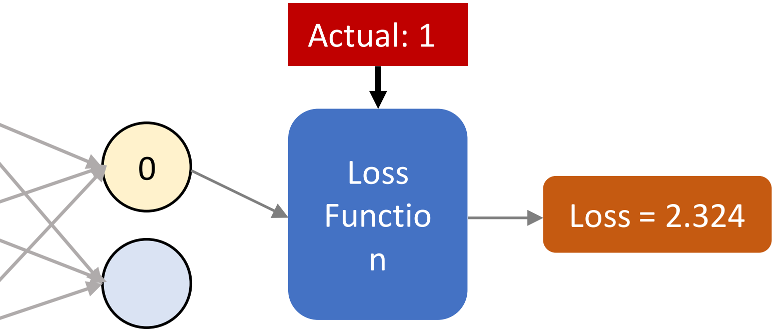
3. Step

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

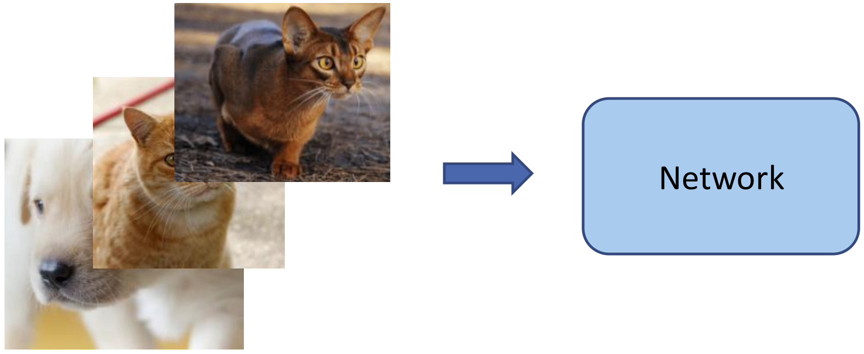


- **Really important in ML and optimization**
- General metric for evaluating performance
- Minimizing an (appropriate) loss metric should cause improved performance

Example: CrossEntropyLoss

- Task: classifying dogs and cats

Observations (batch of 3)



Example: CrossEntropyLoss

- Task: classifying dogs and cats

Observations (batch of 3)

Network

Logits (label confidence)

	Cat	Dog
Img_1	1.01	**1.2**
Img_2	**12.5**	0.02
Img_3	0.23	**1.35**

Size: (batch_size, num_possible_labels)

Example: CrossEntropyLoss

- Task: classifying dogs and cats

Observations (batch of 3)

Network

Logits (label confidence)

	Cat	Dog
Img_1	1.01	**1.2**
Img_2	**12.5**	0.02
Img_3	0.23	**1.35**

Notice:
incorrect guess

Size: (batch_size, num_possible_labels)

Example: CrossEntropyLoss

Logits (label confidence)

	Cat	Dog
Img_1	1.01	**1.2**
Img_2	**12.5**	0.02
Img_3	0.23	**1.35**

True labels

0	**Cat**
0	**Cat**
1	**Dog**

Size: (batch_size,)

Example: CrossEntropyLoss

Logits (label confidence)

	Cat	Dog
Img_1	1.01	**1.2**
Img_2	**12.5**	0.02
Img_3	0.23	**1.35**

True labels

0	**Cat**
0	**Cat**
1	**Dog**

CrossEntropyLoss

Example: CrossEntropyLoss

Logits (label confidence)

	Cat	Dog
Img_1	1.01	1.2
Img_2	12.5	0.02
Img_3	0.23	1.35

True labels

0	**Cat**
0	**Cat**
1	**Dog**

CrossEntropyLoss

0.3583	**Average Loss Value for Batch**

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

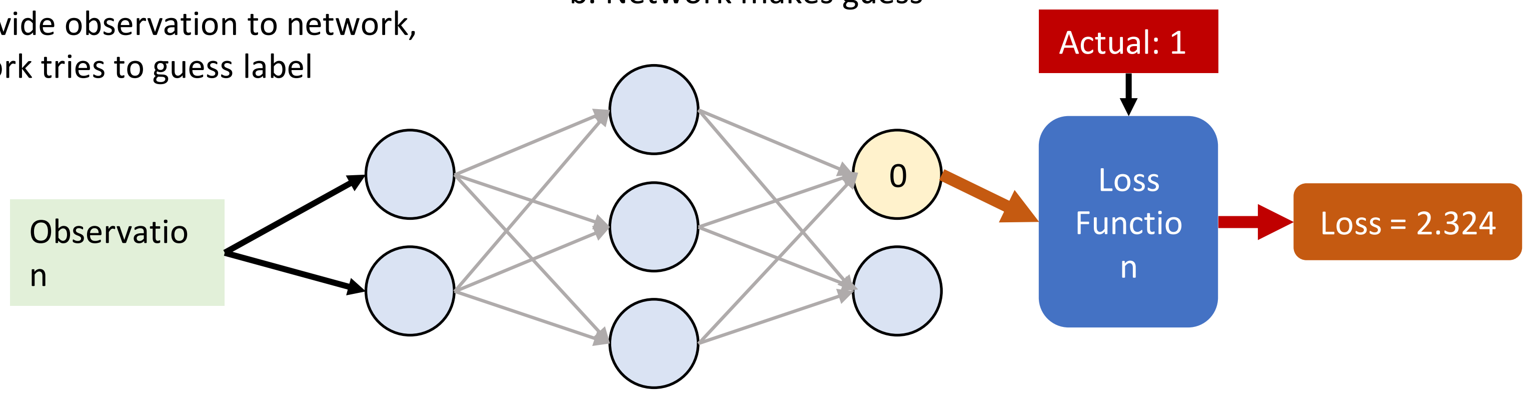
So far:

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.



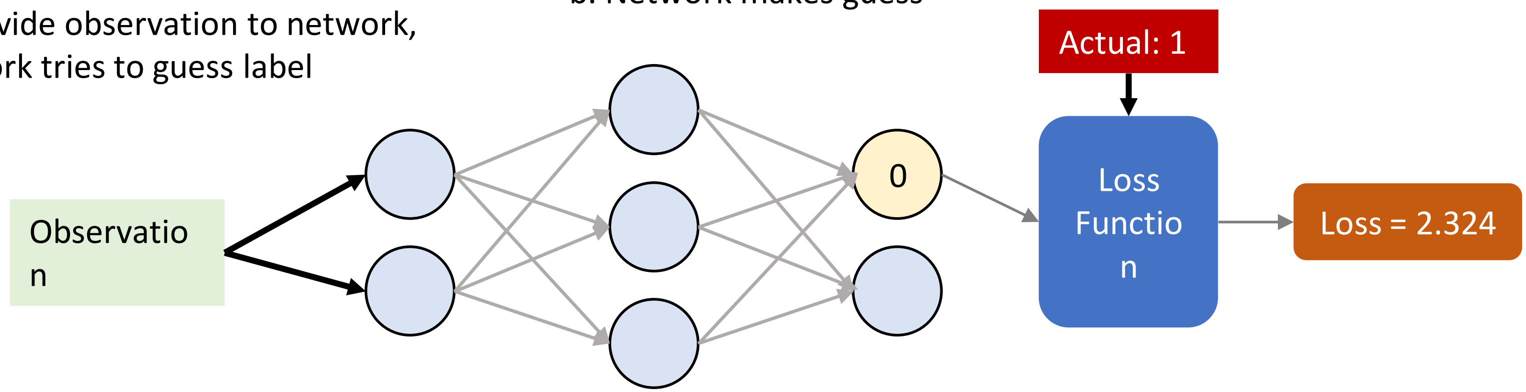
So far:

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.



3. Step

Adjust weights using those gradients

2. Backpropagation

Determine how each weight affects the loss by calculating partial derivative

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 (α definition)

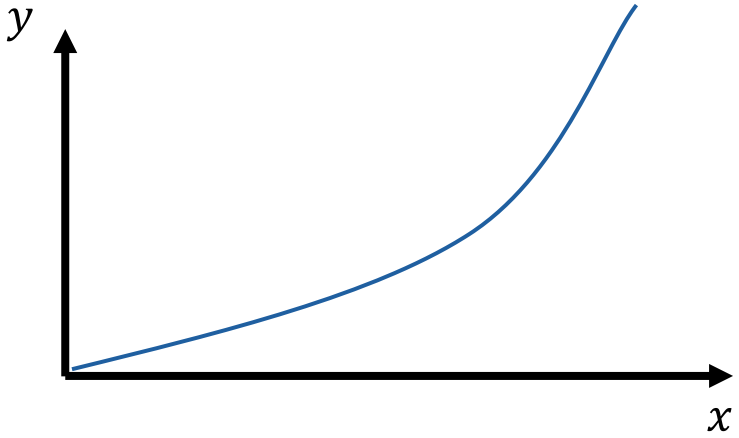
$$f(x) = y$$

x and y are scalars

Scalar Derivatives (α definition)

$$f(x) = y$$

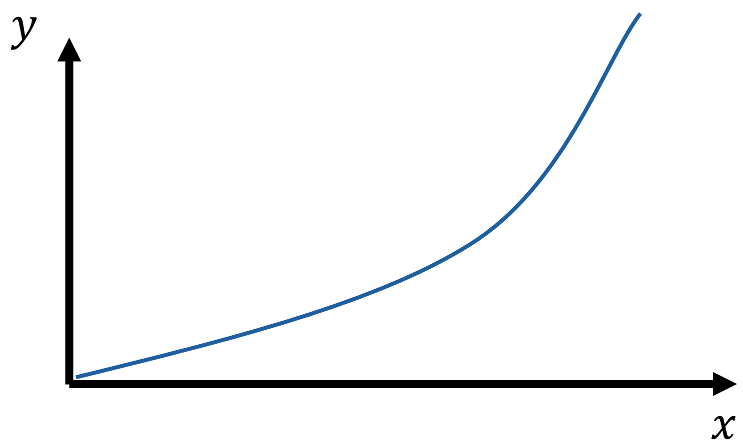
x and y are scalars



Scalar Derivatives (α definition)

$$f(x) = y$$

x and y are scalars

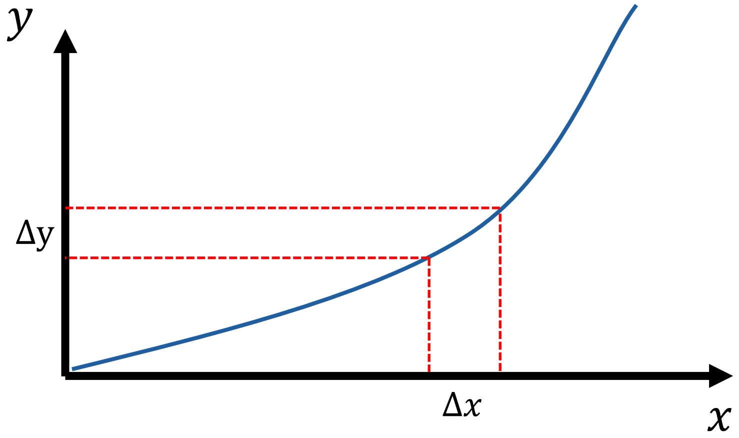


Goal: determine how changing the input affects the output

Scalar Derivatives (α definition)

$$f(x) = y$$

x and y are scalars



Goal: Find Δy given Δx

Scalar Derivatives (α definition)

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

$$\Delta y = \alpha \Delta x$$

- α is some factor multiplied to $\Delta \mathrm{x}$ that gives Δy

Scalar Derivatives (α definition)

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

Derivative $f'(x)$

$$\Delta y = \alpha \Delta x$$

- α is some factor multiplied to Δx that gives Δy.
- **Plot twist: $\boldsymbol{\alpha}$ is the derivative $\boldsymbol{f'(x)}$**

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 Δx onto Δy.

$$f'(x): \Delta x \longmapsto \Delta y$$

$$\mathbb{R}^1 \longmapsto \mathbb{R}^1$$

Derivatives (vector in, scalar out)

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

$$f(x_1, \dots, x_D) = y$$

Vector-scalar derivatives use the same general form as scalar-scalar derivatives.

To do this, group the input variables into a 1-D vector $\mathbf{x}$.

$$\Delta y = \boldsymbol{\alpha} \cdot \mathbf{x}$$

$$= \begin{bmatrix} a_1 & \dots & a_D \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_D \end{bmatrix}$$

Note: vectors are notated in bold and unitalicized font.

Derivatives (vector in, scalar out)

$$\Delta y = \boldsymbol{\alpha} \cdot \mathbf{x}$$

$$= \begin{bmatrix} a_1 & \dots & a_D \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_D \end{bmatrix}$$

Same thing below, but in more familiar notation:

$$\Delta y = \frac{\partial y}{\partial \mathbf{x}} \cdot \mathbf{x}$$

$$= \begin{bmatrix} \frac{\partial y}{\partial x_1} & \dots & \frac{\partial y}{\partial x_D} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_D \end{bmatrix}$$

Derivatives (vector in, scalar out)

$$\Delta y = \boldsymbol{\alpha} \cdot \mathbf{x}$$

$$= [a_1 \quad \dots \quad a_D] \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_D \end{bmatrix}$$

Same thing below, but in more familiar notation:

This is the derivative

(1 x D) row vector

$$\Delta y = \frac{\partial y}{\partial \mathbf{x}} \cdot \mathbf{x}$$

$$= \left[\frac{\partial y}{\partial x_1} \quad \dots \quad \frac{\partial y}{\partial x_D} \right] \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_D \end{bmatrix}$$

Derivatives (vector in, scalar out)

In summary, for a function of multiple arguments $\mathbf{x}$ and scalar output y

$$f(\boldsymbol{x}) = y$$

Its derivative is this:

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \cdots & \frac{\partial y}{\partial x_D} \end{bmatrix}$$

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:

$$\Delta y = \nabla_{\mathbf{x}} y \cdot \mathbf{x}$$

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

By notational convention:

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}\mathbf{b}^T$$

Derivatives (vector in, vector out)

- For a function that inputs and outputs vectors, $\nabla_{\mathbf{x}}\mathbf{y}$ is the **“Jacobian”**.

Input	**Output**

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_K \end{bmatrix} \quad \Rightarrow \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_K}{\partial x_1} & \cdots & \frac{\partial y_K}{\partial x_D} \end{bmatrix}$$

$\mathrm{D} \times 1$ $\quad$ $\mathrm{K} \times 1$ $\quad$ $\mathrm{K} \times \mathrm{D}$

Derivatives (vector in, vector out)

- For a function that inputs and outputs vectors, $\nabla_{\mathbf{x}}\mathbf{y}$ is the **"Jacobian"**.

Each row is a vector-scalar derivative

Input	**Output**
$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix}$	$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_K \end{bmatrix}$
$\mathrm{D} \times 1$	$\mathrm{K} \times 1$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \boxed{\frac{\partial y_1}{\partial x_1} \quad \cdots \quad \frac{\partial y_1}{\partial x_D}} \\ \vdots \quad \ddots \quad \vdots \\ \frac{\partial y_K}{\partial x_1} \quad \cdots \quad \frac{\partial y_K}{\partial x_D} \end{bmatrix}$$

$\mathrm{K} \times \mathrm{D}$

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 $\frac{\partial y}{\partial \mathbf{x}} = \left[\frac{\partial y}{\partial x_1} \quad \dots \quad \frac{\partial y}{\partial x_D}\right]$

3. Vector/vector derivative $\left(\frac{d\mathbf{y}}{d\mathbf{x}}\right)$

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**

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

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

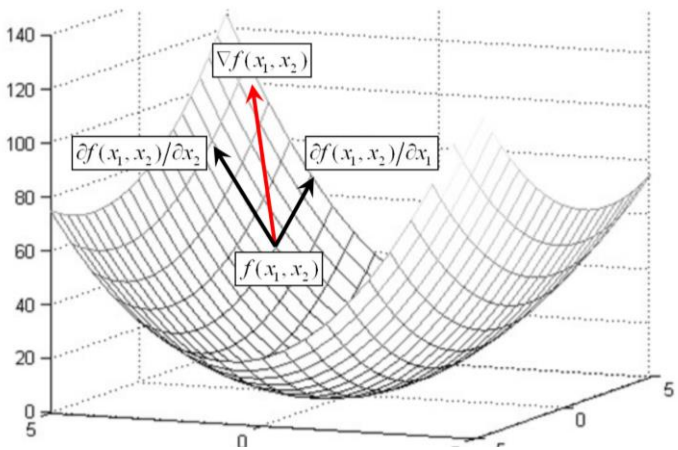
One difference: interpretation

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

- If **maximizing**, follow the gradient.
- If **minimizing**, go in the opposite direction (gradient descent)

Partial vs. Total Derivatives

$$\frac{dy}{d\mathbf{x}} \quad \text{vs} \quad \frac{\partial y}{\partial x_i}$$

- The total influence of $\mathbf{x}$ on y
- (Was today's topic, same as $\boldsymbol{\alpha}$ or ∇)

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

Remember before:

$$\frac{\partial y}{\partial \mathbf{x}} = \left[\frac{\partial y}{\partial x_1} \quad \dots \quad \frac{\partial y}{\partial x_D}\right]$$

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

Things get messy

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

Things get messy

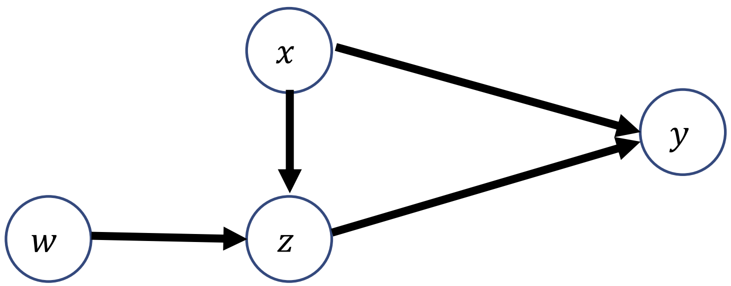
Find $\frac{dy}{dx}$ for $\boldsymbol{f(x, z) = y}$, where $\boldsymbol{z = g(x, w)}$

x affects y twice; directly in f, and indirectly through z.

Things get messy

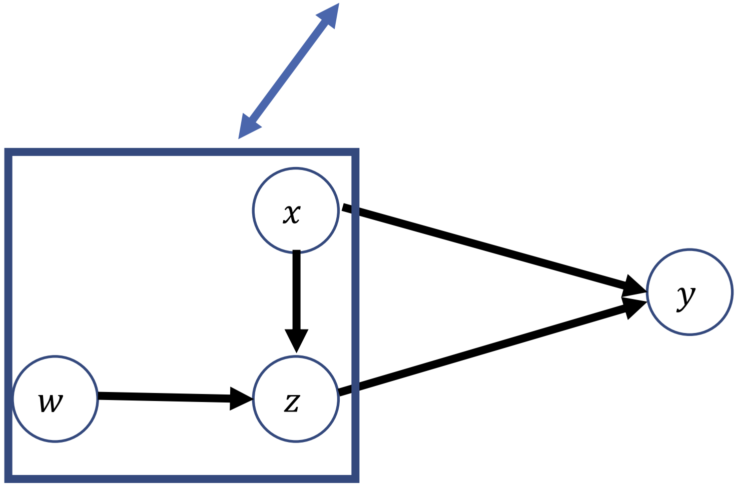
Find $\frac{dy}{dx}$ **for** $\boldsymbol{f(x,z) = y}$**, where** $\boldsymbol{z = g(x,w)}$

x affects y twice; directly in f, and indirectly through z.



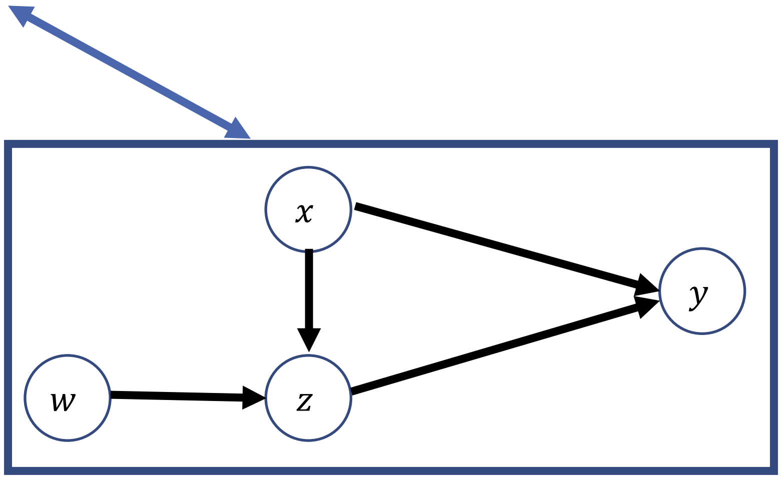
Things get messy

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



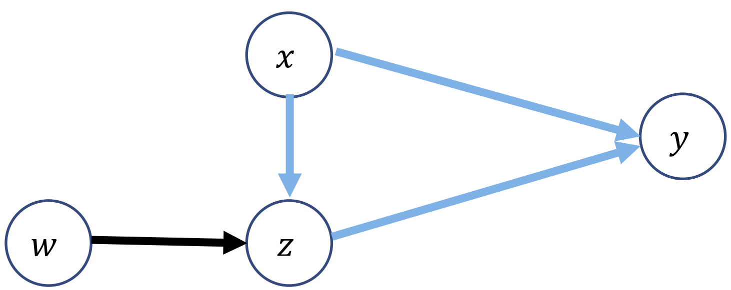
Things get messy

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



Things get messy

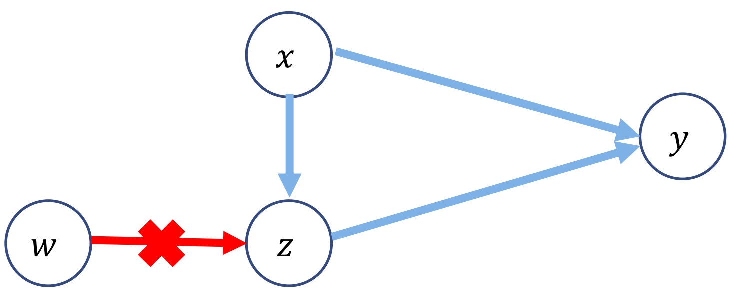
Find $\frac{dy}{dx}$ for $f(x, z) = y$, where $z = g(x, w)$



Goal: get only x's influence on y

Things get messy

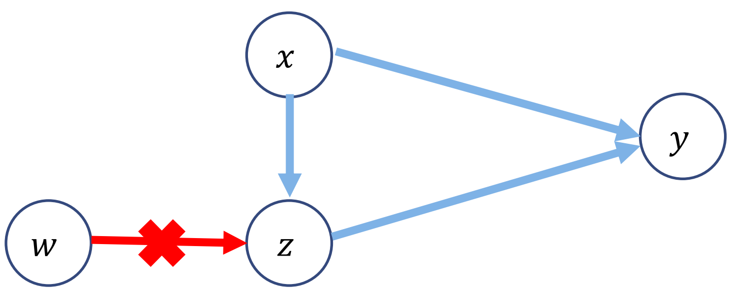
Find $\boldsymbol{\frac{dy}{dx}}$ for $\boldsymbol{f(x, z) = y}$, where $\boldsymbol{z = g(x, w)}$



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 $\boldsymbol{f(x,z) = y}$, where $\boldsymbol{z = g(x,w)}$



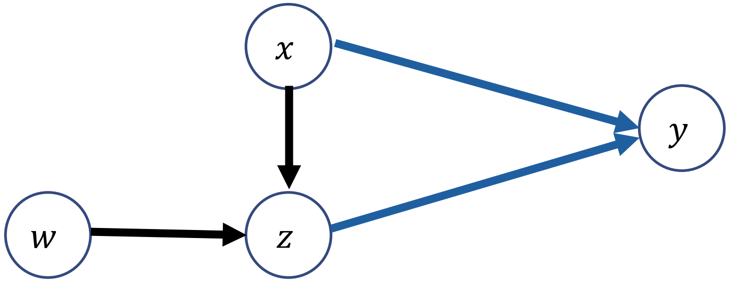
It's time for... "***<u>the chain rule</u>***"

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

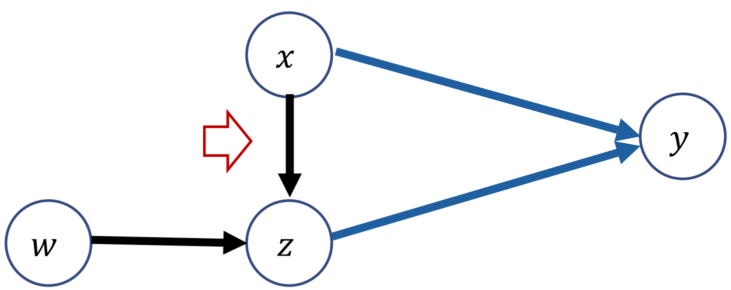
- The chain rule is used to properly **account for influences in nested functions**
  - Recursively calculates derivatives on nested functions w.r.t. target



$$\frac{dy}{dx} = \frac{\partial y}{\partial x} + \frac{\partial y}{\partial z}$$

The Chain Rule

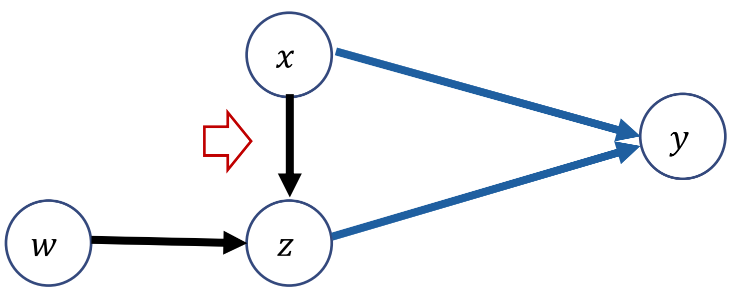
- The chain rule is used to properly **account for influences in nested functions**
  - Recursively calculates derivatives on nested functions w.r.t. target



$$\frac{dy}{dx} = \frac{\partial y}{\partial x} + \frac{\partial y}{\partial z} ?$$

The Chain Rule

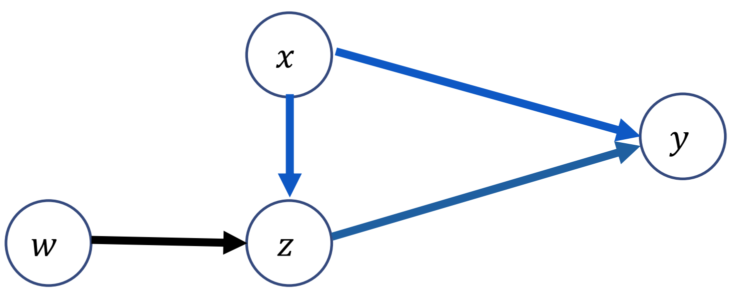
- The chain rule is used to properly **account for influences in nested functions**
  - Recursively calculates derivatives on nested functions w.r.t. target



$$\frac{dy}{dx} = \frac{\partial y}{\partial x} + \frac{\partial y}{\partial z}\frac{dz}{dx}$$

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

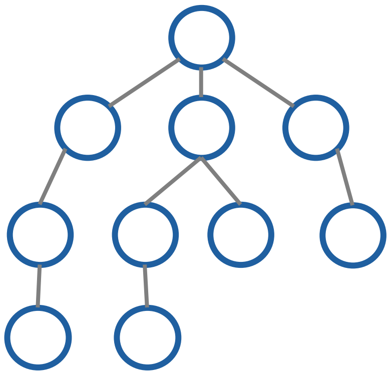


$$\frac{dy}{dx} = \frac{\partial y}{\partial x} + \frac{\partial y}{\partial z}\frac{dz}{dx}$$

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

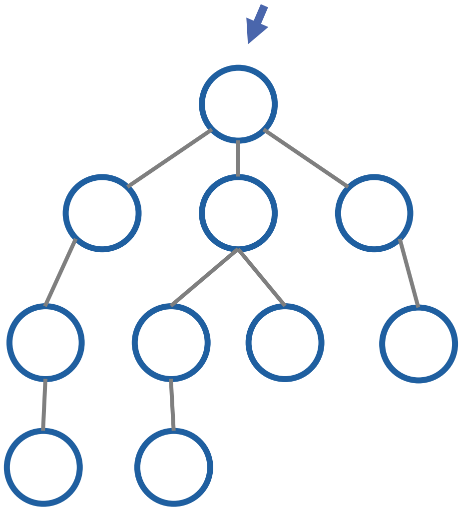


Example of a graph

Depth-First Search (DFS)

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

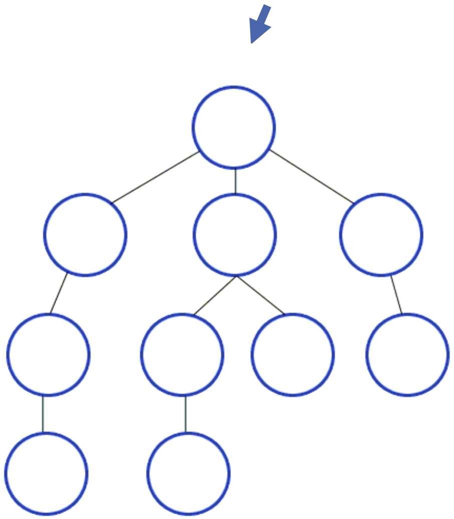
i.e. Start from here



Depth-First Search (DFS)

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

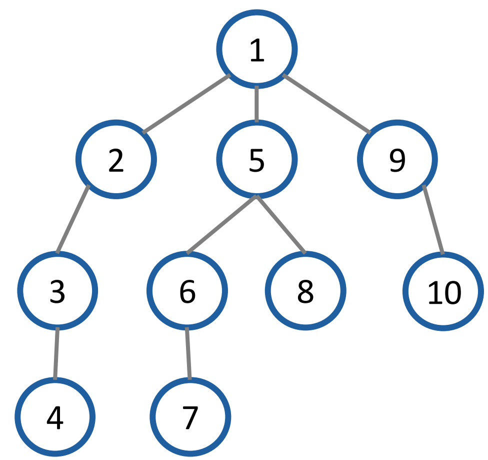
i.e. Start from here



[(Animated GIF source)]

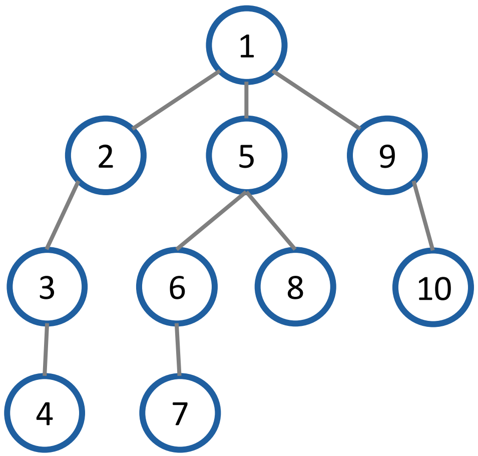
Depth-First Search (DFS)

- 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)

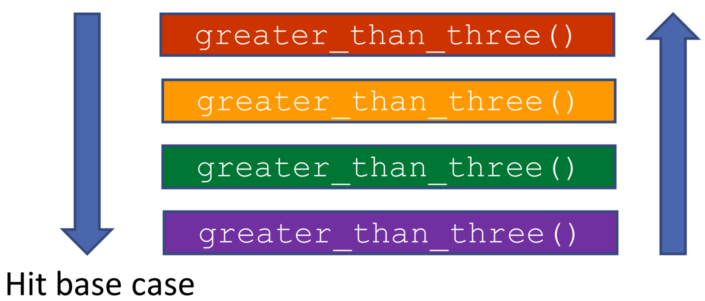
```
def greater_than_three(x):
        print("Recursive call, x=" + str(x))
        if x < 3:
                result = greater_than_three(x + 1)
                print("Received: x=" + str(result) + " and returning 
upward.")
                return result
        else:
                print("Hit base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

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
```



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]
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]
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

Rule	$f(x)$	**Scalar derivative notation with respect to** x	**Example**
Constant	c	0	$\frac{d}{dx}99 = 0$
Multiplication by constant	cf	$c\frac{df}{dx}$	$\frac{d}{dx}3x = 3$
Power Rule	x^n	nx^{n-1}	$\frac{d}{dx}x^3 = 3x^2$
Sum Rule	$f + g$	$\frac{df}{dx} + \frac{dg}{dx}$	$\frac{d}{dx}(x^2 + 3x) = 2x + 3$
Difference Rule	$f - g$	$\frac{df}{dx} - \frac{dg}{dx}$	$\frac{d}{dx}(x^2 - 3x) = 2x - 3$
Product Rule	fg	$f\frac{dg}{dx} + \frac{df}{dx}g$	$\frac{d}{dx}x^2x = x^2 + x2x = 3x^2$
Chain Rule	$f(g(x))$	$\frac{df(u)}{du}\frac{du}{dx}$, let $u = g(x)$	$\frac{d}{dx}ln(x^2) = \frac{1}{x^2}2x = \frac{2}{x}$

Table Source

Good Resources

[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:

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

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

Good Resources

[Stanford CS231N – Vector, Matrix, and Tensor Derivatives]

5 The chain rule in combination with vectors and matrices

Now that we have worked through a couple of basic examples, let's combine these ideas with an example of the chain rule. Again, assuming $\vec{y}$ and $\vec{x}$ are column vectors, let's start with the equation

$$\vec{y} = VW\vec{x},$$

and try to compute the derivative of $\vec{y}$ with respect to $\vec{x}$. We could simply observe that the product of two matrices V and W is simply another matrix, call it U, and therefore

$$\frac{d\vec{y}}{d\vec{x}} = VW = U.$$

Clear rules and examples of how to take matrix derivatives.

Good Resources

- https://en.wikipedia.org/wiki/Matrix_calculus
  - Another excellent reference; just be careful about notation
- Khan Academy’s article on gradients
  - **Simple/intuitive** visualizations and explanation
- https://en.wikipedia.org/wiki/Backpropagation
- https://en.wikipedia.org/wiki/Automatic_differentiation
- https://numpy.org/doc/stable/reference/routines.linalg.html
  - NumPy’s matrix operations documentation

Proof by Examples: Computing Derivatives Can be Trivial

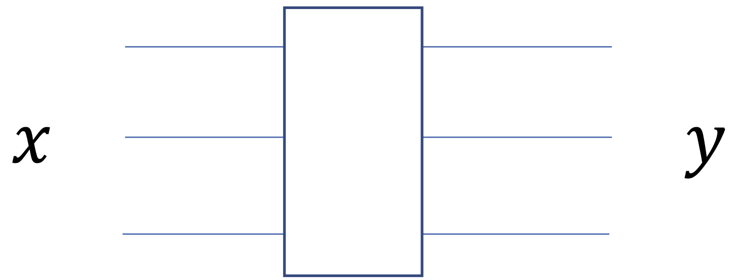
Influence Diagrams

A New (Made Up) Activation in Town

$$y_i = \cos(\frac{e^{x_i} \sum_j x_j}{\sum_j \ln(x_j)})$$

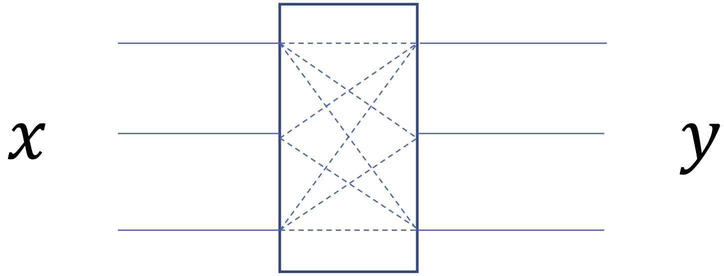
A New (Made Up) Activation in Town

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



A New (Made Up) Activation in Town

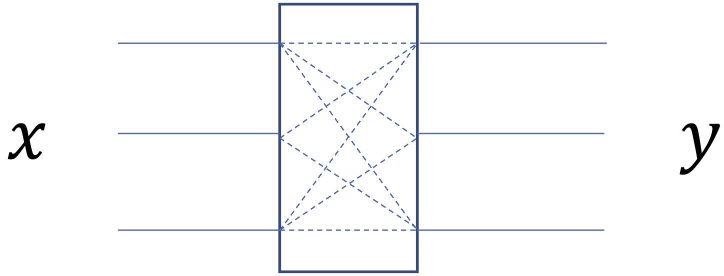
$$y_i = \cos\left(\frac{e^{x_i} \sum_j x_j}{\sum_j \ln(x_j)}\right)$$



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

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

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

$$a = \sum_j x_j$$

$$b = \sum_j \ln(x_j)$$

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

First we'll break things up so they're manageable…

A New (Made Up) Activation in Town

$$a = \sum_j x_j$$

$$b = \sum_j \ln(x_j)$$

$$y_i = \cos(\frac{e^{x_i\ a}}{b})$$

Now we'll draw the influence diagram...

A New (Made Up) Activation in Town

$$a = \sum_j x_j$$

$$b = \sum_j \ln(x_j)$$

$$y_i = \cos(\frac{e^{x_i \ a}}{b})$$

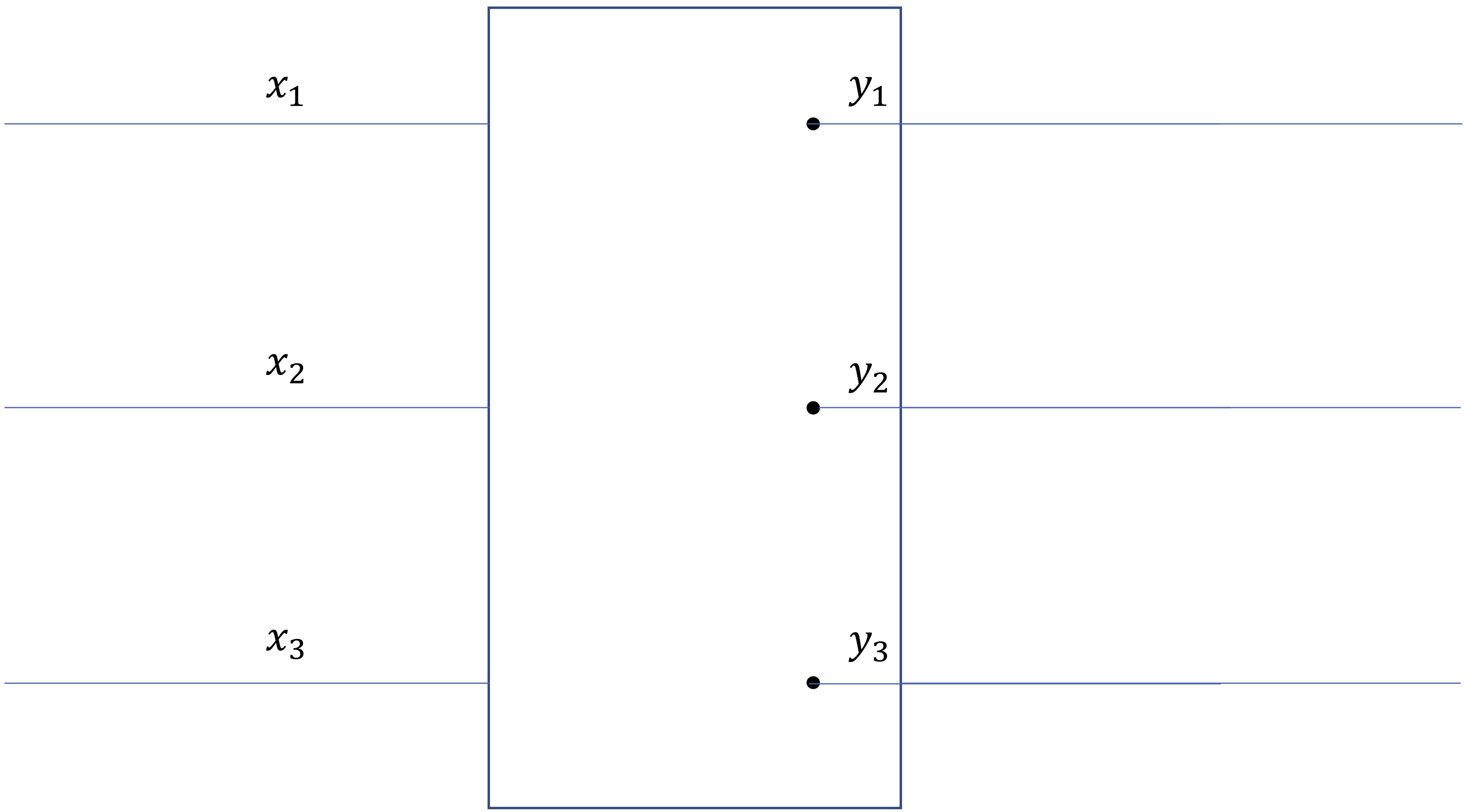
Now we'll draw the influence diagram…

A New (Made Up) Activation in Town

$$a = \sum_j x_j$$

$$b = \sum_j \ln(x_j)$$

$$y_i = \cos(\frac{e^{x_i \ a}}{b})$$

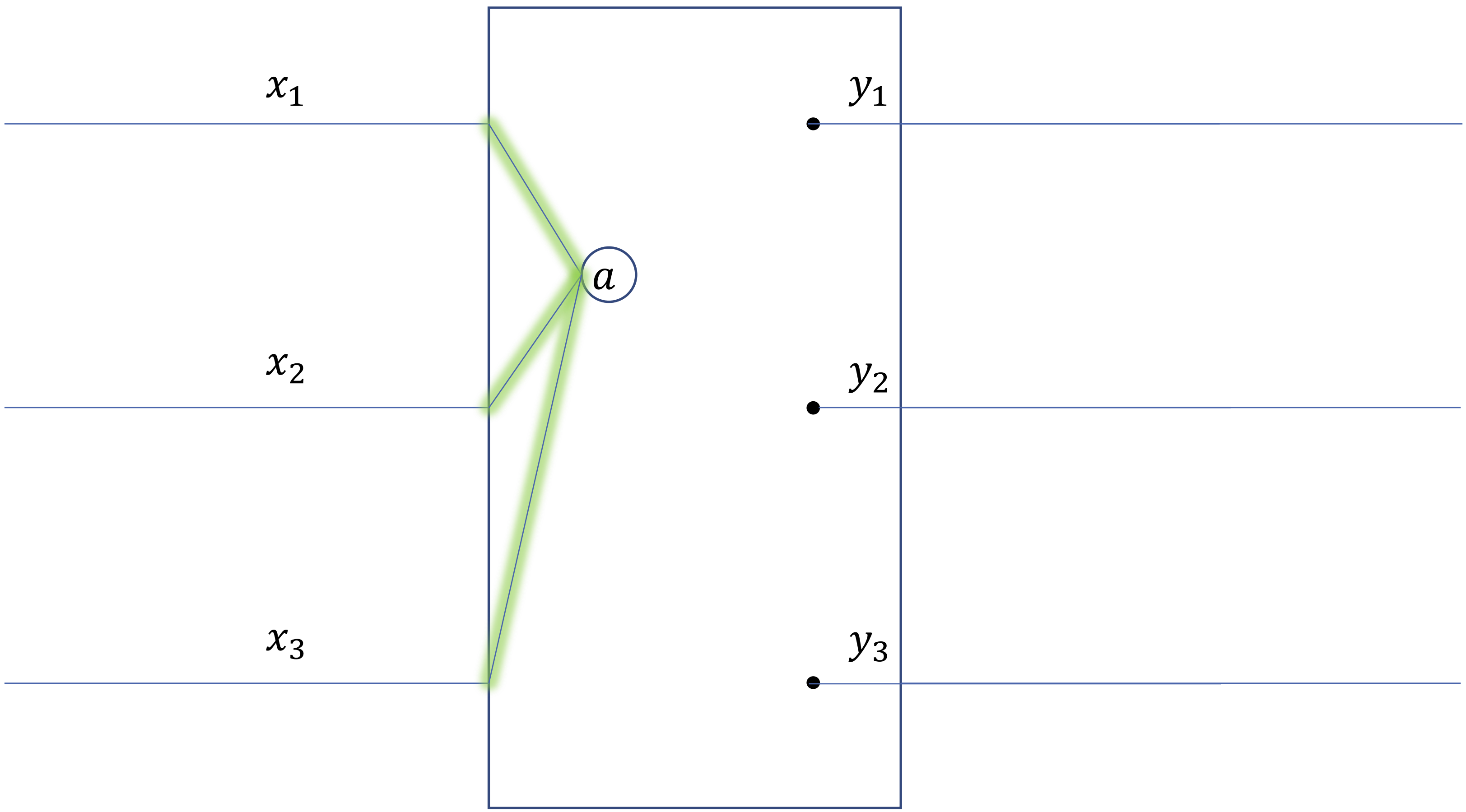


A New (Made Up) Activation in Town

$$a = \sum_j x_j$$

$$b = \sum_j \ln(x_j)$$

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

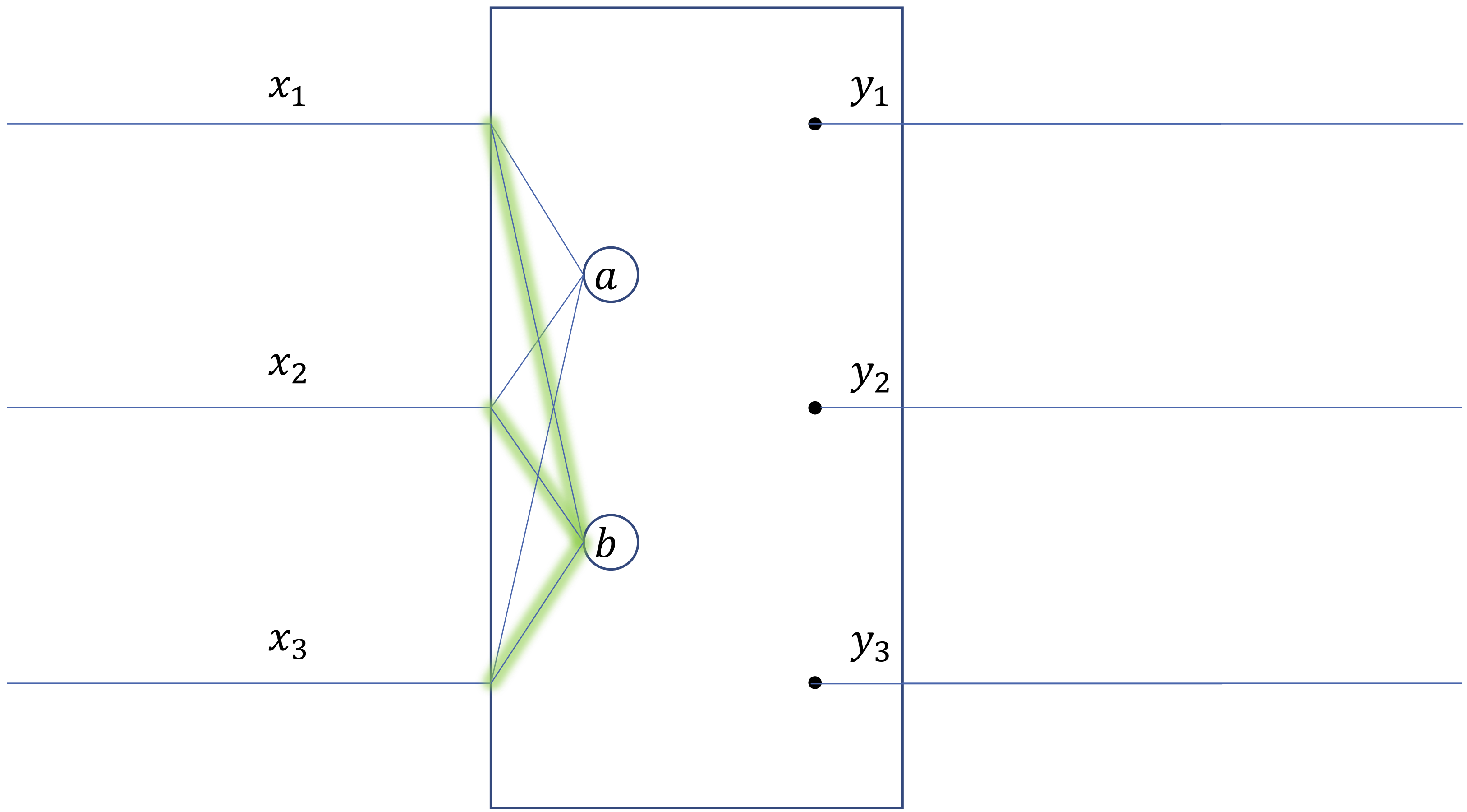


A New (Made Up) Activation in Town

$$a = \sum_j x_j$$

$$b = \sum_j \ln(x_j)$$

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

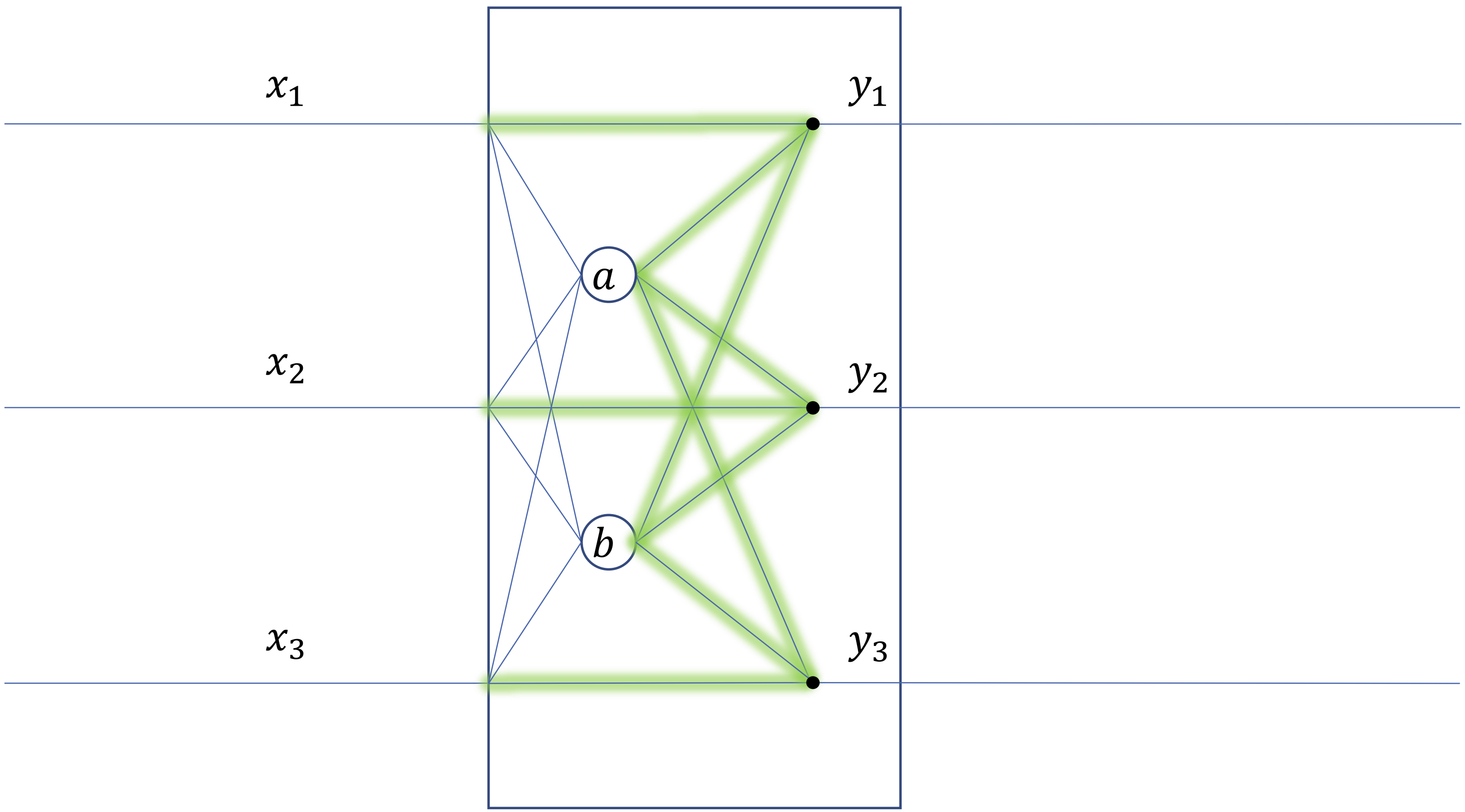


A New (Made Up) Activation in Town

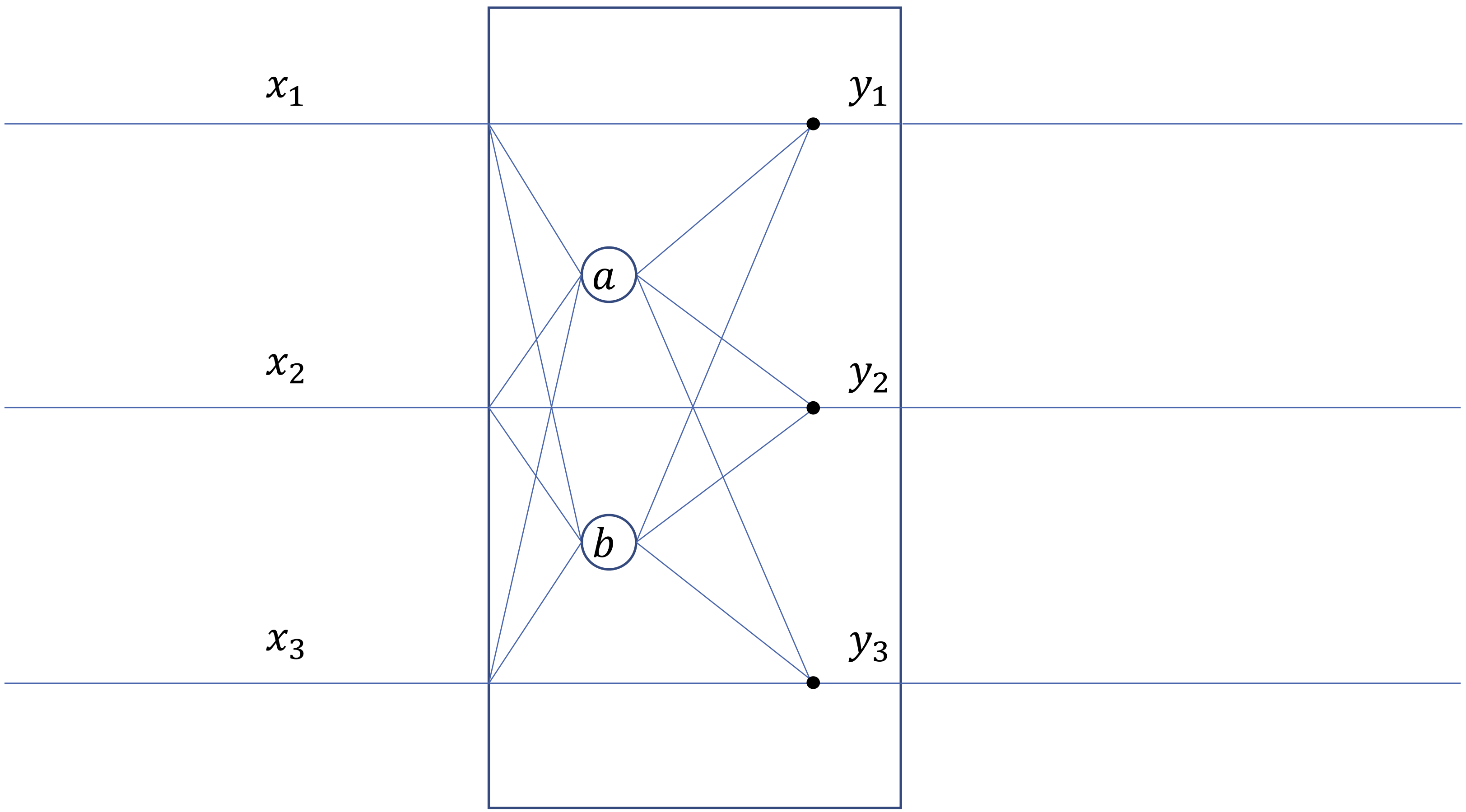
$$a = \sum_j x_j$$

$$b = \sum_j \ln(x_j)$$

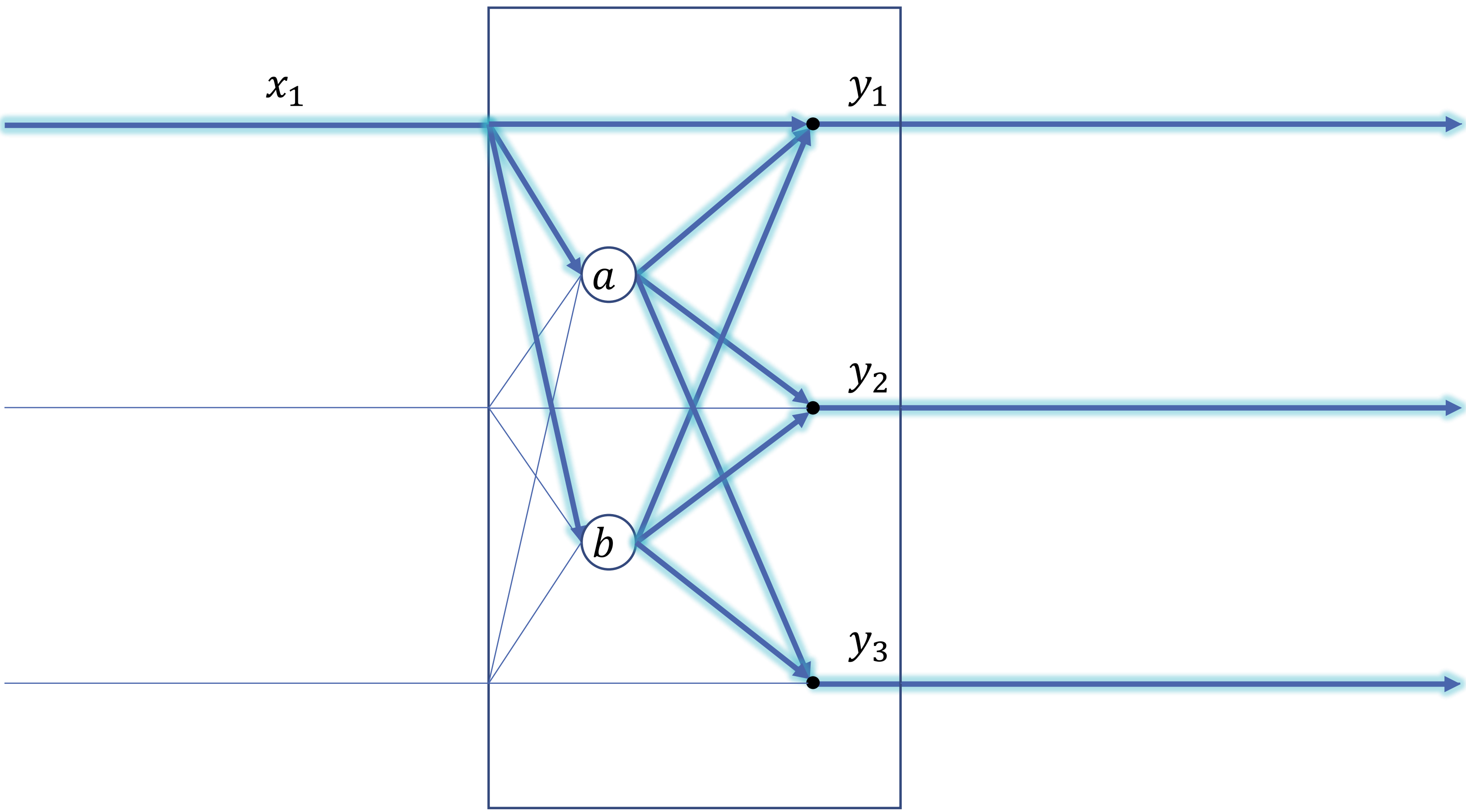
$$y_i = \cos\left(\frac{e^{x_i a}}{b}\right)$$



A New (Made Up) Activation in Town

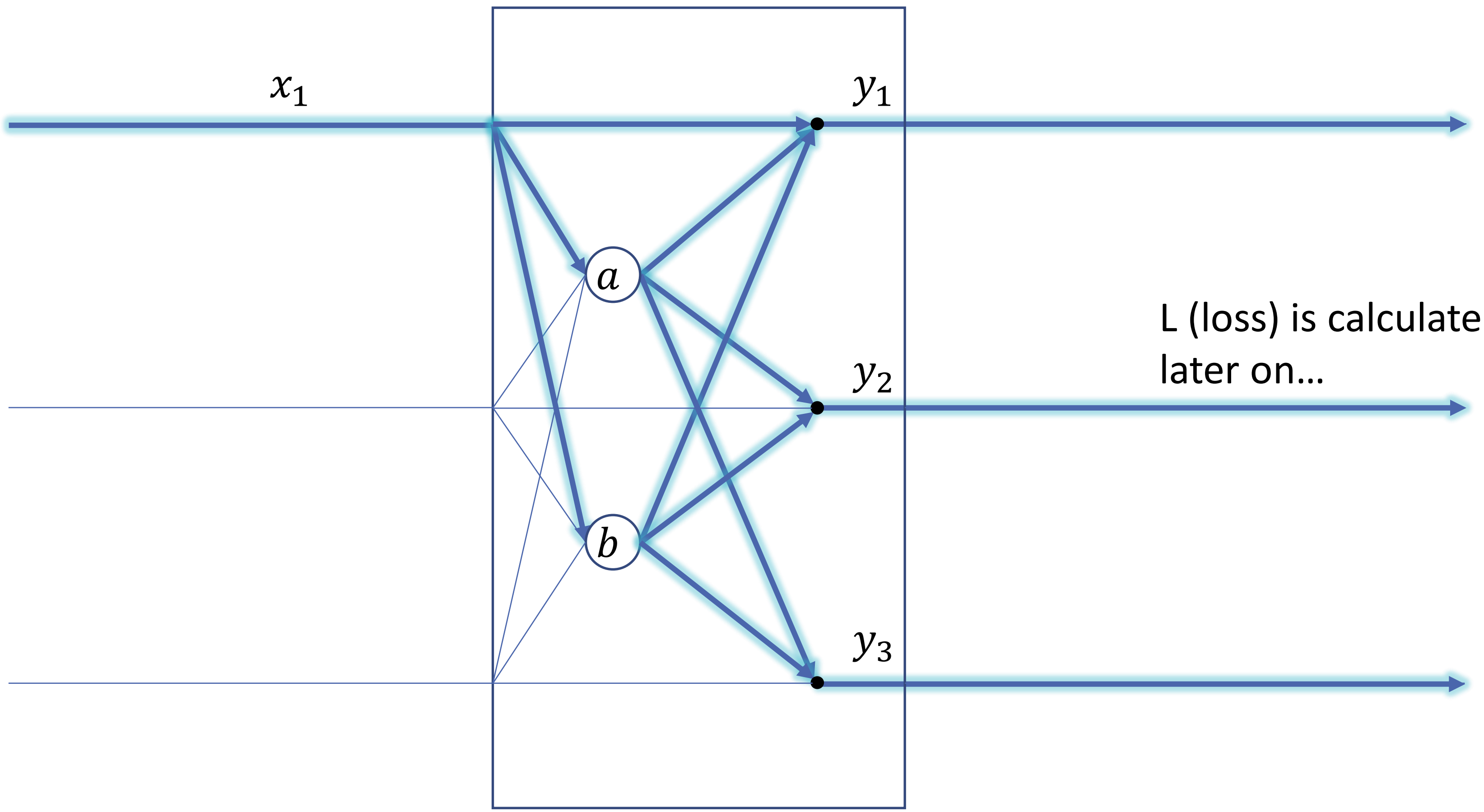


A New (Made Up) Activation in Town



Notice x_1's different paths of influence

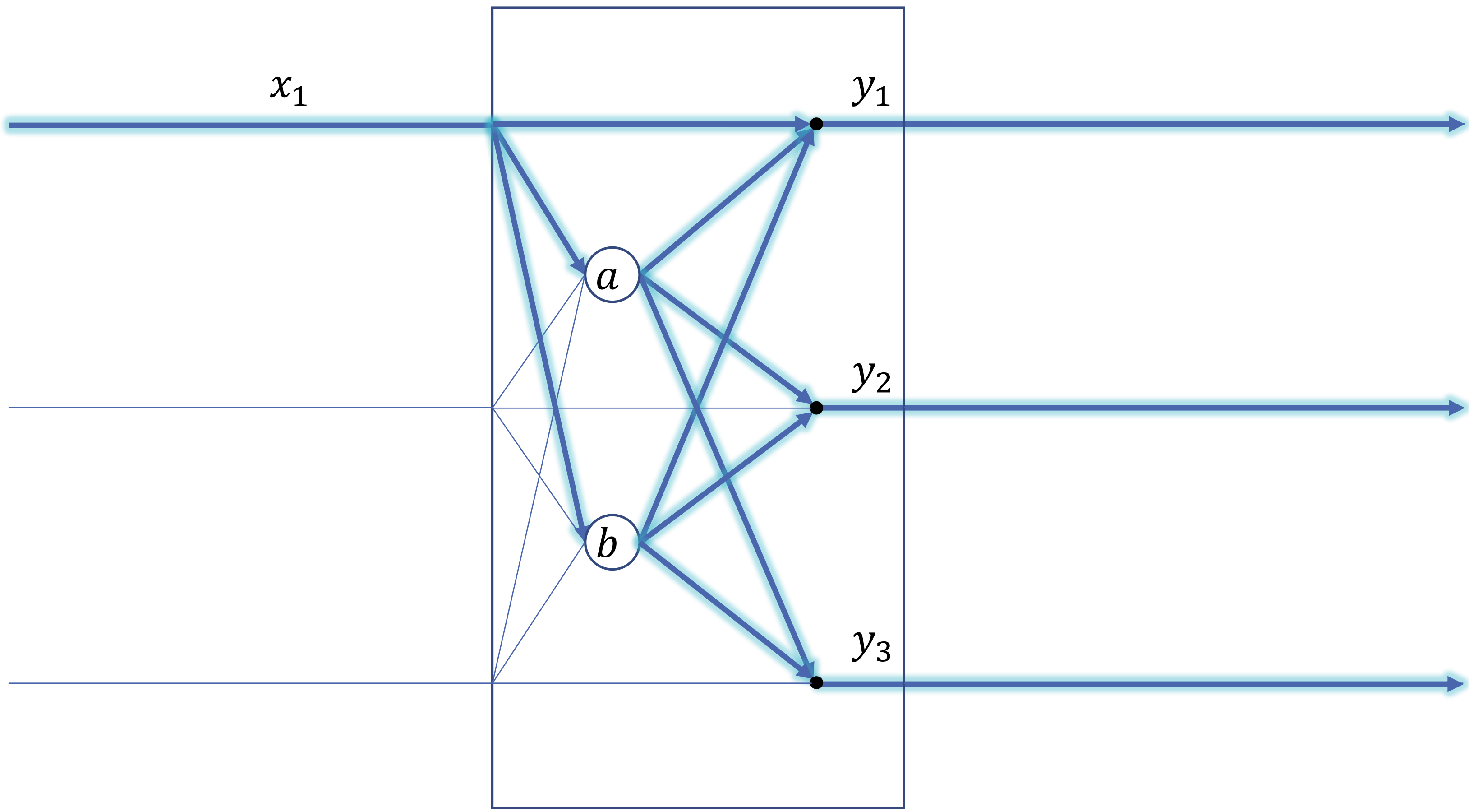
A New (Made Up) Activation in Town



L (loss) is calculated later on...

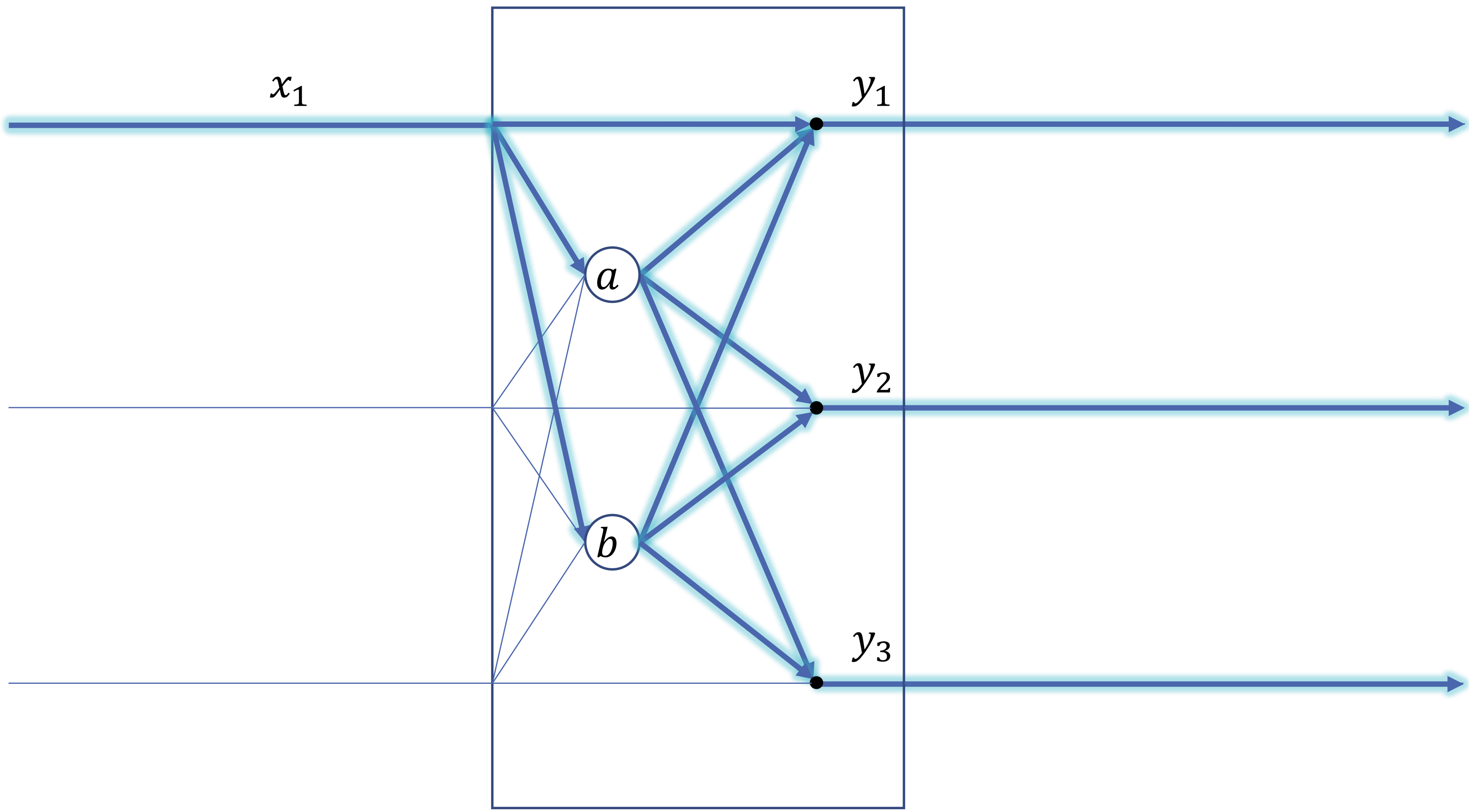
Notice x_1's different paths of influence

A New (Made Up) Activation in Town



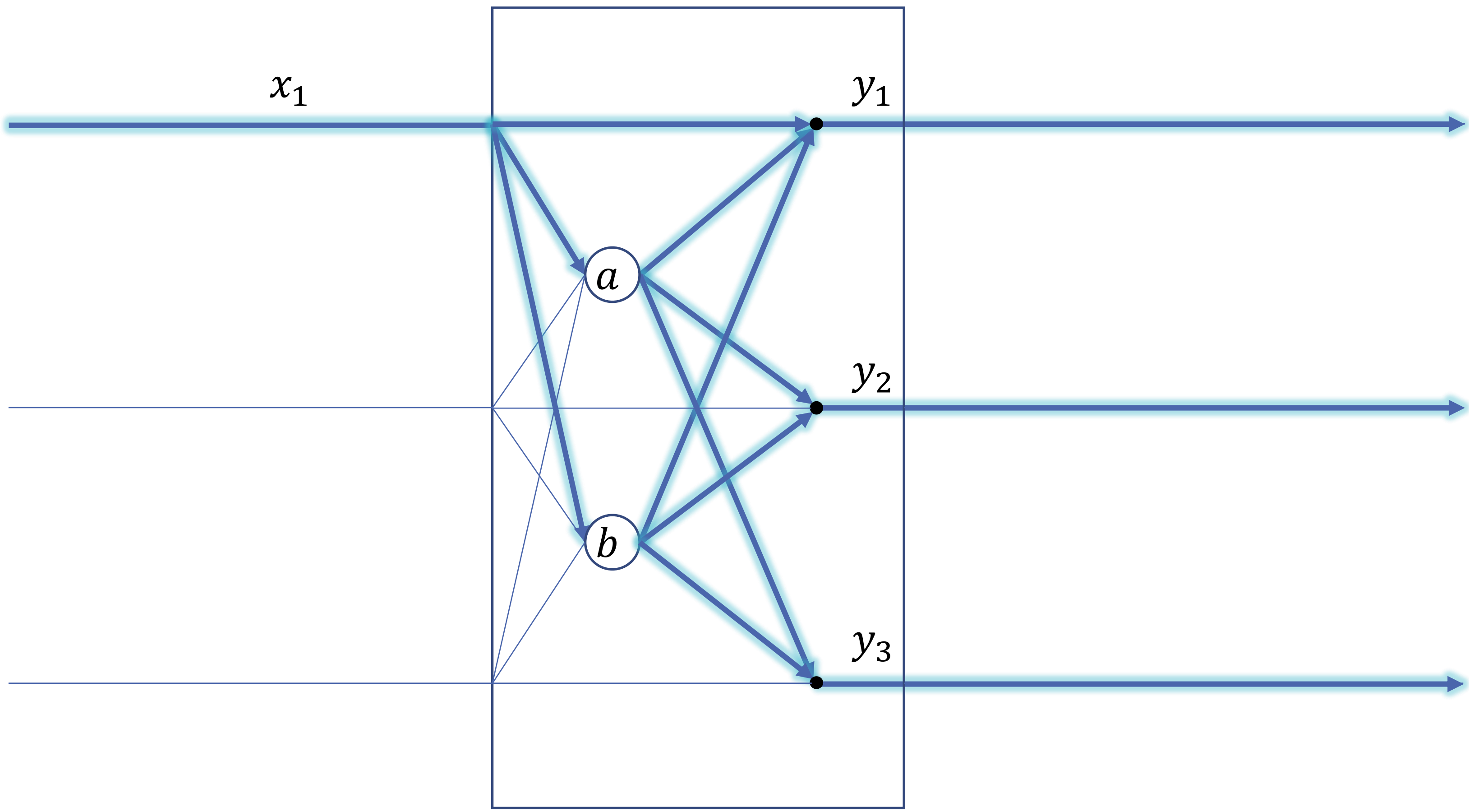
The derivative $\nabla_{x1} L$ is the sum of derivatives along these paths

A New (Made Up) Activation in Town



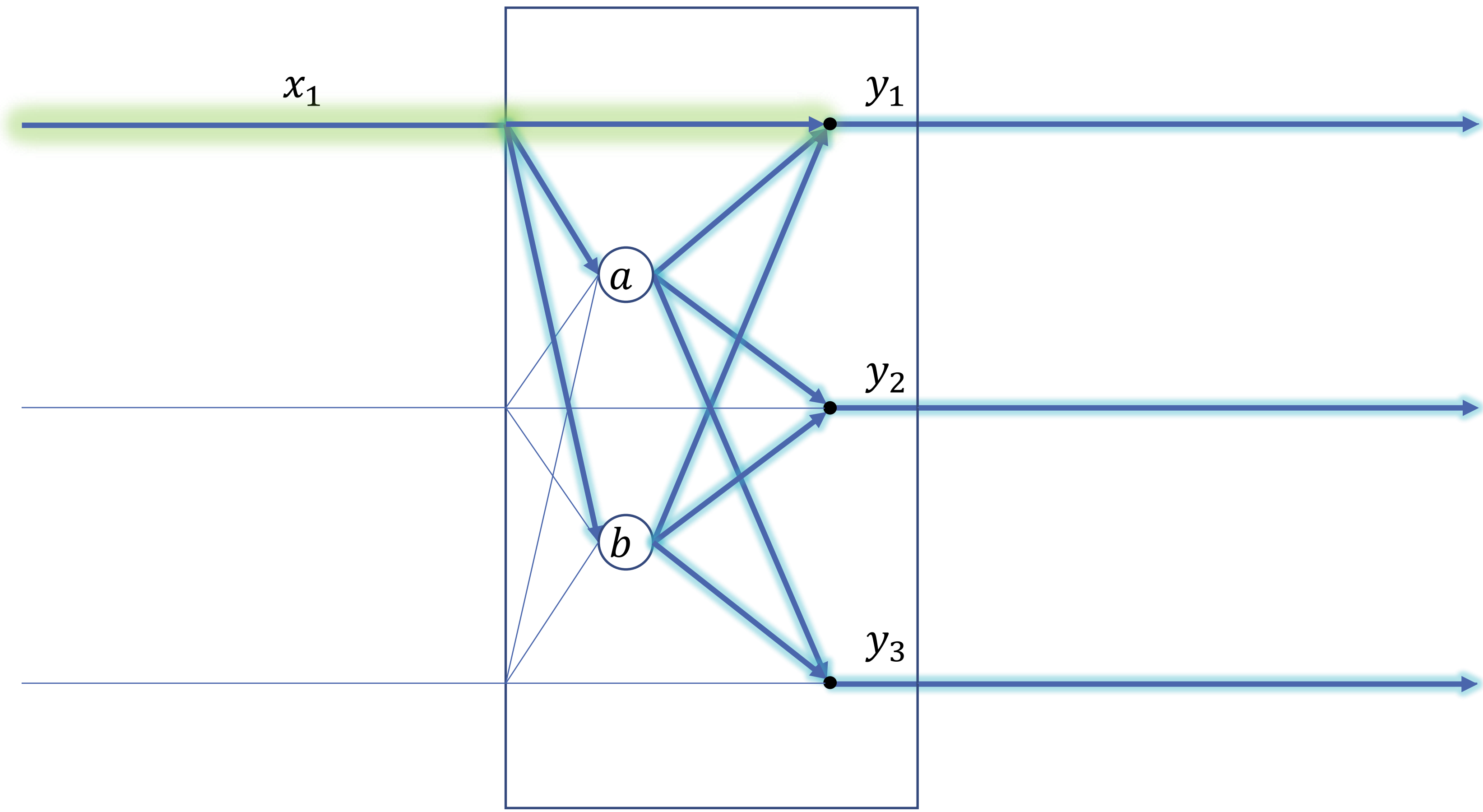
We will apply the chain rule at each node

A New (Made Up) Activation in Town



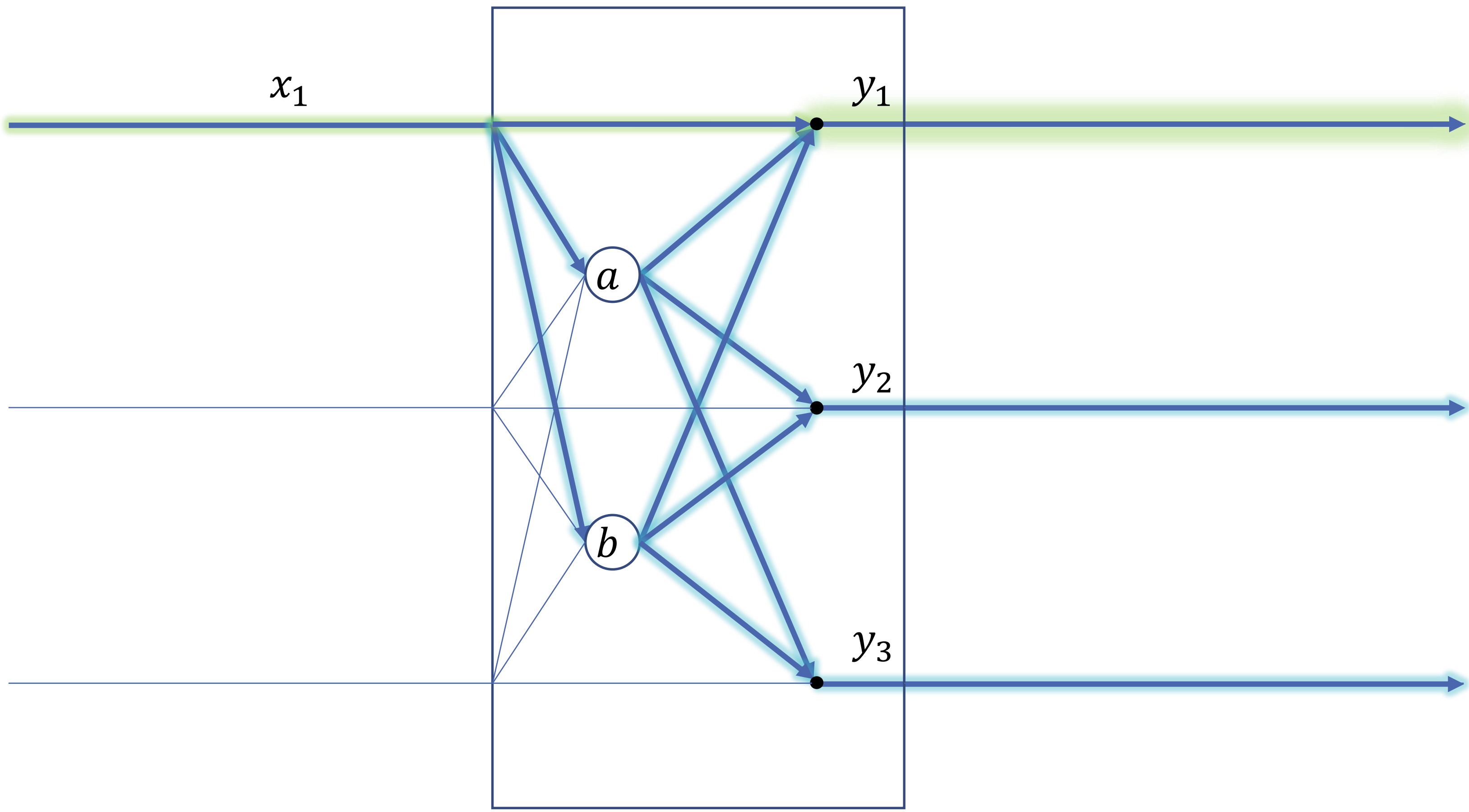
$\nabla_{x1} L =$

A New (Made Up) Activation in Town



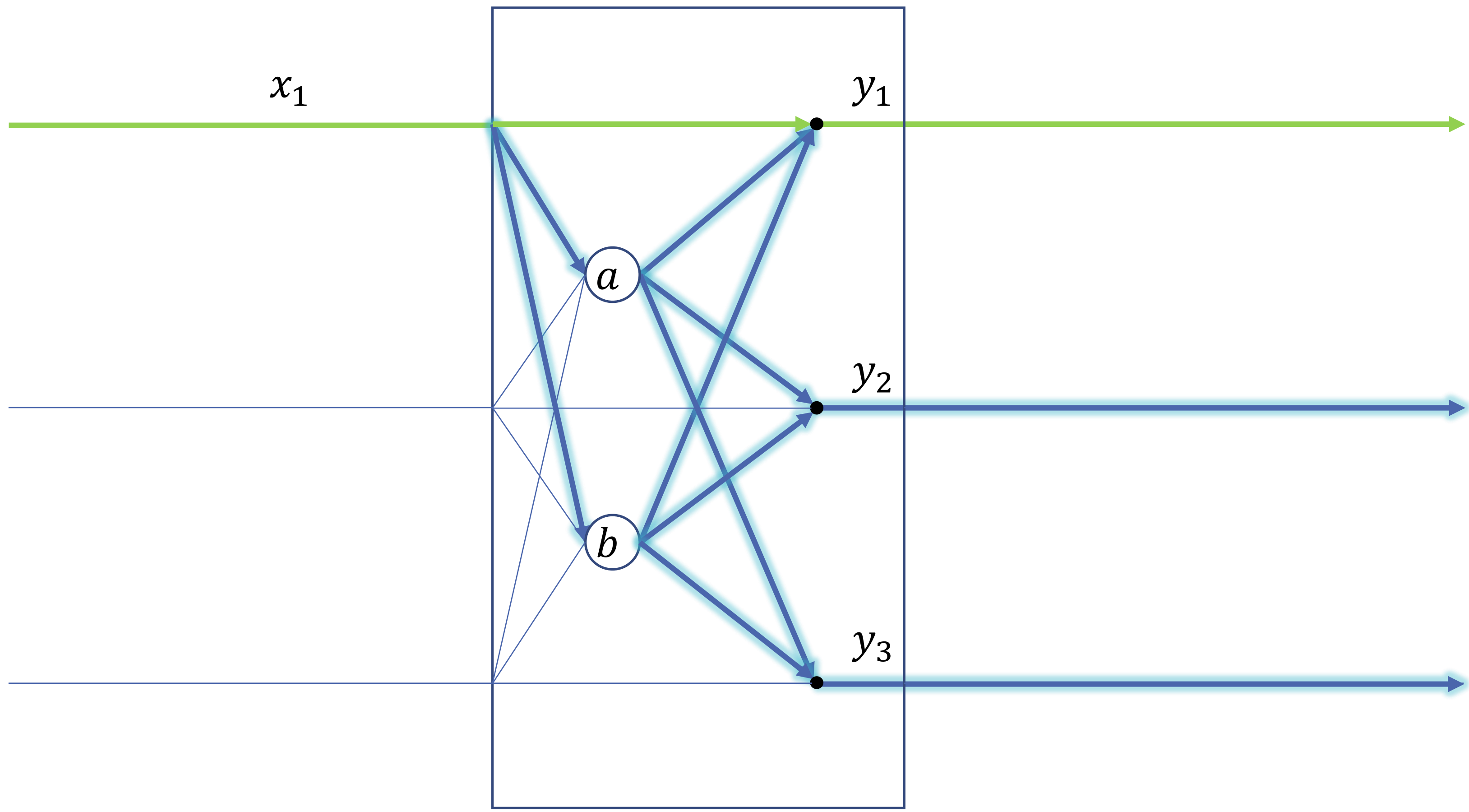
$$\nabla_{x1} L = \frac{dy_1}{dx_1}$$

A New (Made Up) Activation in Town



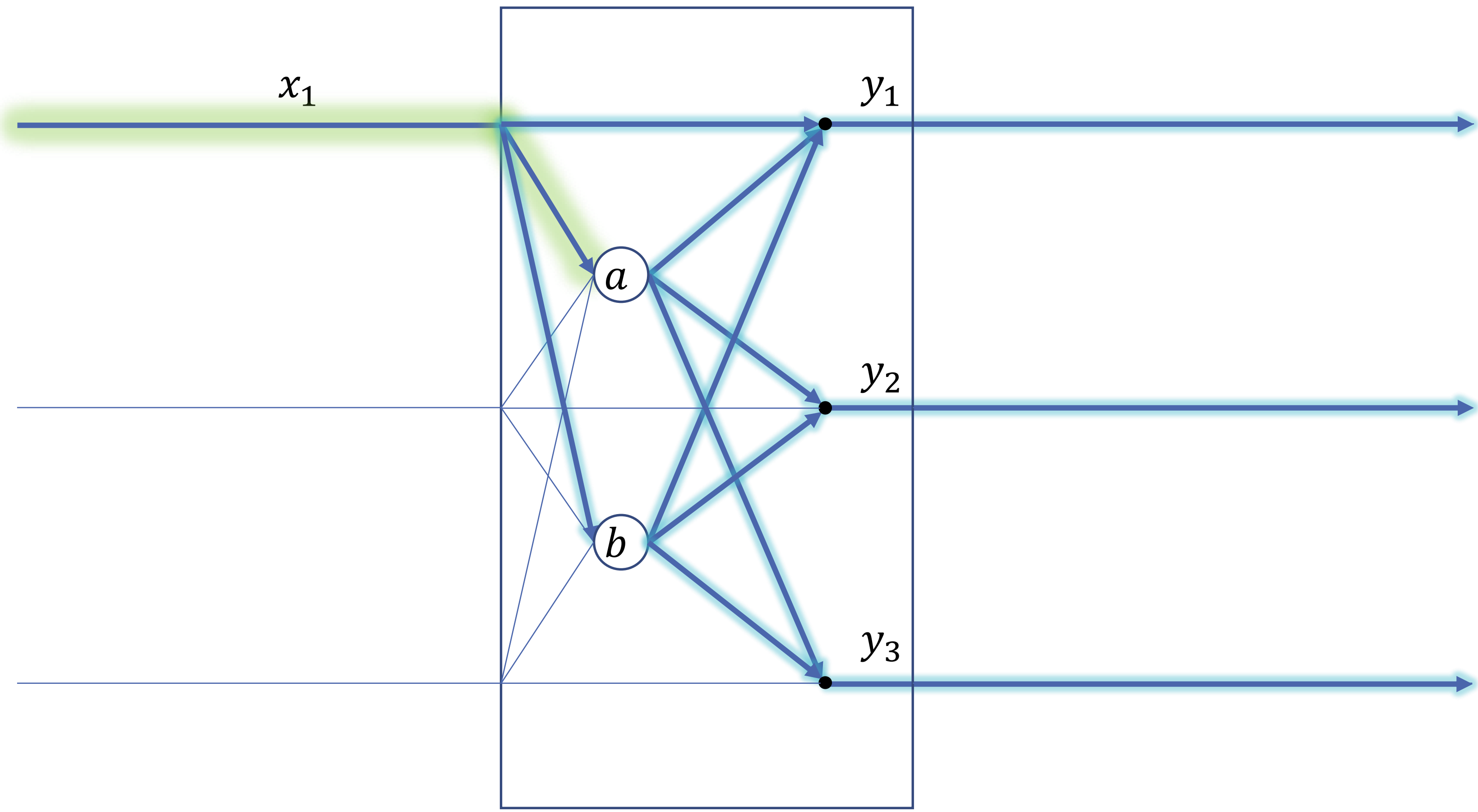
$$\nabla_{x1} L = \frac{dy_1}{dx_1} \frac{dL}{dy_1}$$

A New (Made Up) Activation in Town



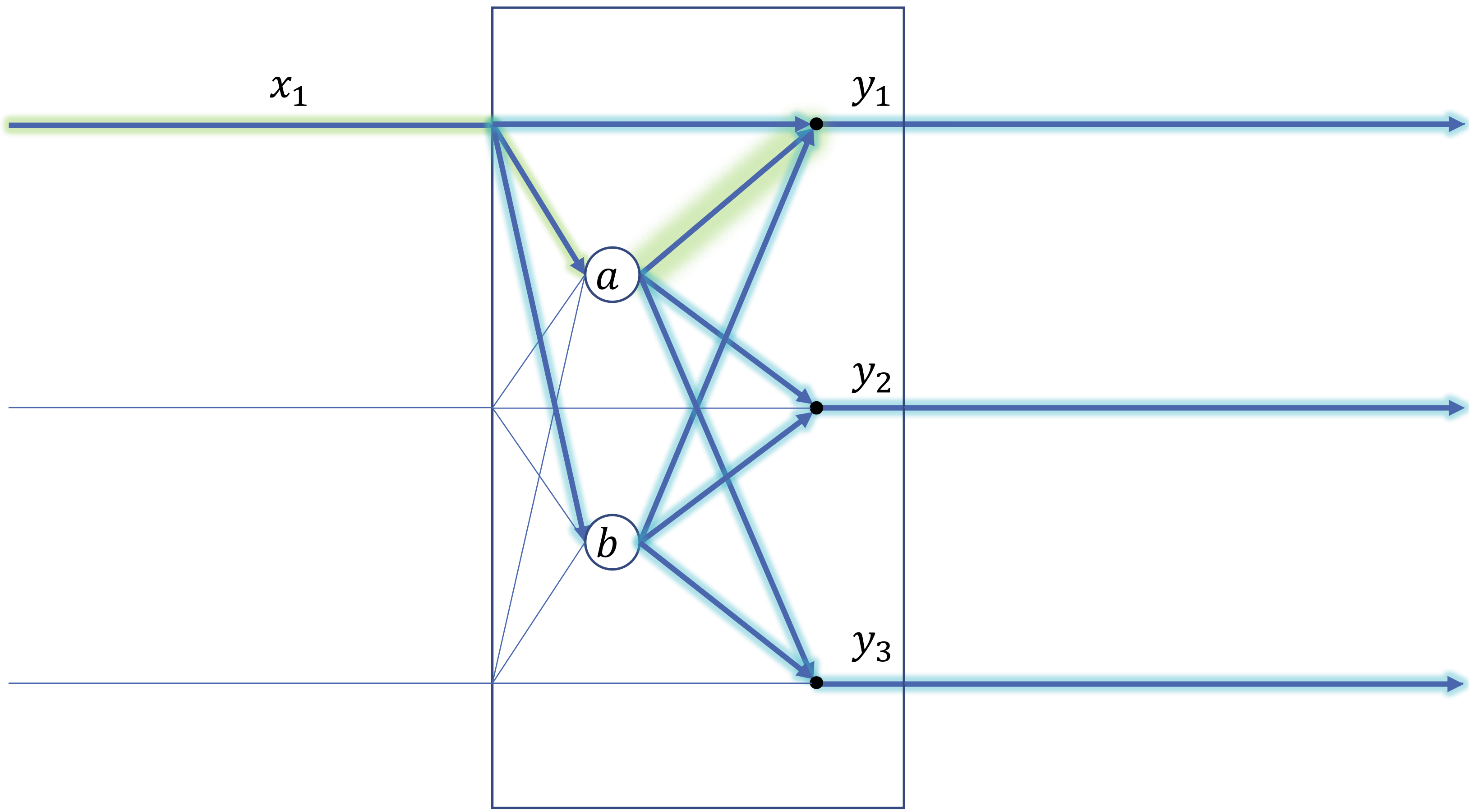
$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1}$$

A New (Made Up) Activation in Town



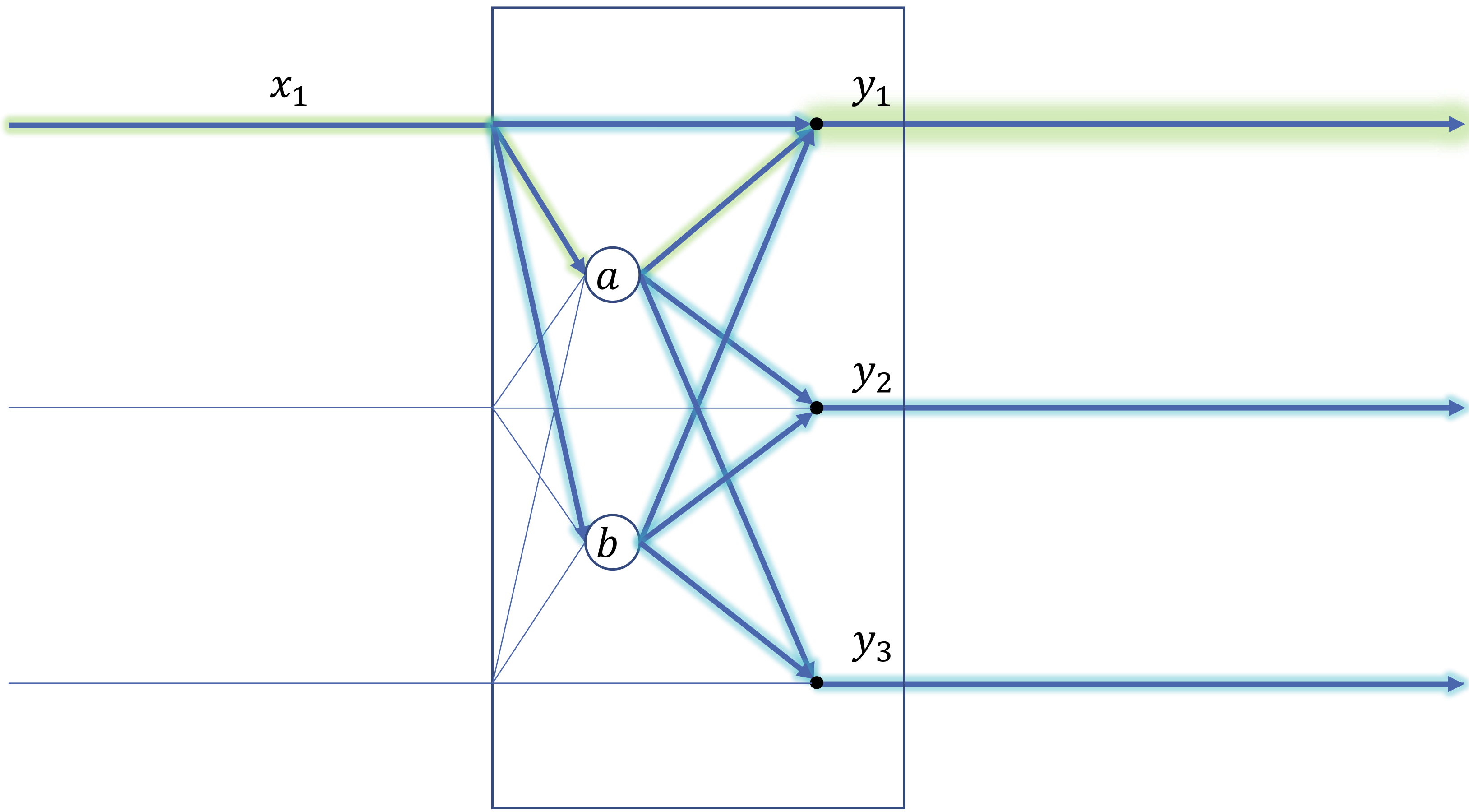
$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}$$

A New (Made Up) Activation in Town



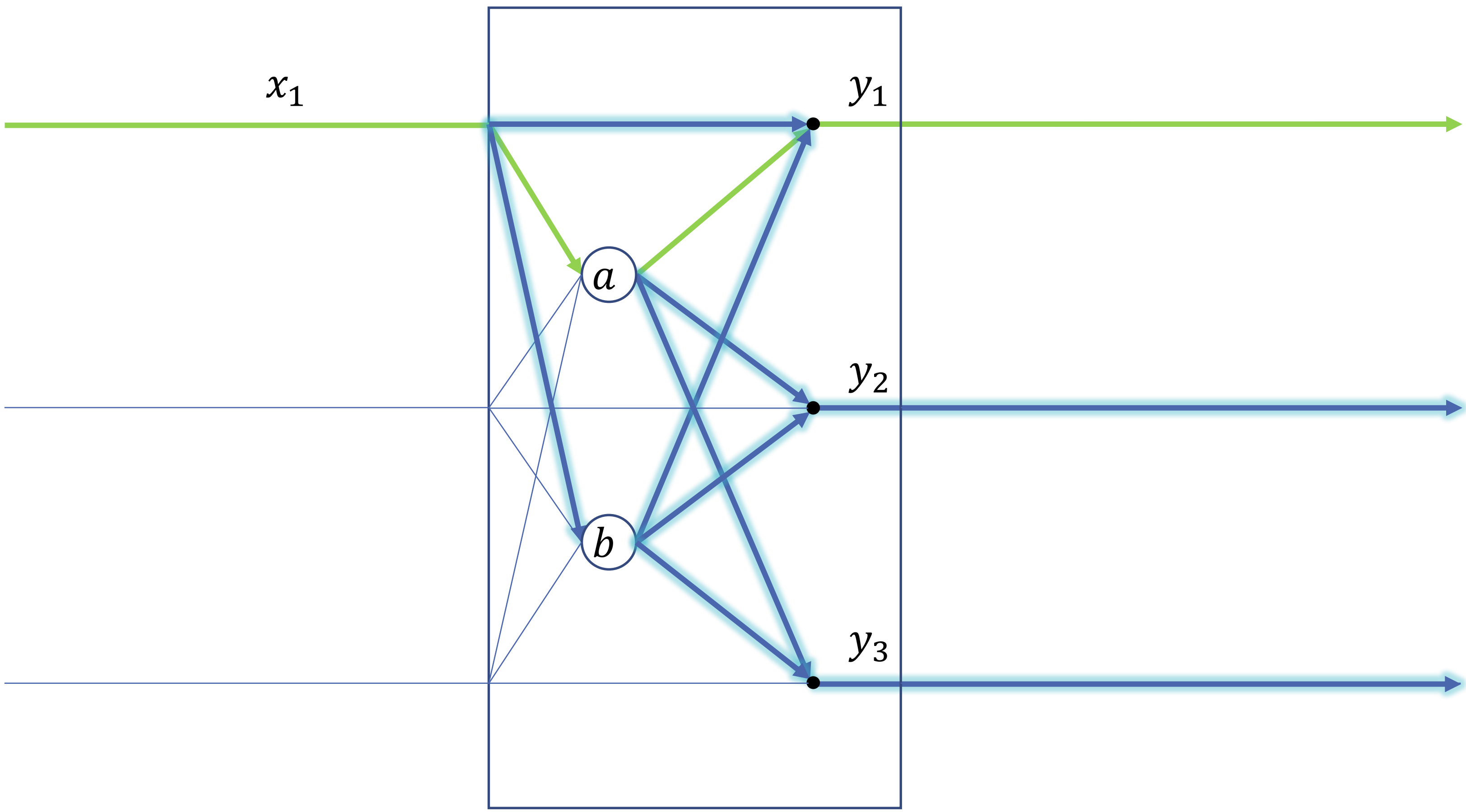
$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}$$

A New (Made Up) Activation in Town



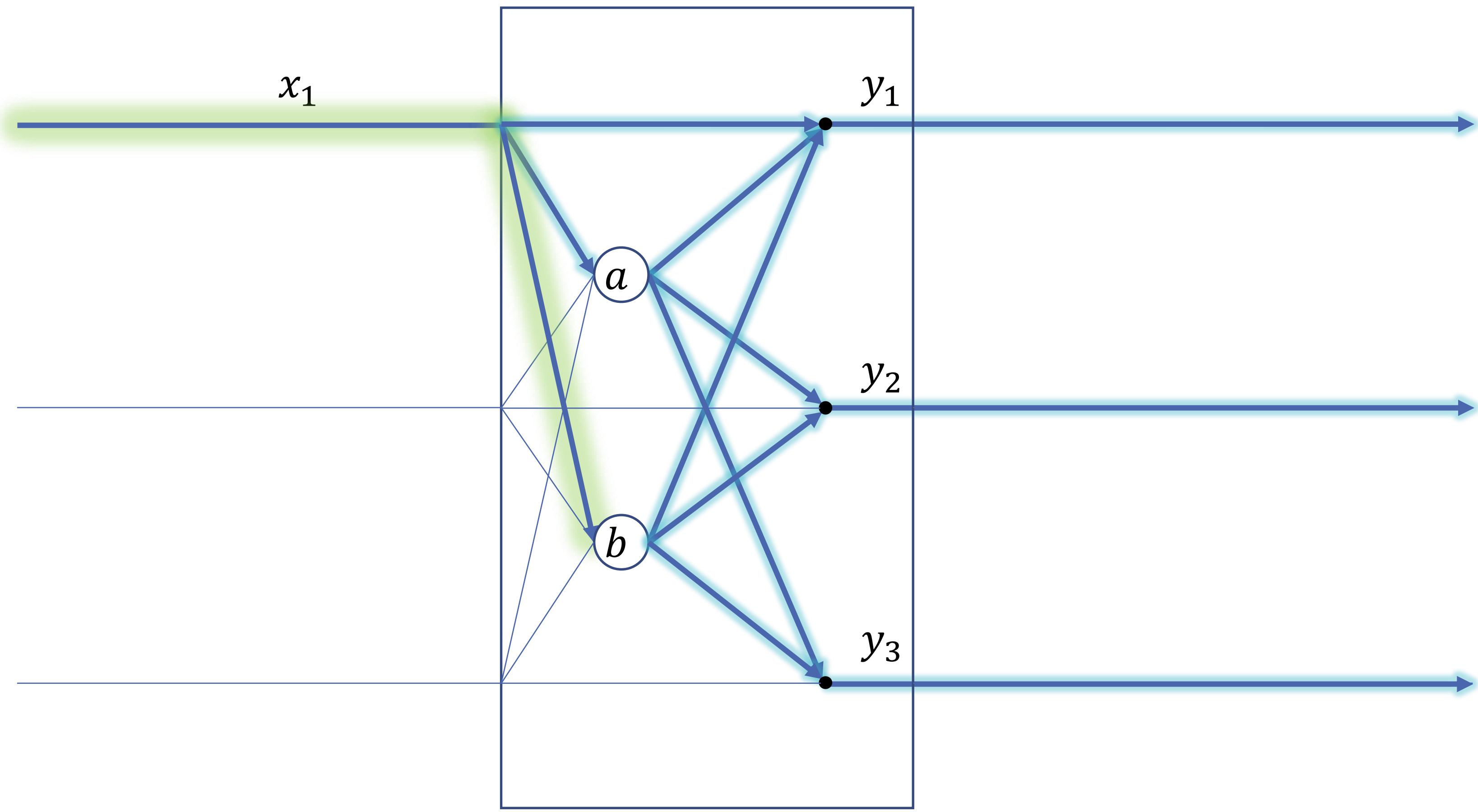
$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1}$$

A New (Made Up) Activation in Town



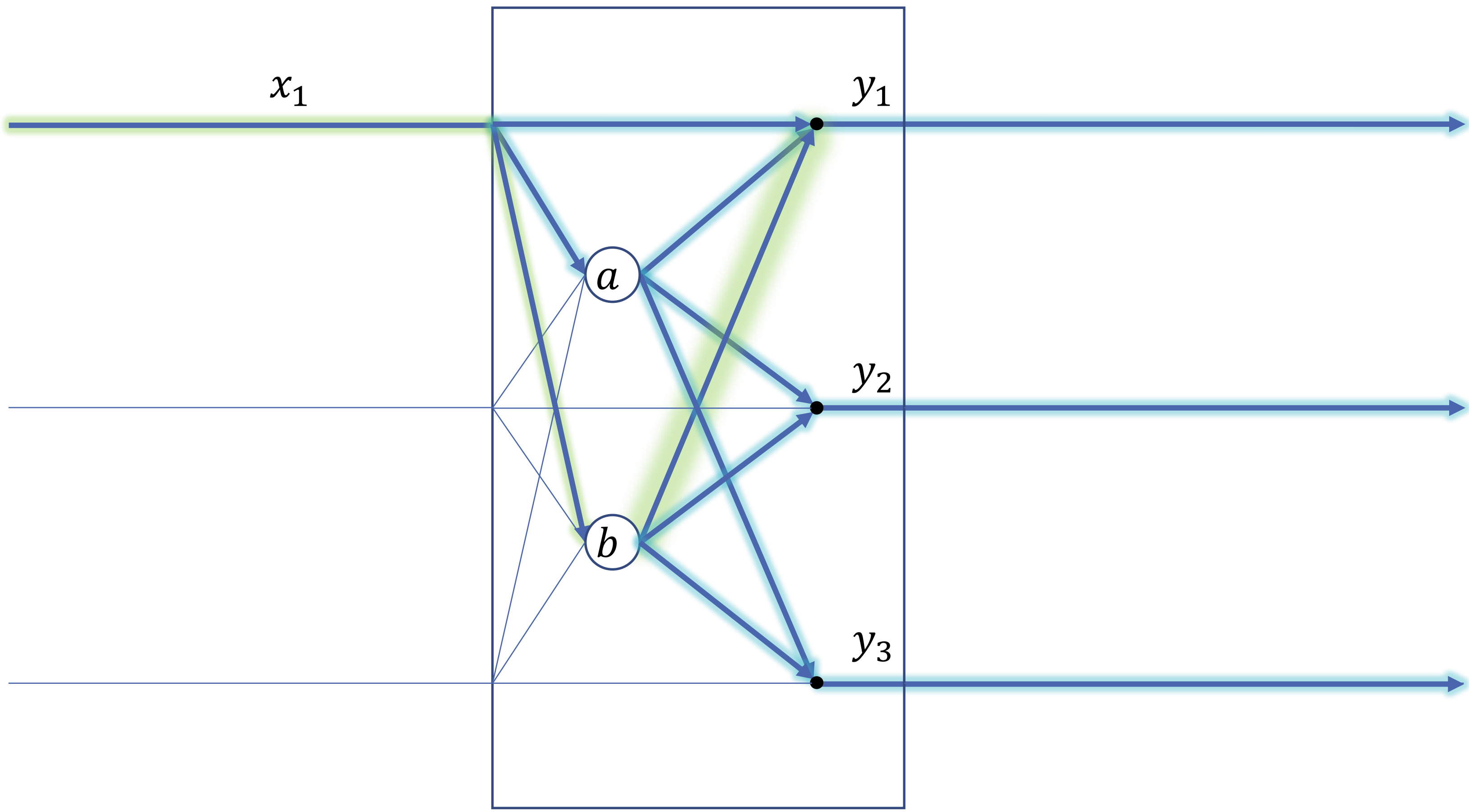
$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1}$$

A New (Made Up) Activation in Town



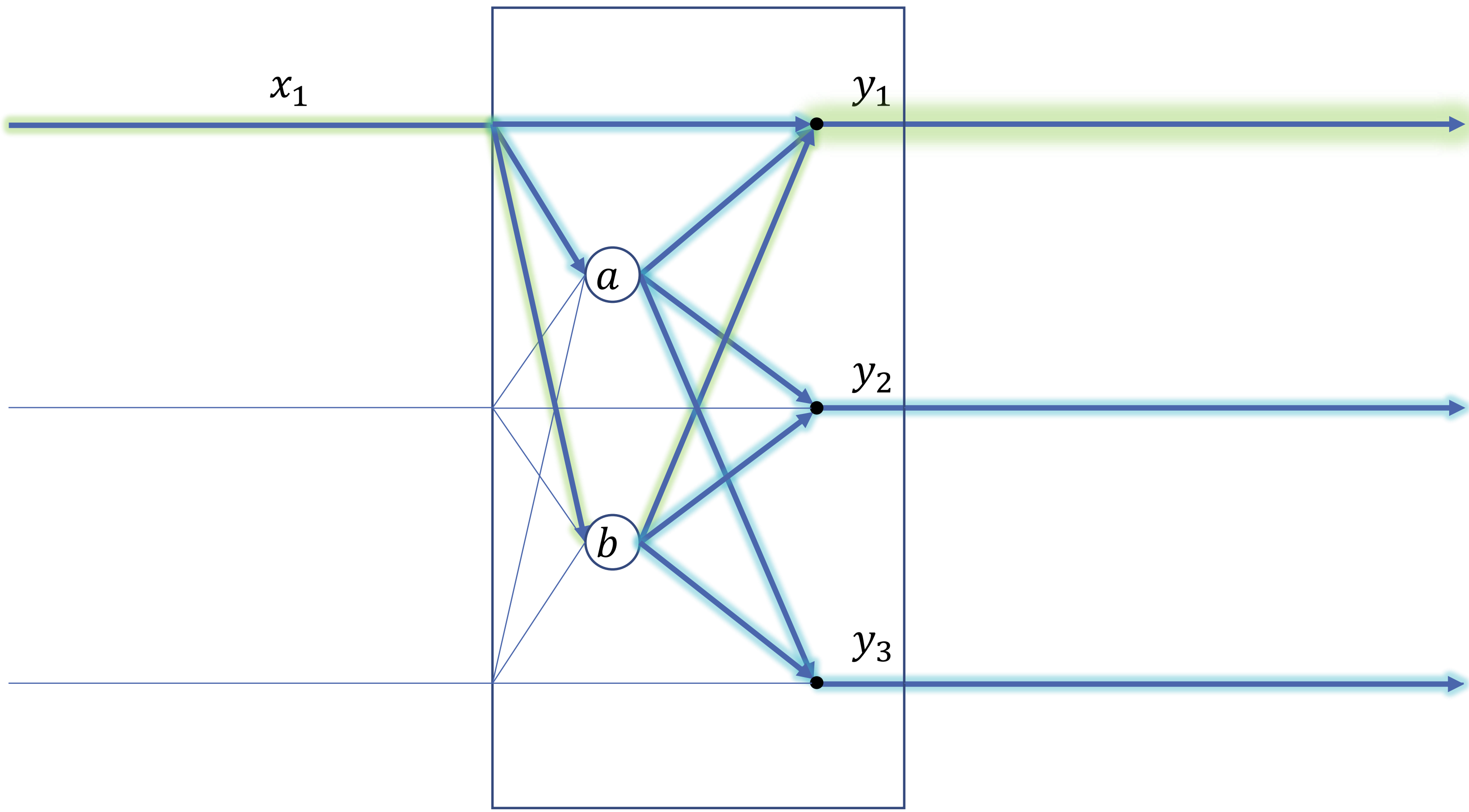
$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}$$

A New (Made Up) Activation in Town



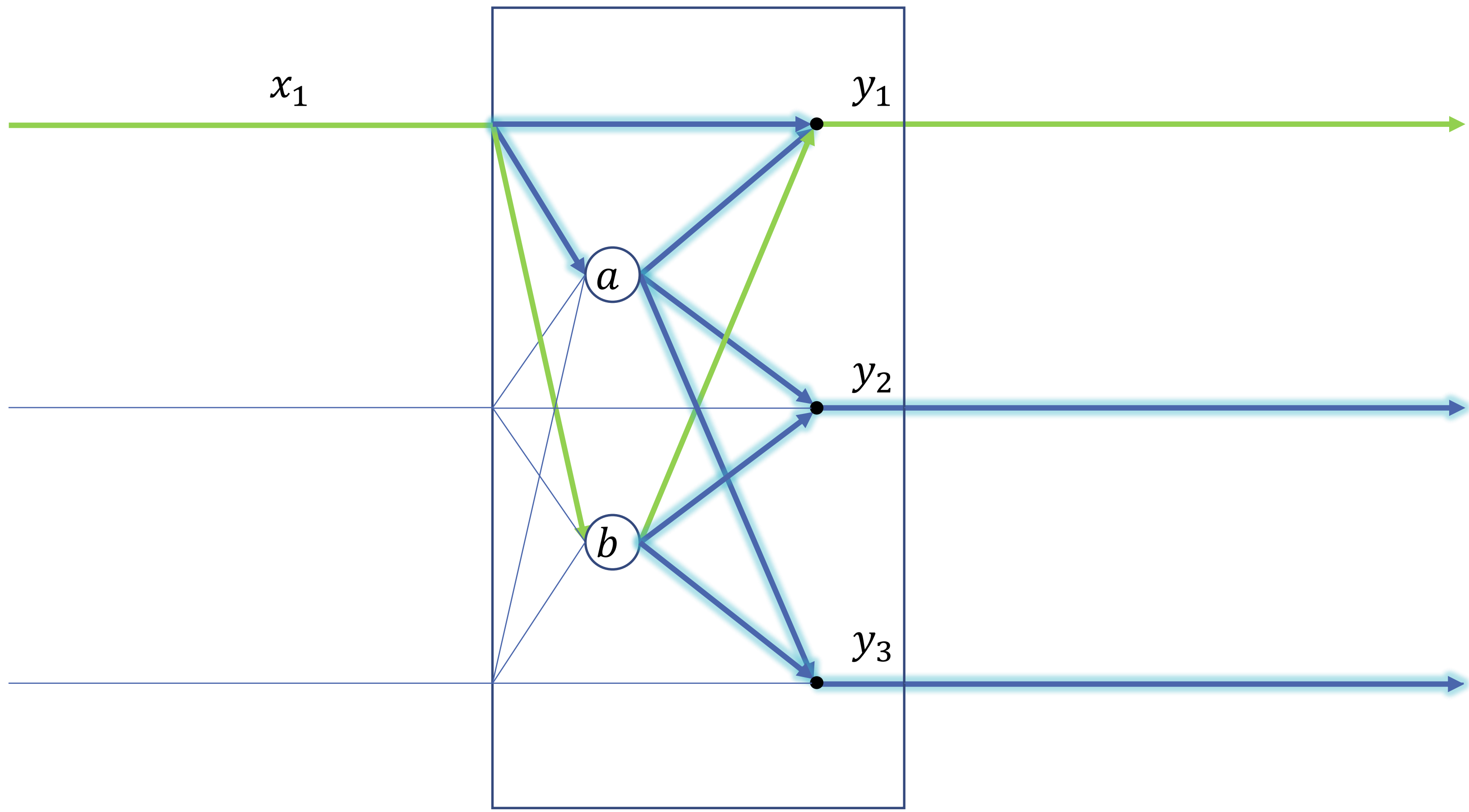
$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}$$

A New (Made Up) Activation in Town



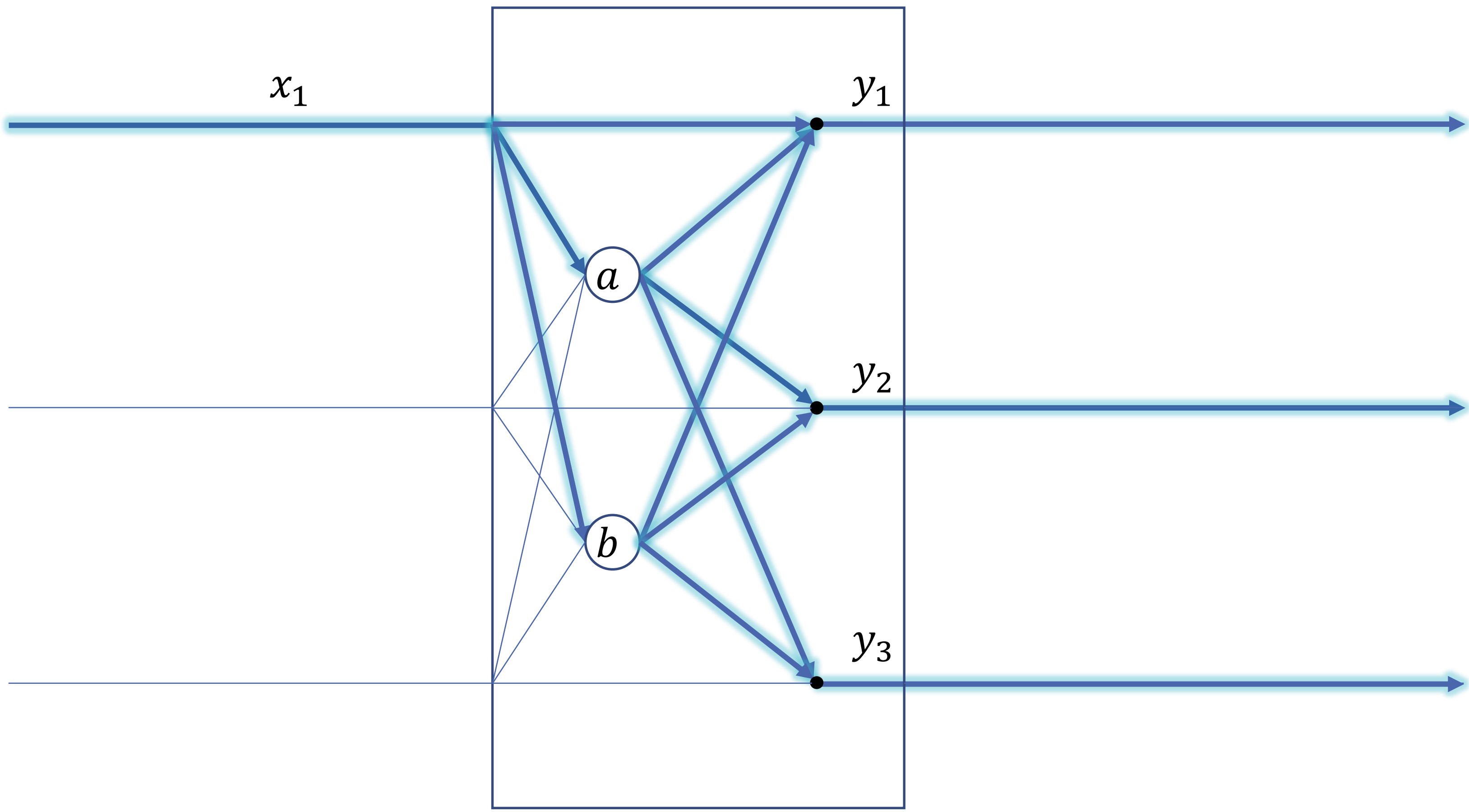
$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1}$$

A New (Made Up) Activation in Town



$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1}$$

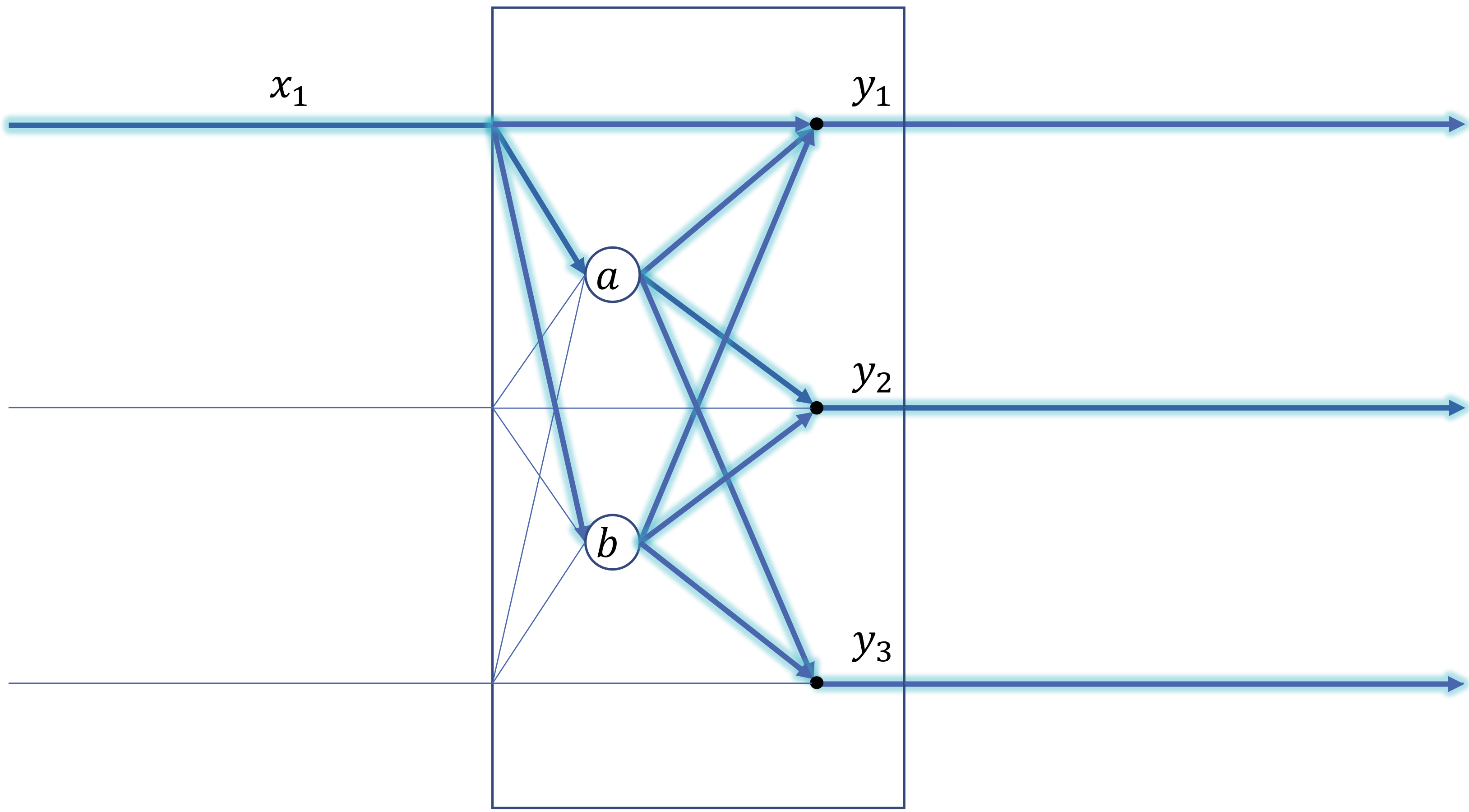
A New (Made Up) Activation in Town



$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \cdots$$

We can do this for the rest of the paths…

A New (Made Up) Activation in Town

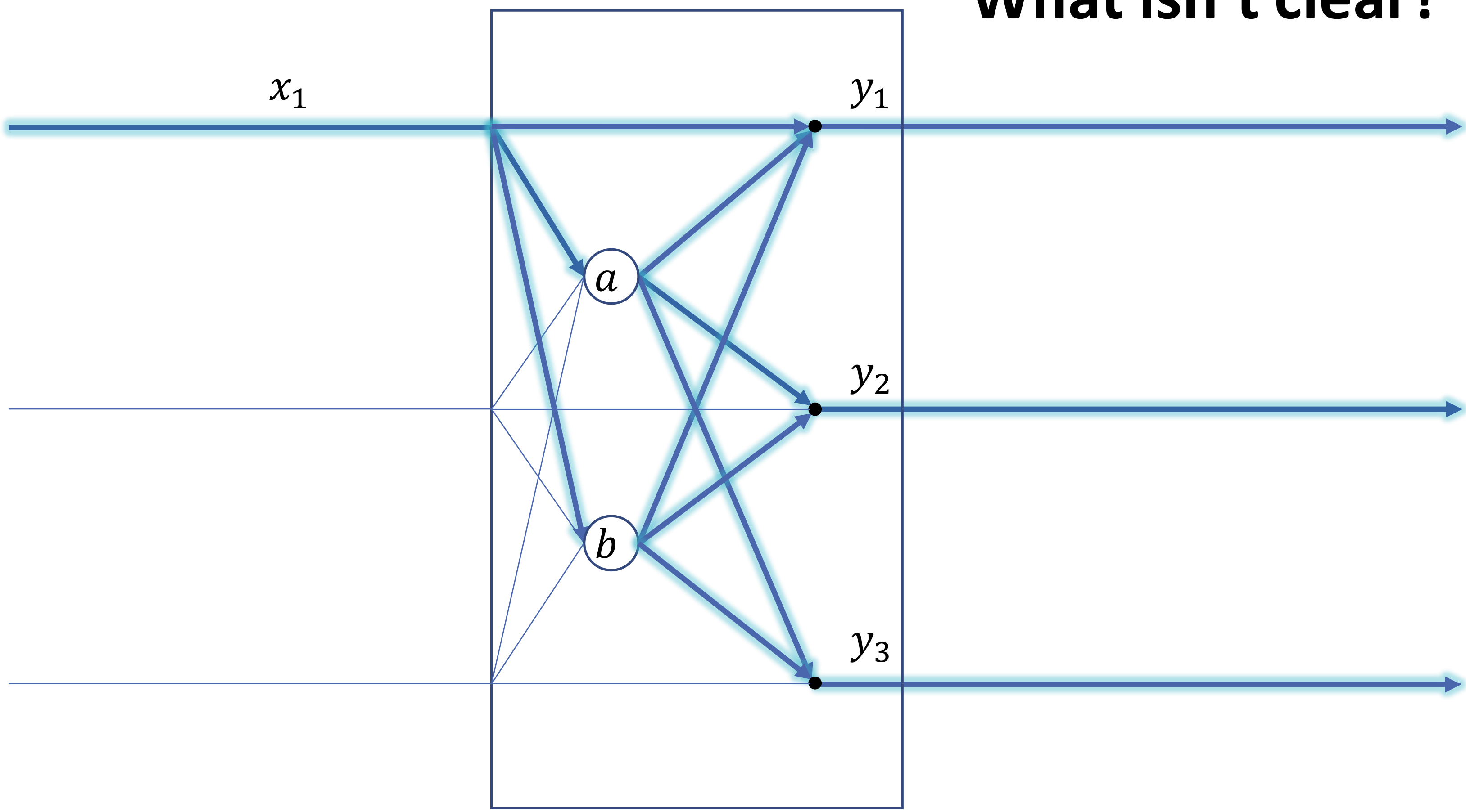


$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Seven terms, seven paths

A New (Made Up) Activation in Town

Questions?
What isn't clear?



$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Seven terms, seven paths

A New (Made Up) Activation in Town

$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Now we’re done with the influence diagram

A New (Made Up) Activation in Town

$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Time to calculate the necessary derivatives…

A New (Made Up) Activation in Town

$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Time to calculate the necessary derivatives…

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

A New (Made Up) Activation in Town

$$\nabla_{x1}L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Time to calculate the necessary derivatives…

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

(don't focus on the following math, just the process)

A New (Made Up) Activation in Town

$$\nabla_{x1} L = \boldsymbol{\frac{dy_1}{dx_1}} \frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Time to calculate the necessary derivatives…

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

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

A New (Made Up) Activation in Town

$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{\boldsymbol{dL}}{\boldsymbol{dy_1}} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Time to calculate the necessary derivatives…

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

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

A New (Made Up) Activation in Town

$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Time to calculate the necessary derivatives…

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

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

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

A New (Made Up) Activation in Town

$$\nabla_{x1}L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{\boldsymbol{dy_1}}{\boldsymbol{da}}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Time to calculate the necessary derivatives…

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

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

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

$$\frac{dy_1}{da} = -\sin(\frac{e^{x_1\ a}}{b})\frac{x_1 e^{x_1\ a}}{b}$$

A New (Made Up) Activation in Town

$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{\mathbf{dL}}{\mathbf{dy_1}} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Time to calculate the necessary derivatives…

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

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

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

$$\frac{dy_1}{da} = -\sin(\frac{e^{x_1\ a}}{b})\,\frac{x_1 e^{x_1\ a}}{b}$$

A New (Made Up) Activation in Town

$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Time to calculate the necessary derivatives…

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

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

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

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

A New (Made Up) Activation in Town

$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Time to calculate the necessary derivatives…

$$a = \sum_j x_j \qquad b = \sum_j \ln(x_j) \qquad y_i = \cos(\frac{e^{x_i\, a}}{b})$$

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

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

$$\frac{dy_1}{da} = -\sin(\frac{e^{x_1\, a}}{b})\,\frac{x_1 e^{x_1\, a}}{b}$$

$$\frac{db}{dx_1} = \frac{1}{x_1}$$

$$\frac{dy_1}{db} = \sin(\frac{e^{x_1\, a}}{b})\,\frac{x_1 e^{x_1\, a}}{b^2}$$

$$\frac{dy_2}{da} = -\sin(\frac{e^{x_2\, a}}{b})\,\frac{x_2 e^{x_2\, a}}{b}$$

$$\frac{dy_2}{db} = \sin(\frac{e^{x_2\, a}}{b})\,\frac{x_2 e^{x_2\, a}}{b^2}$$

$$\frac{dy_3}{da} = -\sin(\frac{e^{x_3\, a}}{b})\,\frac{x_3 e^{x_3\, a}}{b}$$

$$\frac{dy_3}{db} = \sin(\frac{e^{x_3\, a}}{b})\,\frac{x_3 e^{x_3\, a}}{b^2}$$

A New (Made Up) Activation in Town

$$\nabla_{x1}L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

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

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

$$\frac{dy_1}{da} = -\sin(\frac{e^{x_1}{}^a}{b})\frac{x_1e^{x_1}{}^a}{b}$$

$$\frac{db}{dx_1} = \frac{1}{x_1}$$

$$\frac{dy_1}{db} = \sin(\frac{e^{x_1}{}^a}{b})\frac{x_1e^{x_1}{}^a}{b^2}$$

$$\frac{dy_2}{da} = -\sin(\frac{e^{x_2}{}^a}{b})\frac{x_2e^{x_2}{}^a}{b}$$

$$\frac{dy_2}{db} = \sin(\frac{e^{x_2}{}^a}{b})\frac{x_2e^{x_2}{}^a}{b^2}$$

$$\frac{dy_3}{da} = -\sin(\frac{e^{x_3}{}^a}{b})\frac{x_3e^{x_3}{}^a}{b}$$

$$\frac{dy_3}{db} = \sin(\frac{e^{x_3}{}^a}{b})\frac{x_3e^{x_3}{}^a}{b^2}$$

Now we plug things in / simplify

A New (Made Up) Activation in Town

$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

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

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

$$\frac{dy_1}{da} = -\sin(\frac{e^{x_1\,a}}{b})\frac{x_1 e^{x_1\,a}}{b}$$

$$\frac{db}{dx_1} = \frac{1}{x_1}$$

$$\frac{dy_1}{db} = \sin(\frac{e^{x_1\,a}}{b})\frac{x_1 e^{x_1\,a}}{b^2}$$

$$\frac{dy_2}{da} = -\sin(\frac{e^{x_2\,a}}{b})\frac{x_2 e^{x_2\,a}}{b}$$

$$\frac{dy_2}{db} = \sin(\frac{e^{x_2\,a}}{b})\frac{x_2 e^{x_2\,a}}{b^2}$$

$$\frac{dy_3}{da} = -\sin(\frac{e^{x_3\,a}}{b})\frac{x_3 e^{x_3\,a}}{b}$$

$$\frac{dy_3}{db} = \sin(\frac{e^{x_3\,a}}{b})\frac{x_3 e^{x_3\,a}}{b^2}$$

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

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

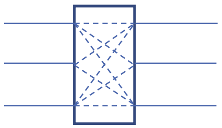
Influence Diagrams

A little painful, but algorithmic:

Break things up

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

Draw the influence diagram



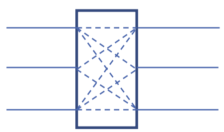
Influence Diagrams

A little painful, but algorithmic:

Break things up

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

Draw the influence diagram



Write out paths using the diagram / chain rule

$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

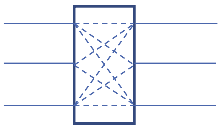
Influence Diagrams

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

$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Calculate necessary derivatives

$\frac{dy_1}{dx_1} = -\sin\left(\frac{e^{x_1\, a}}{b}\right)\frac{a e^{x_1\, a}}{b}$ $\quad \frac{da}{dx_1} = 1$ $\quad \frac{dy_1}{da} = -\sin\left(\frac{e^{x_1\, a}}{b}\right)\frac{x_1 e^{x_1\, a}}{b}$ $\quad \frac{db}{dx_1} = \frac{1}{x_1}$ $\quad \frac{dy_1}{db} = \sin\left(\frac{e^{x_1\, a}}{b}\right)\frac{x_1 e^{x_1\, a}}{b^2}$ etc…

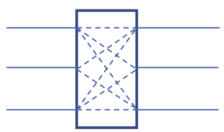
Influence Diagrams

A little painful, but algorithmic:

Break things up

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

Draw the influence diagram



Write out paths using the diagram / chain rule

$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Calculate necessary derivatives

$\frac{dy_1}{dx_1} = -\sin(\frac{e^{x_1\ a}}{b})\frac{ae^{x_1\ a}}{b}$ $\quad \frac{da}{dx_1} = 1$ $\quad \frac{dy_1}{da} = -\sin(\frac{e^{x_1\ a}}{b})\frac{x_1 e^{x_1\ a}}{b}$ $\quad \frac{db}{dx_1} = \frac{1}{x_1}$ $\quad \frac{dy_1}{db} = \sin(\frac{e^{x_1\ a}}{b})\frac{x_1 e^{x_1\ a}}{b^2}$ etc…

Plug things in / simplify

A mess 😁

Influence Diagrams

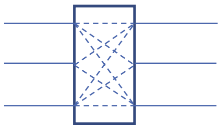
Questions?
What isn't clear?

A little painful, but algorithmic:

Break things up

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

Draw the influence diagram



Write out paths using the diagram / chain rule

$$\nabla_{x1} L = \frac{dy_1}{dx_1}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_1}{da}\frac{dL}{dy_1} + \frac{db}{dx_1}\frac{dy_1}{db}\frac{dL}{dy_1} + \frac{da}{dx_1}\frac{dy_2}{da}\frac{dL}{dy_2} + \frac{db}{dx_1}\frac{dy_2}{db}\frac{dL}{dy_2} + \frac{da}{dx_1}\frac{dy_3}{da}\frac{dL}{dy_3} + \frac{db}{dx_1}\frac{dy_3}{db}\frac{dL}{dy_3}$$

Calculate necessary derivatives

$$\frac{dy_1}{dx_1} = -\sin(\frac{e^{x_1\, a}}{b})\frac{ae^{x_1\, a}}{b} \quad \frac{da}{dx_1} = 1 \quad \frac{dy_1}{da} = -\sin(\frac{e^{x_1\, a}}{b})\frac{x_1 e^{x_1\, a}}{b} \quad \frac{db}{dx_1} = \frac{1}{x_1} \quad \frac{dy_1}{db} = \sin(\frac{e^{x_1\, a}}{b})\frac{x_1 e^{x_1\, a}}{b^2}$$ etc...

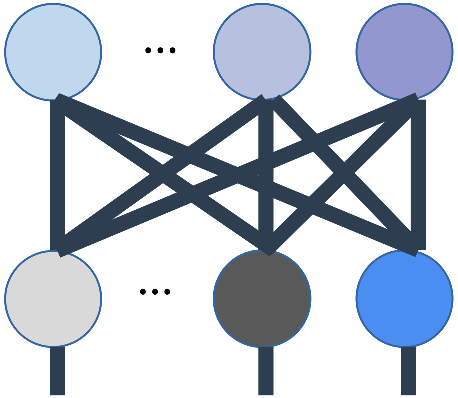
Plug things in / simplify

A mess 😁

Computational Graphs

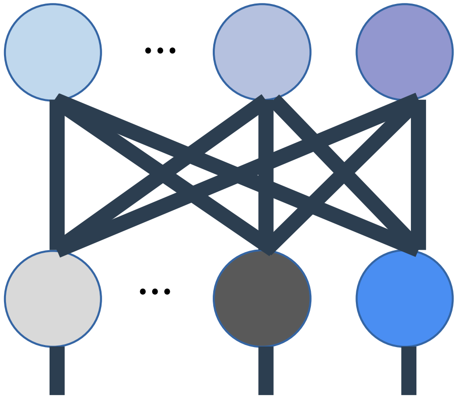
Feel free to follow the backprop part on paper.

Simple MLP



An MLP with one tanh
activated hidden layer

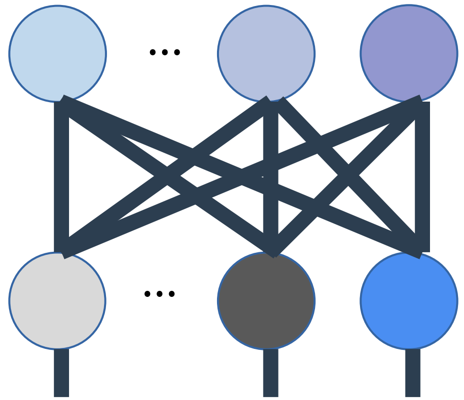
Simple MLP



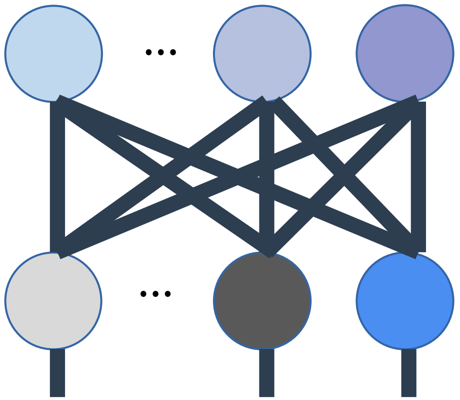
We want to easily compute the derivative with respect to the weights Wi and biases bi

Simple MLP

Linear	$z = W_1 x + b_1$
Activation	$\mathrm{out} = \tanh(z)$



Simple MLP

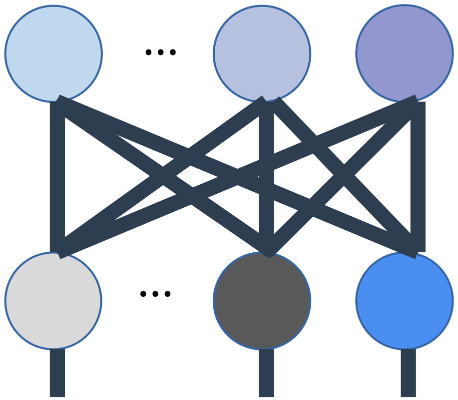


$$z = W_1 x + b_1$$
$$\text{out} = \tanh(z)$$

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

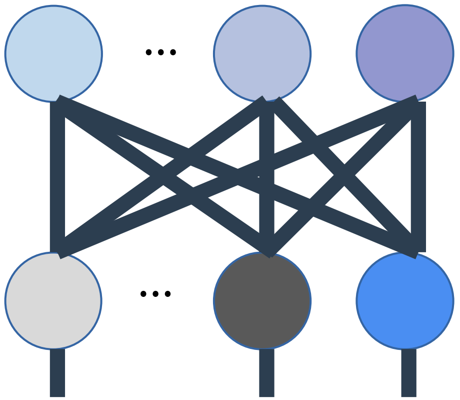
Simple MLP

Linear	$z_1 = W_1 x$ $z_2 = z_1 + b_1$
Activation	$\text{out} = \tanh(z_2)$



Simple MLP

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

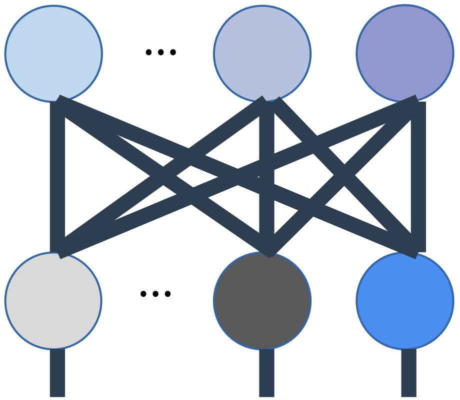


This allows us to **reuse rules for propagating derivatives** through simple functions like +, *

Simple MLP

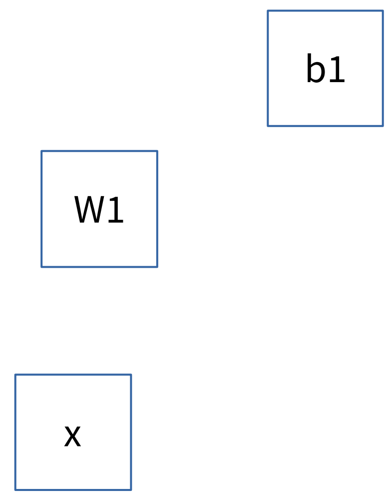
$$z_1 = W_1 x$$
$$z_2 = z_1 + b_1$$
$$\text{out} = \tanh(z_2)$$

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



Simple MLP

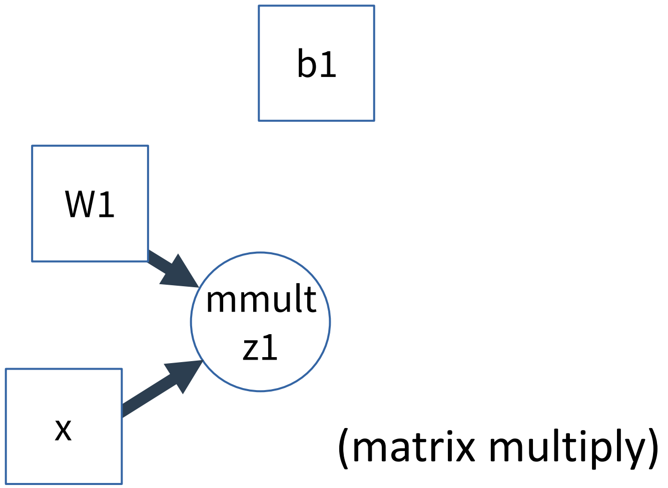
$$z_1 = W_1 x$$
$$z_2 = z_1 + b_1$$
$$\text{out} = \tanh(z_2)$$



Our initial variables

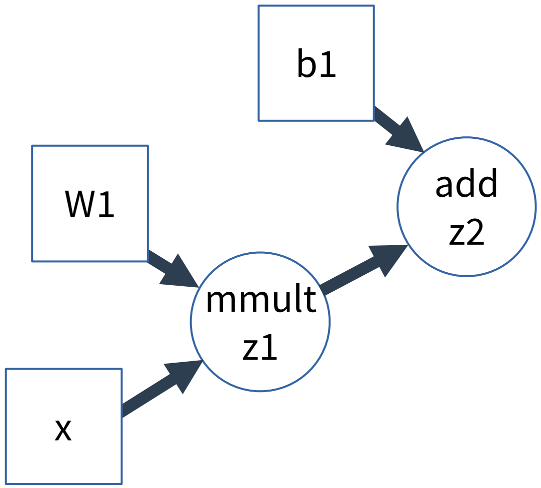
Simple MLP

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



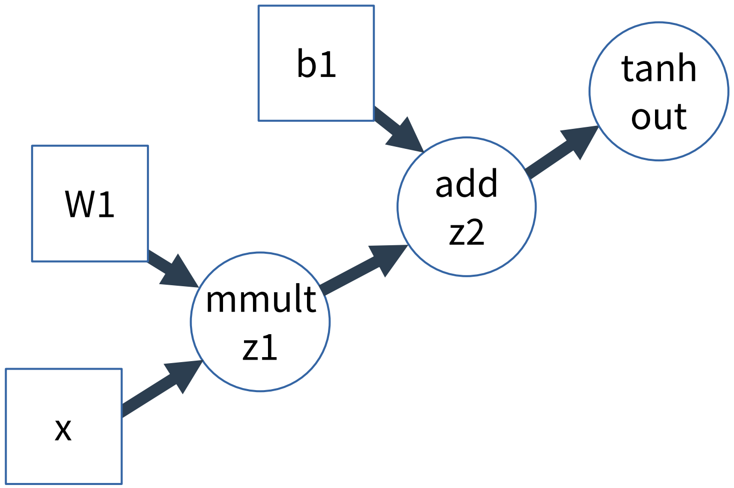
Simple MLP

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



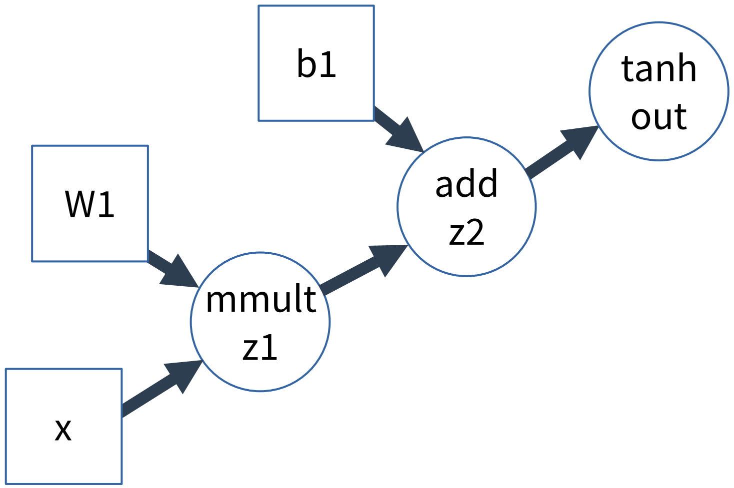
Simple MLP

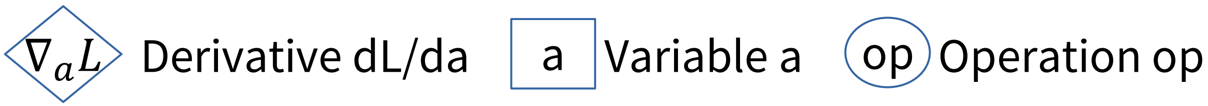
$$z_1 = W_1 x$$
$$z_2 = z_1 + b_1$$
$$\text{out} = \tanh(z_2)$$



Simple MLP

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

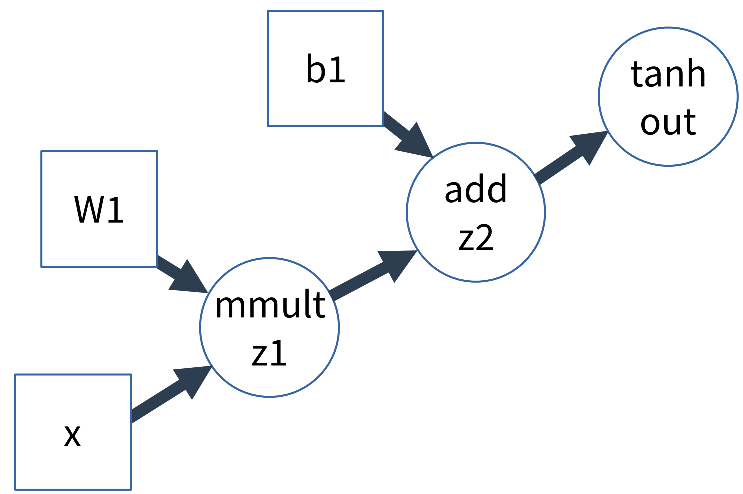




Simple MLP

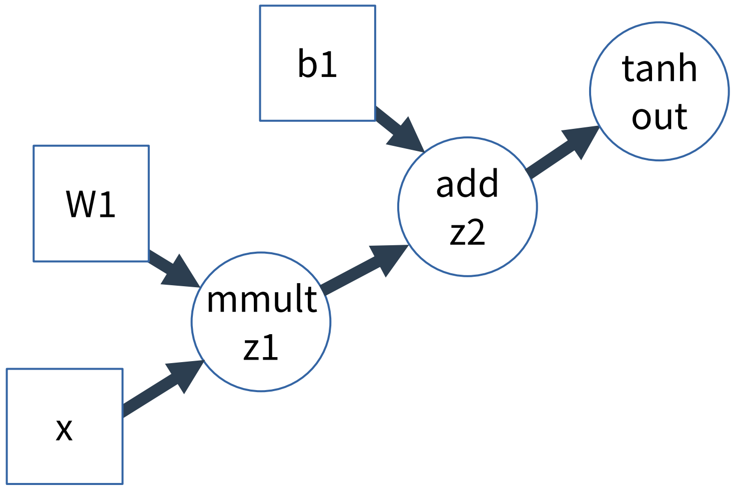
Questions?
What isn't clear?

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



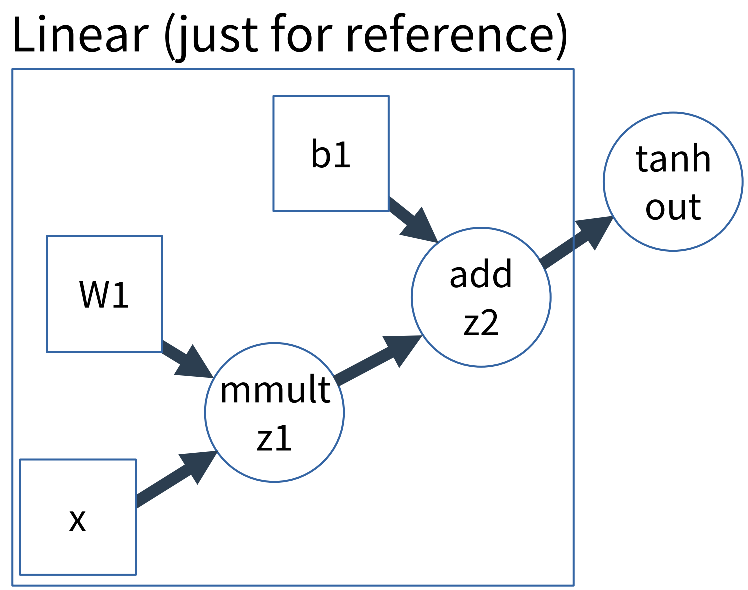
$\nabla_a L$ Derivative dL/da a Variable a op Operation op

Simple MLP


b1
tanh
out
W1
add
z2
mmult
z1
x

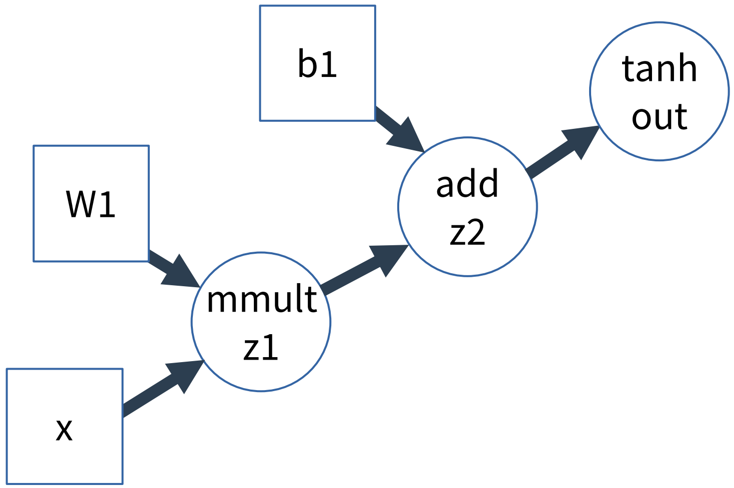

$\nabla_a L$ Derivative dL/da
a Variable a
op Operation

Simple MLP



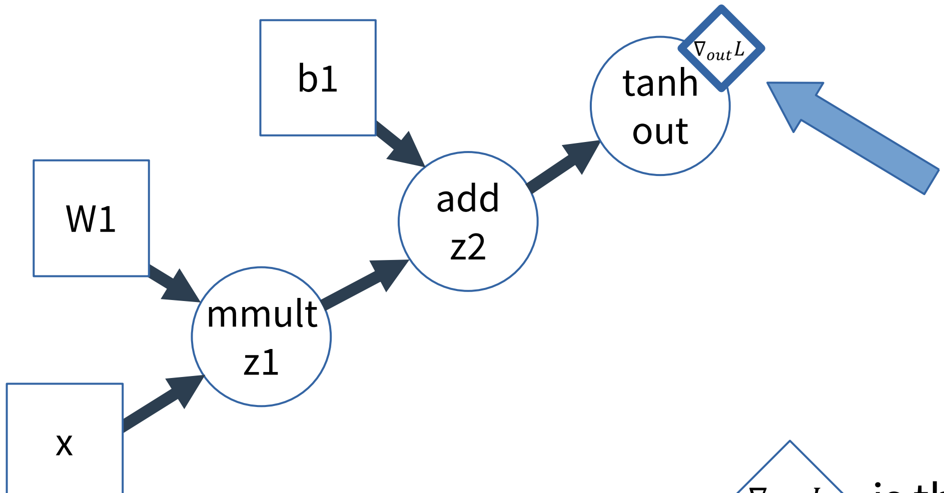
$\nabla_a L$ Derivative dL/da a Variable a op Operation

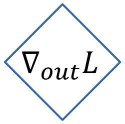
Simple MLP


b1
tanh
out
W1
add
z2
mmult
z1
x


$\nabla_a L$ Derivative dL/da
a Variable a
op Operation

Simple MLP

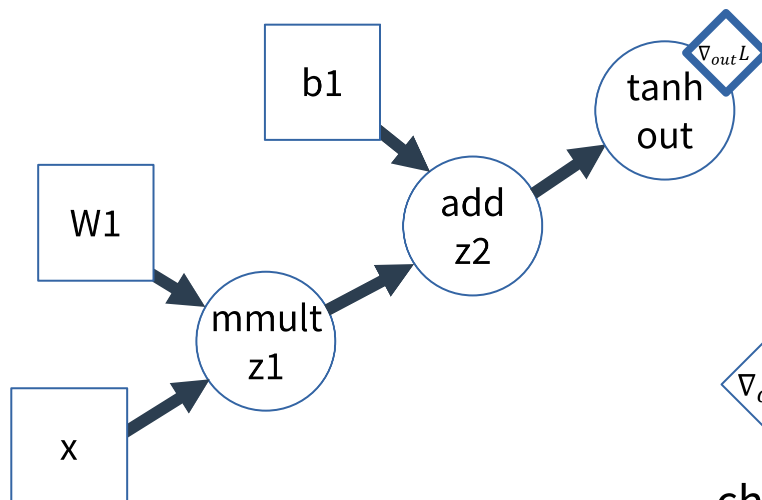




is the derivative of the loss function with respect to the output.

$\nabla_a L$ Derivative dL/da a Variable a op Operation

Simple MLP

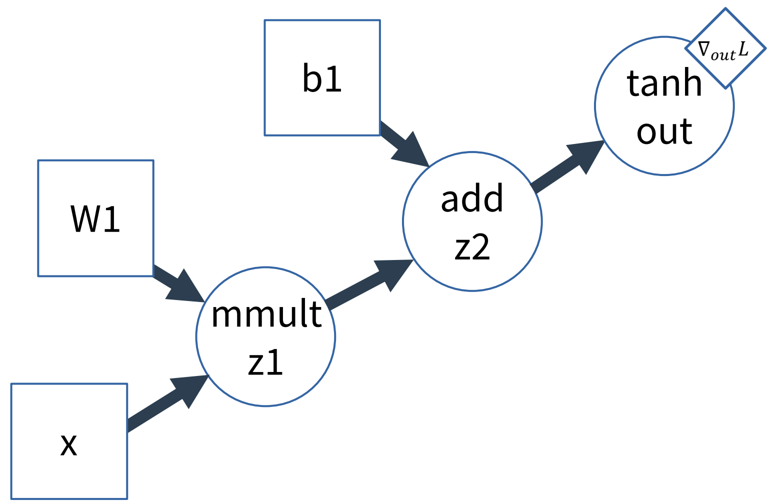


$\nabla_{out}L$ is calculated from our chosen loss function.

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

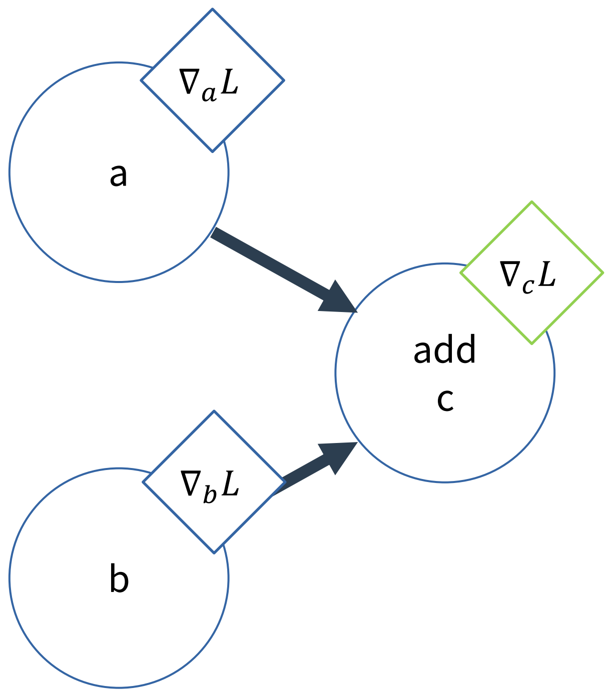
$\nabla_a L$ Derivative dL/da a Variable a op Operation

Simple MLP

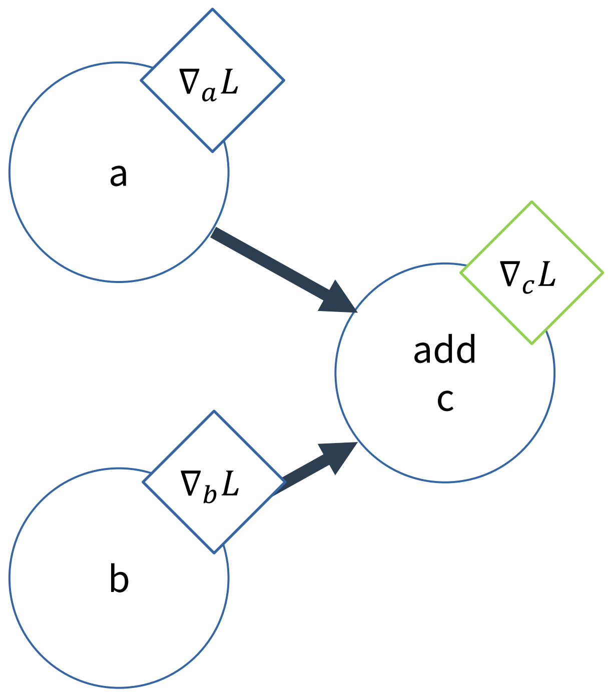


$\nabla_a L$ Derivative dL/da a Variable a op Operation

Aside: Backward Functions



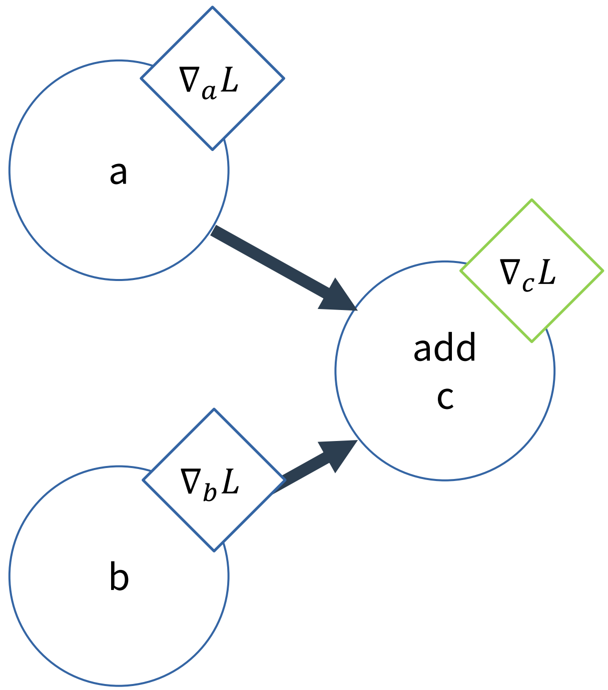
Aside: Backward Functions



With the chain rule, every operation has a **backward function** to calculate its parent's gradients

$\nabla_a L$ Derivative dL/da a Variable a op Operation

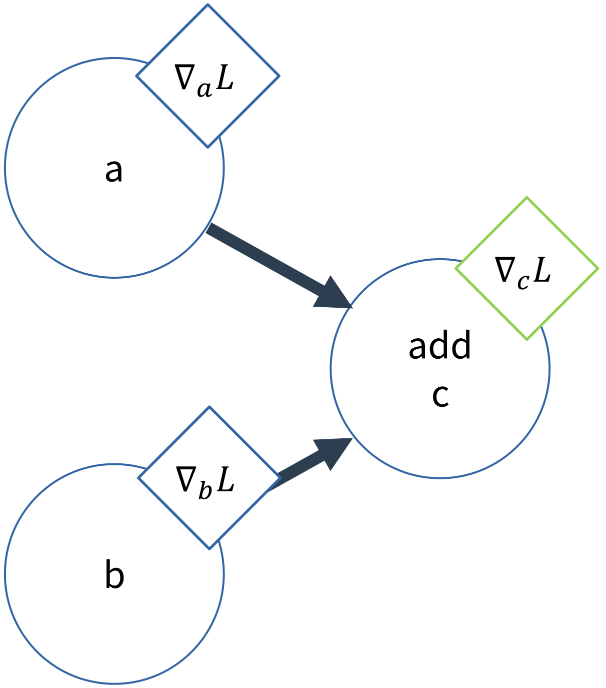
Aside: Backward Functions



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

$\nabla_a L$ Derivative dL/da   a Variable a   op Operation

Aside: Backward Functions

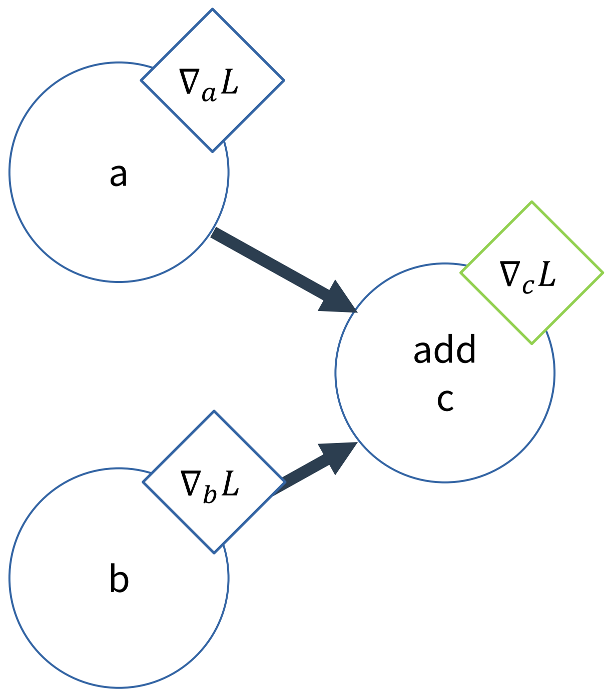


Given $\nabla_c L$, what are $\nabla_a L$ and $\nabla_a L$ in terms of the chain rule?

$$\frac{d\mathrm{L}}{da} = \frac{d\mathrm{L}}{dc} * \frac{dc}{da}$$

$\nabla_a L$ Derivative dL/da a Variable a op Operation

Aside: Backward Functions

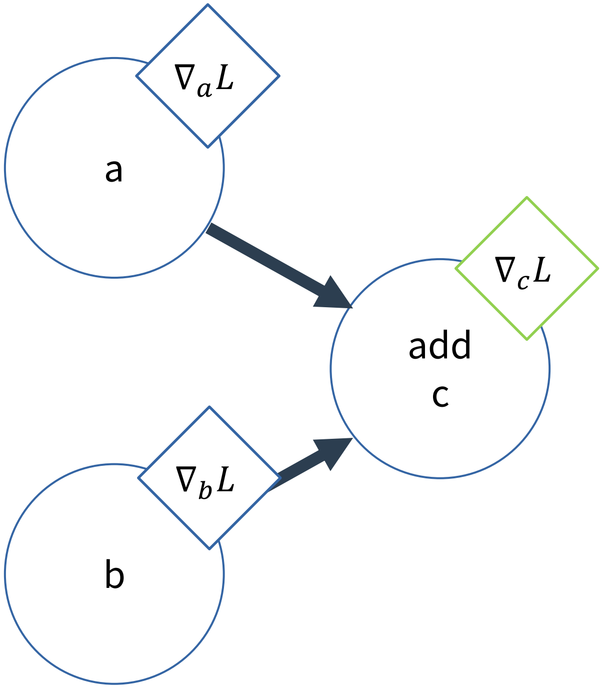


Given $\nabla_c L$, what are $\nabla_a L$ and $\nabla_a L$ in terms of the chain rule?

$$\frac{d\mathrm{L}}{da} = \frac{d\mathrm{L}}{dc} * \frac{dc}{da} \qquad \frac{d\mathrm{L}}{d\mathrm{b}} = \frac{d\mathrm{L}}{dc} * \frac{dc}{d\mathrm{b}}$$

$\nabla_a L$ Derivative dL/da a Variable a op Operation

Aside: Backward Functions

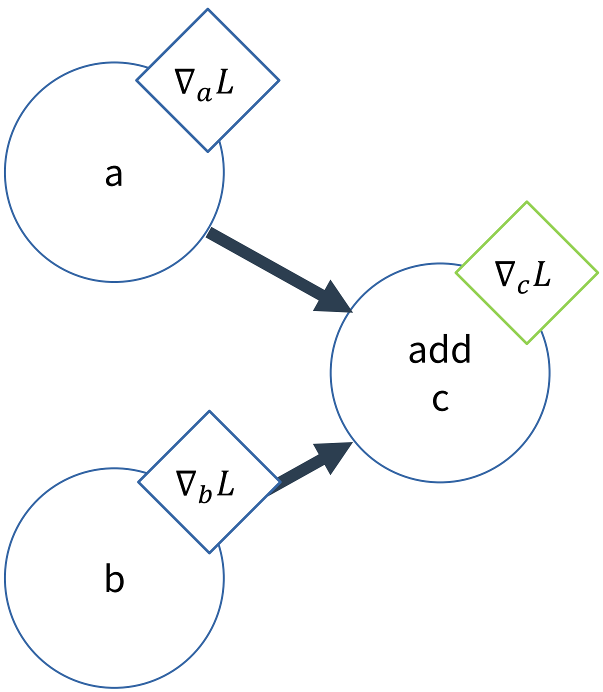


What do these simplify to? Hint: c=a+b, what's dc/da

$$\frac{d\mathrm{L}}{d\mathrm{a}} = \frac{d\mathrm{L}}{d\mathrm{c}} * \frac{d\mathrm{c}}{d\mathrm{a}} \qquad \frac{d\mathrm{L}}{d\mathrm{b}} = \frac{d\mathrm{L}}{d\mathrm{c}} * \frac{d\mathrm{c}}{d\mathrm{b}}$$

$\nabla_a L$ Derivative dL/da a Variable a op Operation

Aside: Backward Functions

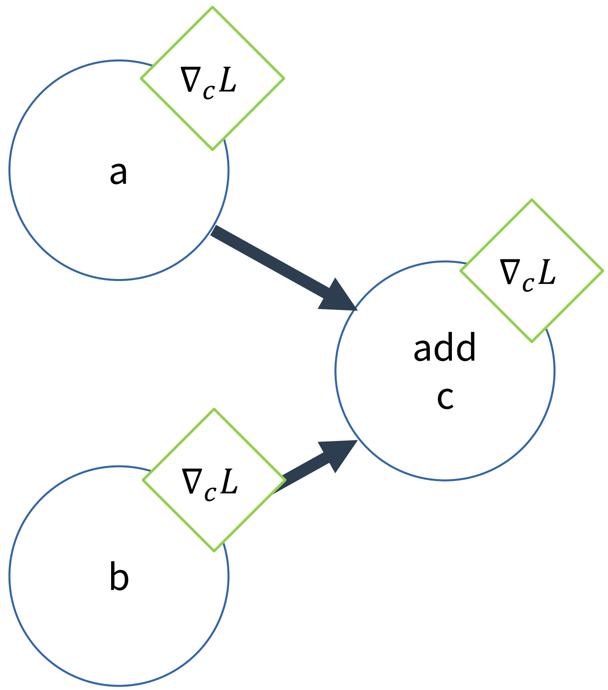


What do these simplify to? Hint: c=a+b, what's dc/da

$$\frac{dL}{da} = \frac{dL}{dc} * \cancel{\frac{dc}{da}} = \frac{dL}{dc} \qquad \frac{dL}{db} = \frac{dL}{dc} * \cancel{\frac{dc}{db}} = \frac{dL}{dc}$$

$\nabla_a L$ Derivative dL/da a Variable a op Operation

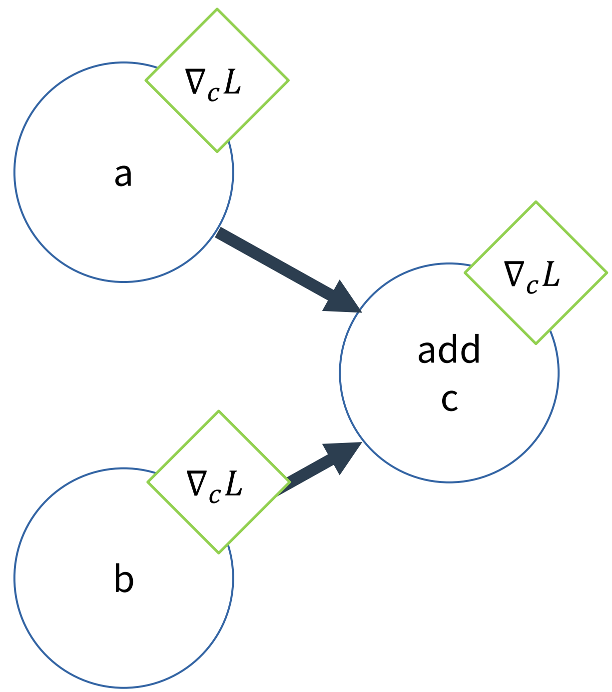
Aside: Backward Functions



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

Aside: Backward Functions

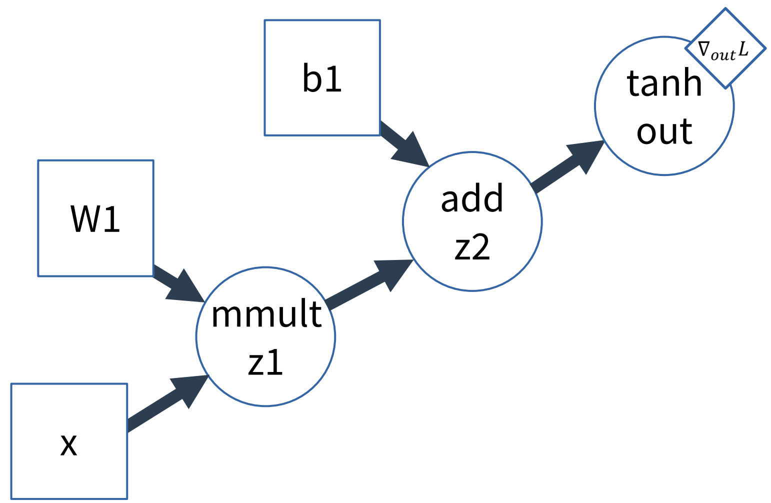
Questions?
What isn't clear?



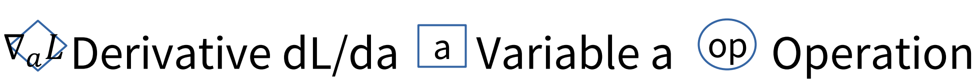
Add's **backward function** is to pass the gradient back unchanged

$\nabla_a L$ Derivative dL/da a Variable a op Operation

Simple MLP



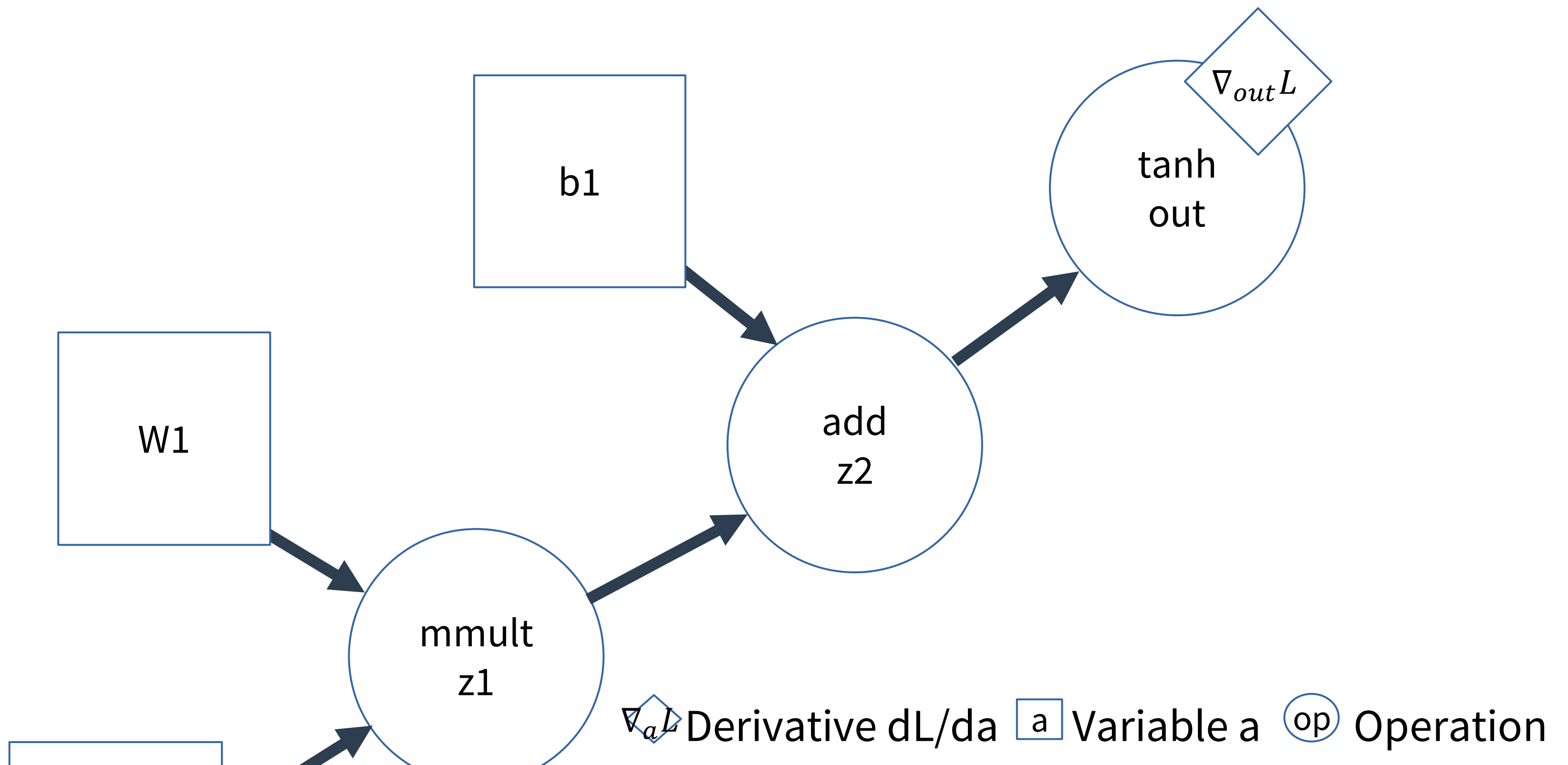
b1
W1
x
mmult
z1
add
z2
tanh
out
$\nabla_{out} L$



$\nabla_a L$ Derivative dL/da
a Variable a
op Operation

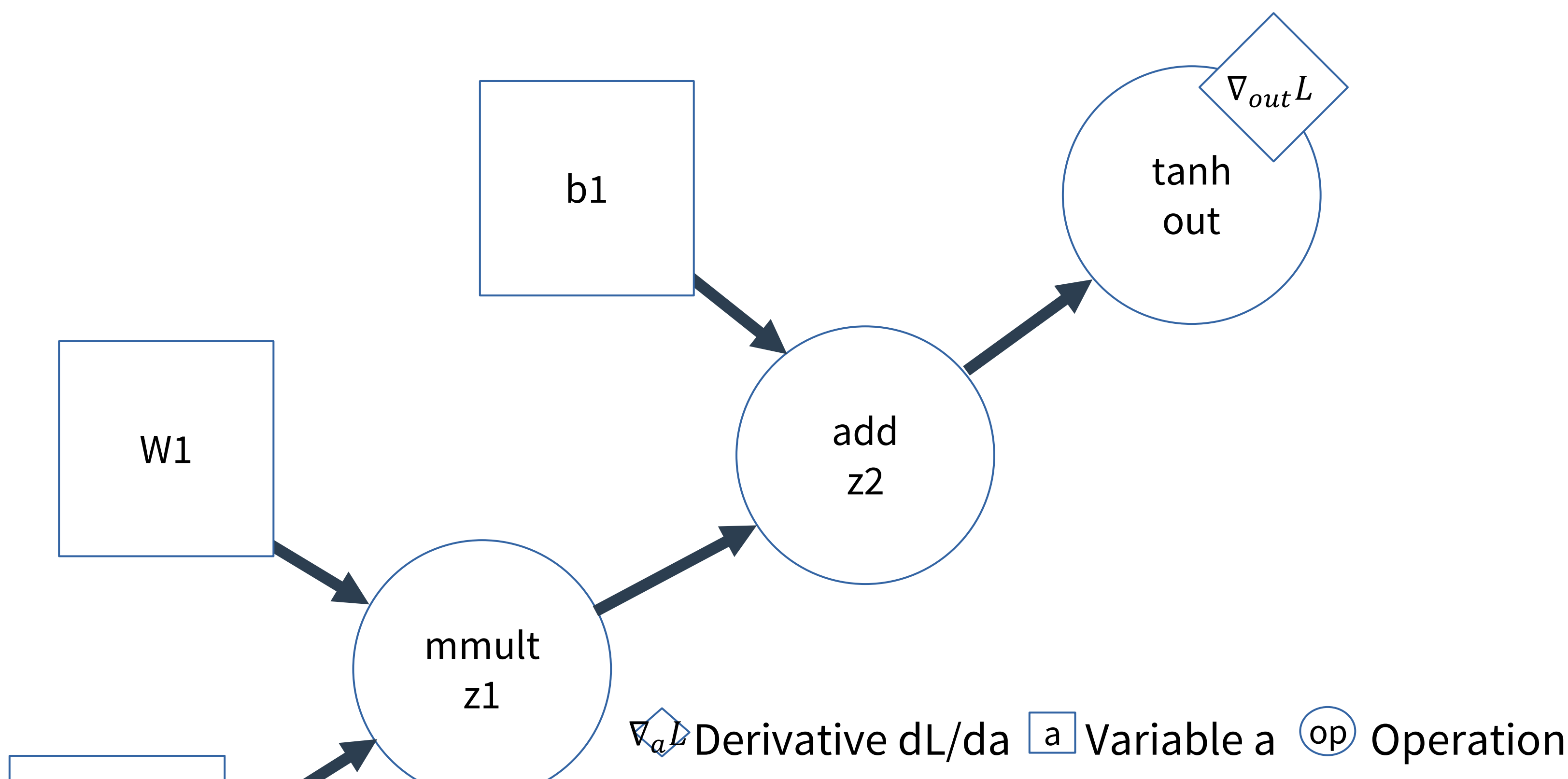
Simple MLP

We will perform a graph search from the end, updating derivatives as we go. DFS is easiest.



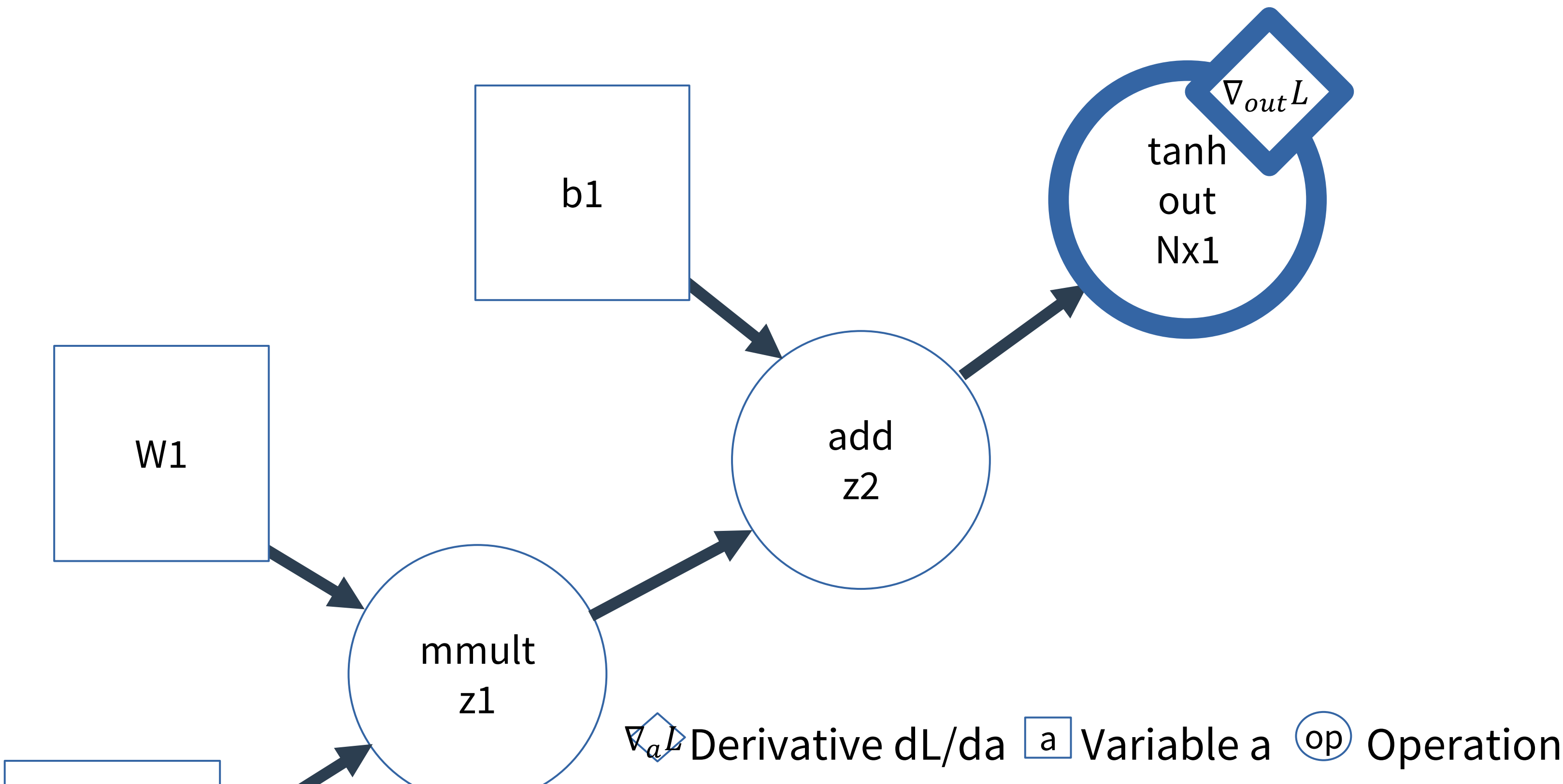
Simple MLP

Feel free to follow along on paper.



Simple MLP

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

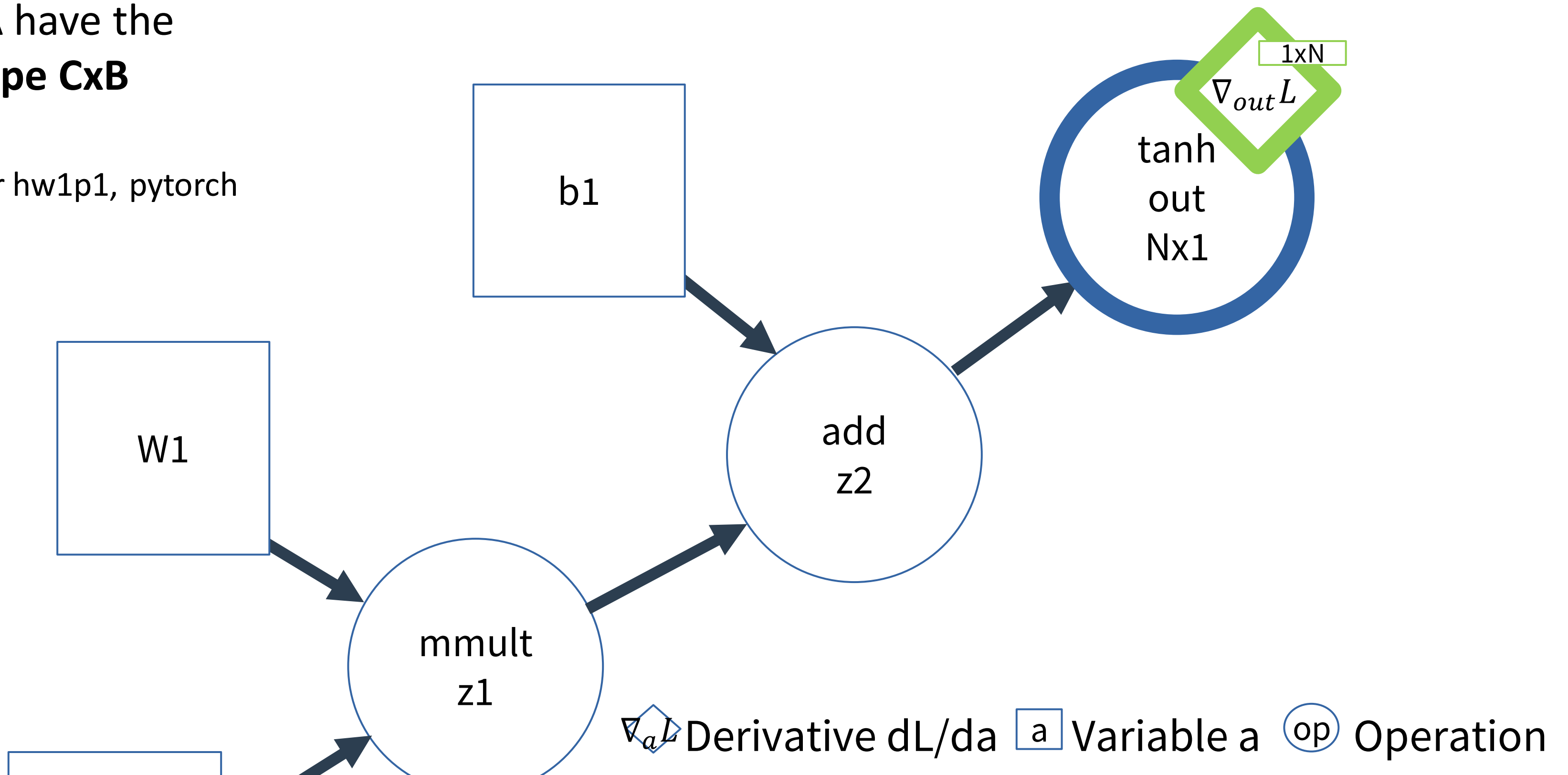


Simple MLP

If the output, 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**

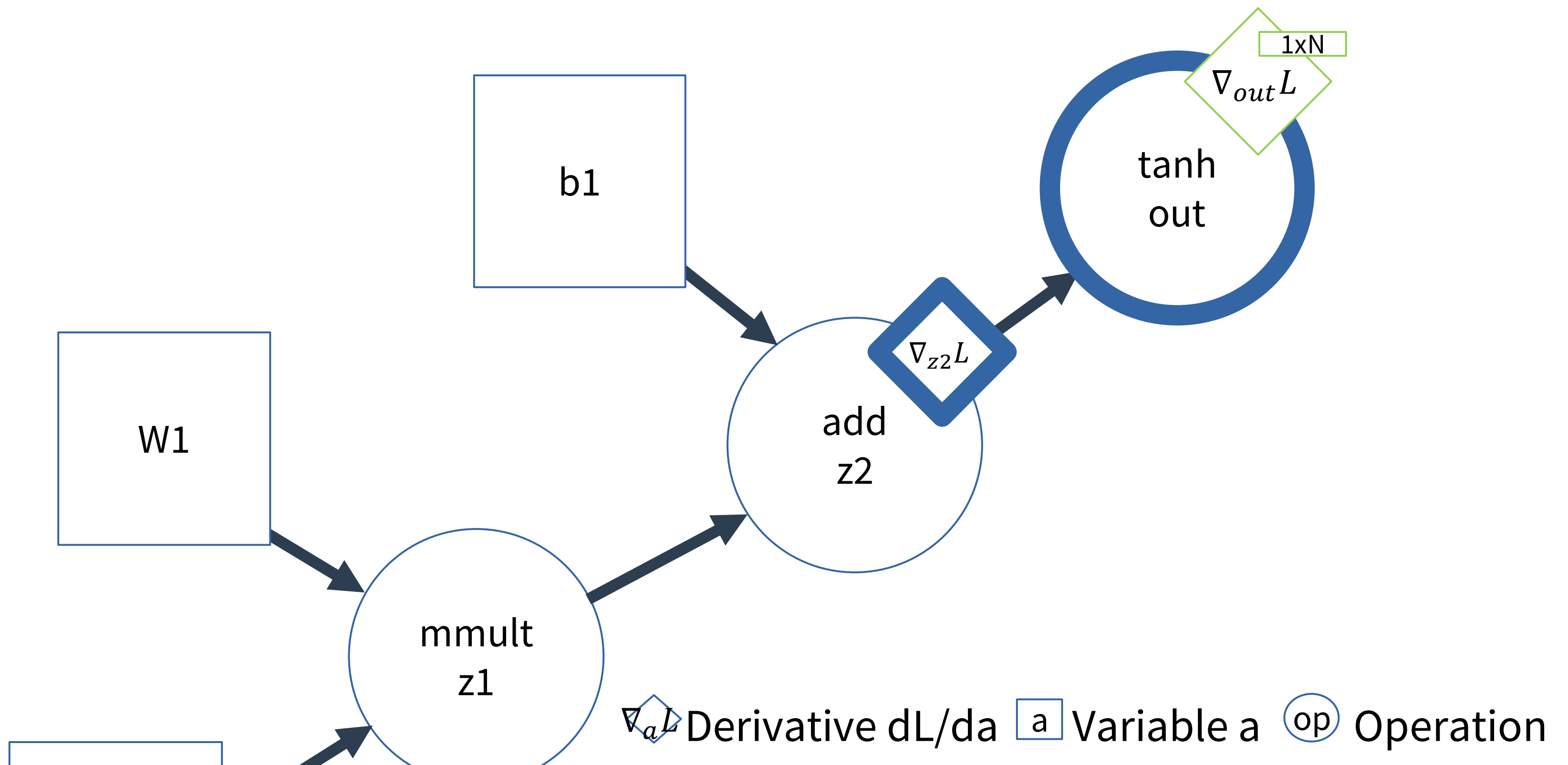
* shape isn't transpose for hw1p1, pytorch



Simple MLP

In terms of the chain rule, what is the backward function of tanh out ?

I.e., in terms of the chain rule, what is $\nabla_{z2} L$?



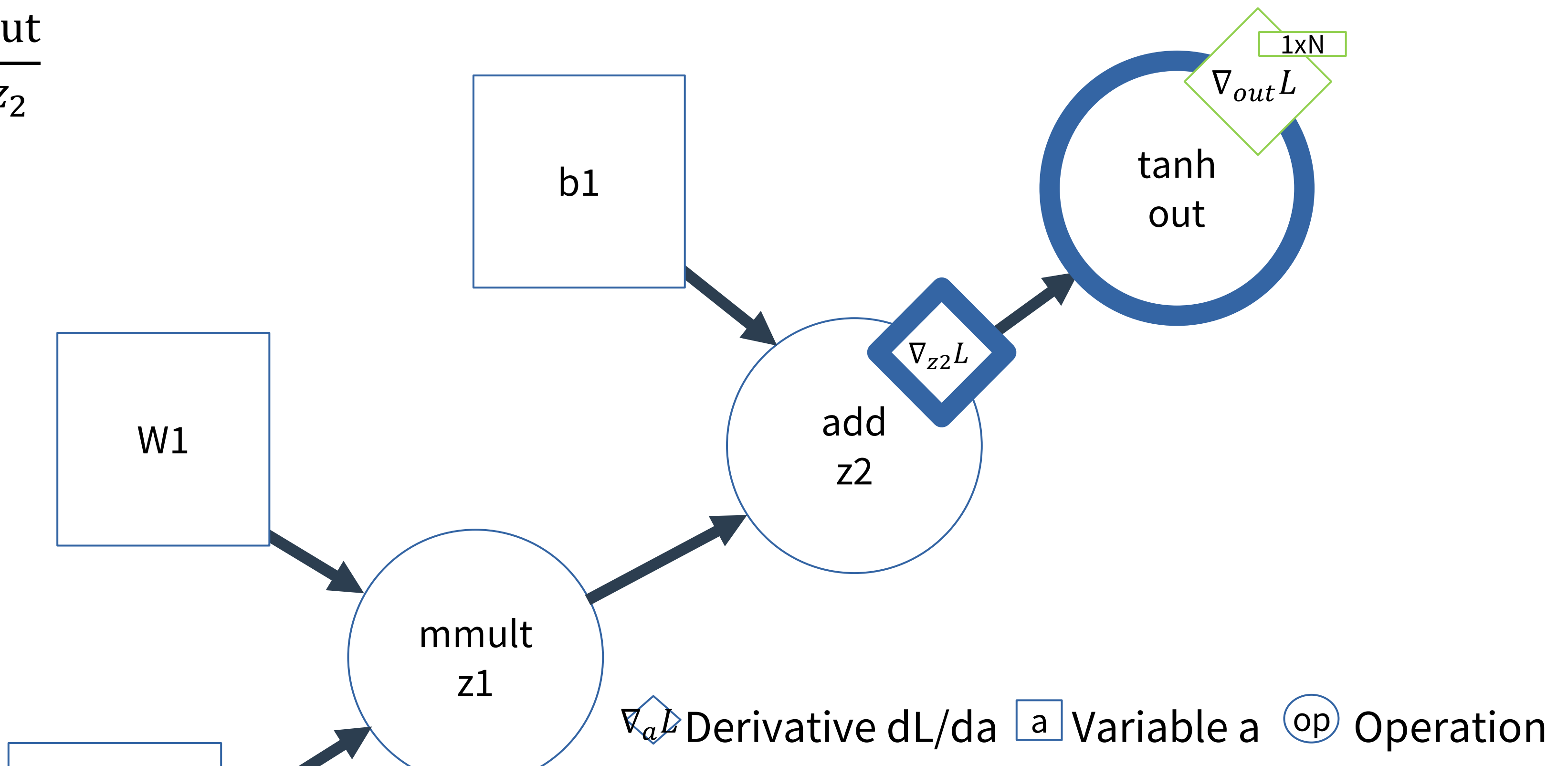
Simple MLP

In terms of the chain rule, what is the backward

function of tanh out ?

I.e., in terms of the chain rule, what is $\nabla_{z2}L$?

$$\frac{d\mathrm{L}}{dz_2} = \frac{d\mathrm{L}}{\mathrm{dout}} * \frac{dout}{dz_2}$$



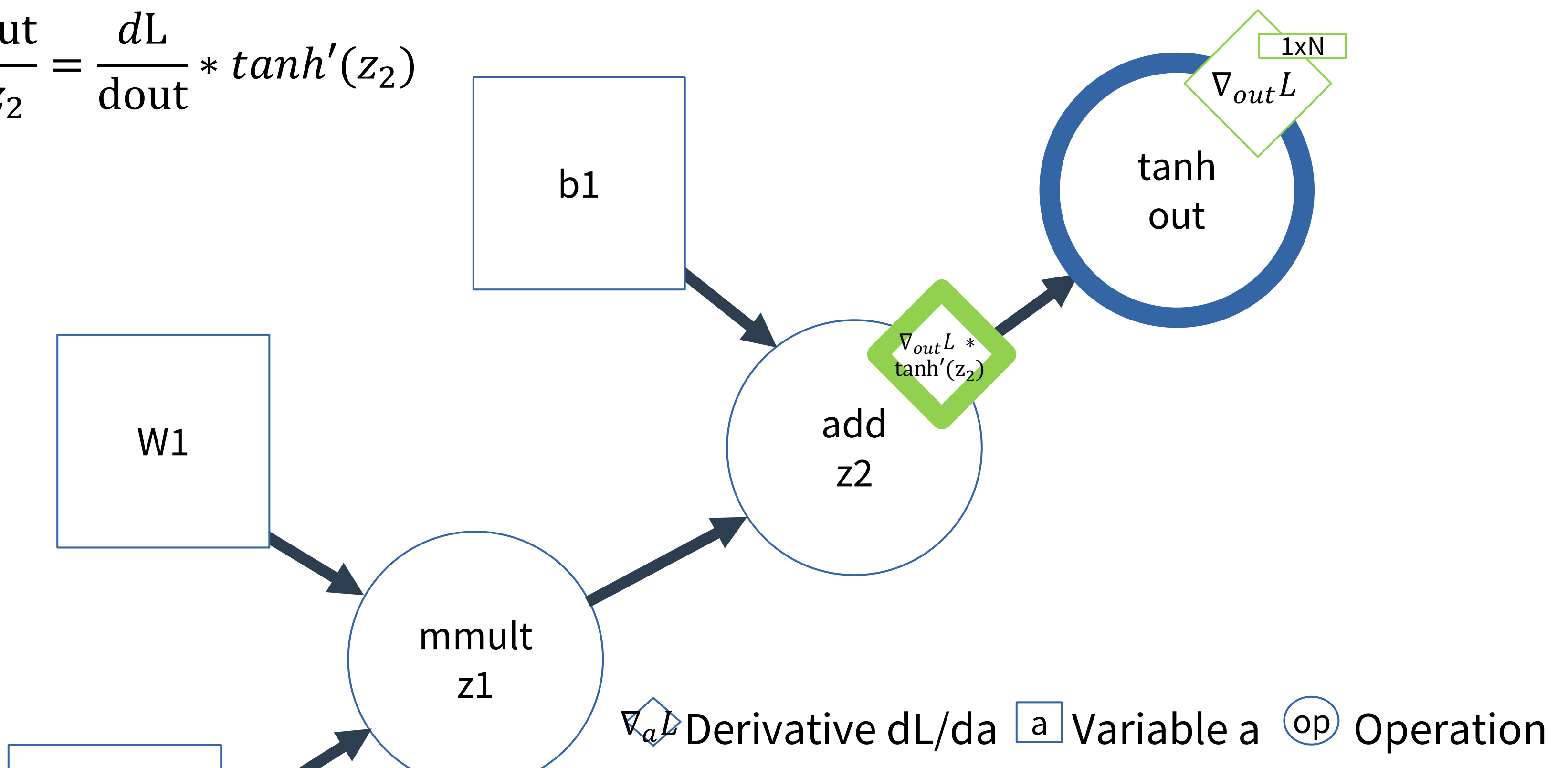
Simple MLP

In terms of the chain rule, what is the backward

function of tanh out ?

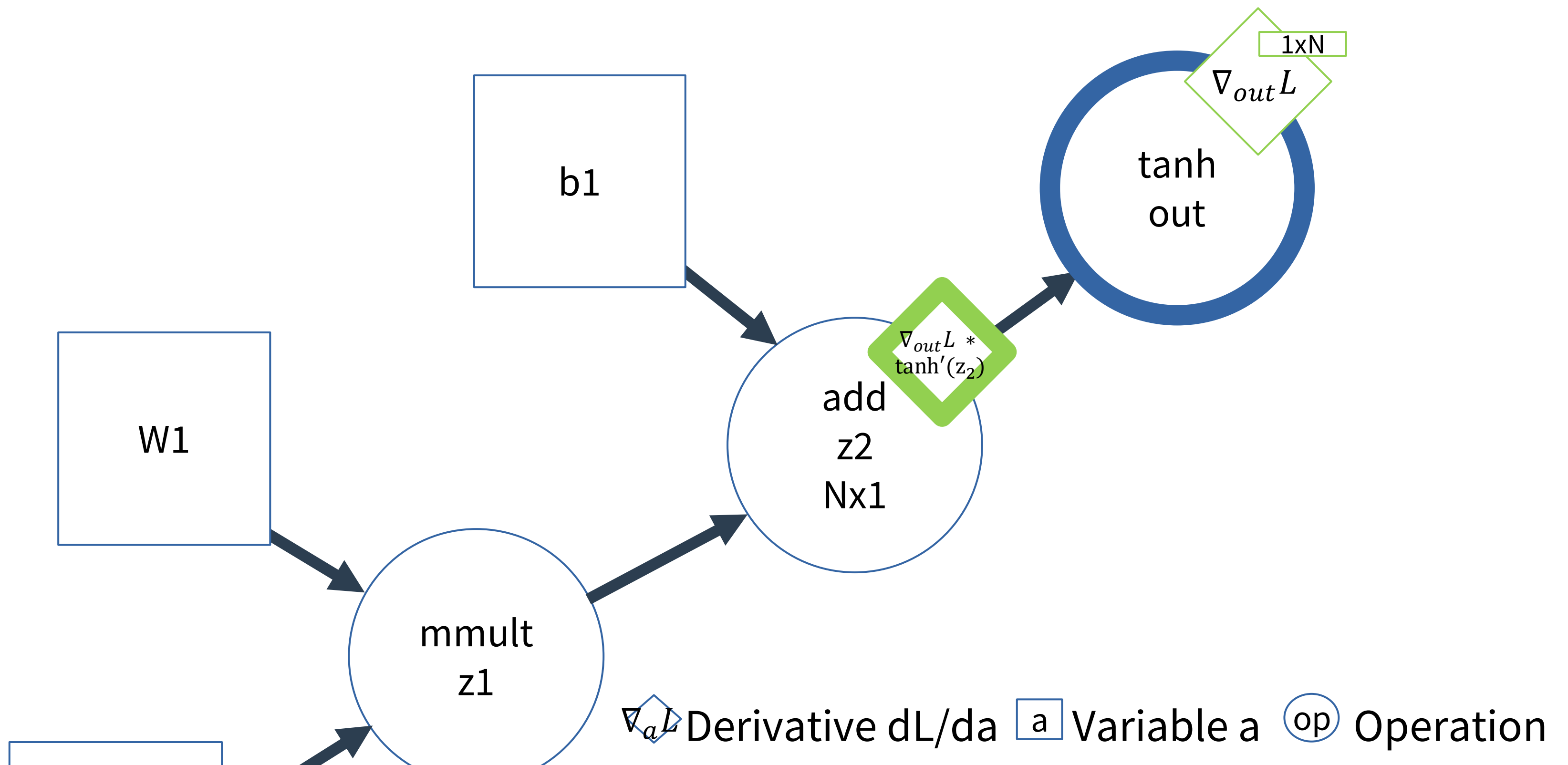
I.e., in terms of the chain rule, what is $\nabla_{z2}L$?

$$\frac{d\mathrm{L}}{dz_2} = \frac{d\mathrm{L}}{\mathrm{dout}} * \frac{\mathrm{dout}}{dz_2} = \frac{d\mathrm{L}}{\mathrm{dout}} * tanh'(z_2)$$



Simple MLP

What is the shape of $\nabla_{z2}L$?

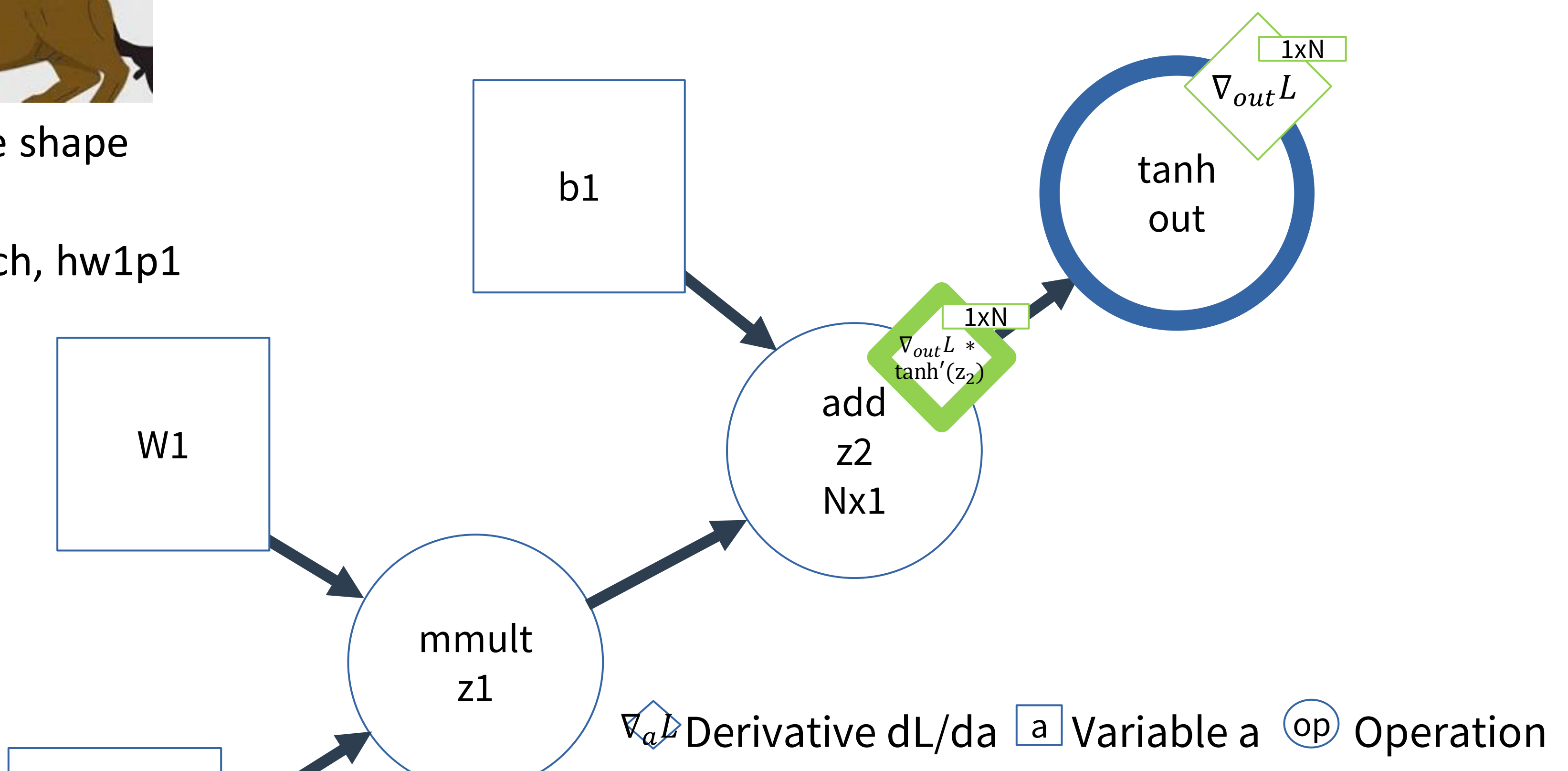


Simple MLP

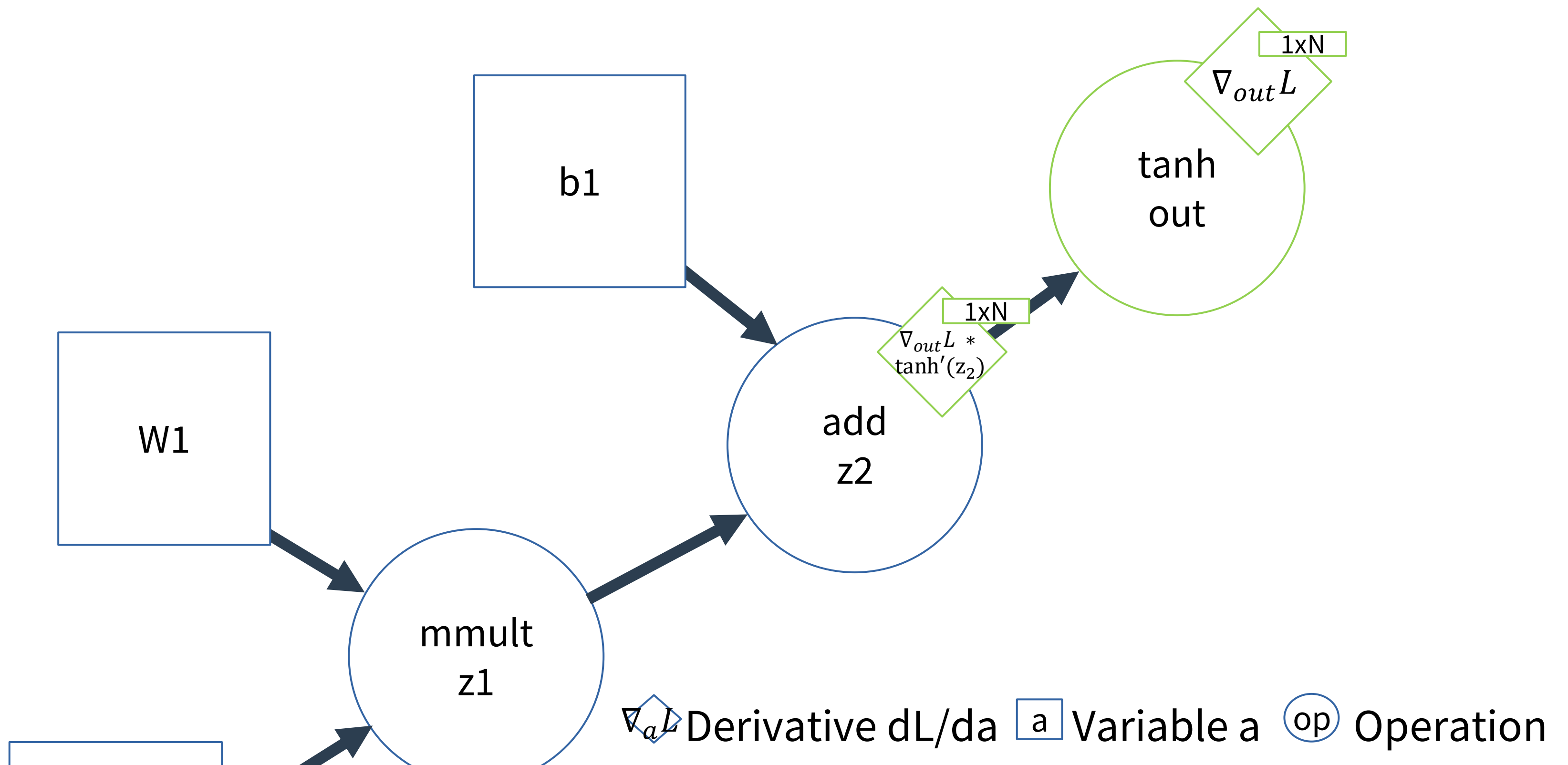
What is the shape of $\nabla_{z2} L$?

The transpose shape

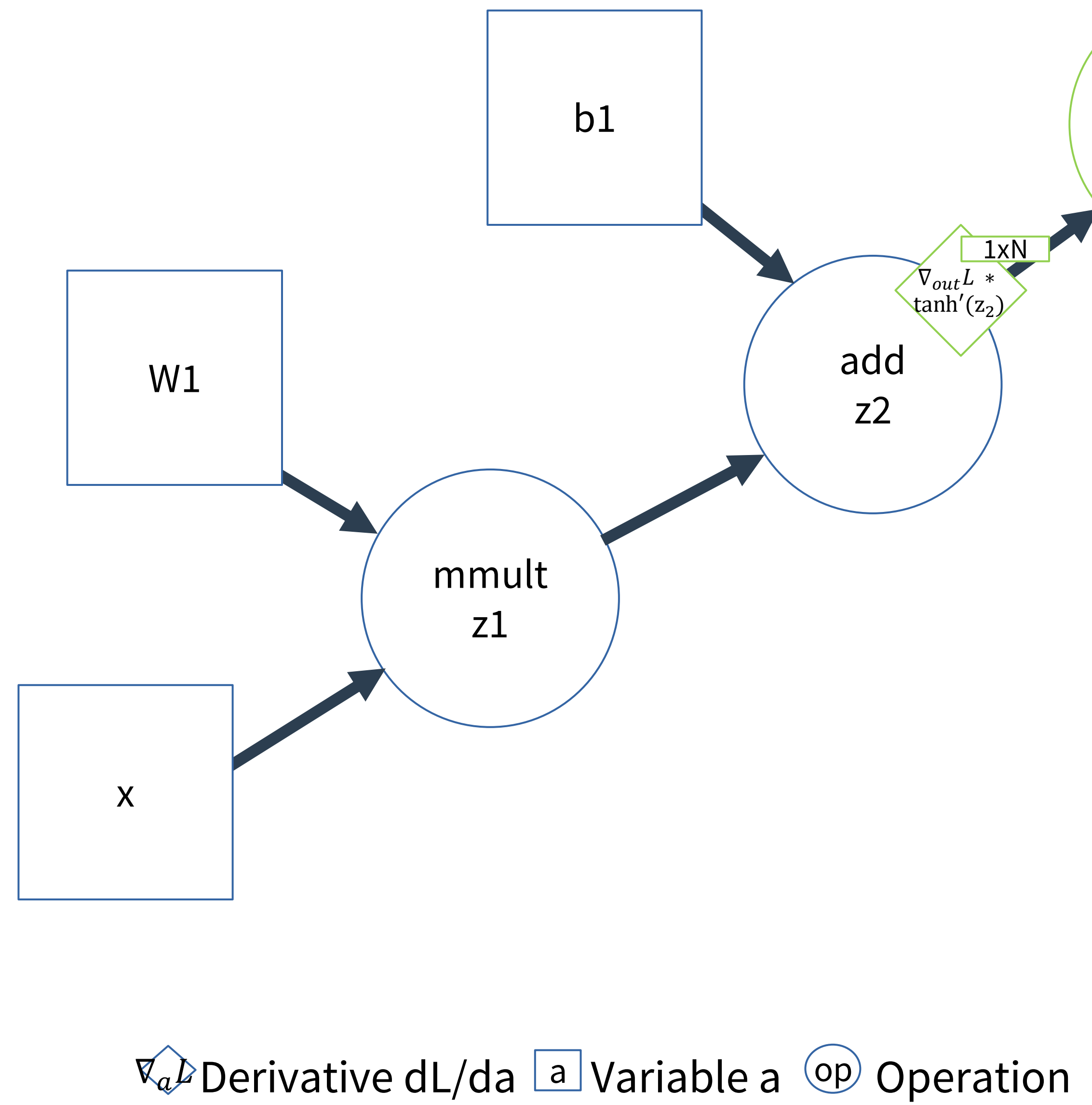
*except in pytorch, hw1p1



Simple MLP


1xN
$\nabla_{out}L$
tanh
out
b1
1xN
$\nabla_{out}L$ *
$\tanh'(z_2)$
add
z2
W1
mmult
z1
$\nabla_a L$ Derivative dL/da
a Variable a
op Operation

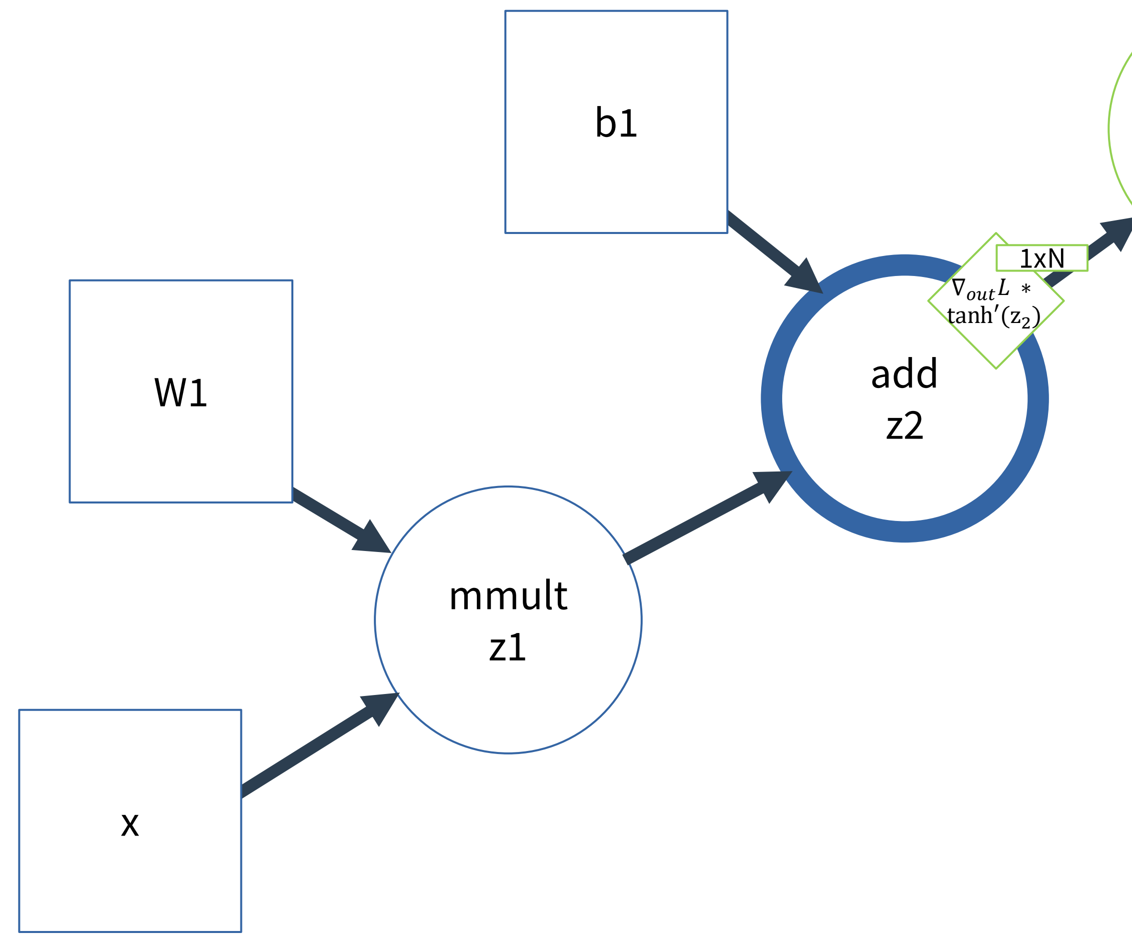
Simple MLP



b1
W1
x
mmult
z1
add
z2
1xN
$\nabla_{out}L * \tanh'(z_2)$
$\nabla_a L$ Derivative dL/da
a Variable a
op Operation

Simple MLP

We will continue the graph search by visiting add z2 .



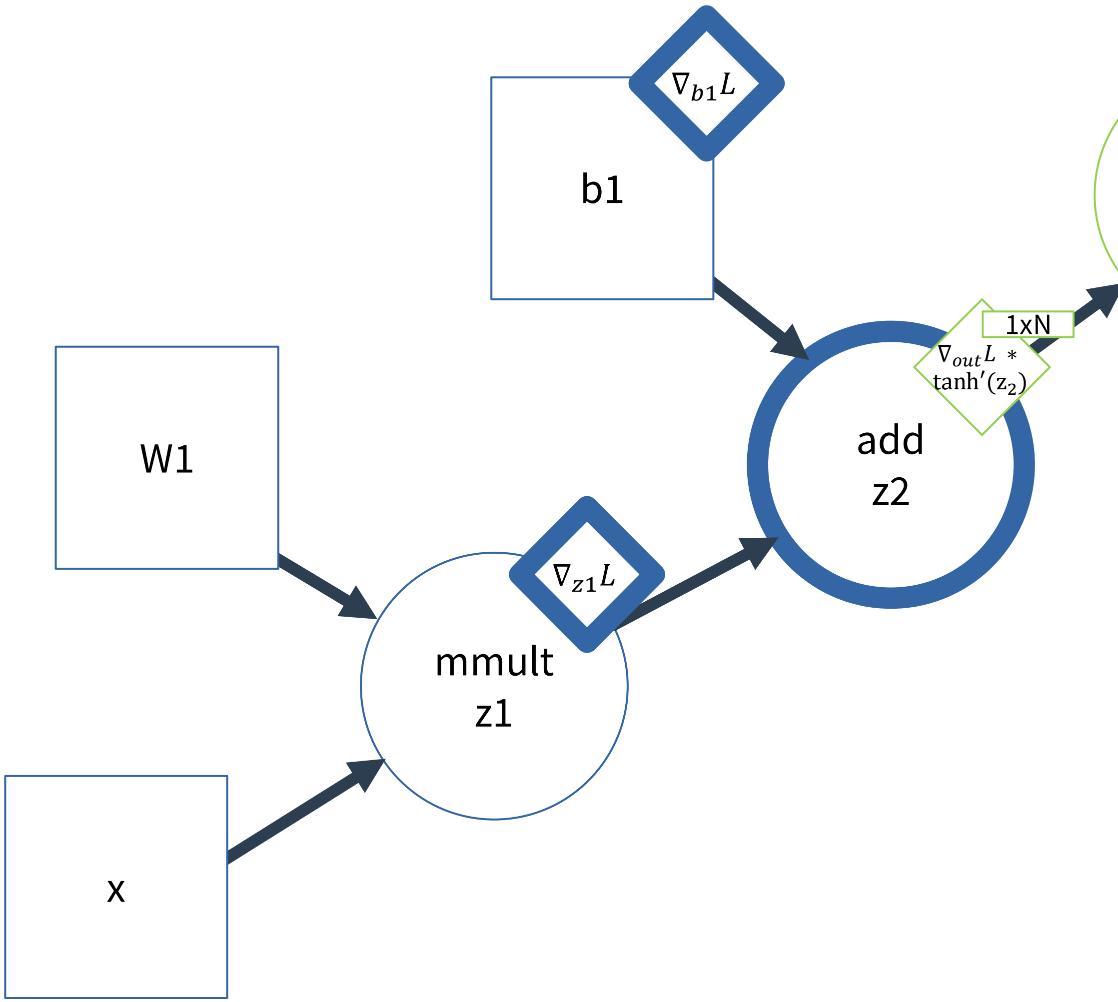
$\nabla_a L$ Derivative dL/da a Variable a op Operation

Simple MLP

What is the backward function of add z2 ?

I.e., what are $\nabla_{b1}L$ and $\nabla_{z1}L$?

Hint: $c = a + b,\quad \frac{dL}{da} = \frac{dL}{dc}\frac{dc}{da}$



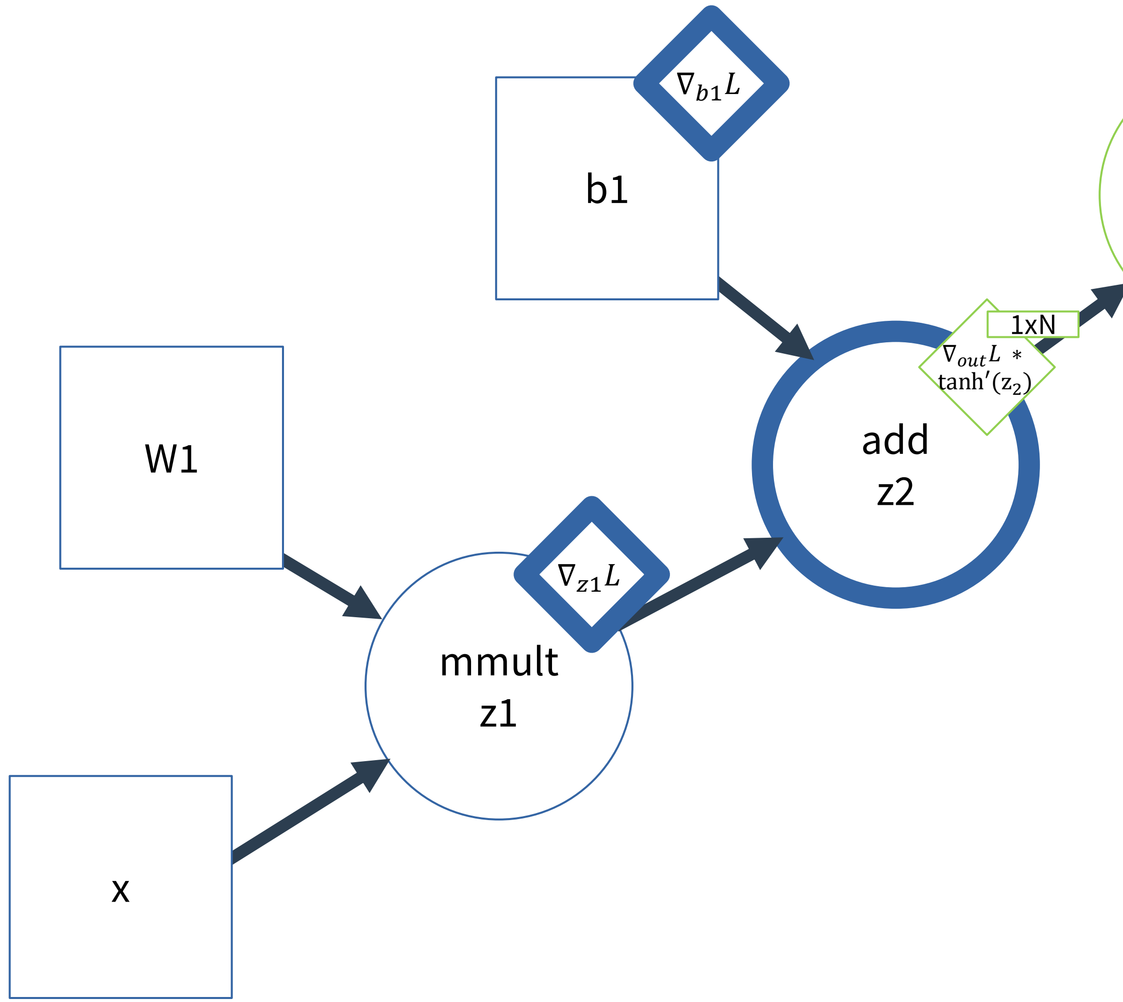
$\nabla_a L$ Derivative dL/da a Variable a op Operation

Simple MLP

What is the backward function of add z2 ?

I.e., what are $\nabla_{b1}L$ and $\nabla_{z1}L$?

Hint: $c = a + b,\quad \frac{dL}{da} = \frac{dL}{dc}\cancel{\frac{dc}{da}}$



$\nabla_a L$ Derivative dL/da a Variable a op Operation

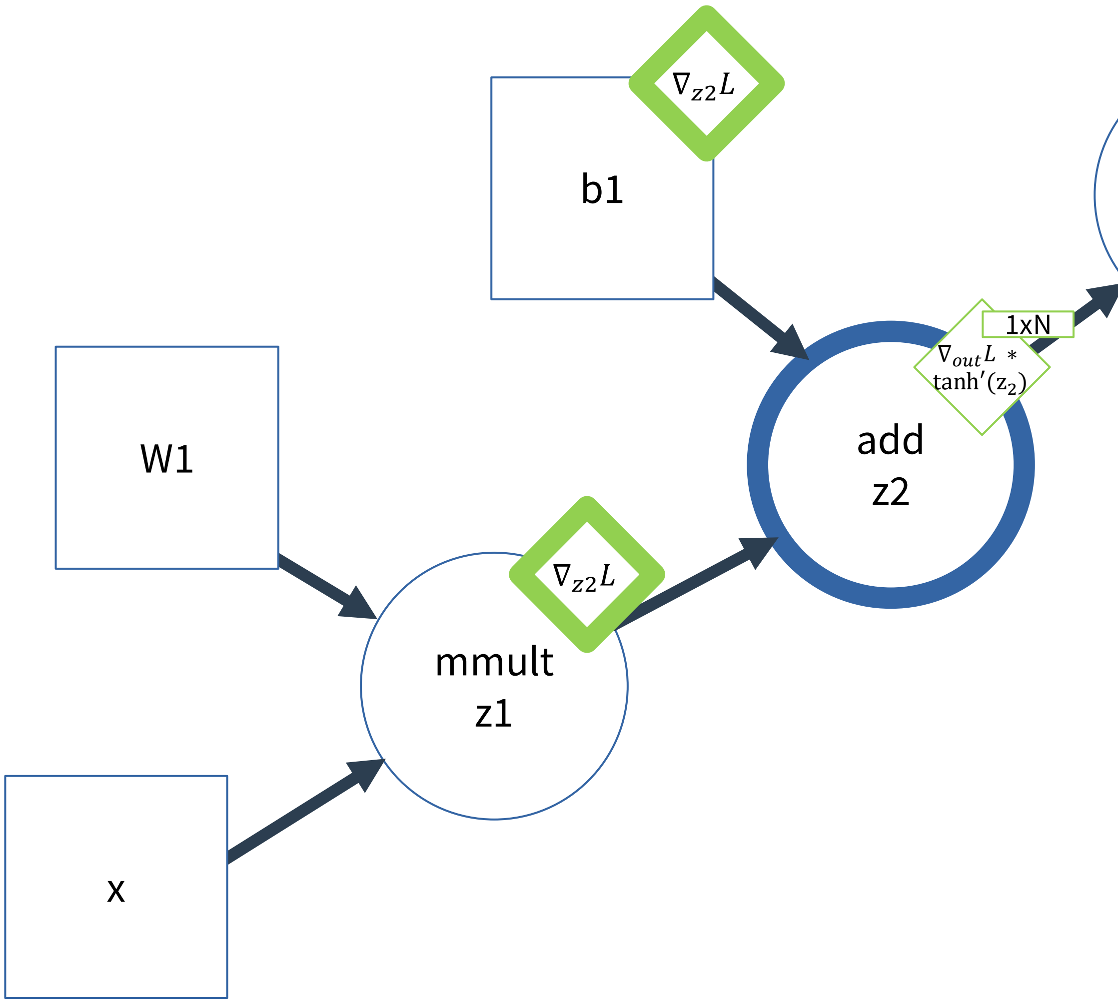
Simple MLP

What is the backward function of add z2 ?

I.e., what are $\nabla_{b1}L$ and $\nabla_{z1}L$?

Hint: $c = a + b$, $\frac{dL}{da} = \frac{dL}{dc}\cancel{\frac{dc}{da}}$

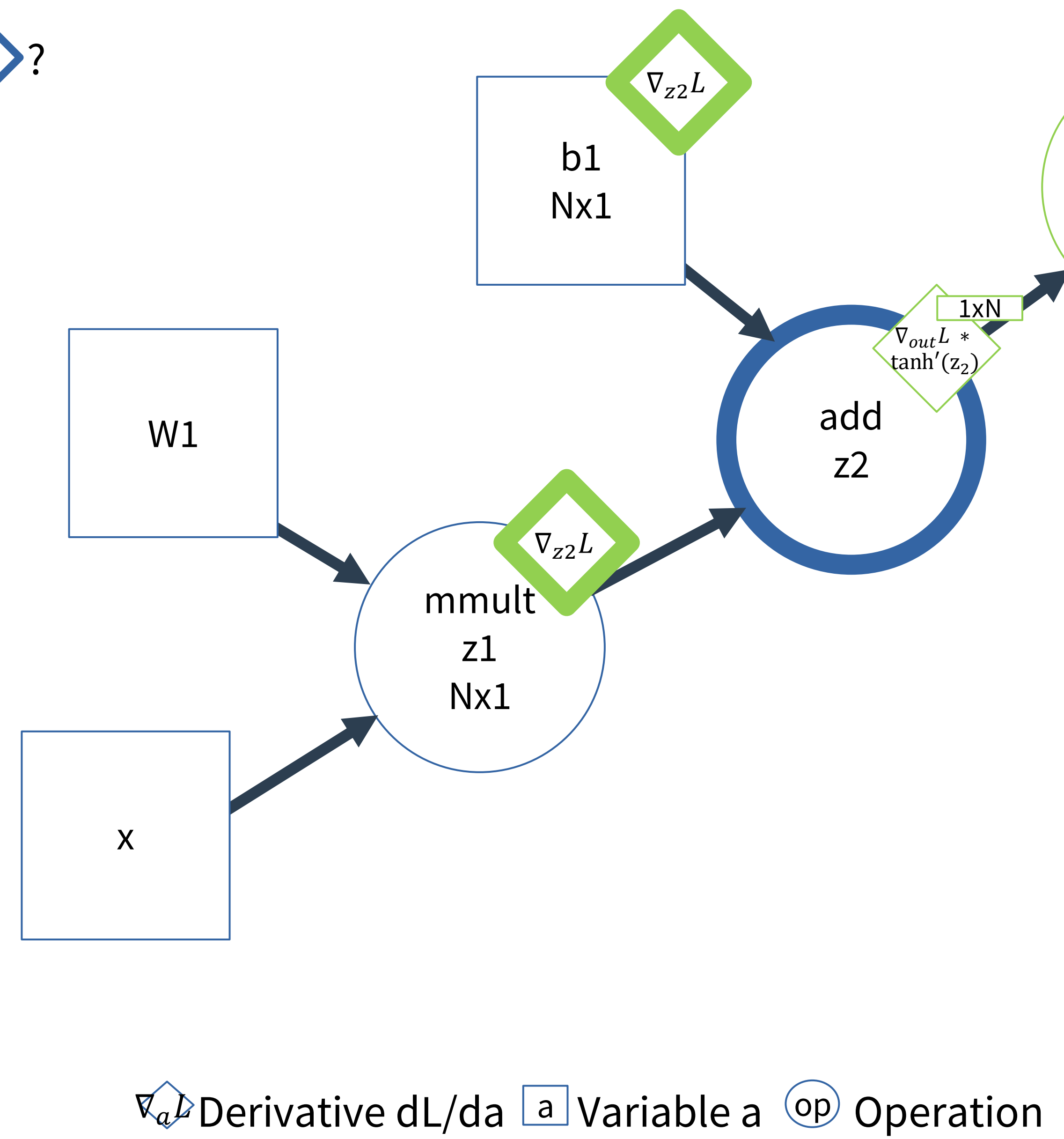
$$\frac{dL}{db_1} = \frac{dL}{dz_2}, \qquad \frac{dL}{dz_1} = \frac{dL}{dz_2}$$



$\nabla_a L$ Derivative dL/da a Variable a op Operation

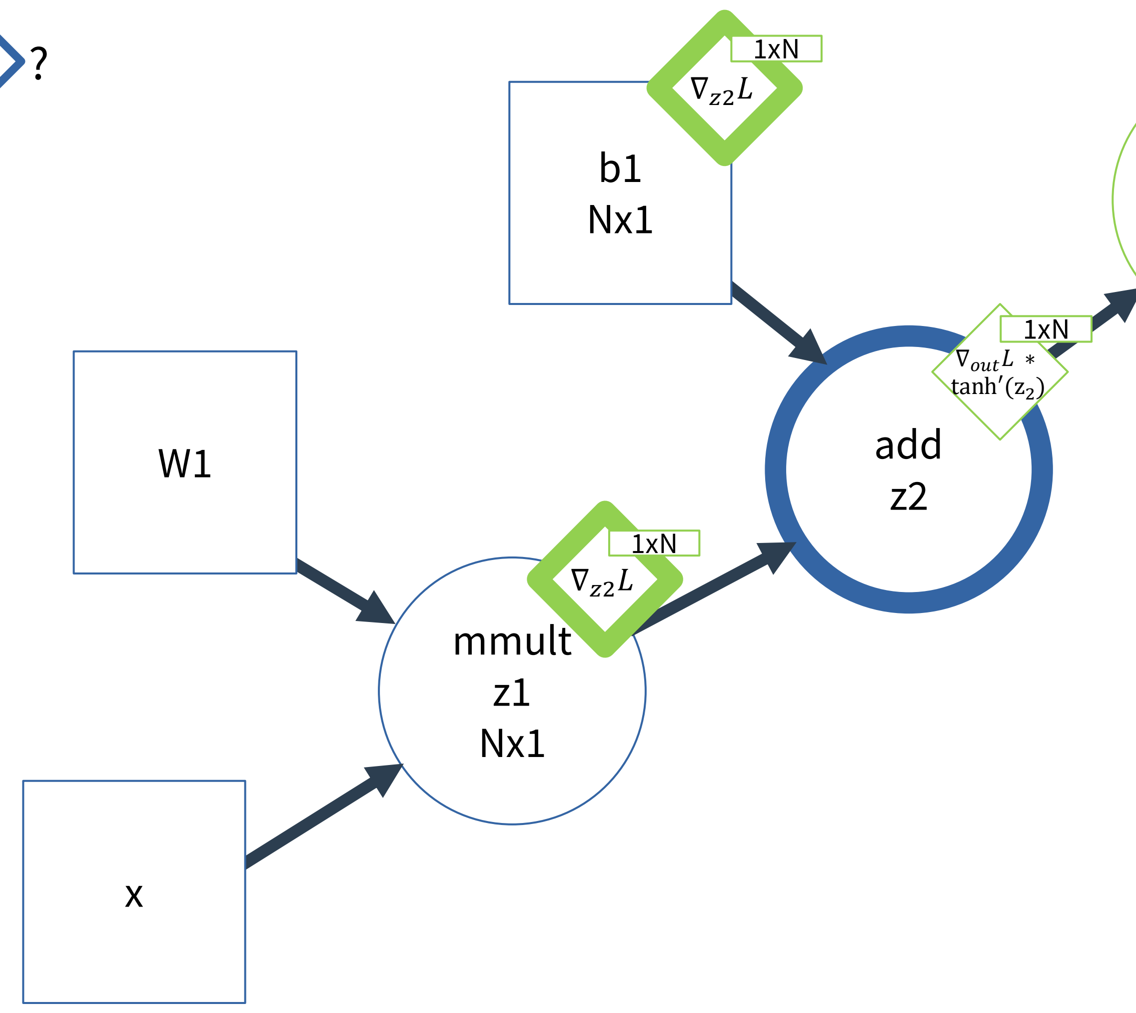
Simple MLP

What are the shapes of $\nabla_{b1}L$ and $\nabla_{z1}L$?



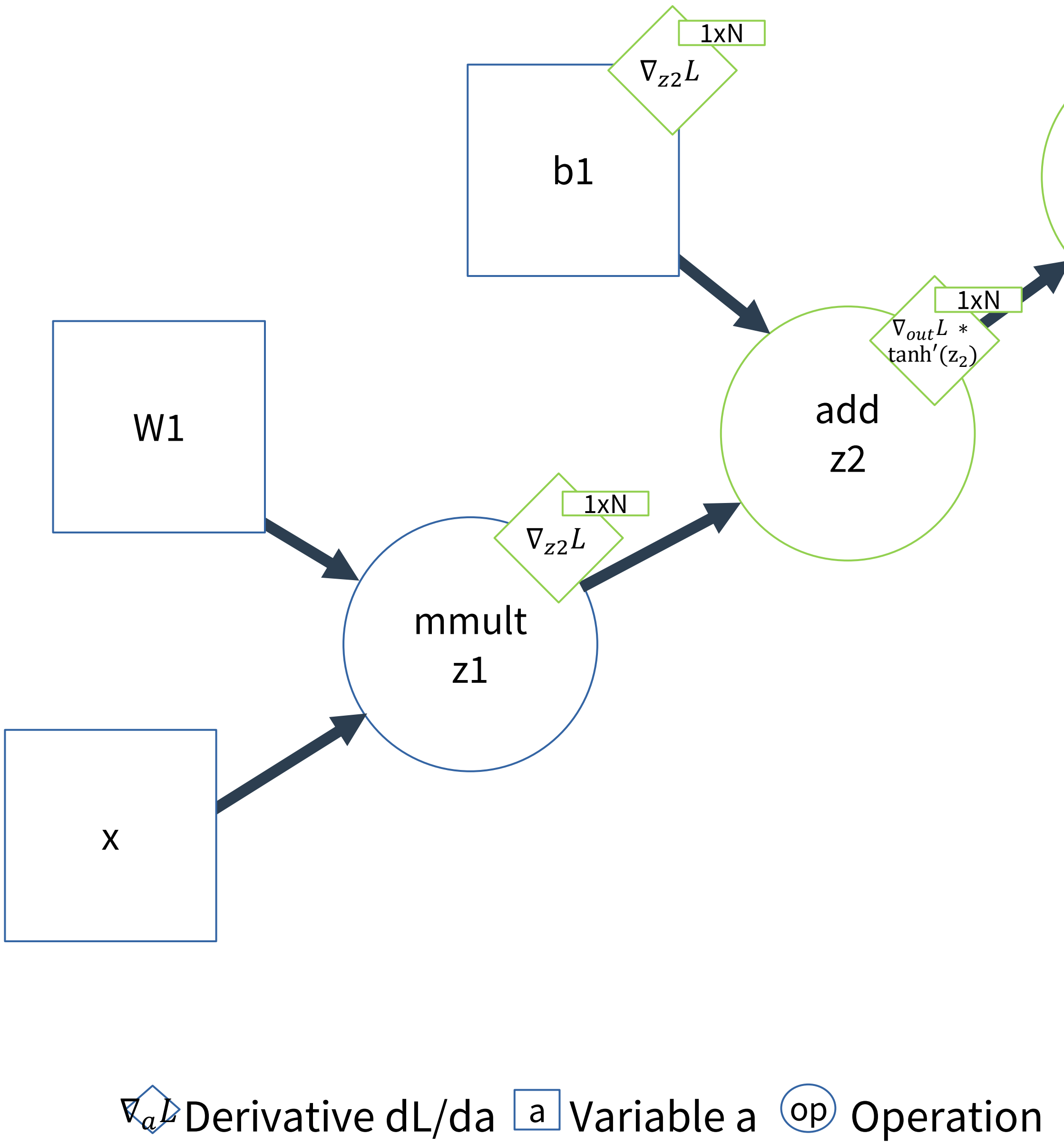
Simple MLP

What are the shapes of $\nabla_{b1}L$ and $\nabla_{z1}L$?



$\nabla_a L$ Derivative dL/da a Variable a op Operation

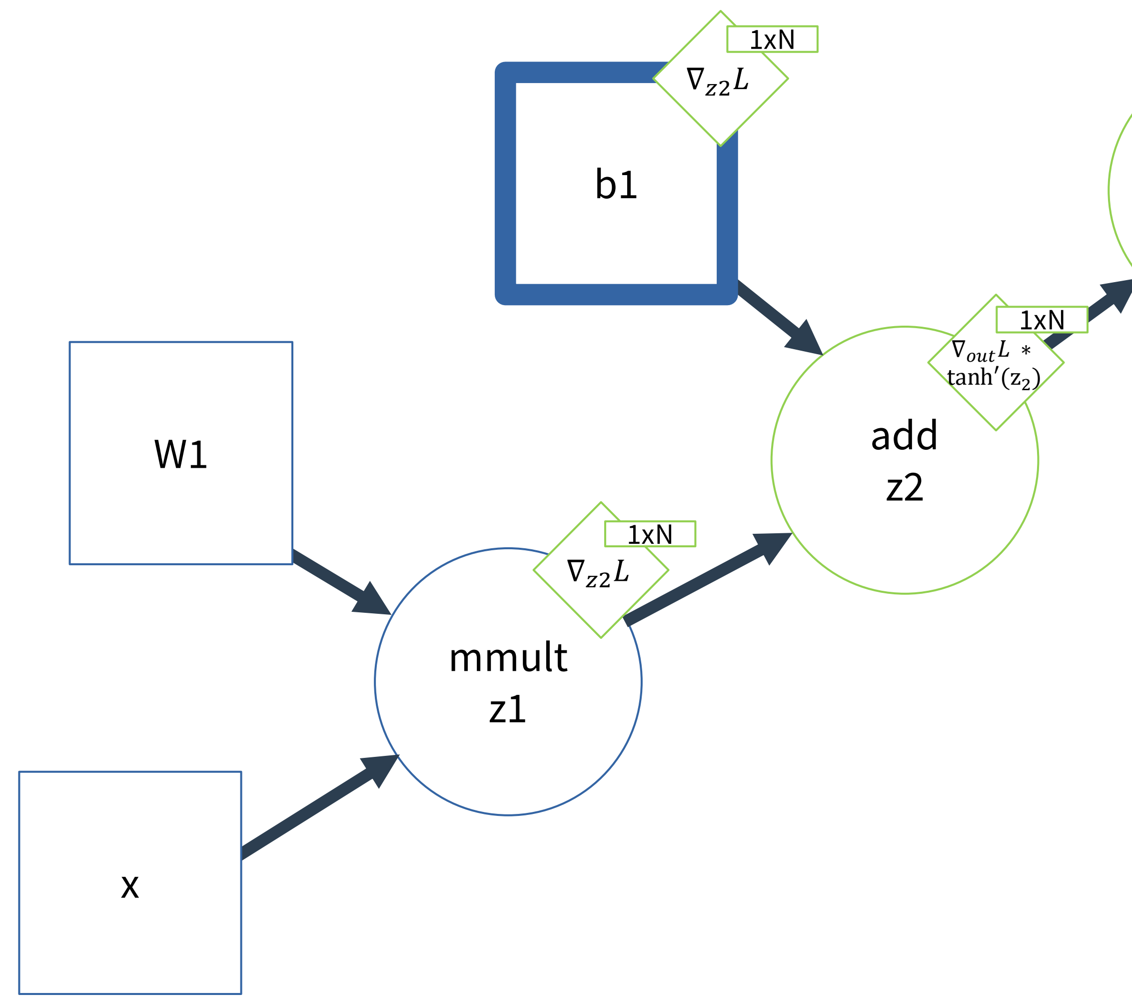
Simple MLP



1xN
$\nabla_{z2}L$
b1
1xN
$\nabla_{out}L$ *
$\tanh'(z_2)$
add
z2
W1
1xN
$\nabla_{z2}L$
mmult
z1
x
$\nabla_a L$ Derivative dL/da
a Variable a
op Operation

Simple MLP

We will continue the graph search by visiting $\boxed{\text{b1}}$.

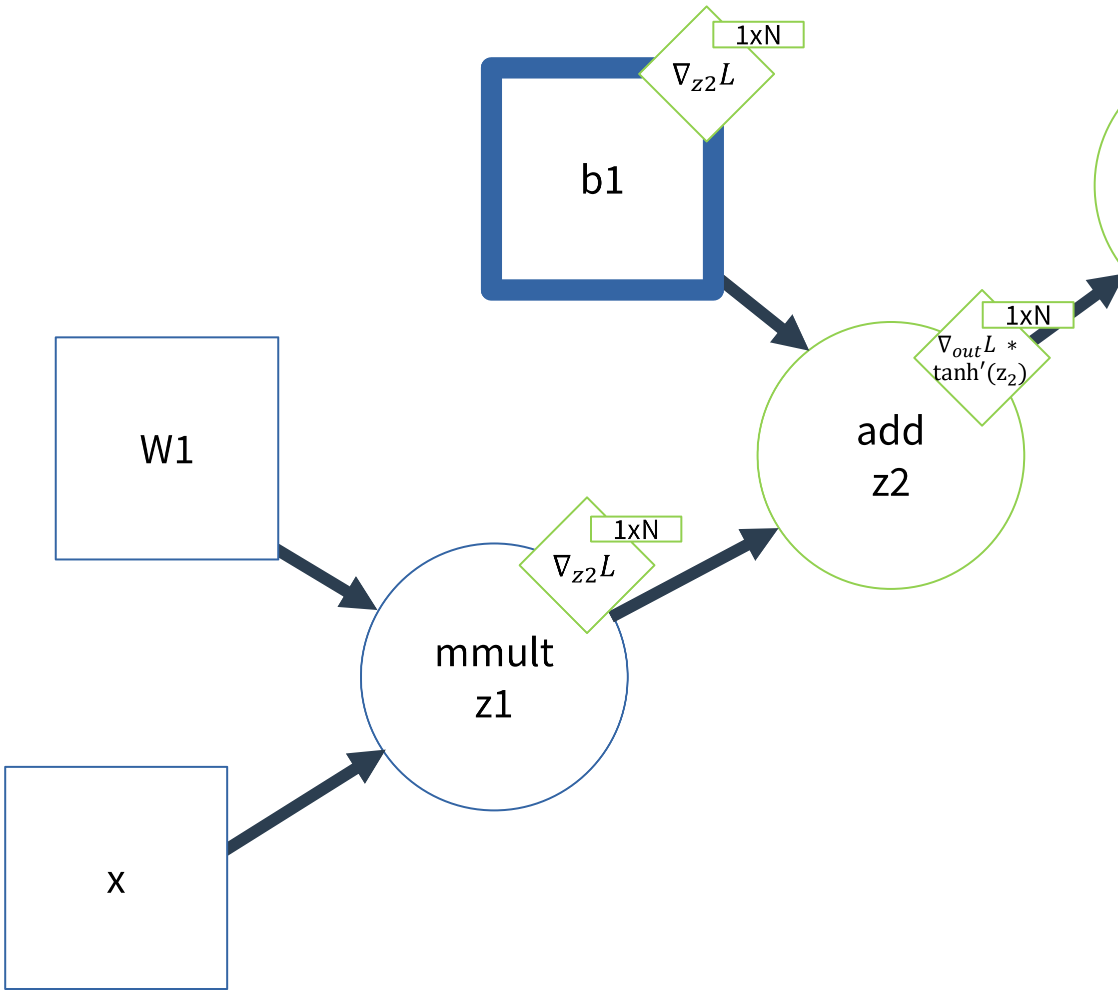


$\nabla_a L$ Derivative dL/da $\boxed{a}$ Variable a op Operation

Simple MLP

$\boxed{\text{b1}}$ has no gradient-enabled parents,

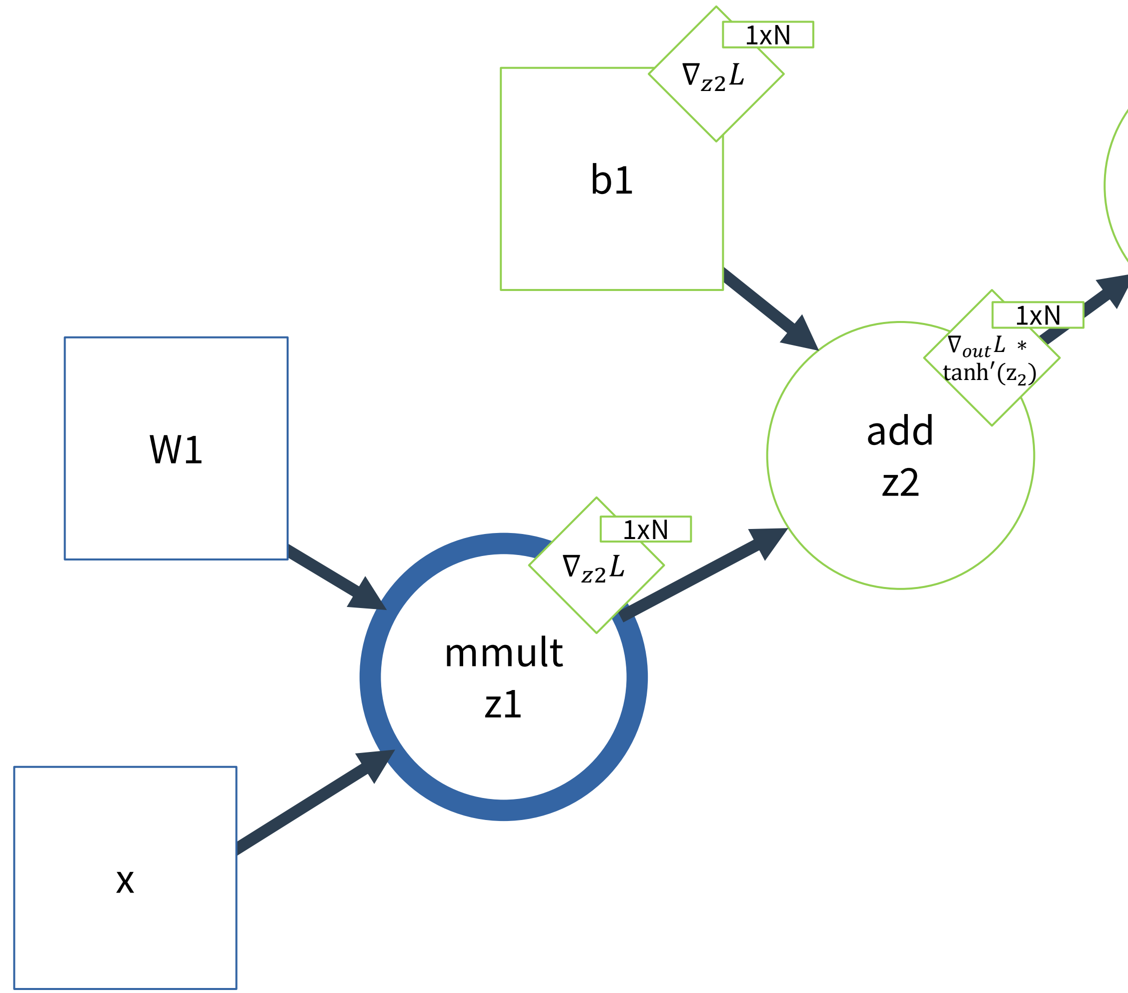
and we want it's gradient, so its backward function is to **accumulate** (i.e. save) the gradient passed to it.



Simple MLP

We will continue the graph search by

visiting mmult z1 .

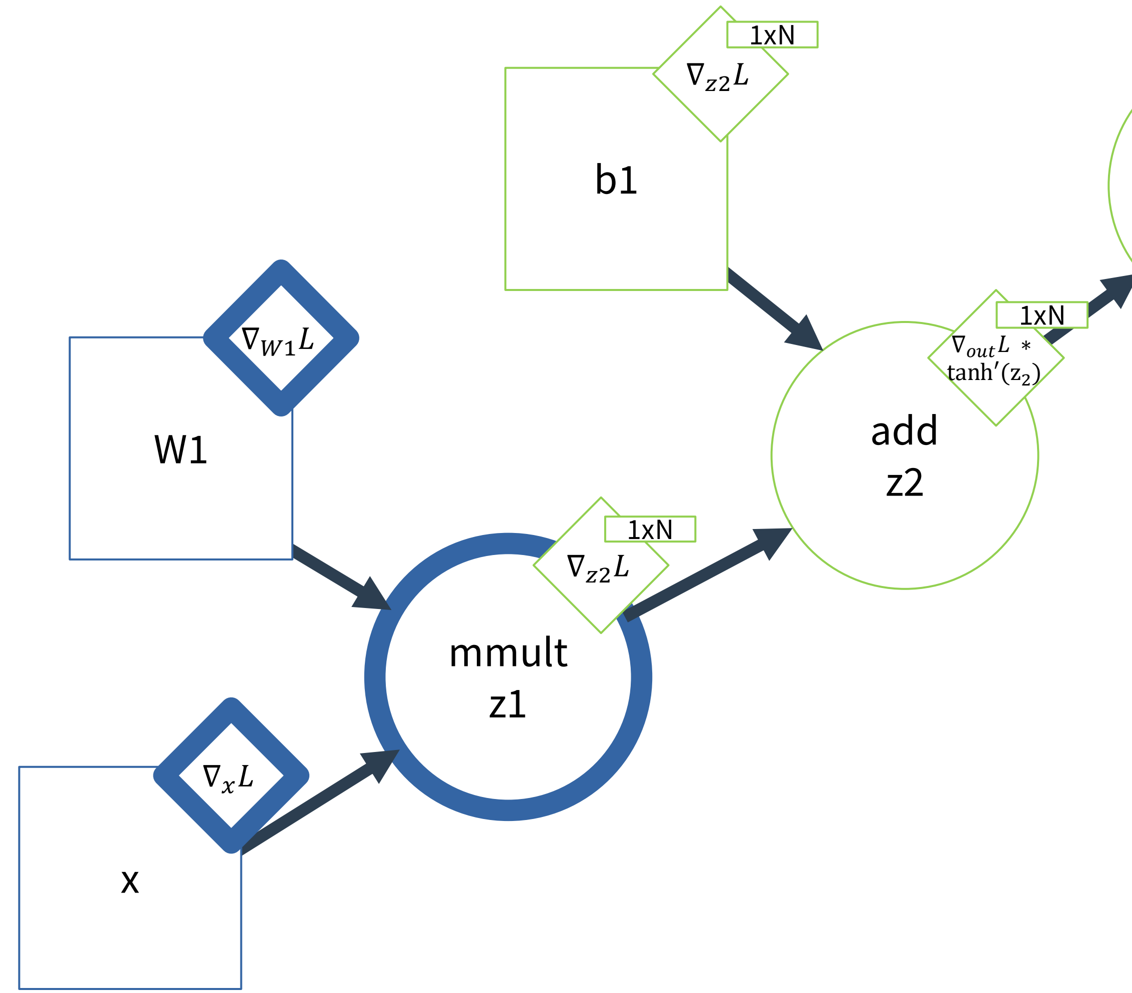


$\nabla_a L$ Derivative dL/da a Variable a op Operation

Simple MLP

What is the backward function of mmult z1 ?

I.e., what are $\nabla_{W1}L$ and $\nabla_{x}L$?



$\nabla_{a}L$ Derivative dL/da a Variable a op Operation

Simple MLP

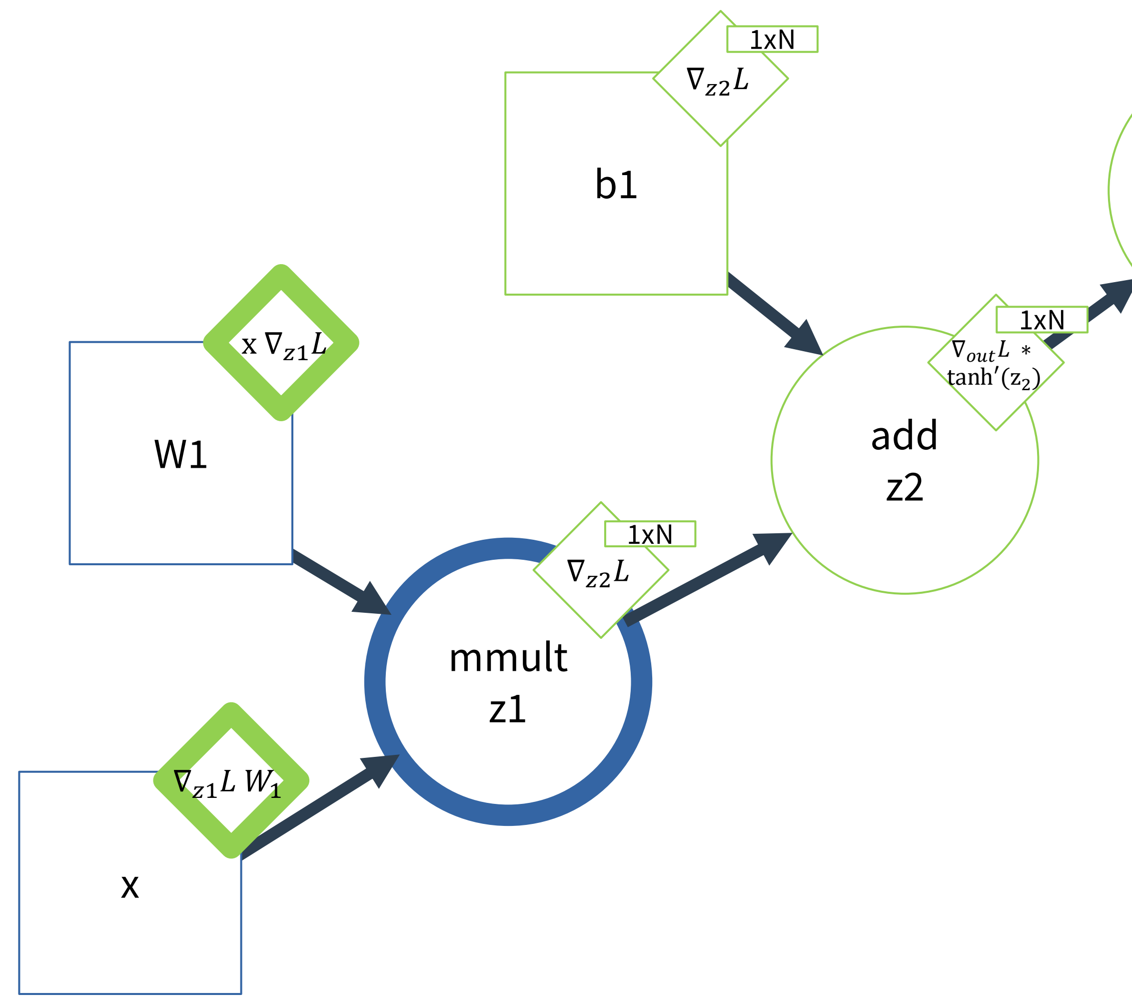
What is the backward function of mmult z1 ?

I.e., what are $\nabla_{W1}L$ 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!



$\nabla_a L$ Derivative dL/da a Variable a op Operation

Simple MLP

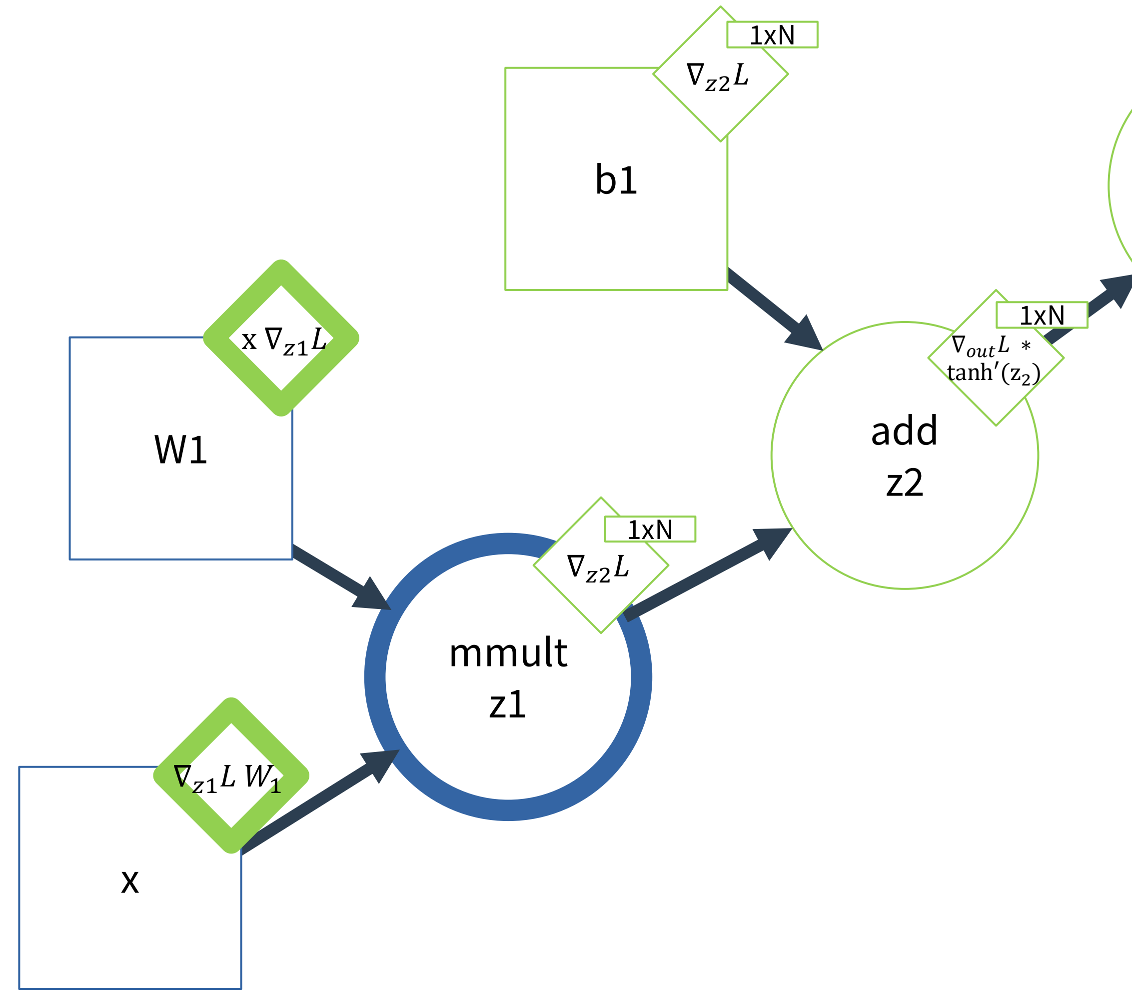
What is the backward function of mmult z1 ?

I.e., what are $\nabla_{W1} L$ and $\nabla_x L$?

Given matrix $\nabla_{AB} L$:

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

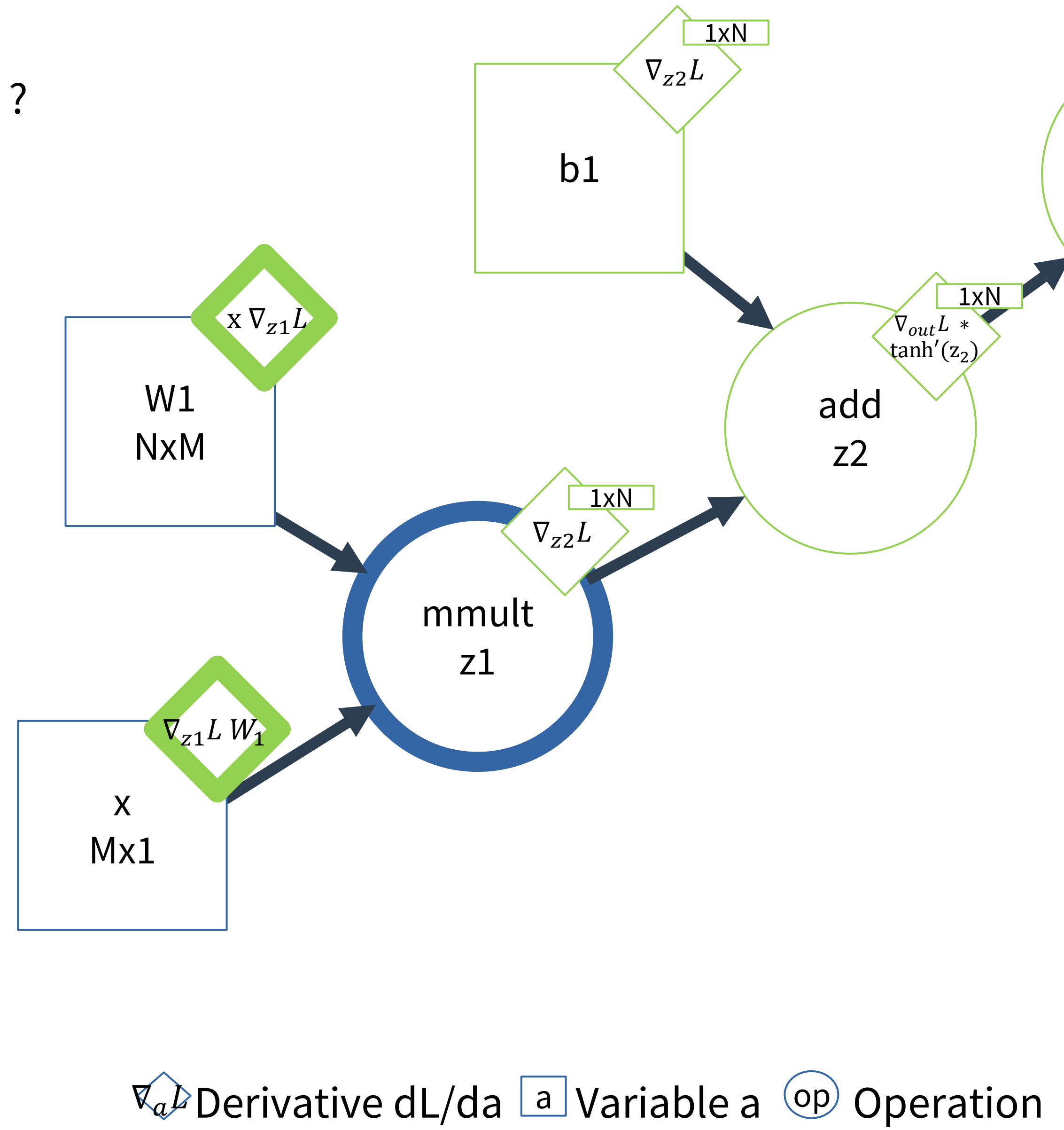
confirm for yourself!



$\nabla_a L$ Derivative dL/da a Variable a op Operation

Simple MLP

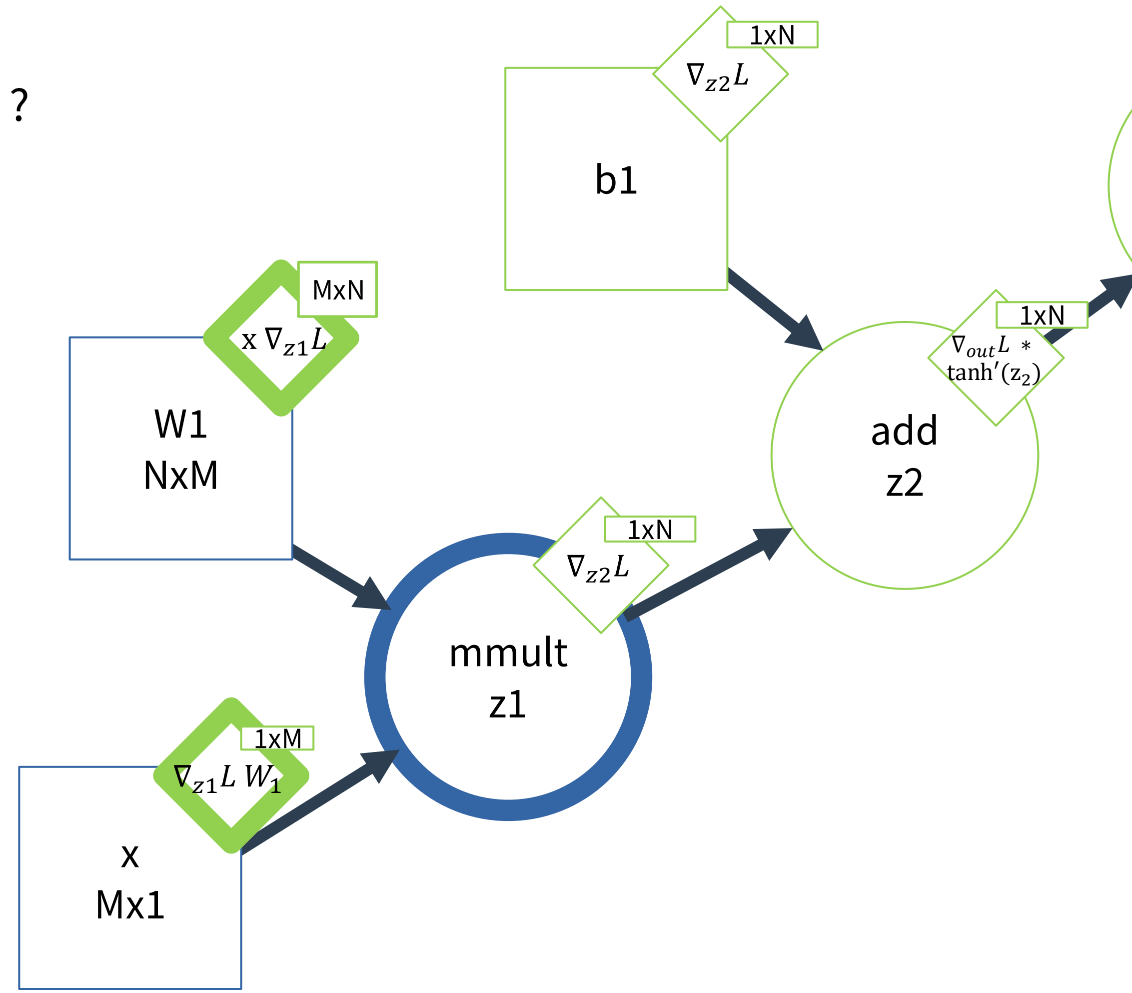
What are the shapes of $\nabla_{W1} L$ and $\nabla_x L$?



Simple MLP

What are the shapes of $\nabla_{W1} L$ and $\nabla_x L$?

… **transpose** except in hw1p1, pytorch …

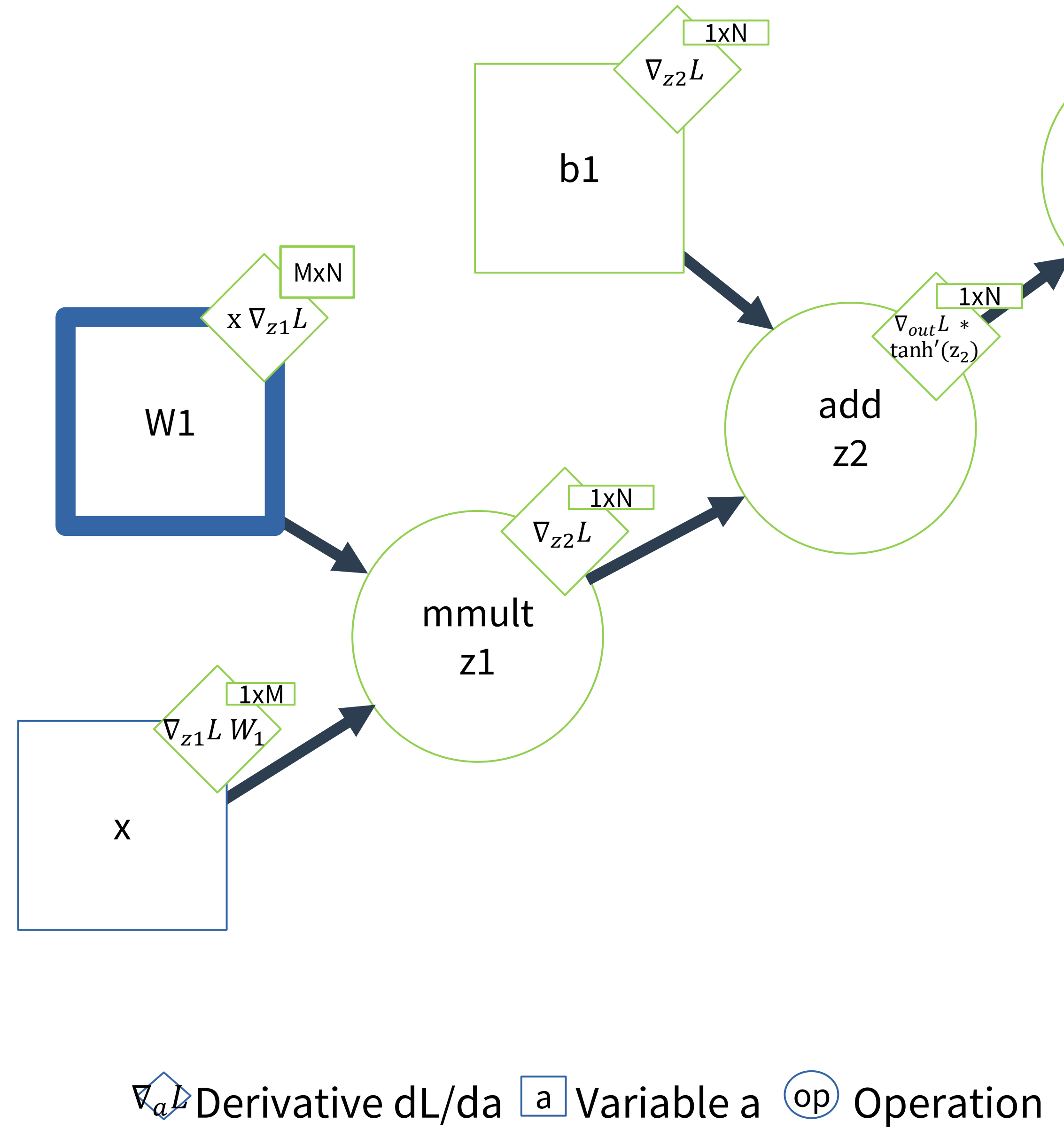


$\nabla_a L$ Derivative dL/da a Variable a op Operation

Simple MLP

We will continue the graph search by

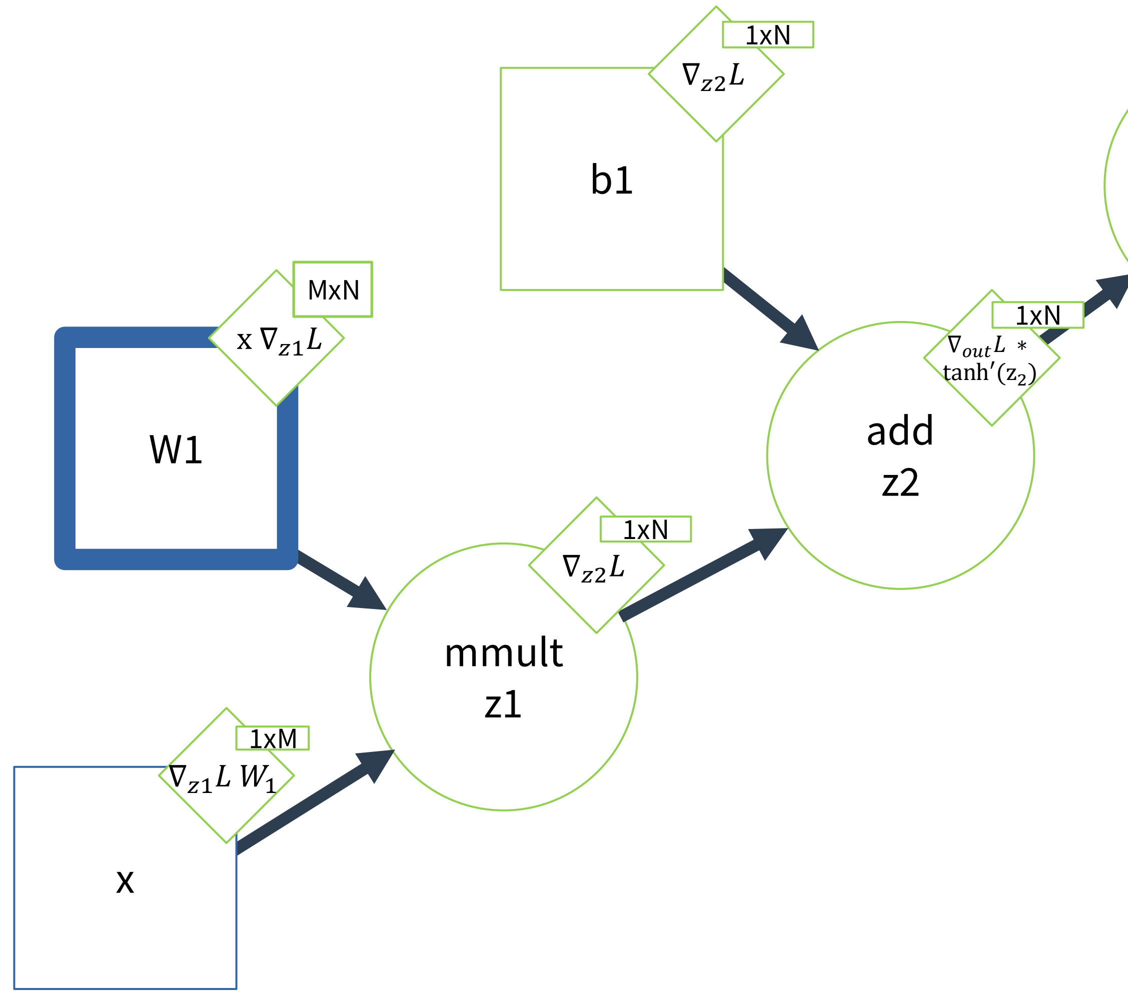
visiting $\boxed{\text{W1}}$.



Simple MLP

$\boxed{\text{W1}}$ has no gradient-enabled parents,

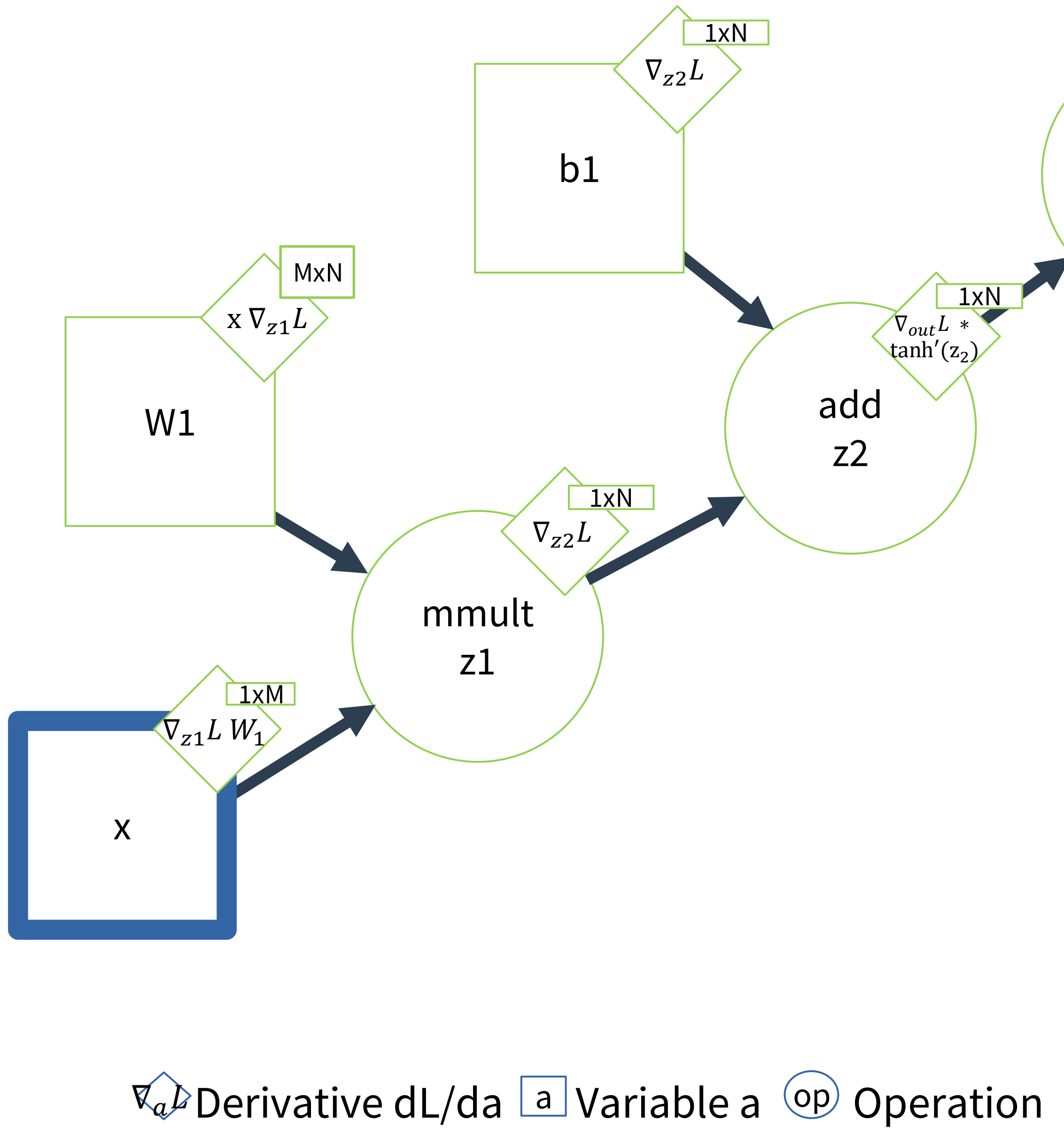
and we want it's gradient, so its backward function is to **accumulate** (i.e. save) the gradient passed to it.



$\nabla_a L$ Derivative dL/da a Variable a op Operation

Simple MLP

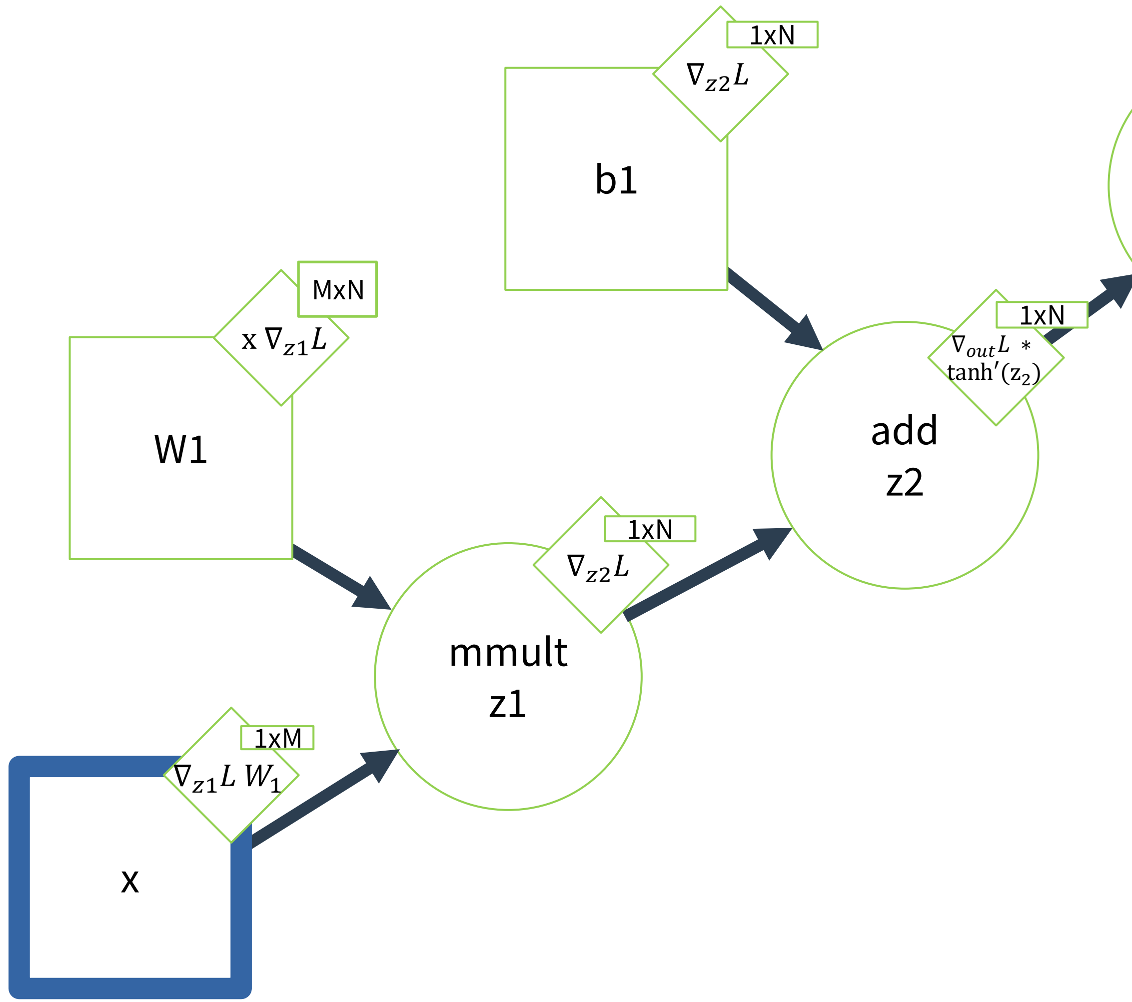
We will continue the graph search by visiting $\boxed{x}$.



Simple MLP

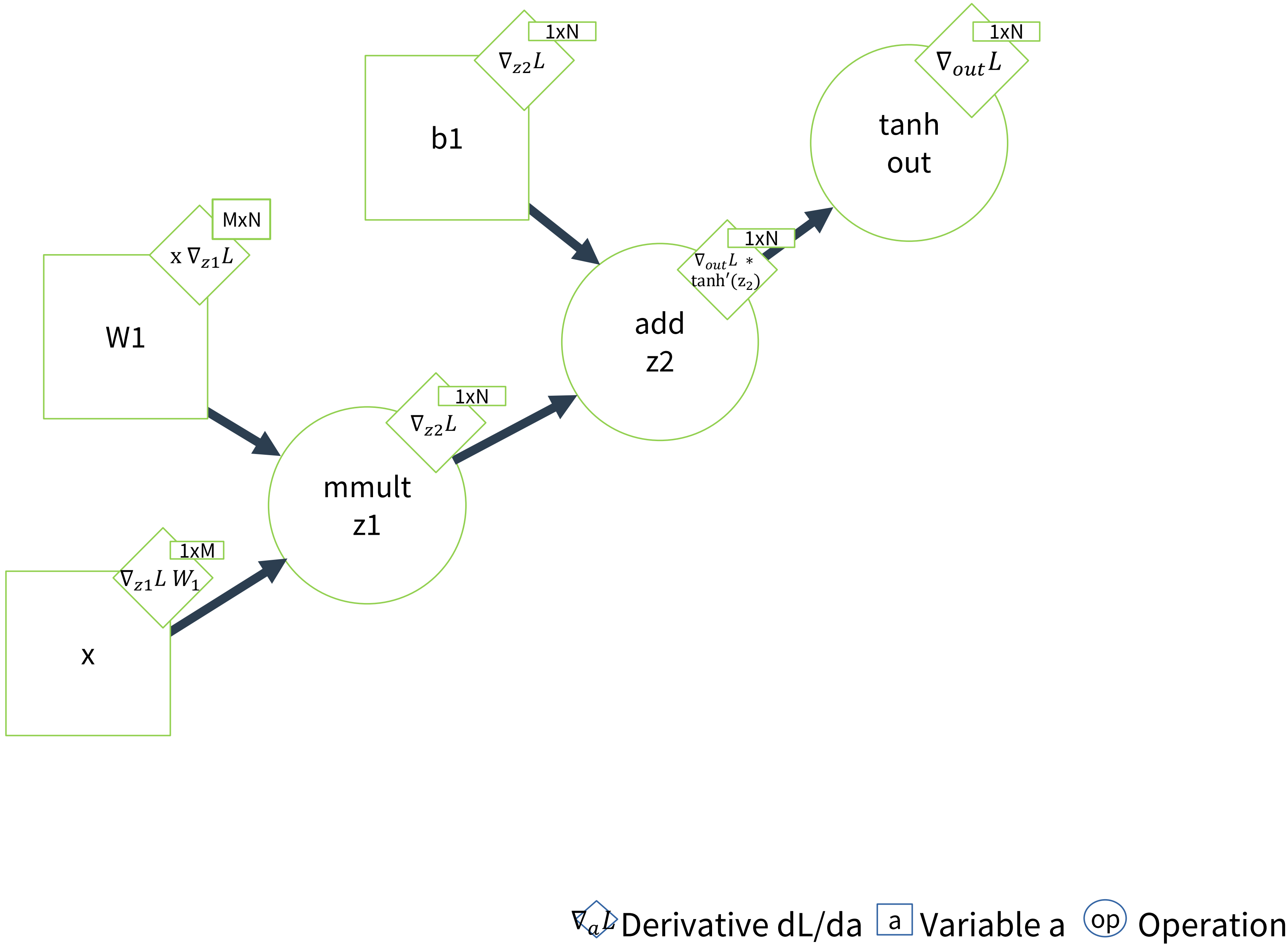
$\boxed{x}$ has no gradient-enabled parents,

and we don't care about its gradient, so we do nothing.



$\nabla_a L$ Derivative dL/da $\boxed{a}$ Variable a (op) Operation

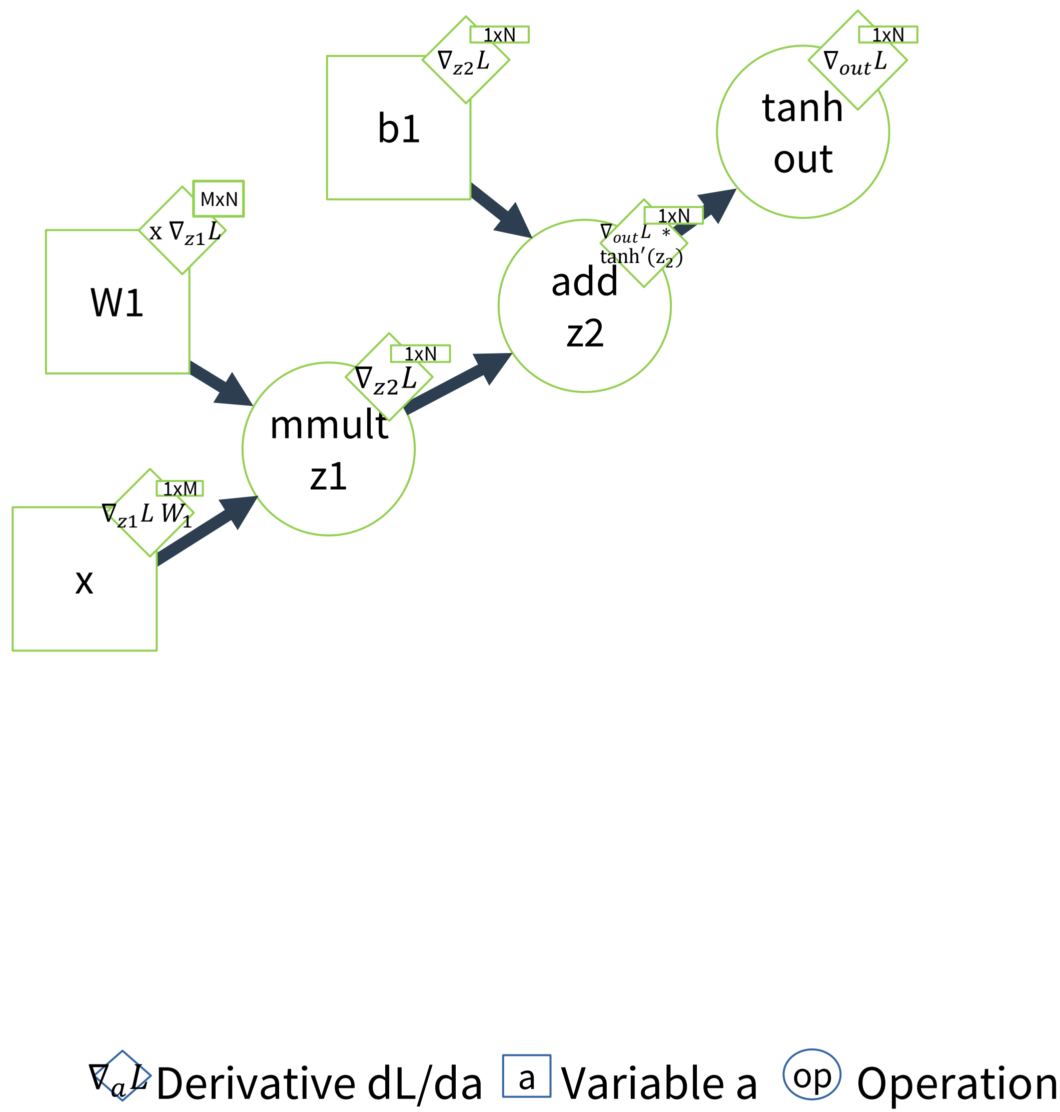
Simple MLP



1xN
$\nabla_{z2}L$
b1
1xN
$\nabla_{out}L$
tanh
out
MxN
x $\nabla_{z1}L$
W1
1xN
$\nabla_{out}L$ * tanh′(z₂)
add
z2
1xN
$\nabla_{z2}L$
mmult
z1
1xM
$\nabla_{z1}L\ W_1$
x
$\nabla_a L$ Derivative dL/da
a Variable a
op Operation

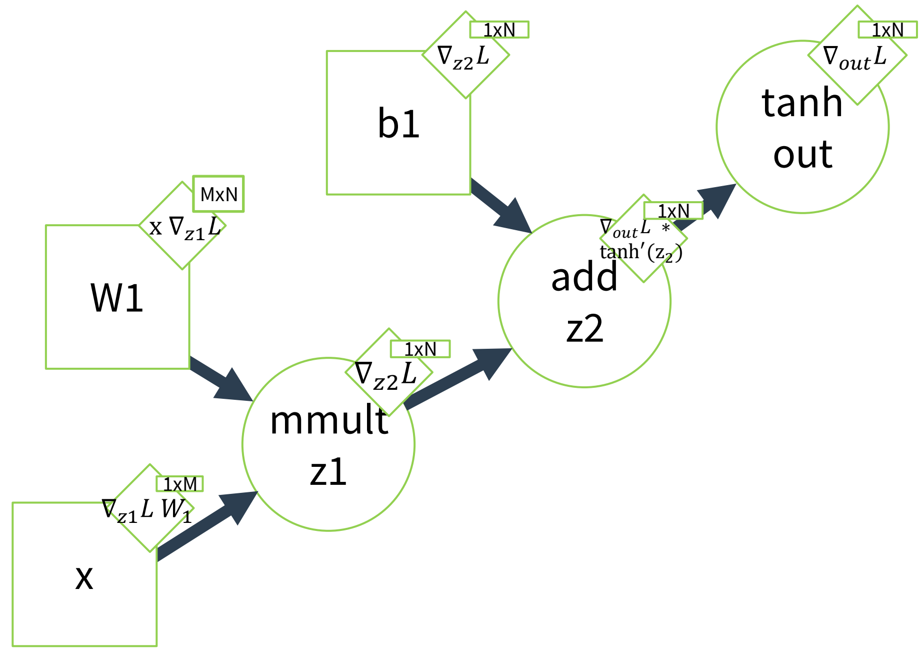
Simple MLP

$\nabla_{out}L$	1xN	from loss function
$\nabla_{z2}L$	1xM	$= \nabla_{z3}L\ tanh'(z_2)$
$\nabla_{b1}L$	1xM	$= \nabla_{z2}L$
$\nabla_{z1}L$	1xM	$= \nabla_{z2}L$
$\nabla_{W1}L$	PxM	$= x\nabla_{z1}L$
$\nabla_{x}L$	1xP	$= \nabla_{z1}L\ W_1$



Simple MLP

$\nabla_{out}L$	1xN	from loss function
$\nabla_{z2}L$	1xM	$= \nabla_{z3}L \; tanh'(z_2)$
$\nabla_{b1}L$	1xM	$= \nabla_{z2}L$
$\nabla_{z1}L$	1xM	$= \nabla_{z2}L$
$\nabla_{W1}L$	PxM	$= x\nabla_{z1}L$
$\nabla_{x}L$	1xP	$= \nabla_{z1}L \; W_1$



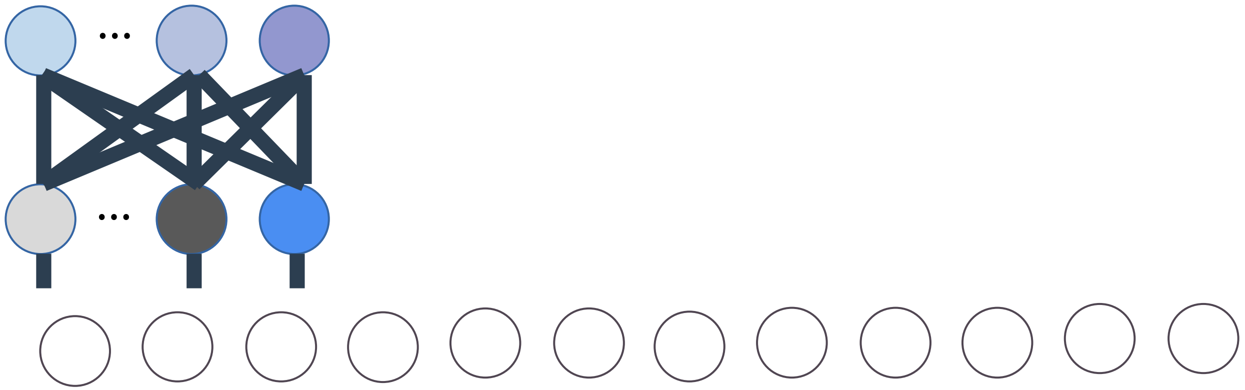
We have gradients for nodes that **accumulated (i.e. saved)** them:
x, W1, b1, W2, b2

$\nabla_a L$ Derivative dL/da   a Variable a   op Operation

What about reusing parameters or intermediate variables?

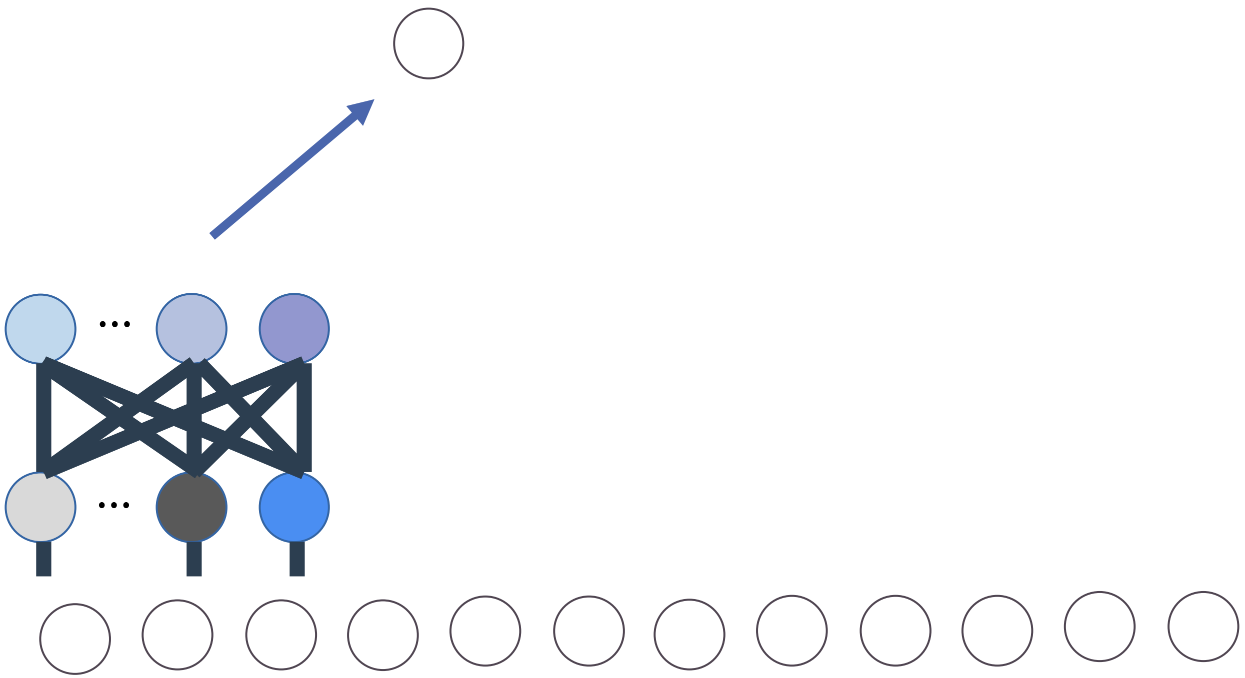
Shared Parameter Networks (Scanning MLP)

The scanning MLP “scans” across some input



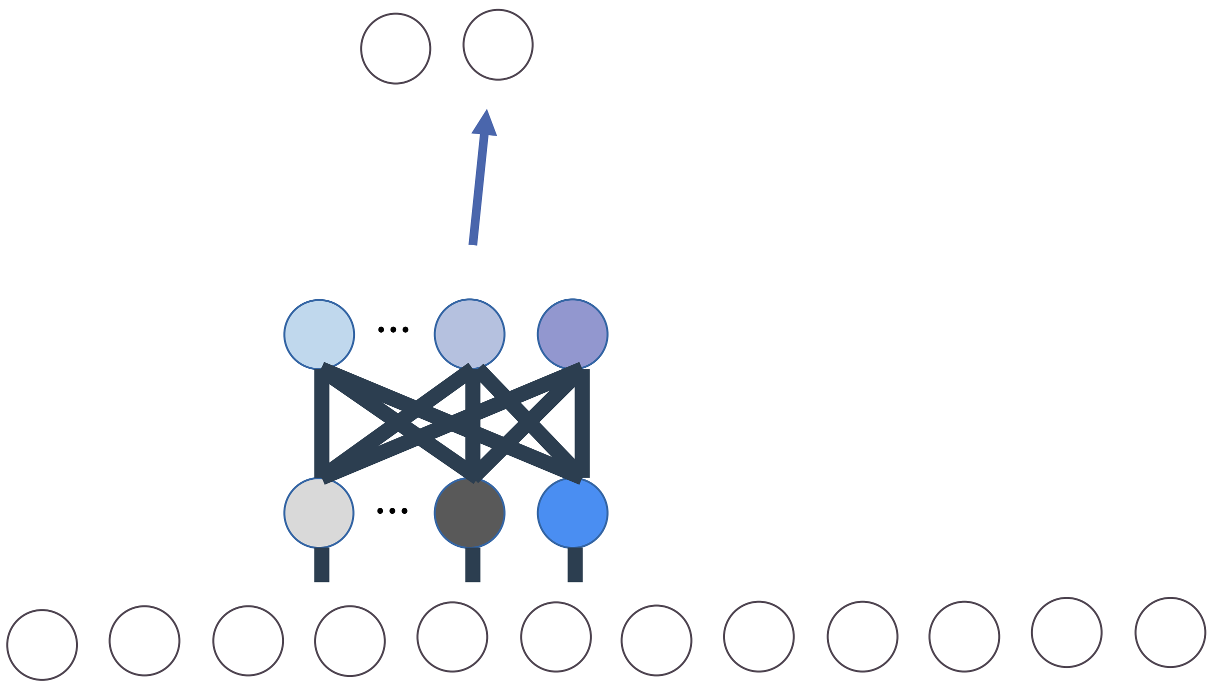
Shared Parameter Networks (Scanning MLP)

The scanning MLP "scans" across some input



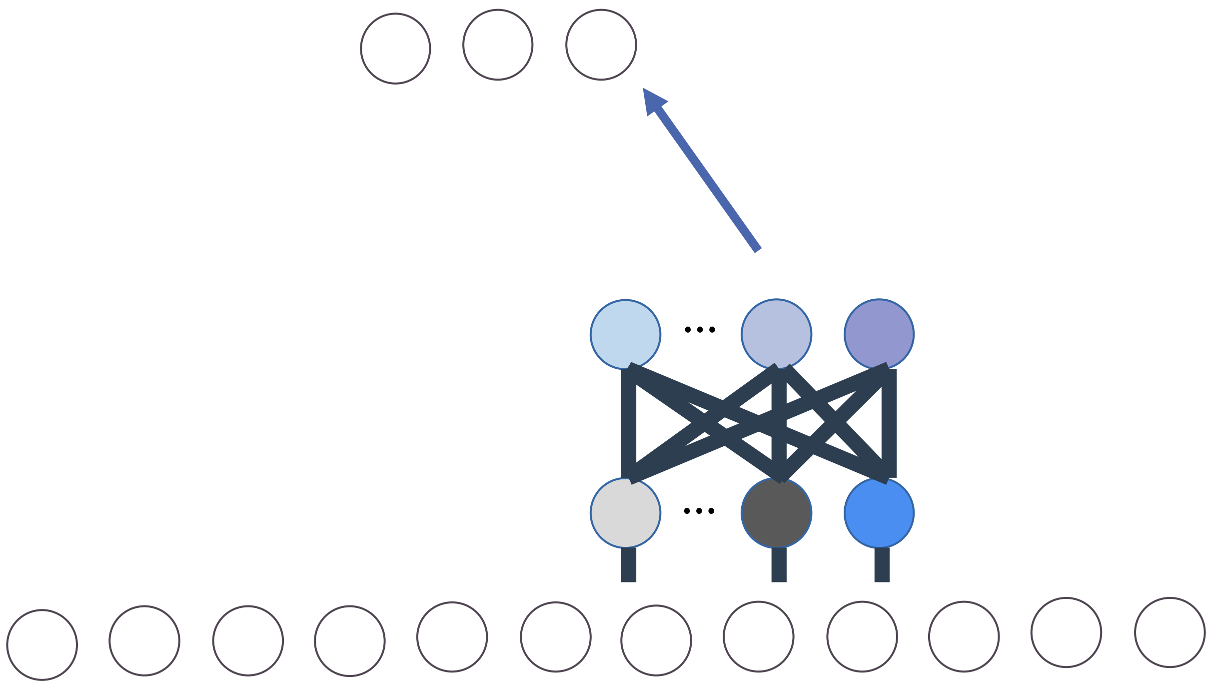
Shared Parameter Networks (Scanning MLP)

The scanning MLP “scans” across some input



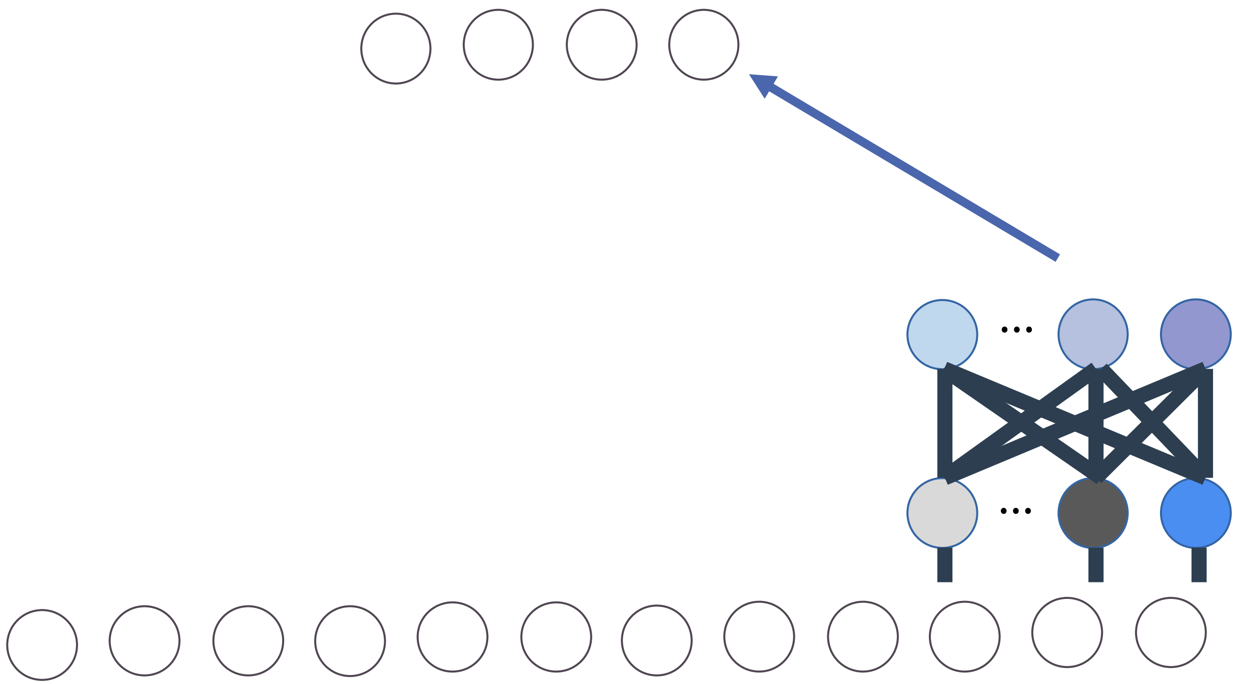
Shared Parameter Networks (Scanning MLP)

The scanning MLP “scans” across some input



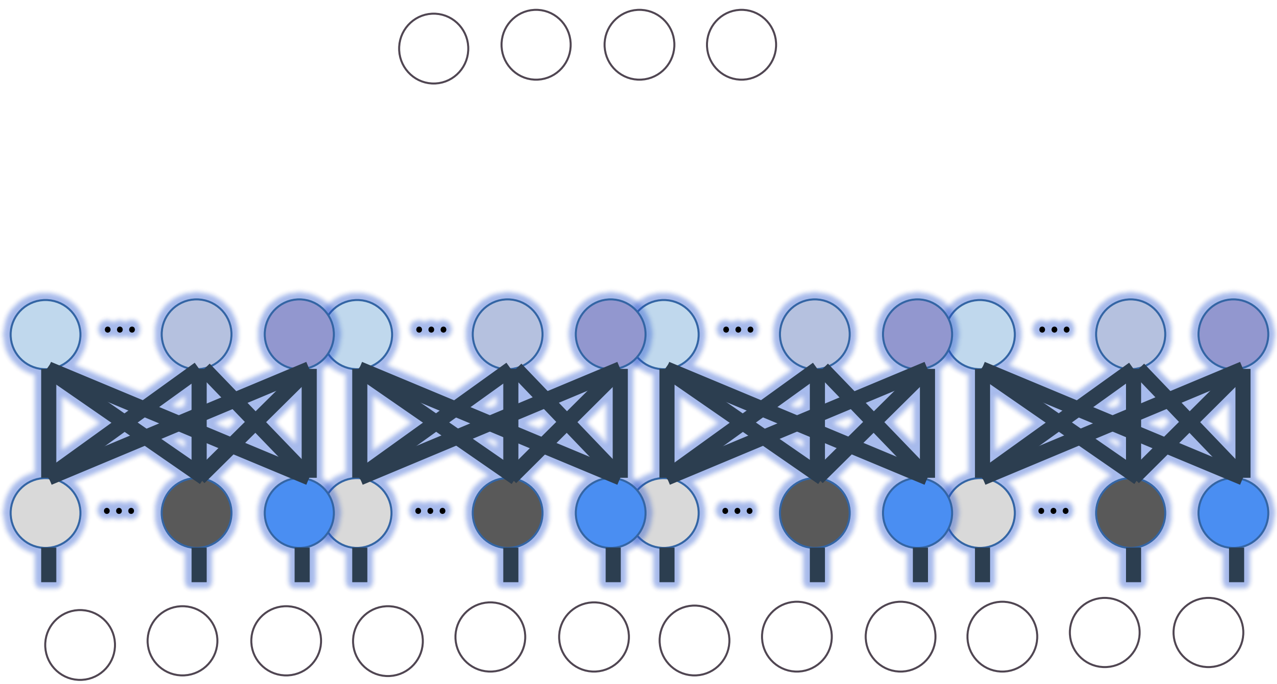
Shared Parameter Networks (Scanning MLP)

The scanning MLP "scans" across some input



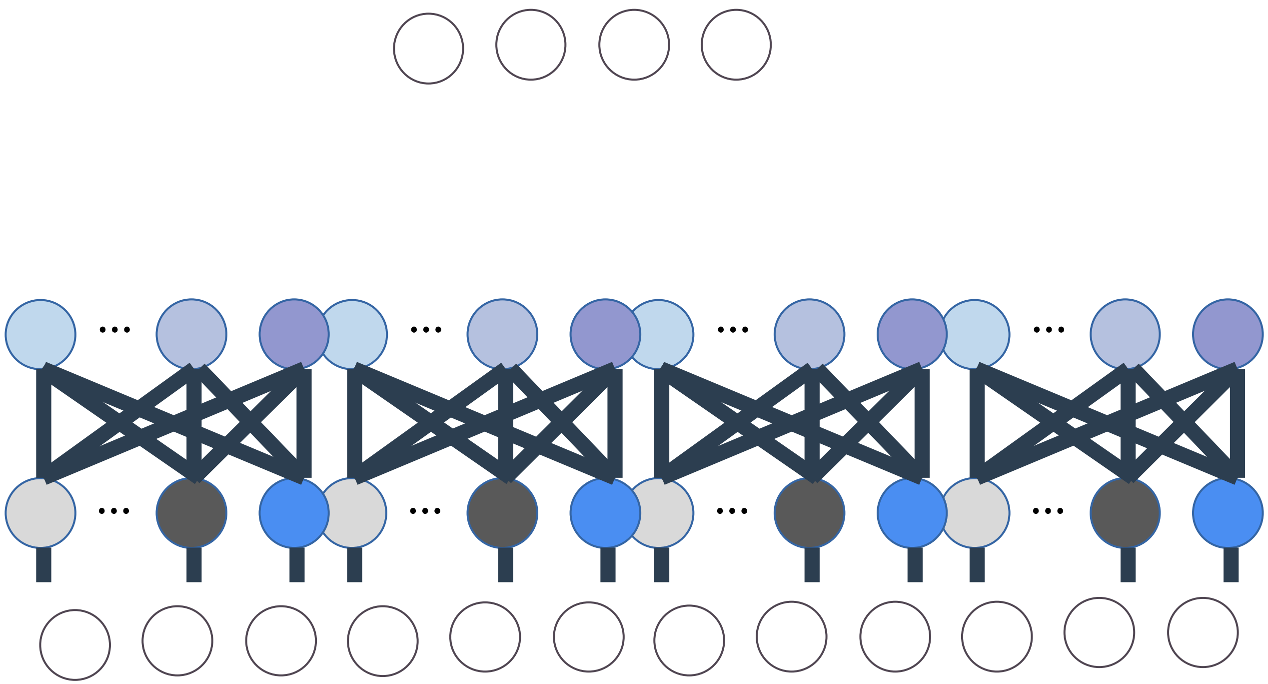
Shared Parameter Networks (Scanning MLP)

One big network with shared parameters



Shared Parameter Networks (Scanning MLP)

One big network with shared parameters



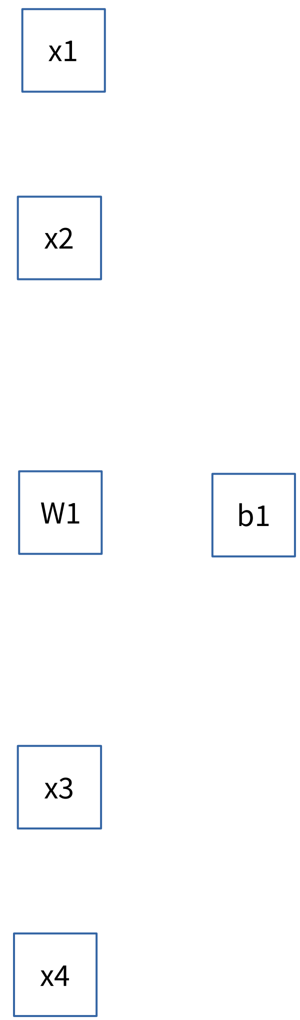
Let's create the graph…

Shared Parameter Networks (Scanning MLP)

```
for position in input:
    MLP(position)
```

Shared Parameter Networks (Scanning MLP)

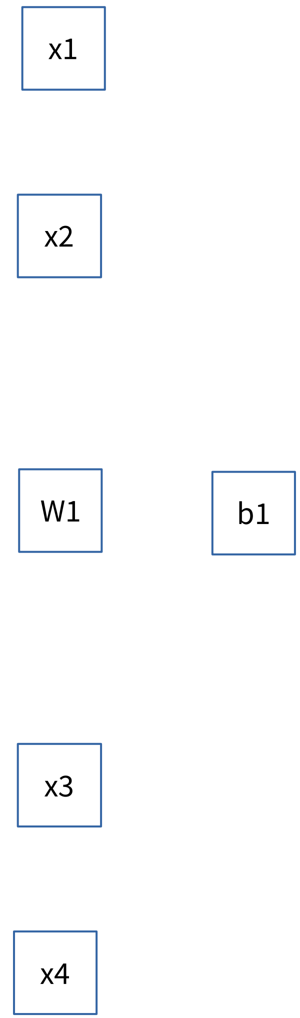
for position in input:
    MLP(position)



Our initial variables

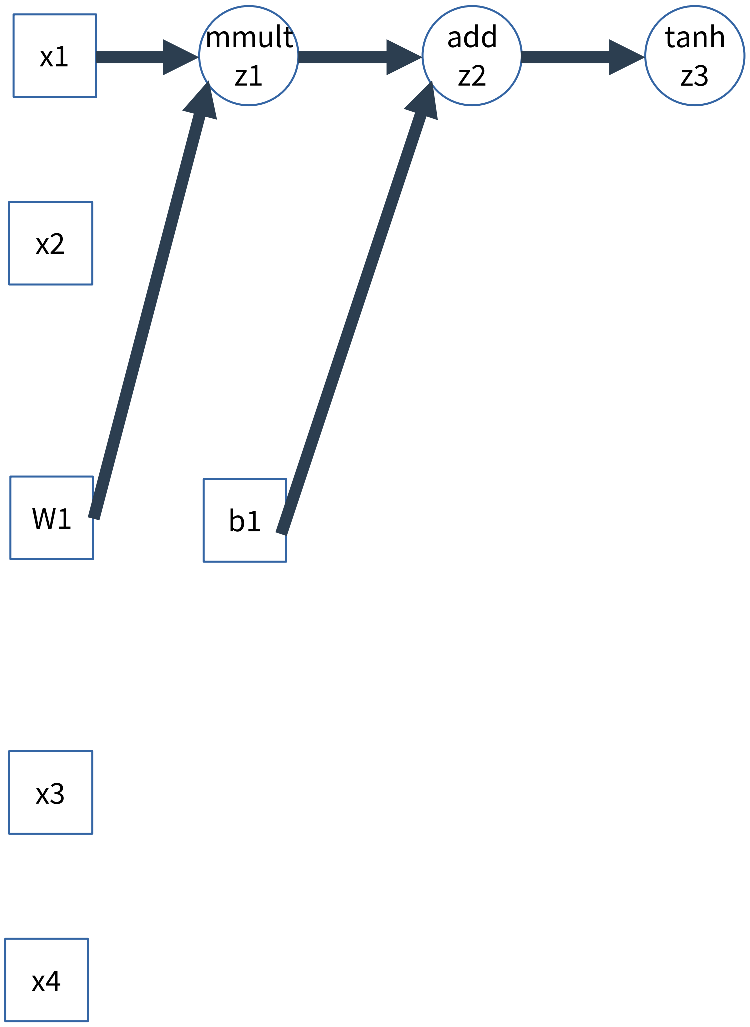
Shared Parameter Networks (Scanning MLP)

for position in input:
    MLP(position)



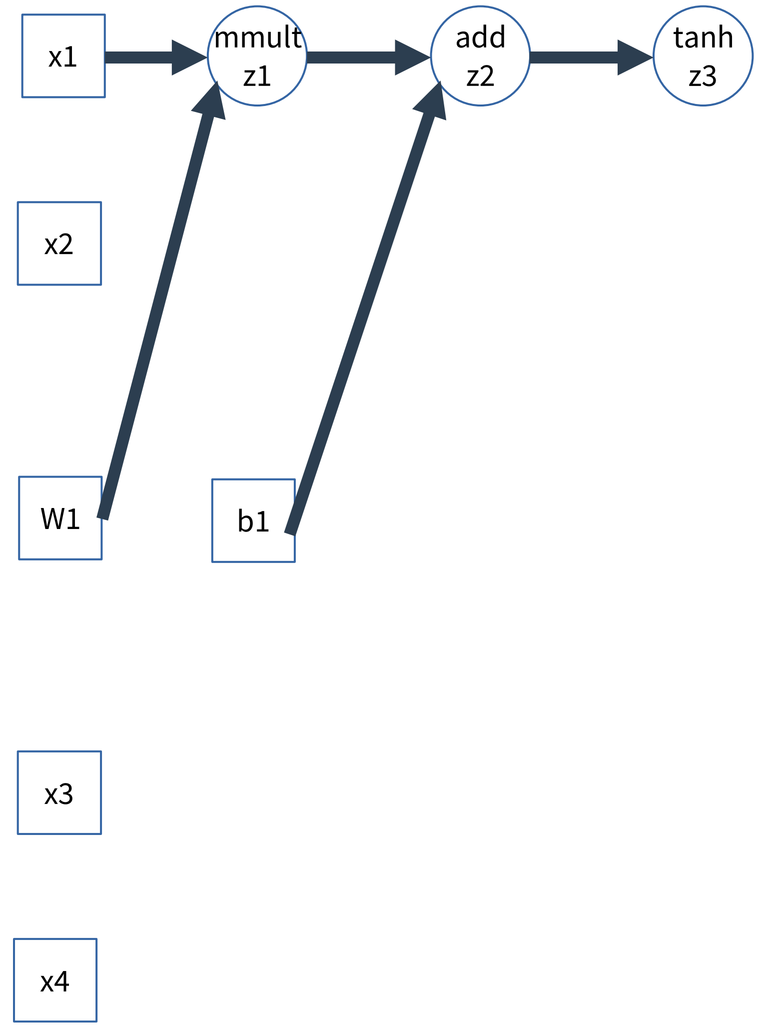
Shared Parameter Networks (Scanning MLP)

for position in input:
MLP(position)



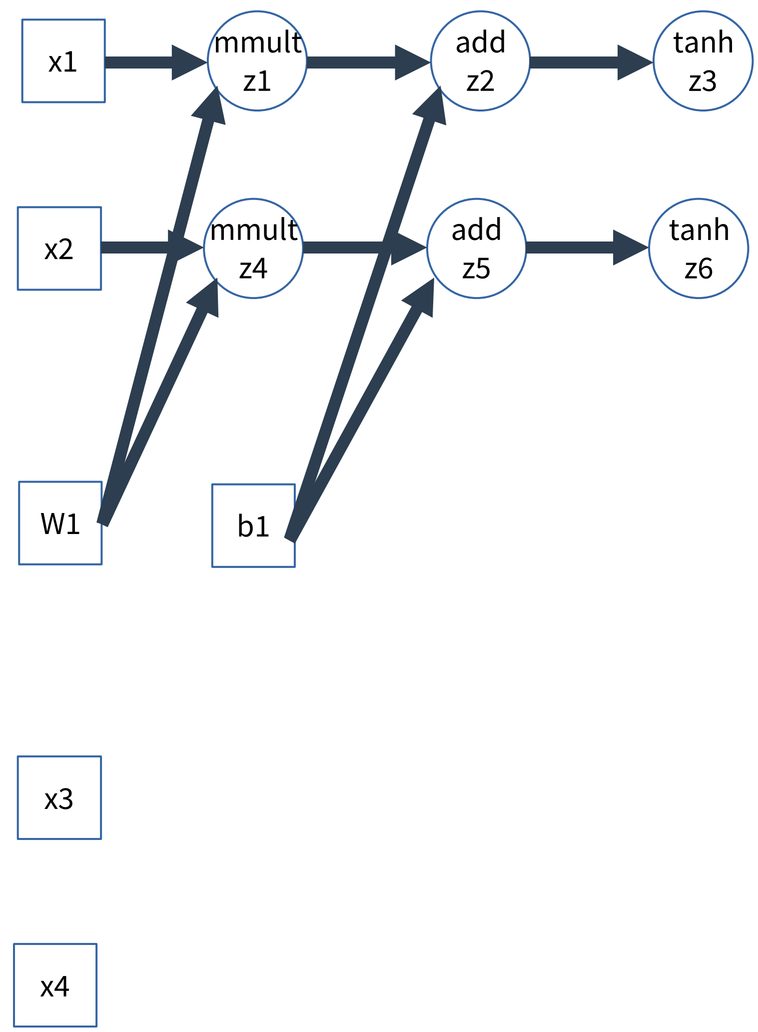
Shared Parameter Networks (Scanning MLP)

- for position in input:
  MLP(position)



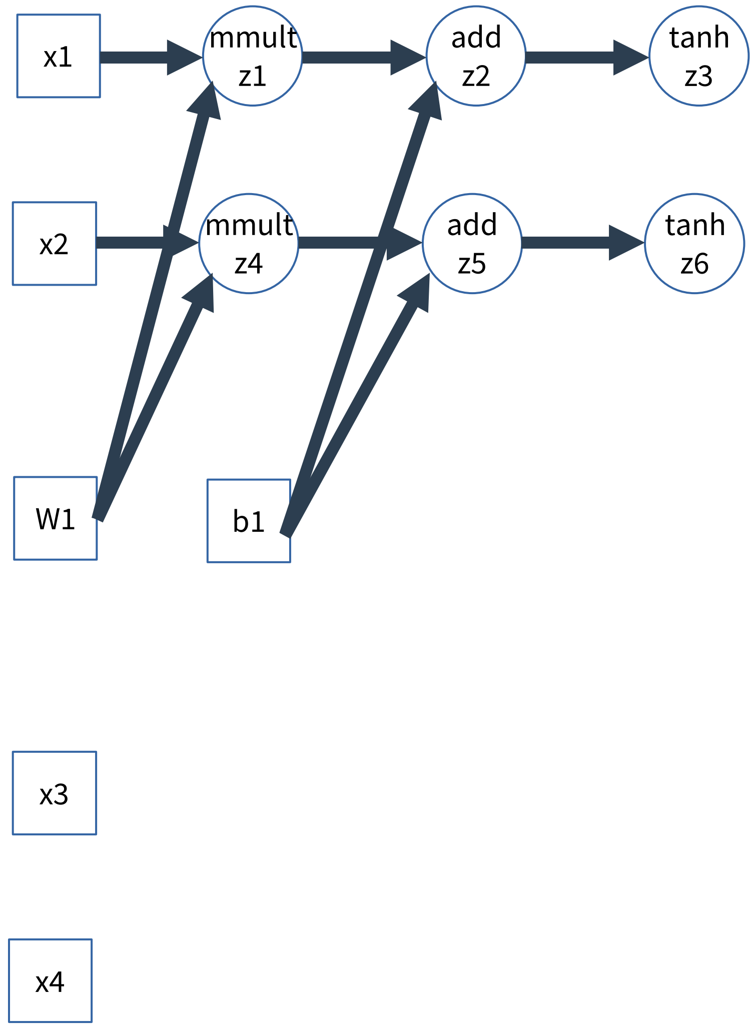
Shared Parameter Networks (Scanning MLP)

for position in input:
    MLP(position)



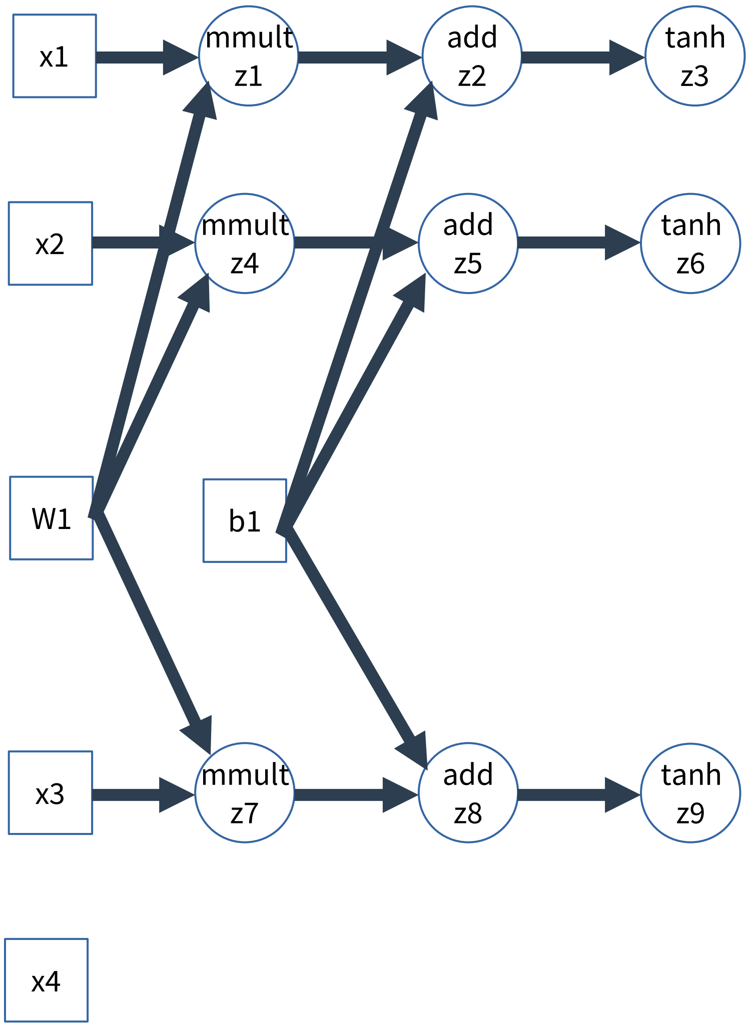
Shared Parameter Networks (Scanning MLP)

for position in input:
    MLP(position)



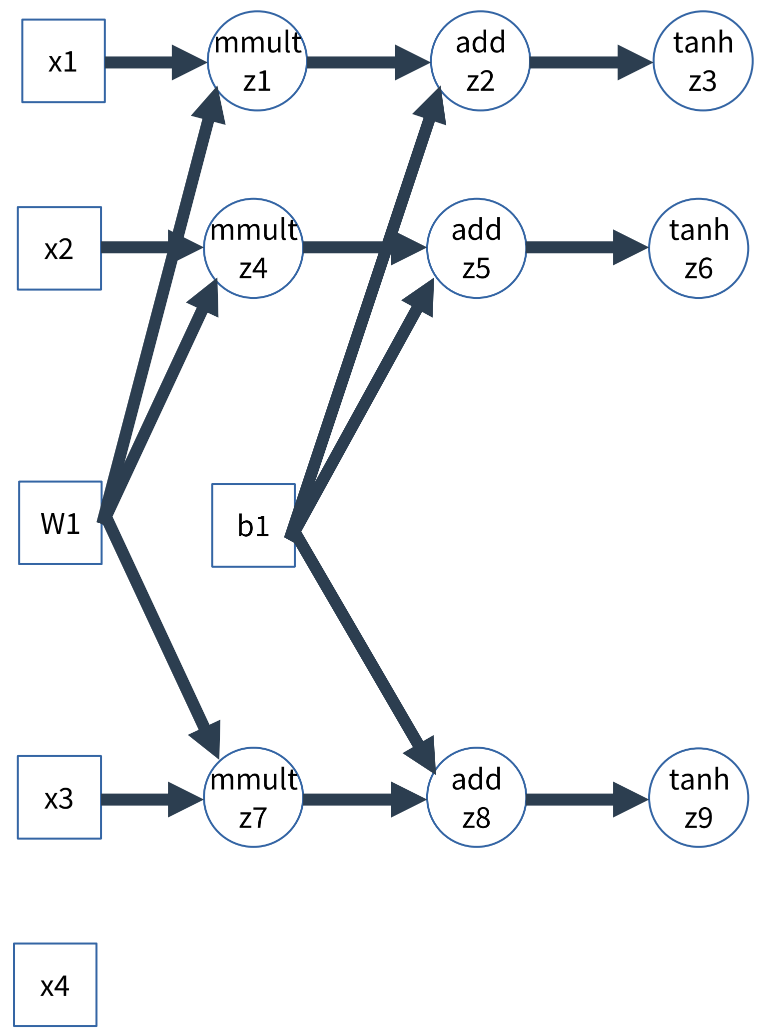
Shared Parameter Networks (Scanning MLP)

for position in input:
    MLP(position)



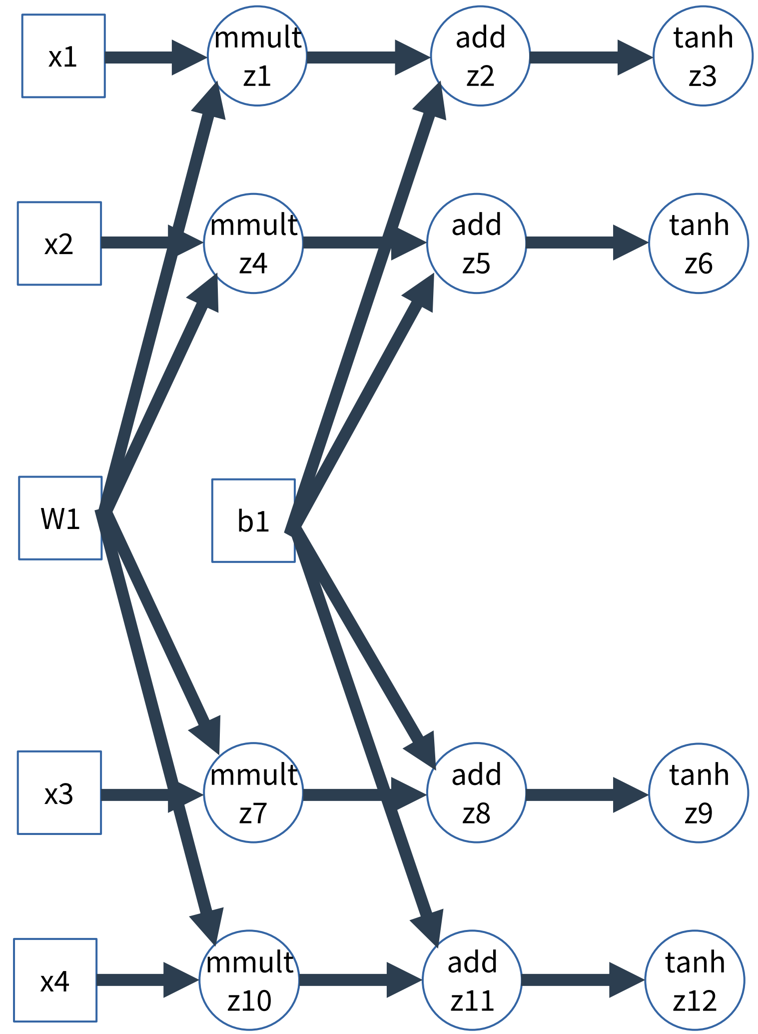
Shared Parameter Networks (Scanning MLP)

- for position in input:
  MLP(position)

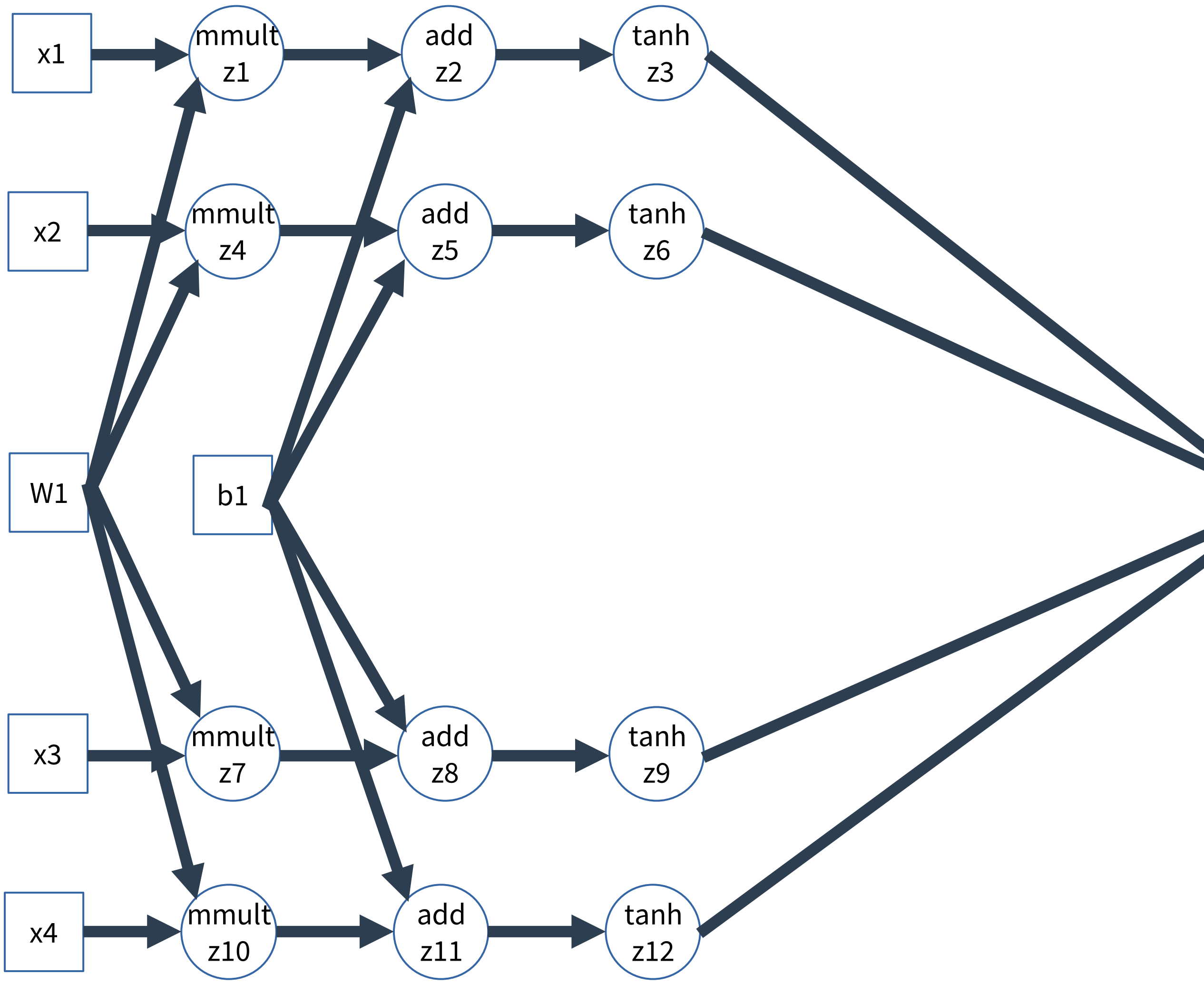


Shared Parameter Networks (Scanning MLP)

for position in input:
    MLP(position)

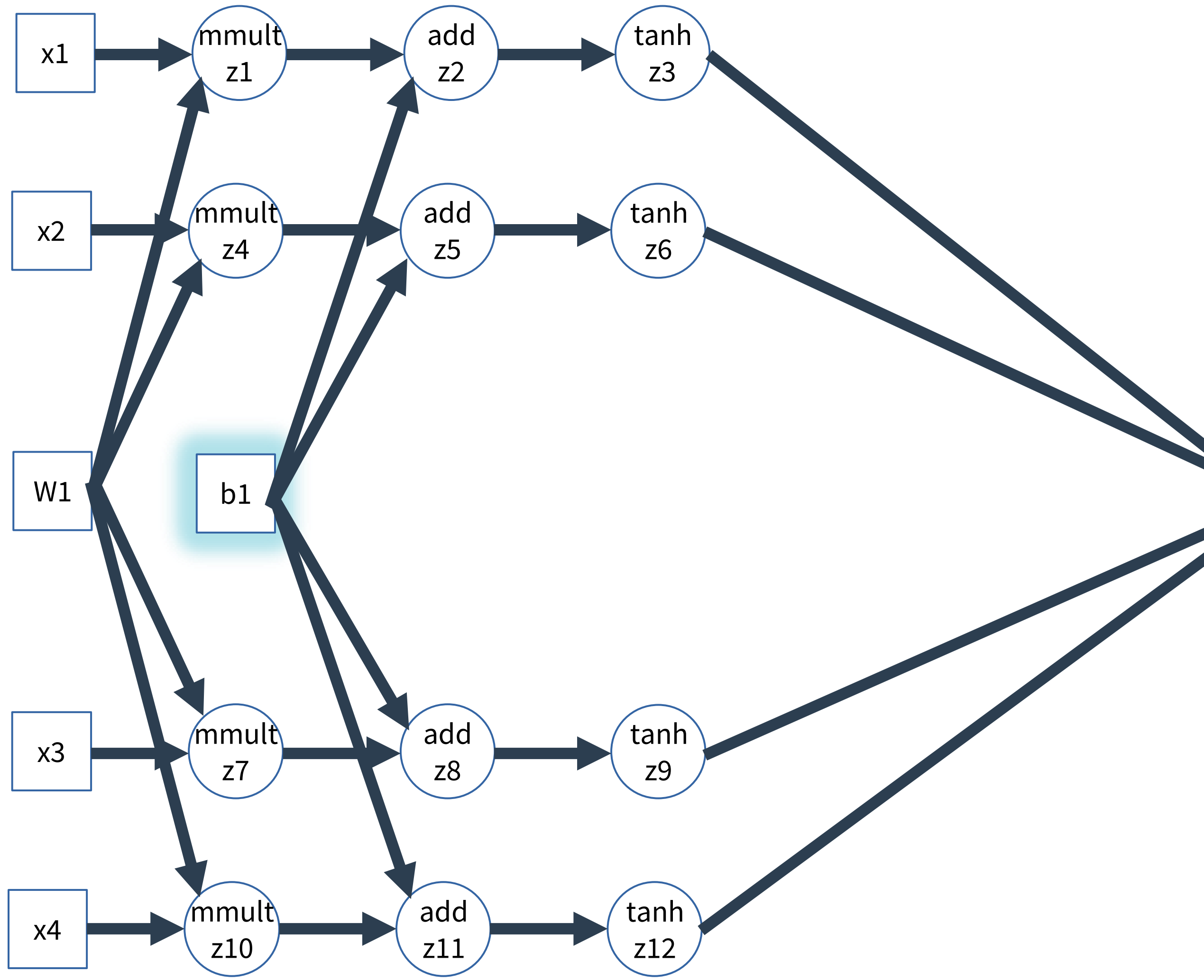


Shared Parameter Networks (Scanning MLP)



Shared Parameter Networks (Scanning MLP)

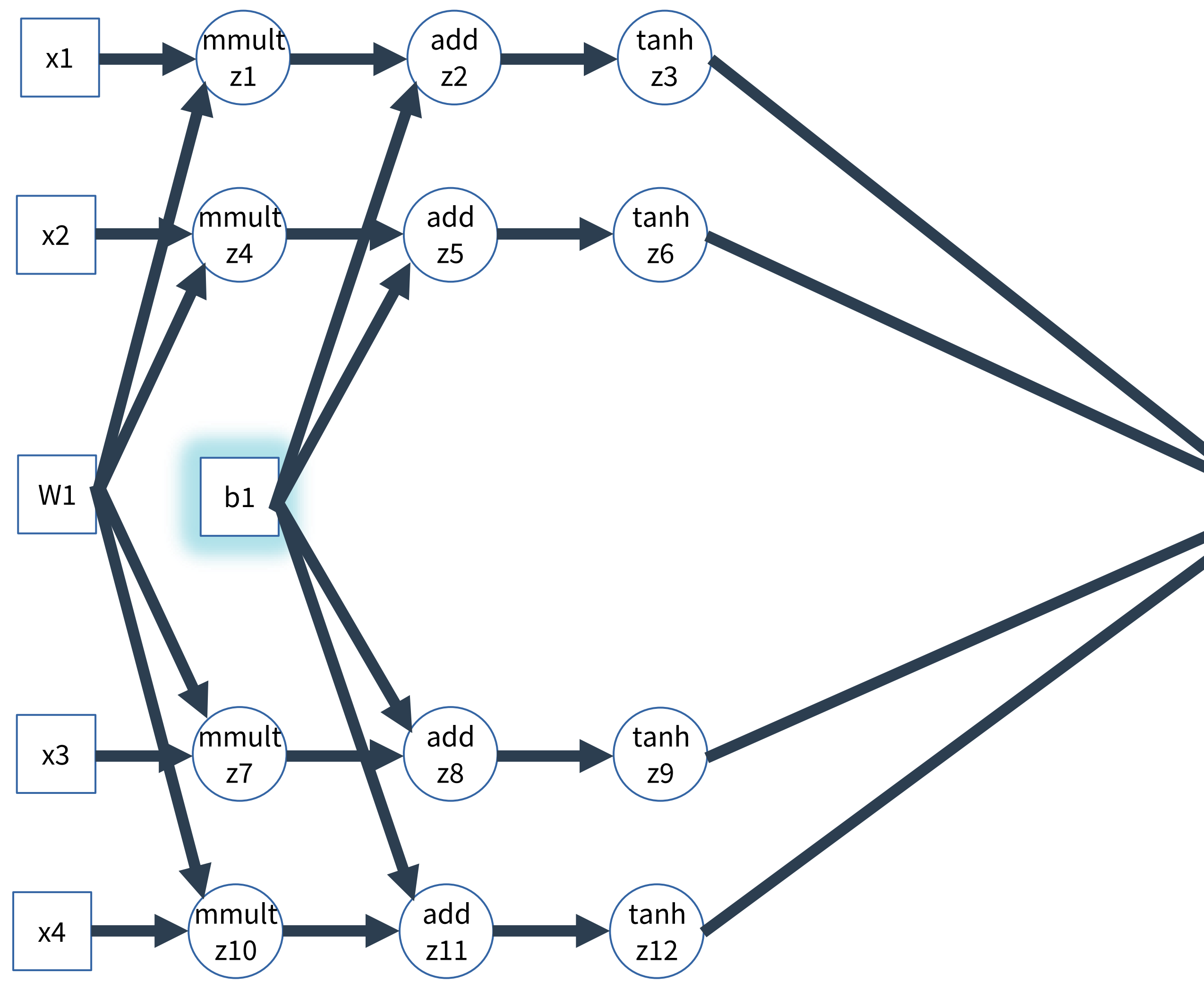
Nodes can have multiple avenues of influence



Shared Parameter Networks (Scanning MLP)

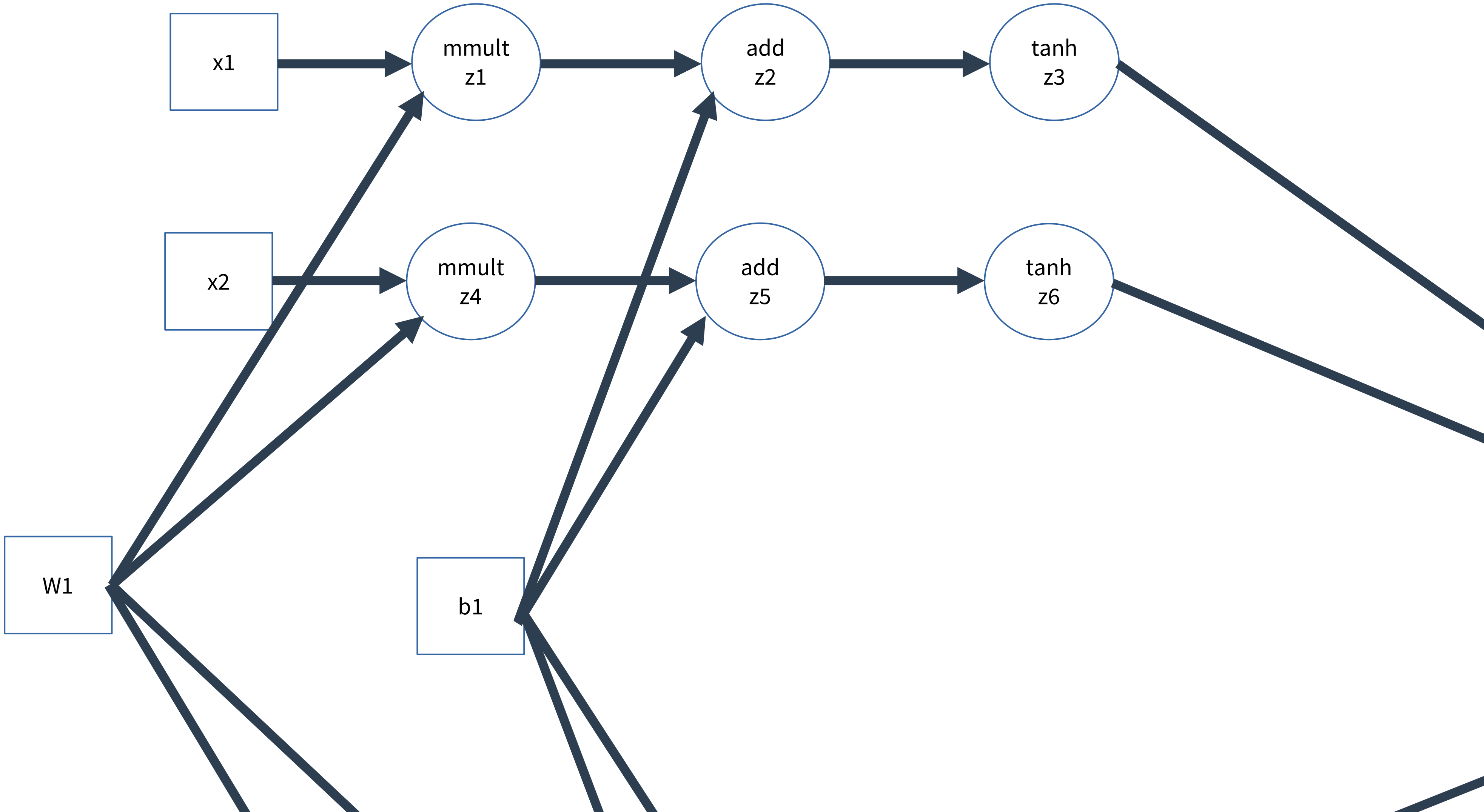
Nodes can have multiple avenues of influence

Gradient **accumulation** is especially important…



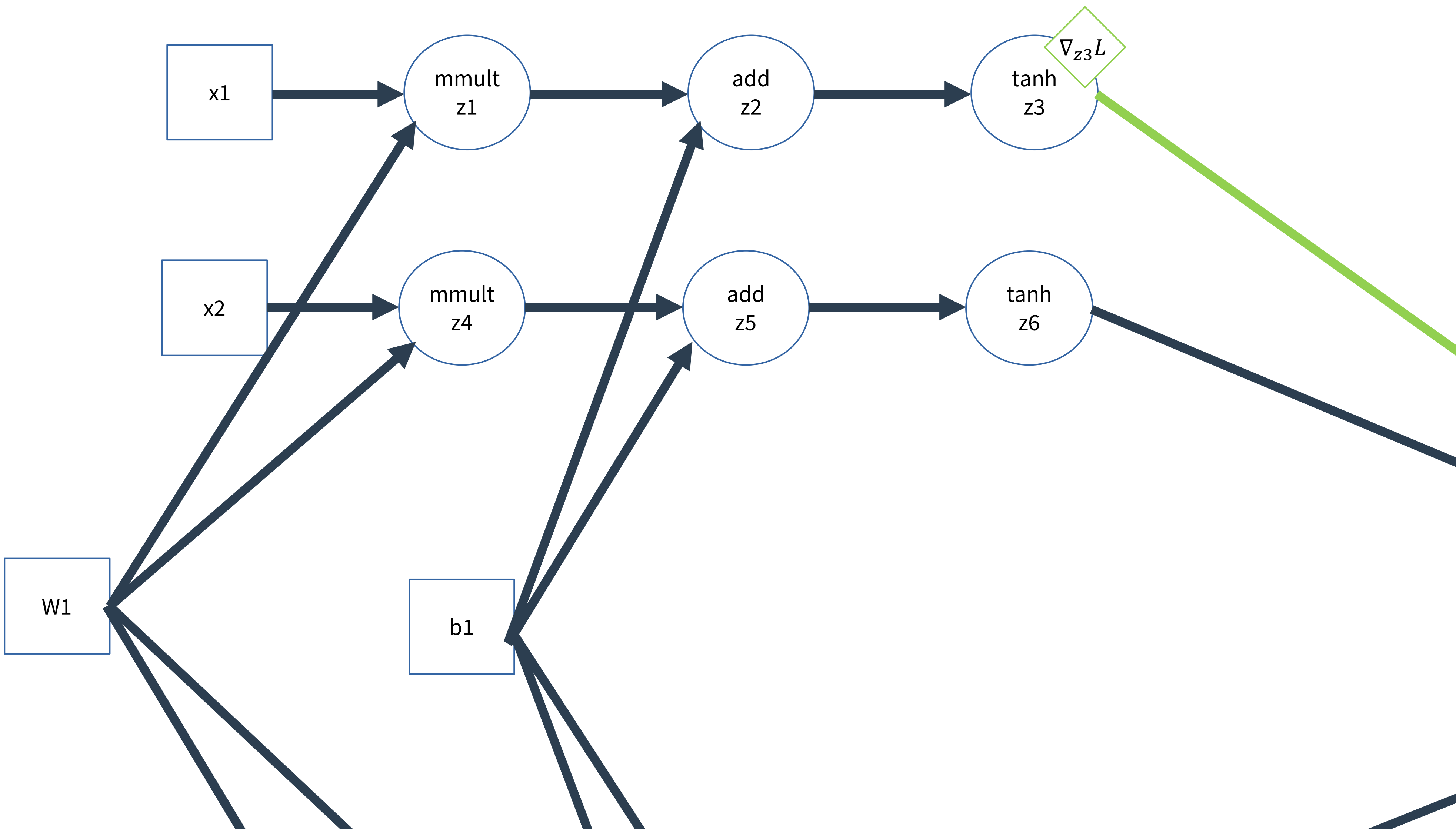
Shared Parameter Networks (Scanning MLP)

Let's DFS…



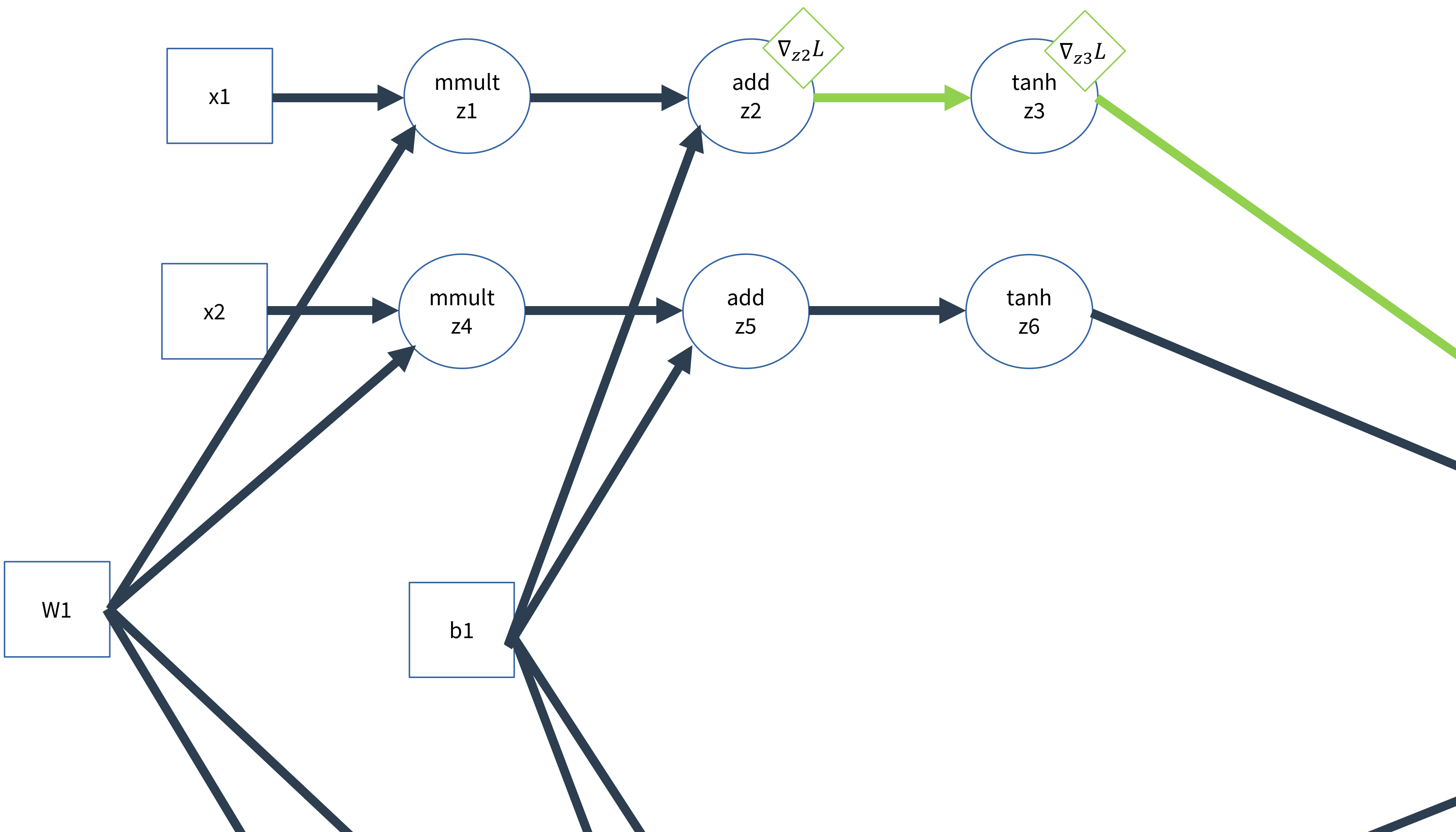
Shared Parameter Networks (Scanning MLP)

Let's DFS…



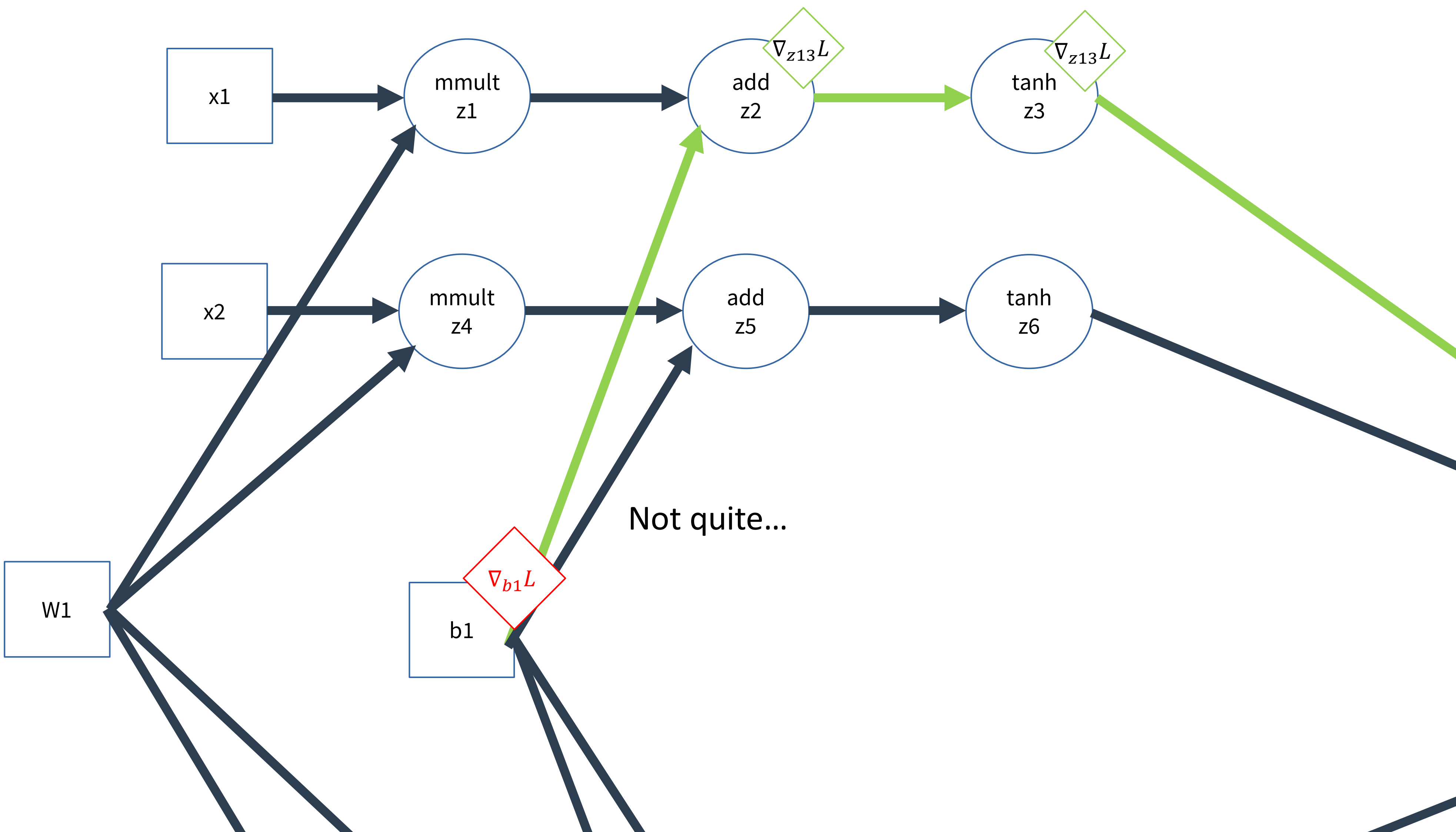
Shared Parameter Networks (Scanning MLP)

Let's DFS...



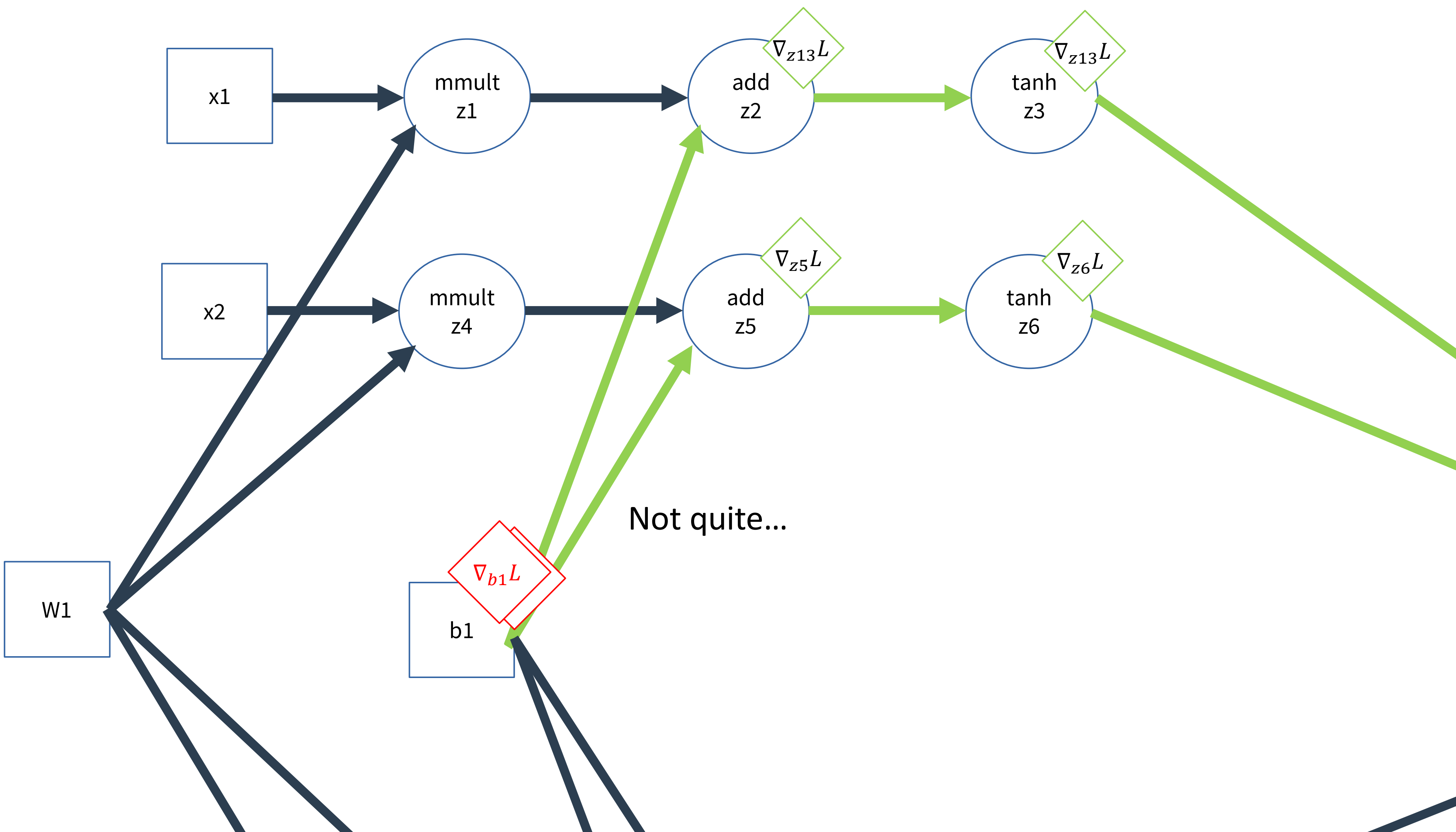
Shared Parameter Networks (Scanning MLP)

Let's DFS...



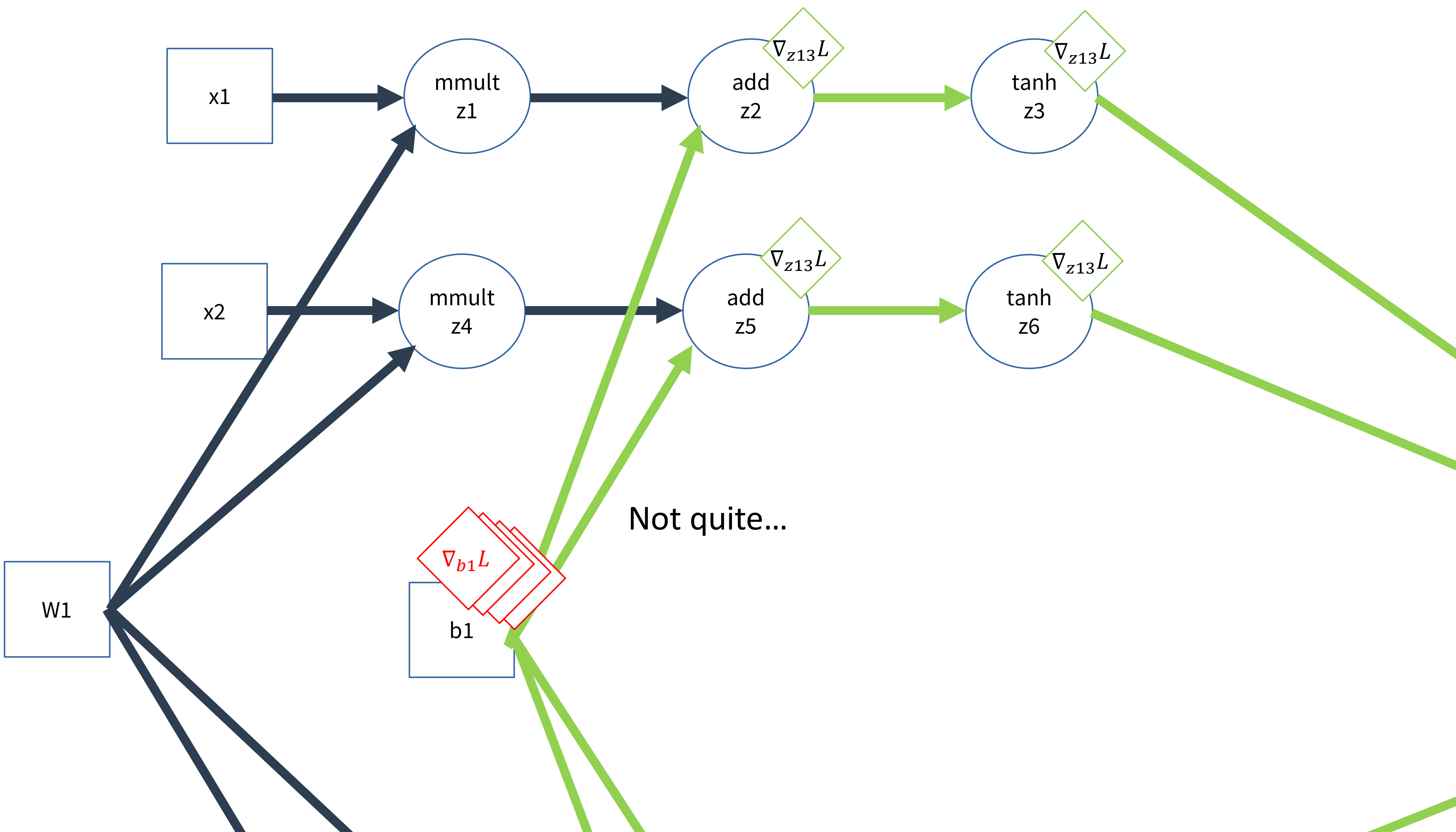
Shared Parameter Networks (Scanning MLP)

Let's DFS…



Shared Parameter Networks (Scanning MLP)

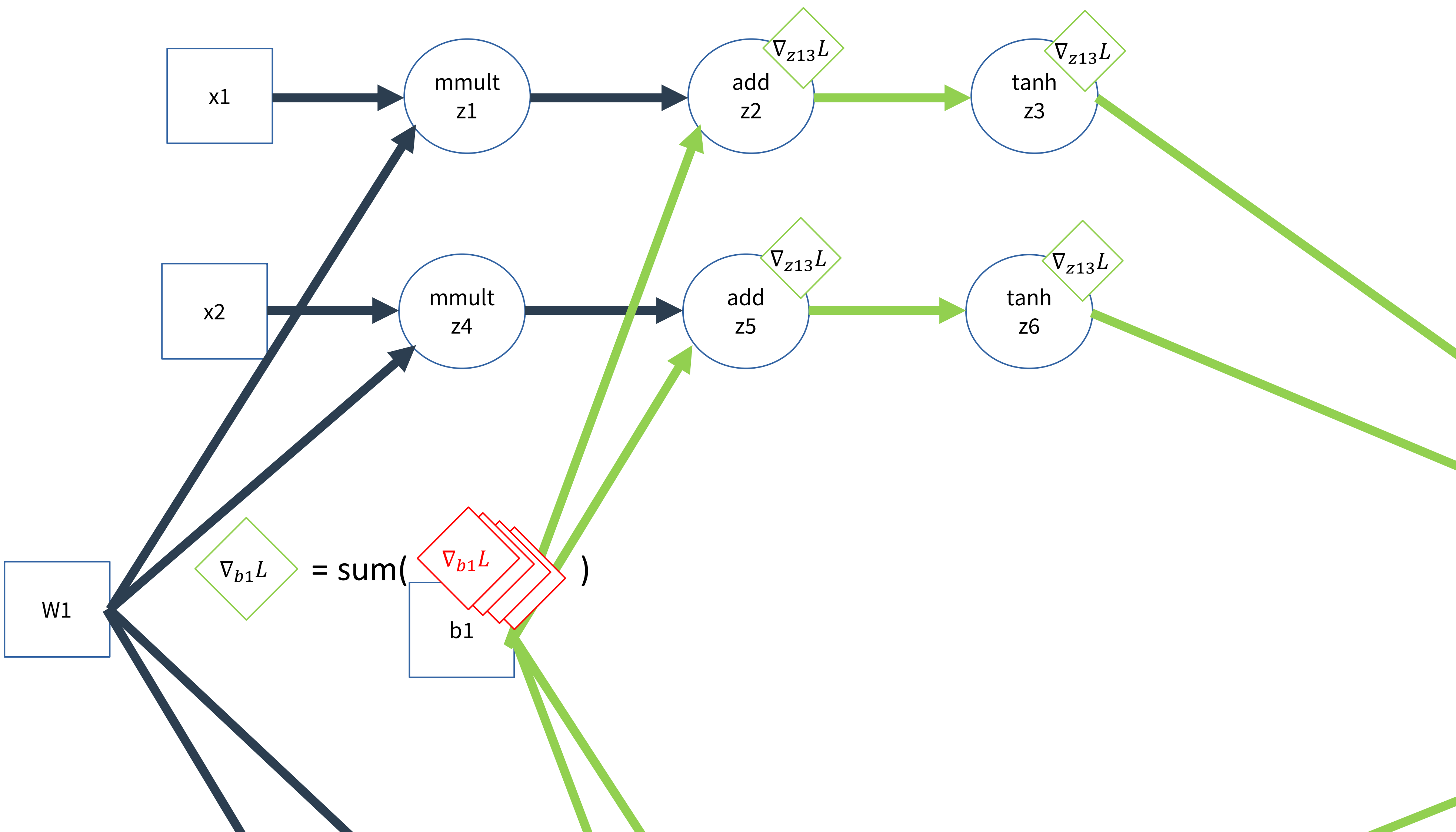
Let's DFS…



Not quite…

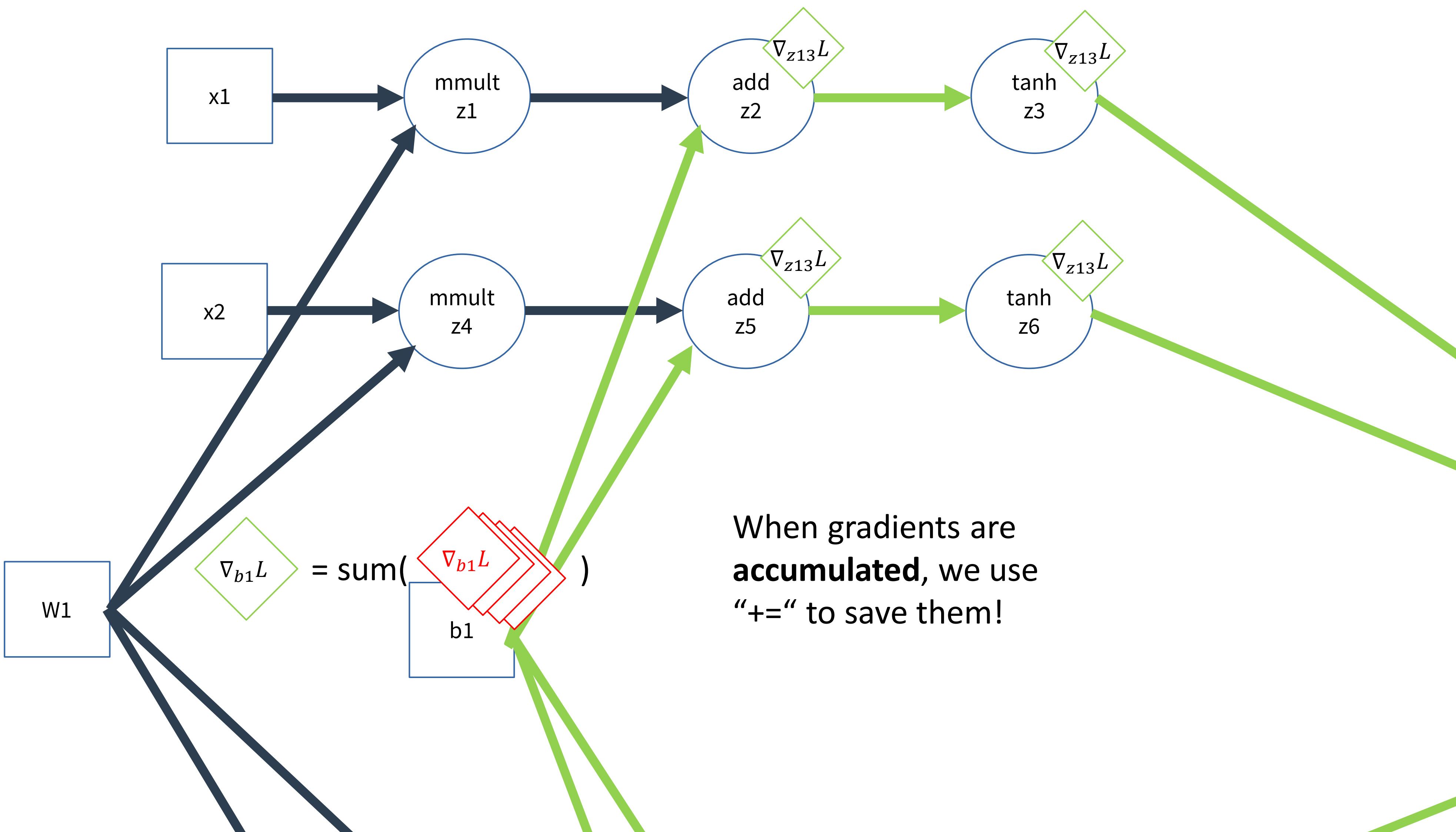
Shared Parameter Networks (Scanning MLP)

Let's DFS…



Shared Parameter Networks (Scanning MLP)

Let's DFS…



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

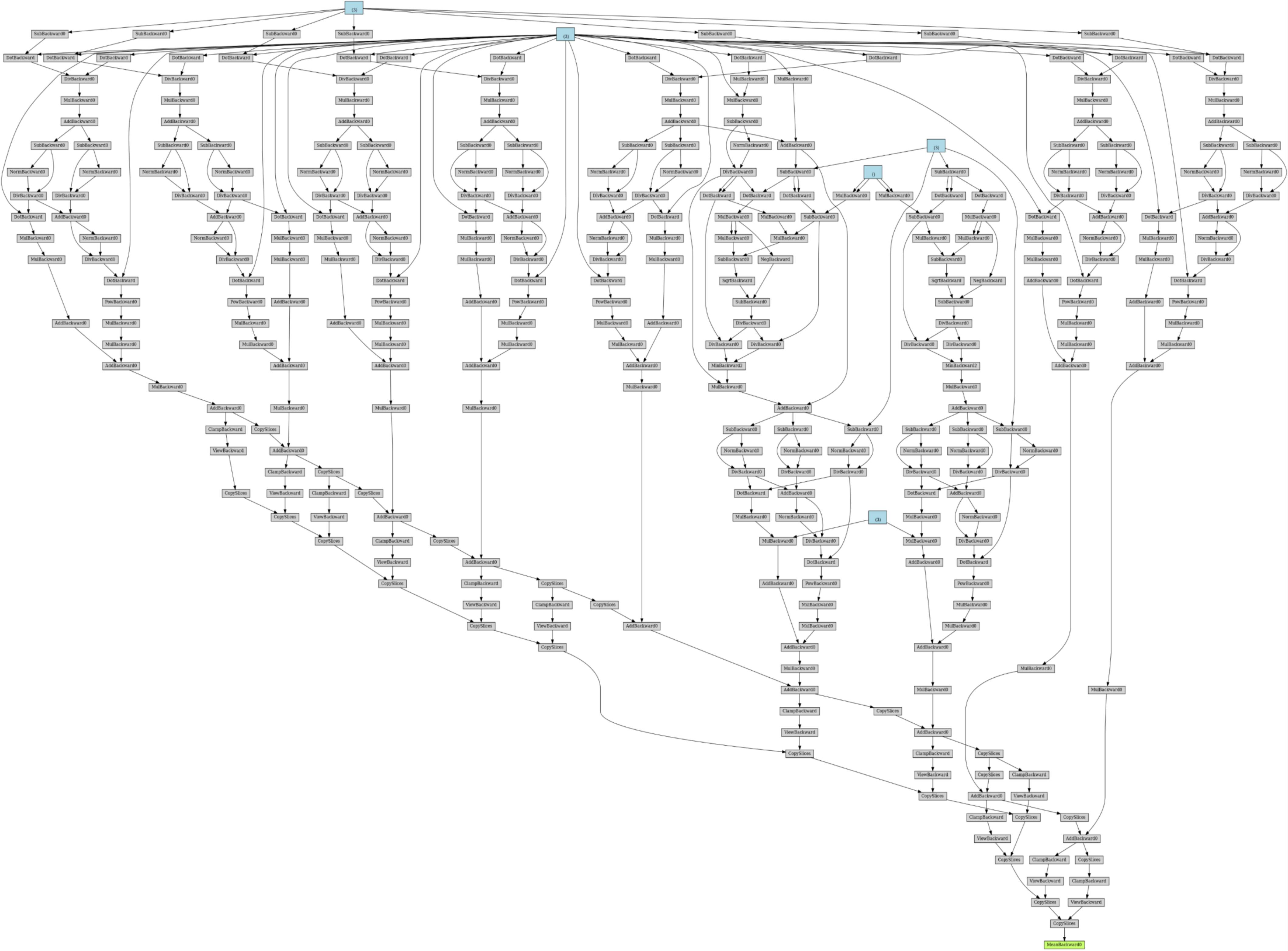
Autograd

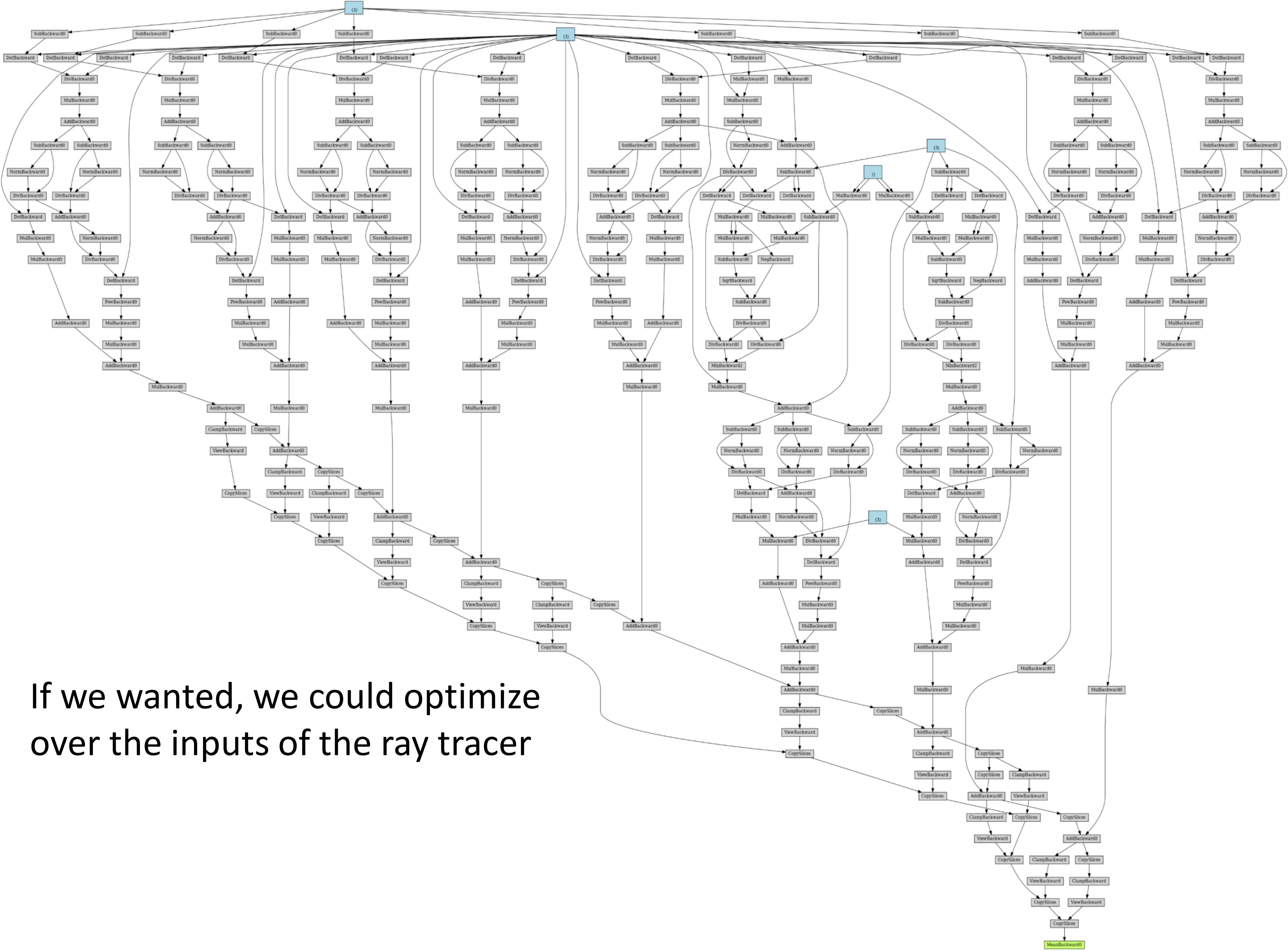
- Pytorch builds an implicit graph when you perform operations (also hw1p1)
  - +, -, *, /
  - Batchnorm, Softmax…
- You can also build this graph on paper to calculate derivatives

As an example, we'll show the graph for a ray tracer for 4x3 images

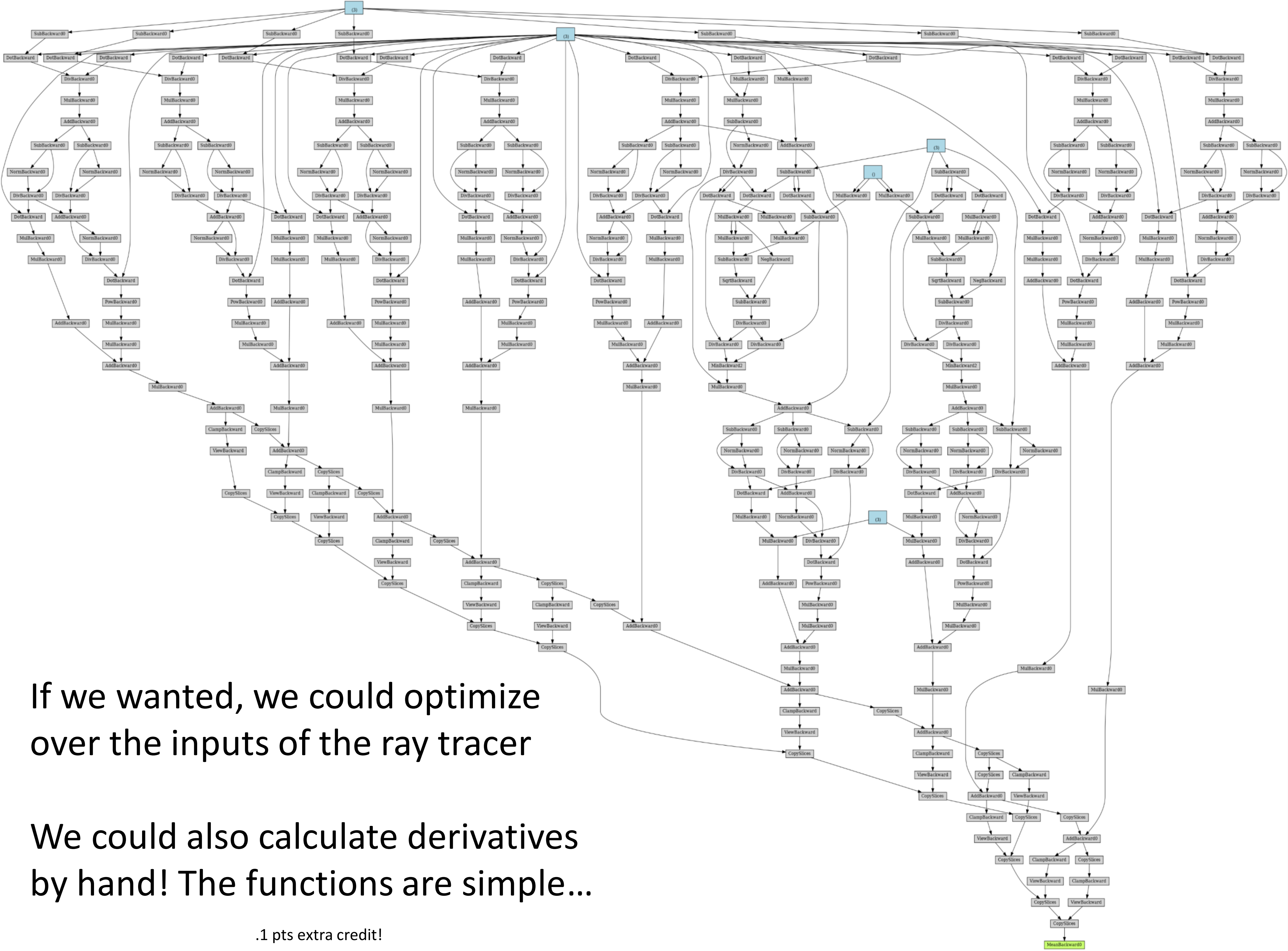
As an example, we'll show the graph for a ray tracer for 4x3 images

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!