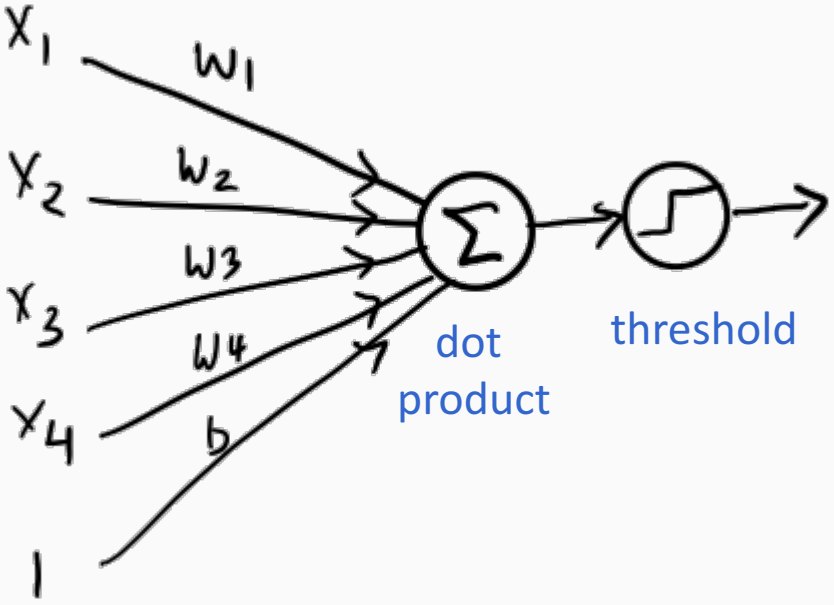
Neural Networks and Computation Graphs

Based on slides and material from Geoffrey Hinton, Richard Socher, Yoav Goldberg, and others.
The computation graph slides are based on a tutorial “Practical Neural Networks for NLP” by Chris Dyer, Yoav Goldberg, Graham Neubig in EMNLP 2016

We have seen linear threshold units



Prediction

$$sgn\ (\boldsymbol{w}^T \boldsymbol{x} + b) = sgn(\sum w_i x_i + b)$$

Learning

various algorithms

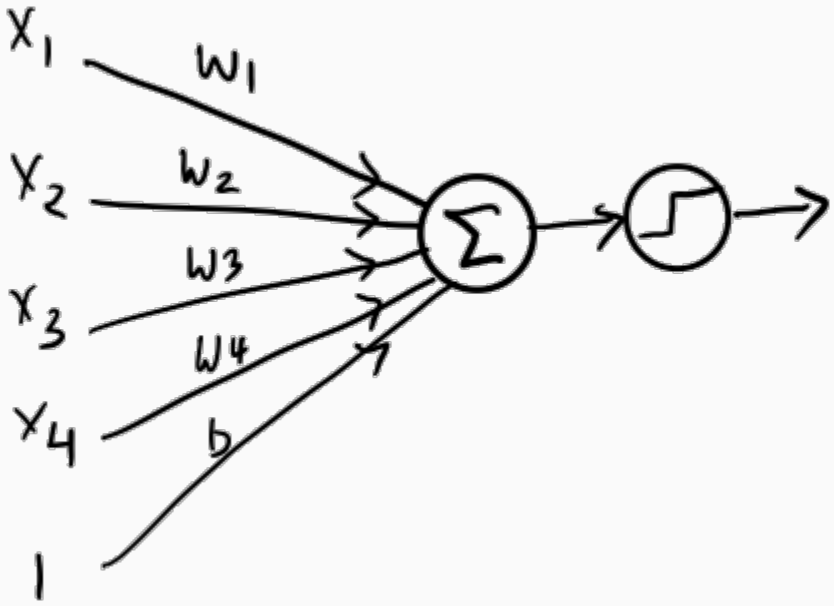
perceptron, SVM, logistic regression,…

in general, minimize loss

But where do these input features come from?

What if the features were outputs of another classifier?

Features from classifiers



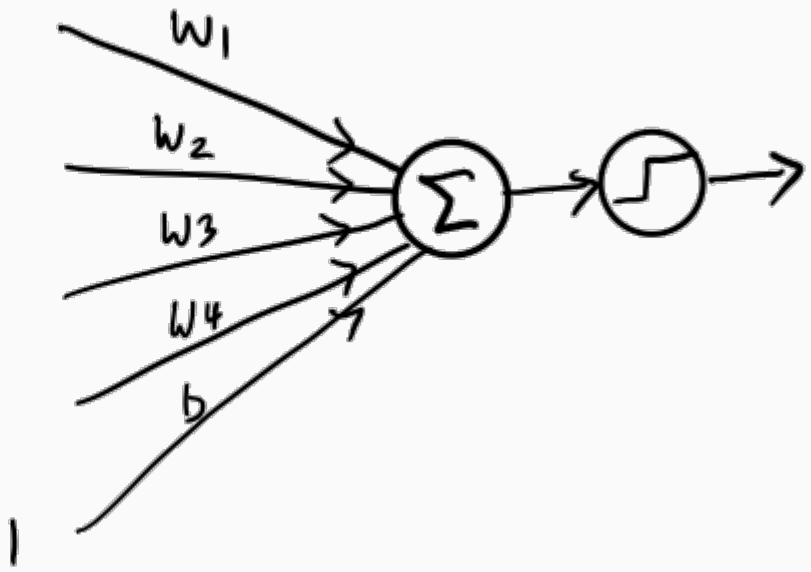
Features from classifiers

x_1

x_2

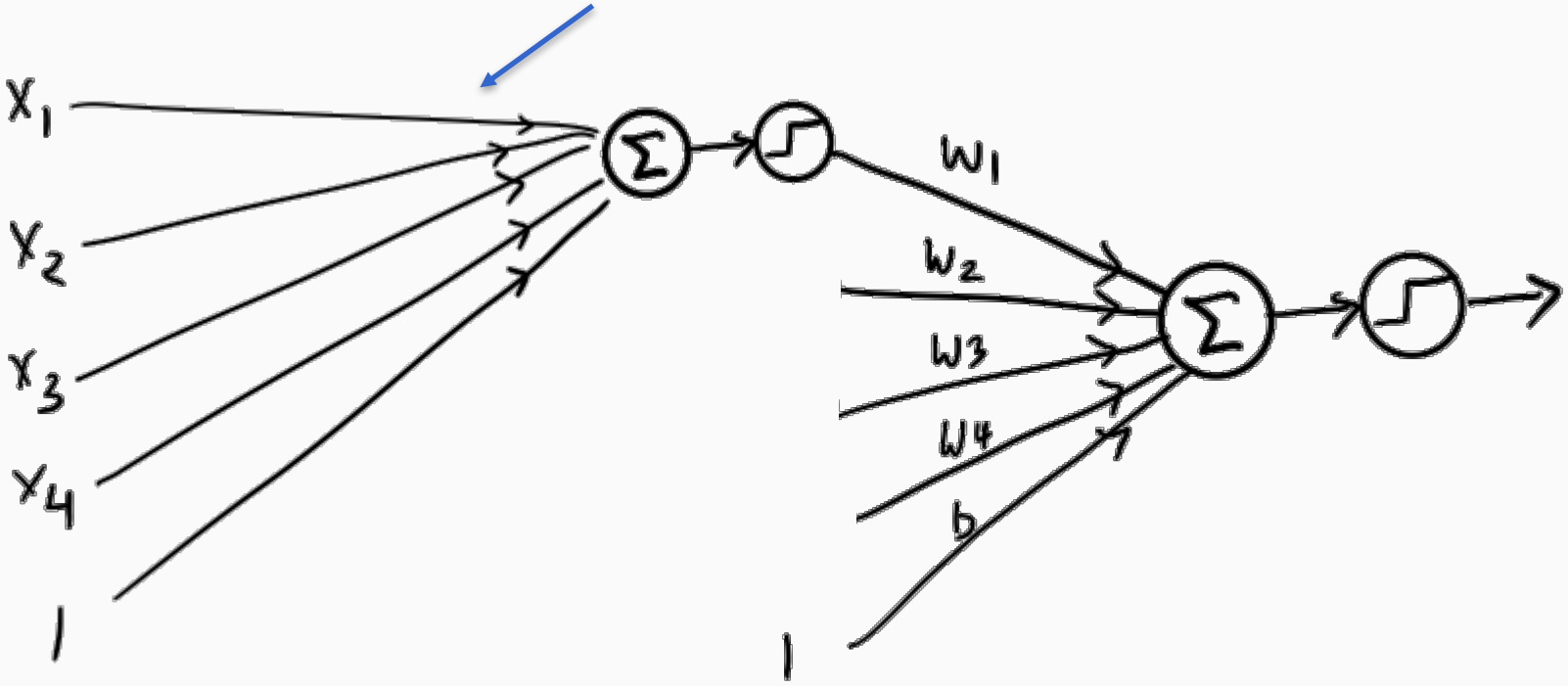
x_3

x_4

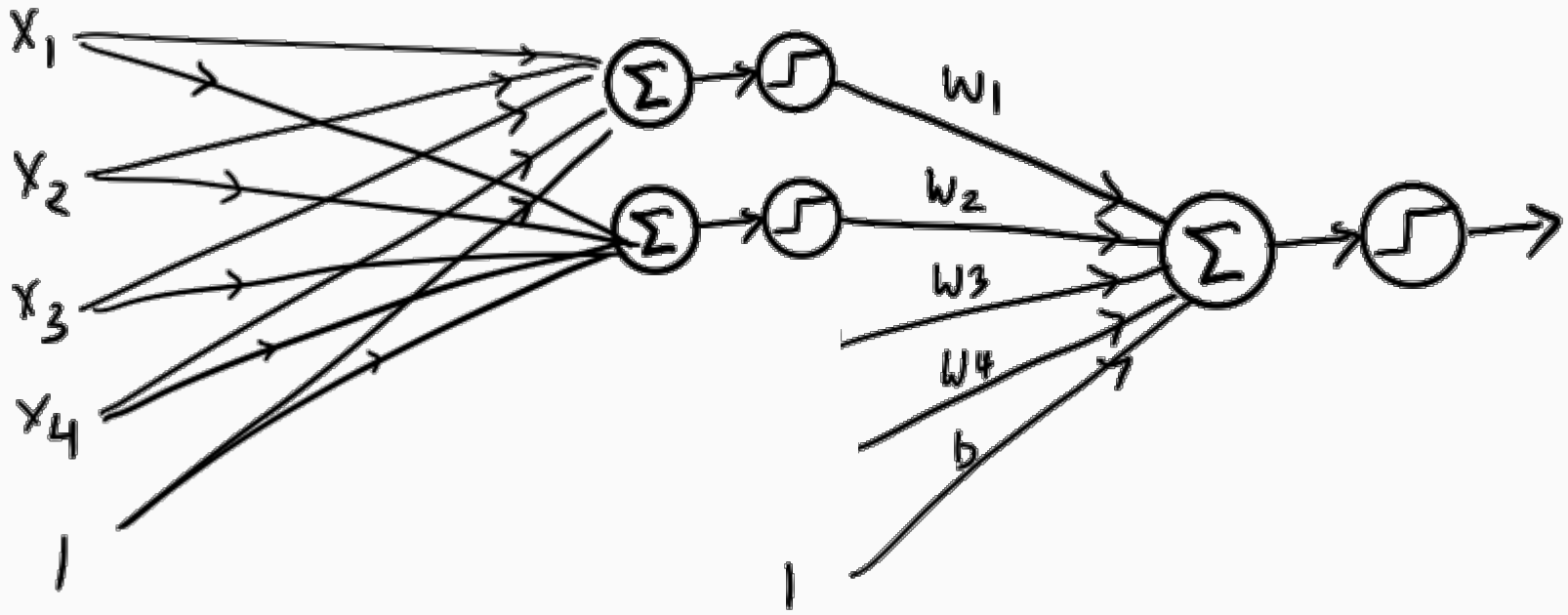


Features from classifiers

Each of these connections have their own weights as well

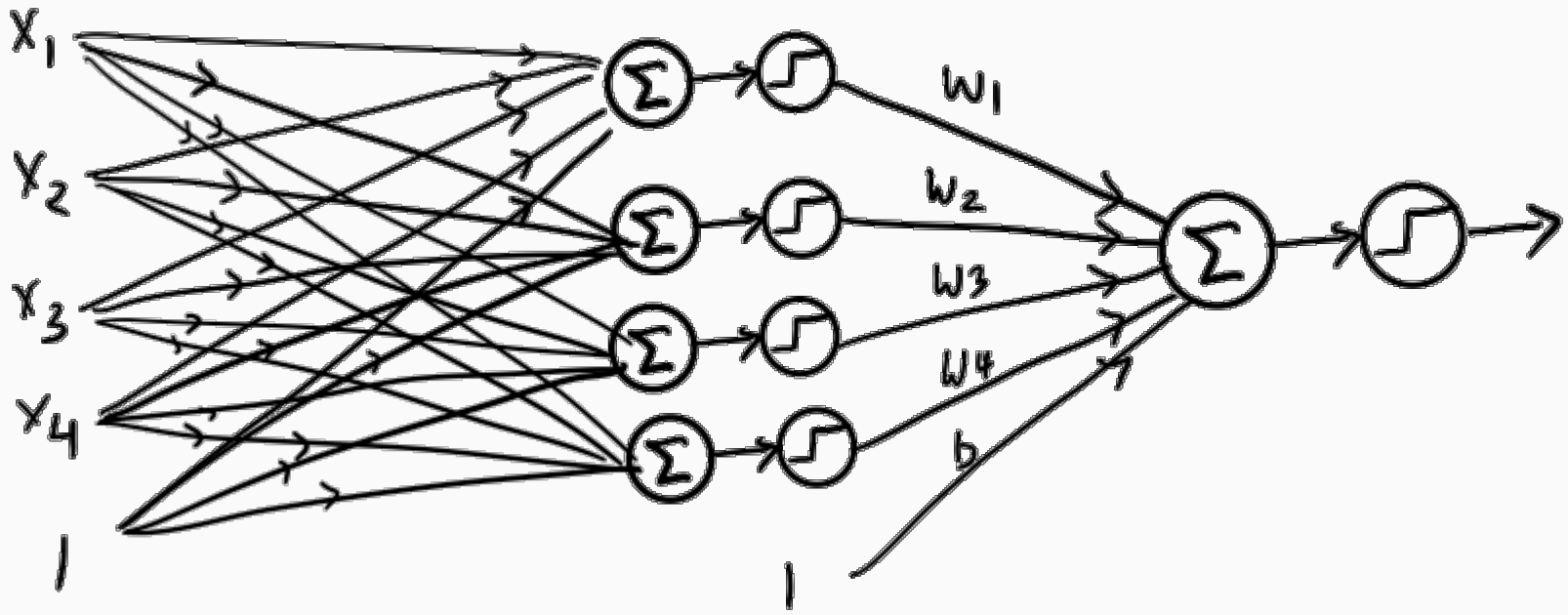


Features from classifiers



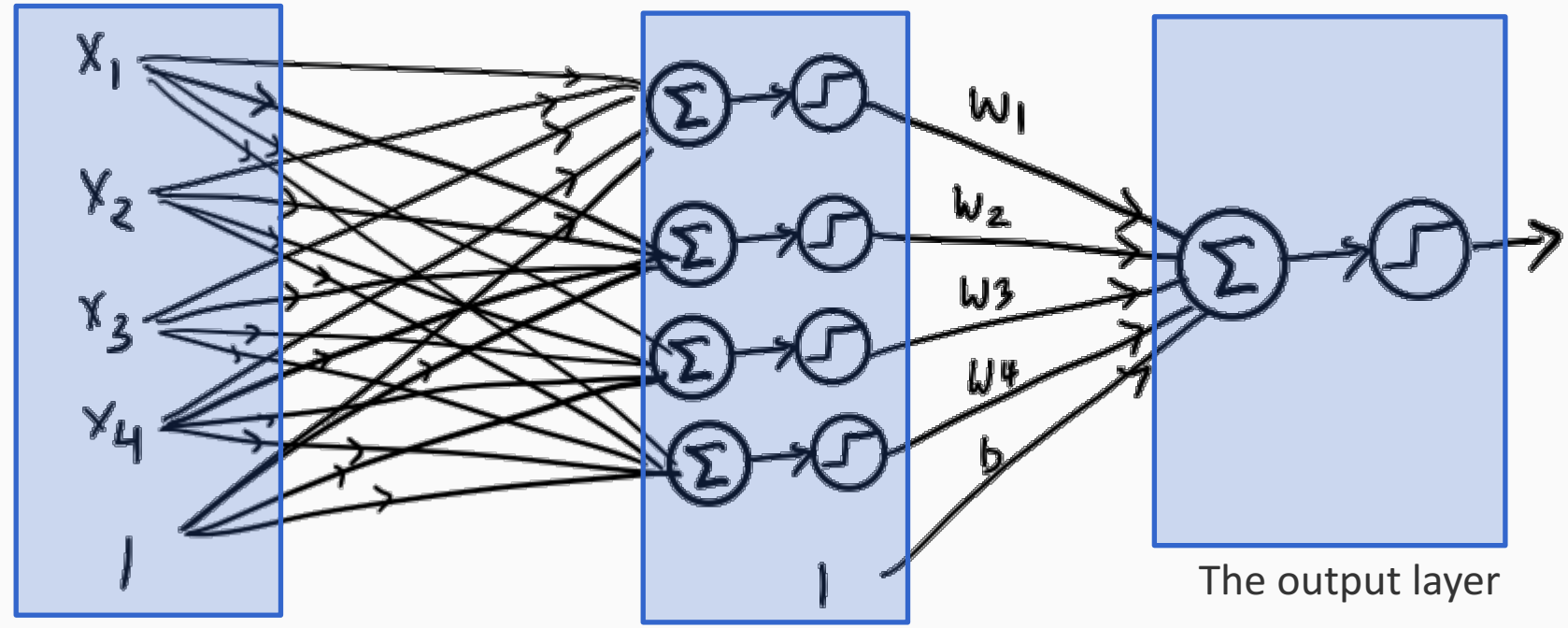
Features from classifiers

This is a **two layer** feed forward neural network



Features from classifiers

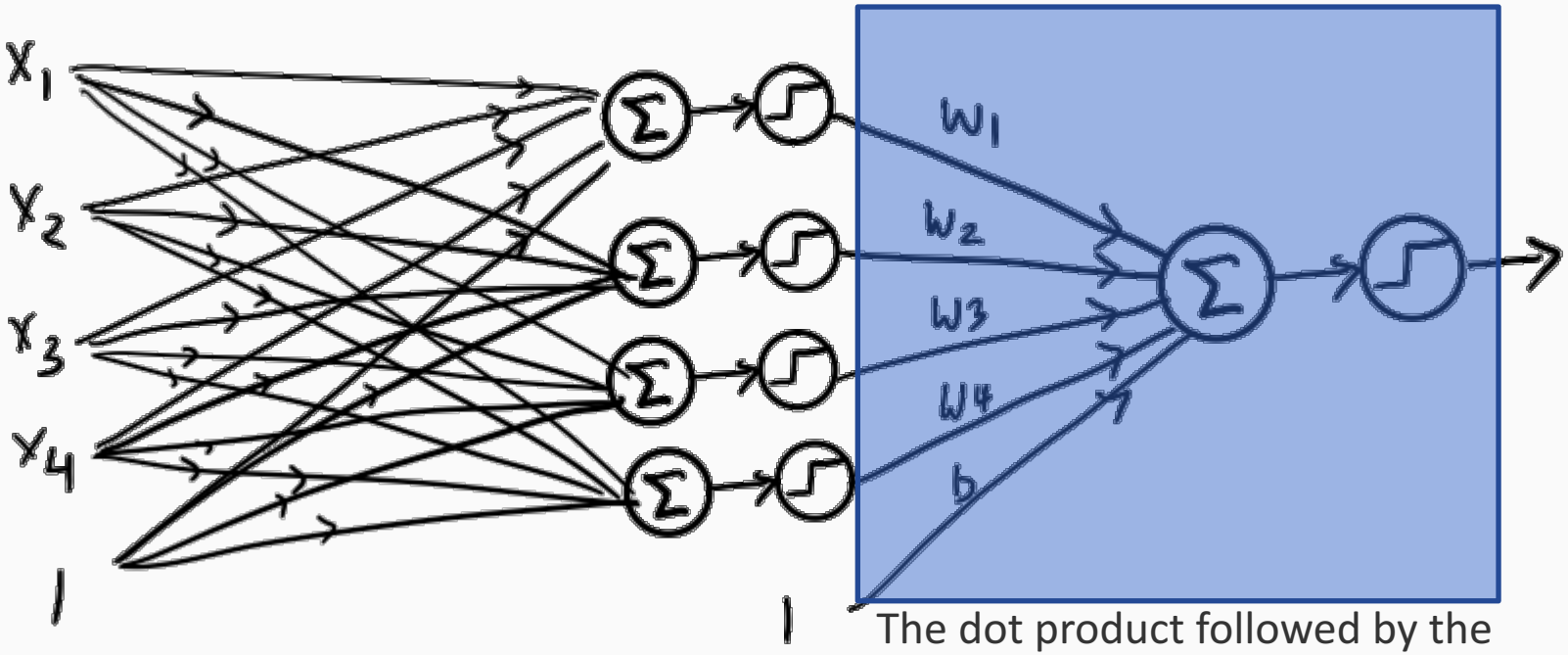
This is a **two layer** feed forward neural network



Think of the hidden layer as learning a good **representation** of the inputs

Features from classifiers

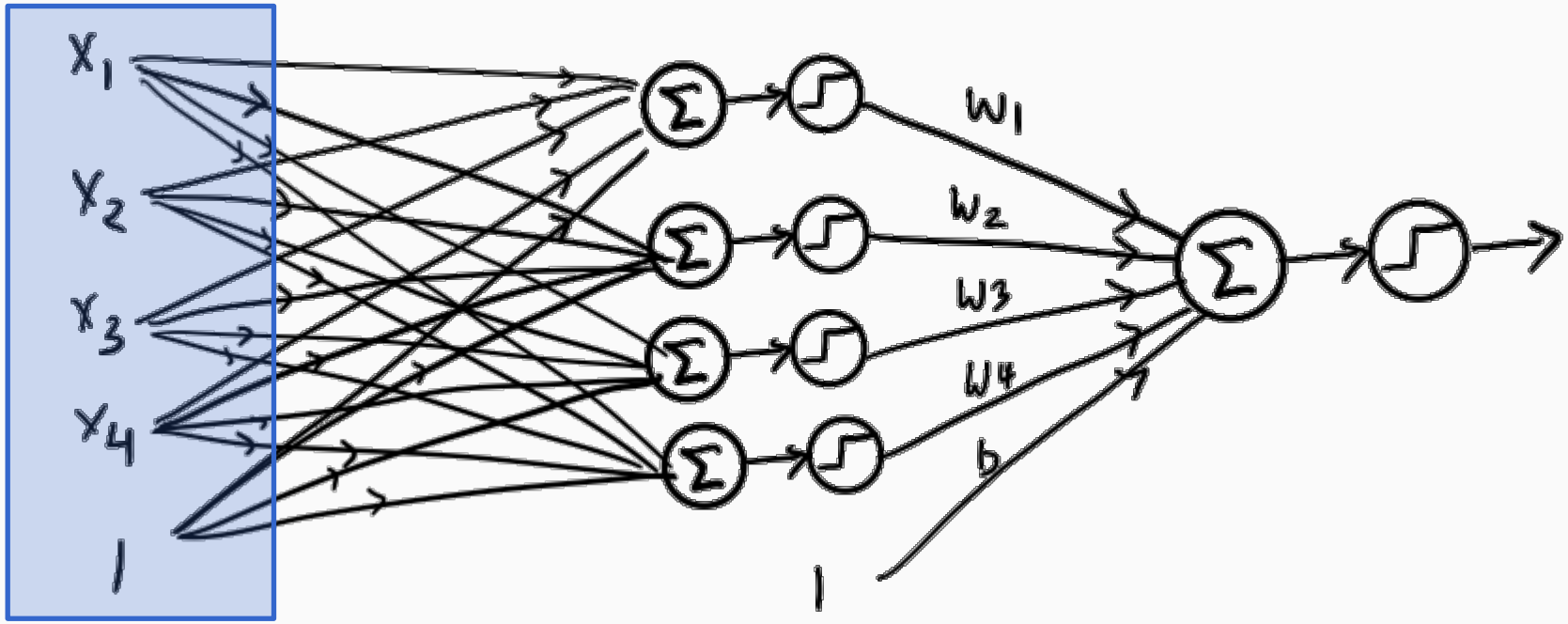
This is a **two layer** feed forward neural network



The dot product followed by the threshold constitutes a ***neuron***

Five neurons in this picture (four in hidden layer and one output)

But where do the inputs come from?



The input layer

What if the inputs were the outputs of a classifier?

We can make a **three** layer network…. And so on.

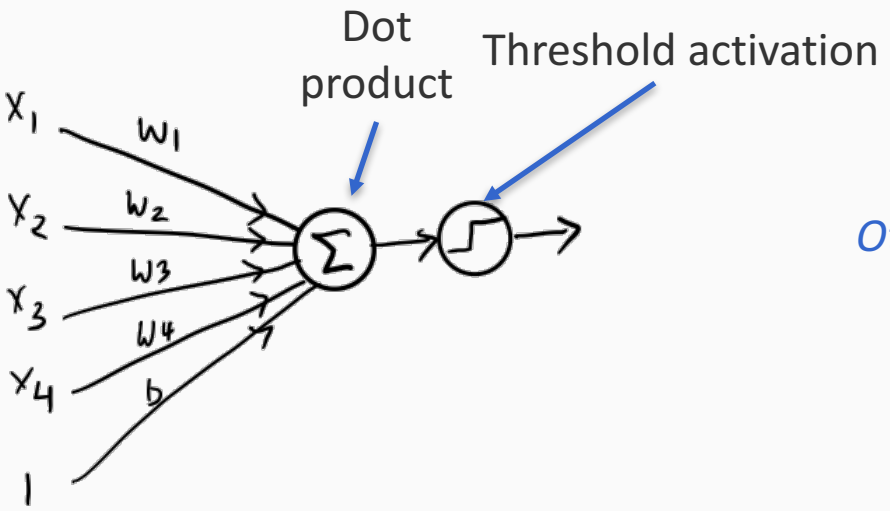
Artificial neurons

*Functions that **very loosely** mimic a biological neuron*

A neuron accepts a collection of inputs (a vector **x**) and produces an output by:

- Applying a dot product with weights **w** and adding a bias b
- Applying a (possibly non-linear) transformation called an ***activation***

$$output = activation(\boldsymbol{w}^T \boldsymbol{x} + b)$$



Other activations are possible

Activation functions

Also called transfer functions

$$output = activation(\boldsymbol{w}^T \boldsymbol{x} + b)$$

Name of the neuron	Activation function: $activation(z)$
Linear unit	z
Threshold/sign unit	$\text{sgn}(z)$
Sigmoid unit	$\dfrac{1}{1+\exp(-z)}$
Rectified linear unit (ReLU)	$\max(0, z)$
Tanh unit	$\tanh(z)$

Many more activation functions exist (sinusoid, sinc, gaussian, polynomial…)

A neural network

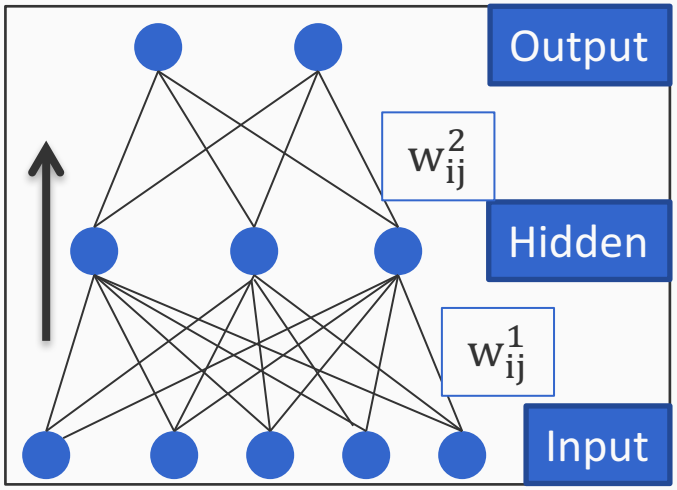
A function that converts inputs to outputs defined by a directed acyclic graph

- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights

A neural network

A function that converts inputs to outputs defined by a directed acyclic graph

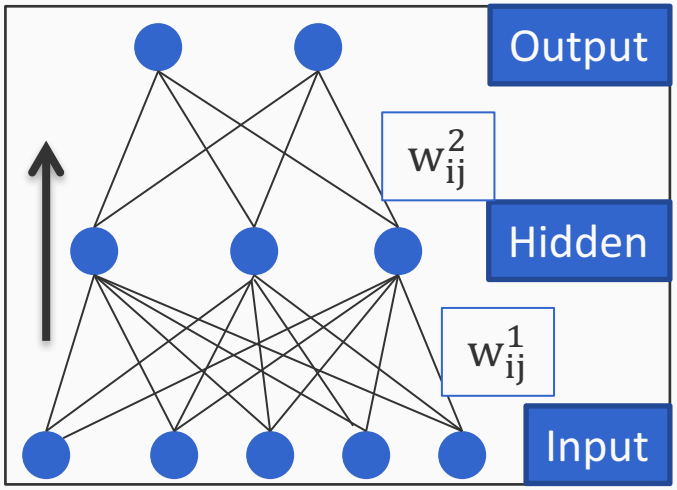
- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights



A neural network

A function that converts inputs to outputs defined by a directed acyclic graph

- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights

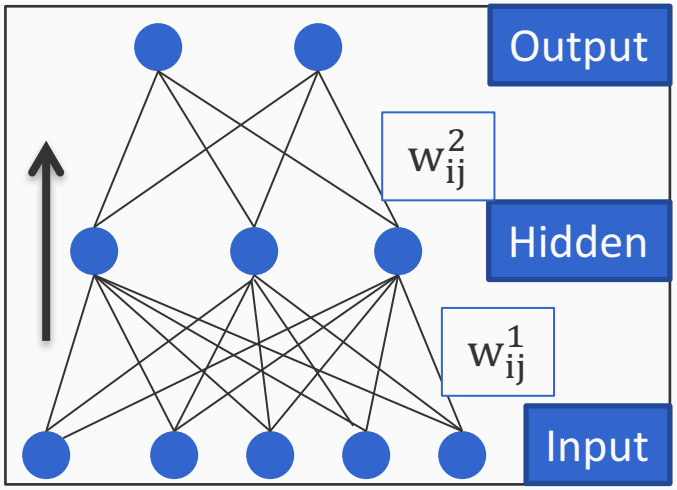


- To define a neural network, we need to specify:
  - The structure of the graph
    - How many nodes, the connectivity
  - The activation function on each node
  - The edge weights

A neural network

A function that converts inputs to outputs defined by a directed acyclic graph

- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights



- To define a neural network, we need to specify:
  - The structure of the graph
    - How many nodes, the connectivity
  - The activation function on each node

  Called the *architecture* of the network
  Typically predefined, part of the design of the classifier

  - The edge weights

  Learned from data

Computation graphs

- A language for constructing deep neural networks
  - A way to think about ***differentiable compute***

- Key ideas:
  - We can represent functions as graphs
  - We can dynamically generate these graphs if necessary
  - We can define algorithms over these graphs that map to learning and prediction
    - Prediction via the forward pass
    - Learning via gradients computed using the backward pass

What we will see

1. What is the semantics of a computation graph?
   - That is, what the nodes and edges mean
2. How to construct them
3. How do perform computations with them

Nodes represent values

Expression **x**

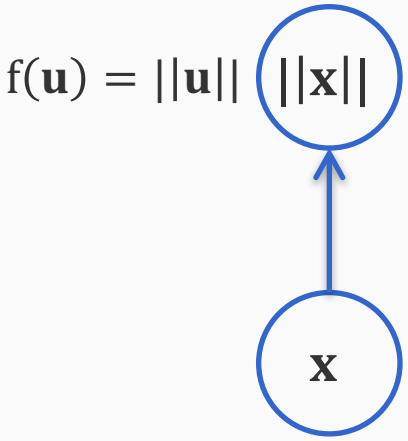
Graph

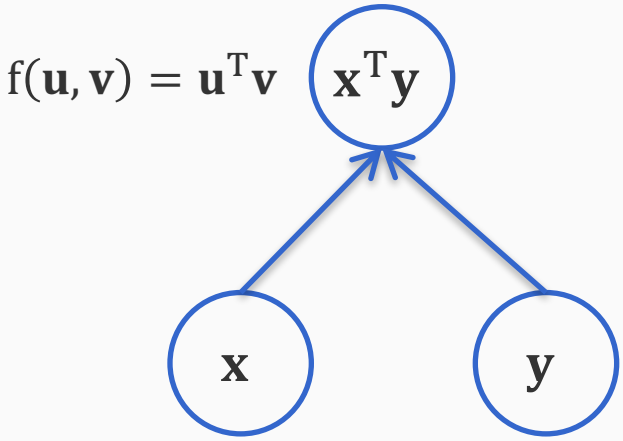
The value is implicitly or explicitly typed.

It could represent a

- Scalar (i.e. a number)
- A vector
- A matrix
- Or more generally, a tensor

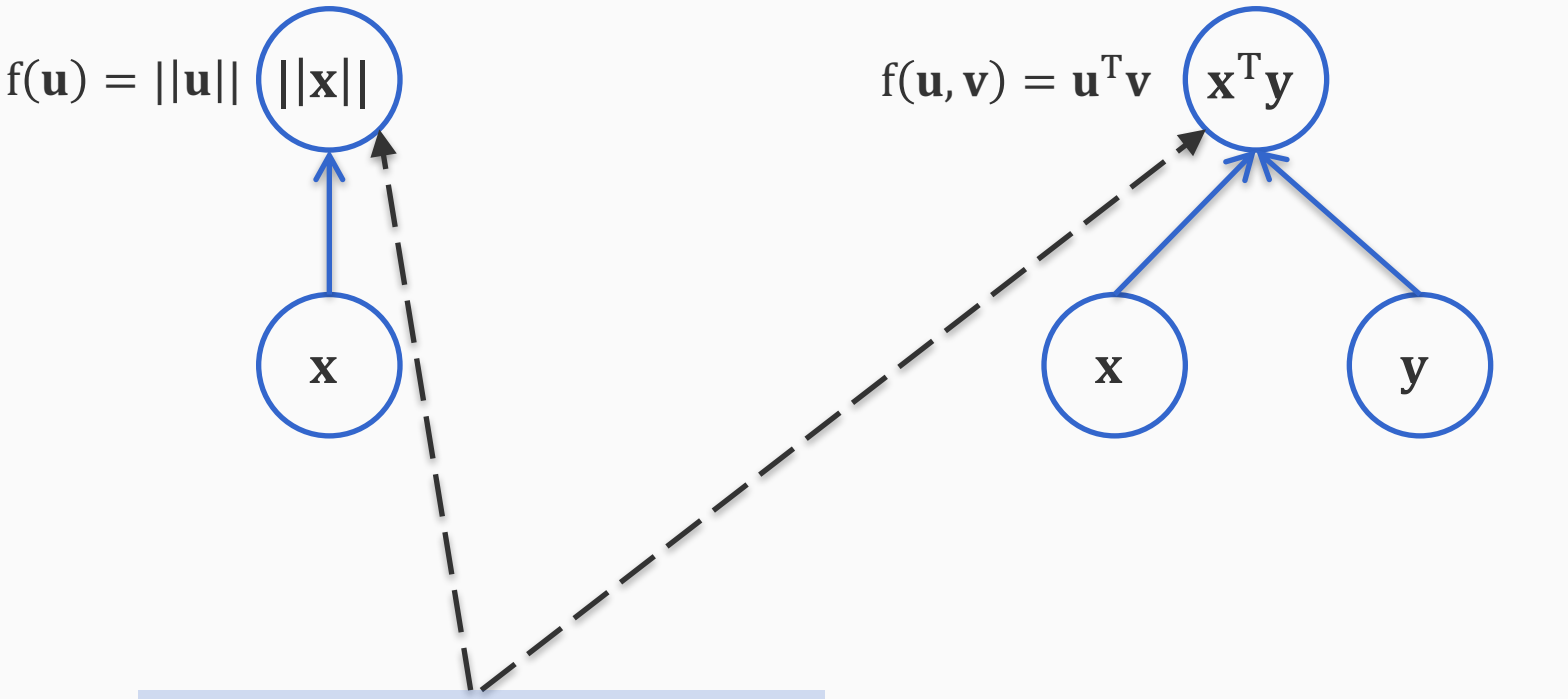
Edges represent function arguments





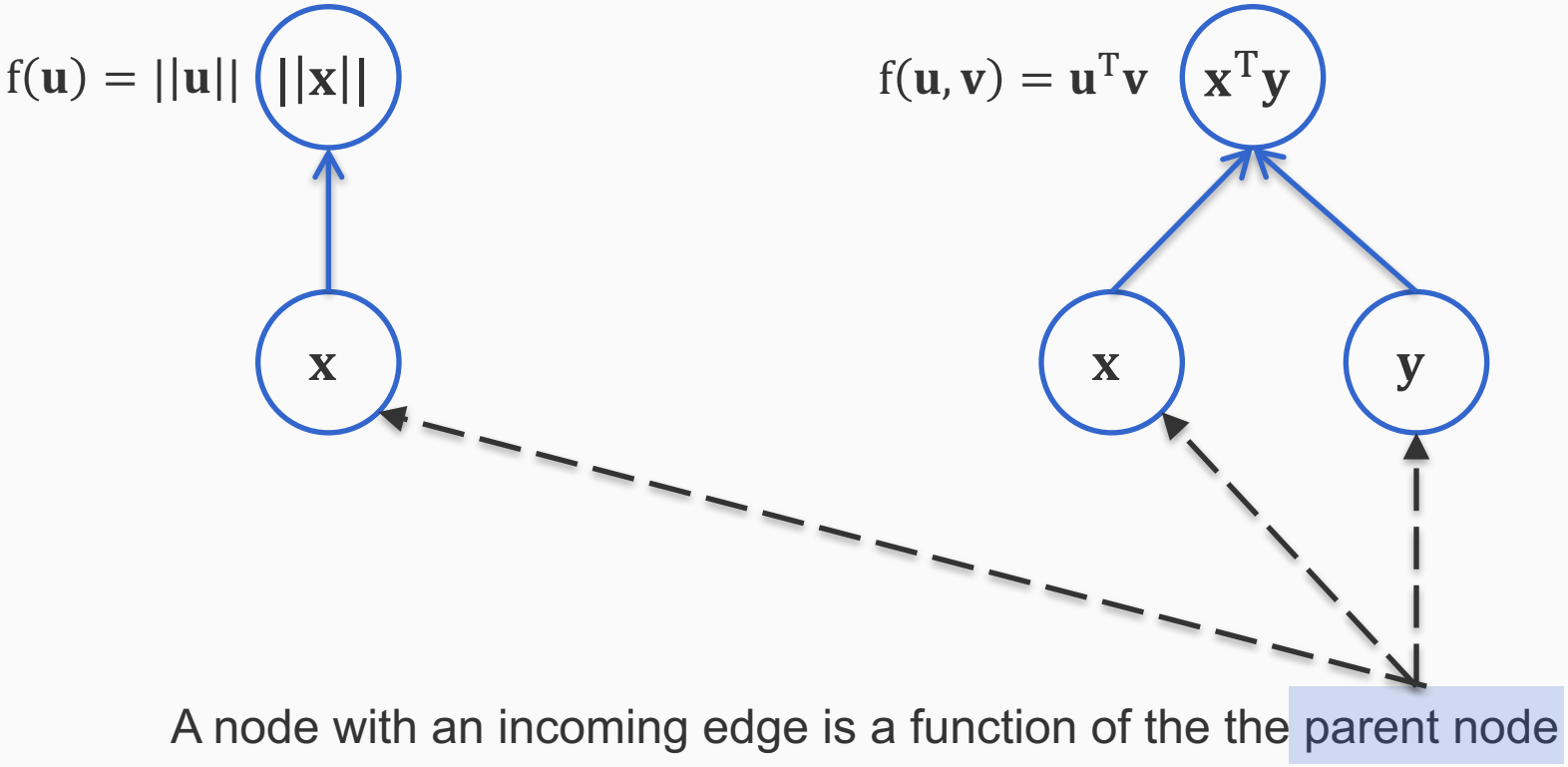
A node with an incoming edge is a function of the the parent node

Edges represent function arguments



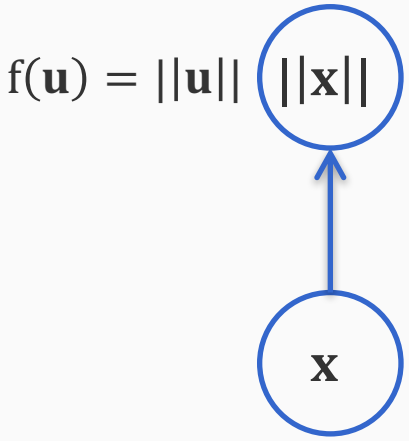
A node with an incoming edge is a function of the the parent node

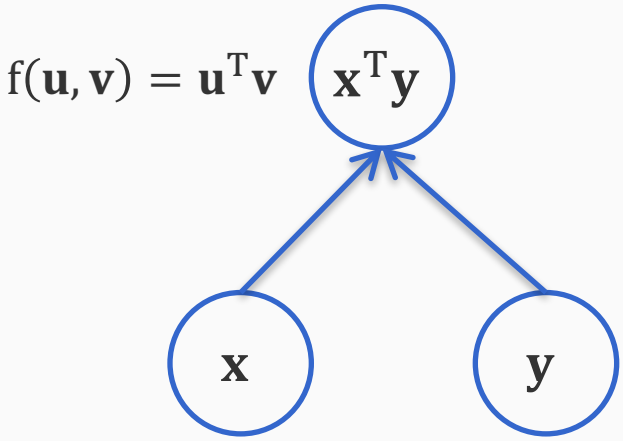
Edges represent function arguments



A node with an incoming edge is a function of the the parent node

Edges represent function arguments





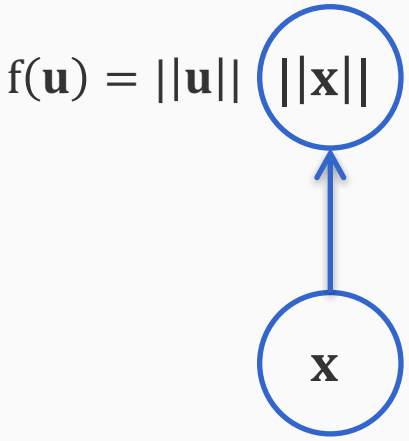
Each node knows how to compute two things:

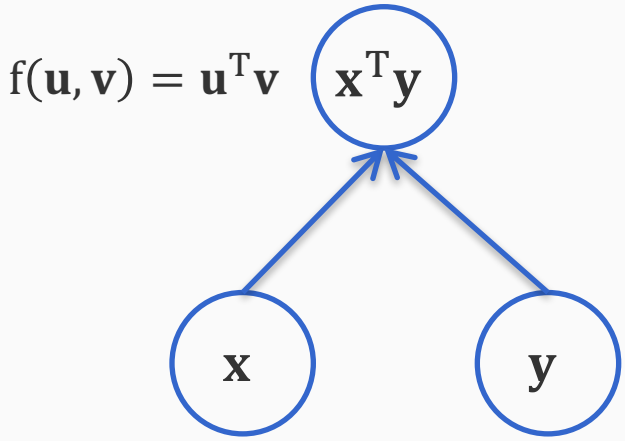
1. Its own value using its inputs
   - In these examples, the nodes on top compute $||\mathbf{x}||$ and $\mathbf{x}^T\mathbf{y}$

Notation: We will write down what that function is next to the node.

When we write this, we will use formal arguments (here, the $\mathbf{u}$ and $\mathbf{v}$). Think of these as similar to the argument names we use when we declare functions while programming.

Edges represent function arguments

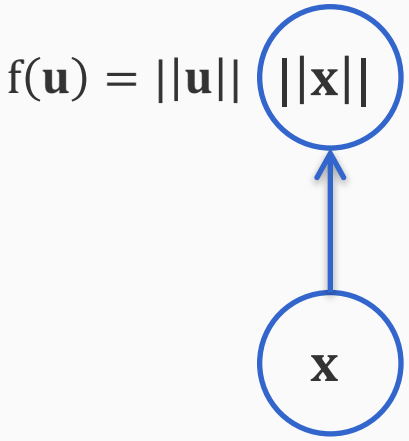


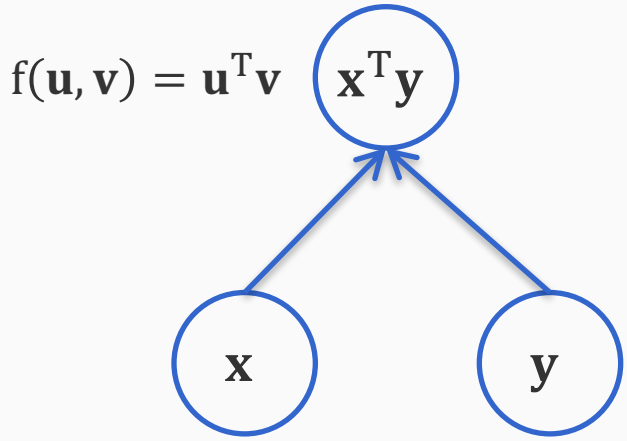


Each node knows how to compute two things:

1. Its own value using its inputs
   - In these examples, the nodes on top compute $||\mathbf{x}||$ and $\mathbf{x}^T\mathbf{y}$
2. The value of its partial derivative with respect to each input
   - Left graph: the node on top knows to compute $\frac{\partial f}{\partial \mathbf{u}}$
   - Right graph: the node on top knows to compute $\frac{\partial f}{\partial \mathbf{u}}$ and $\frac{\partial f}{\partial \mathbf{v}}$

Graphs represent functions





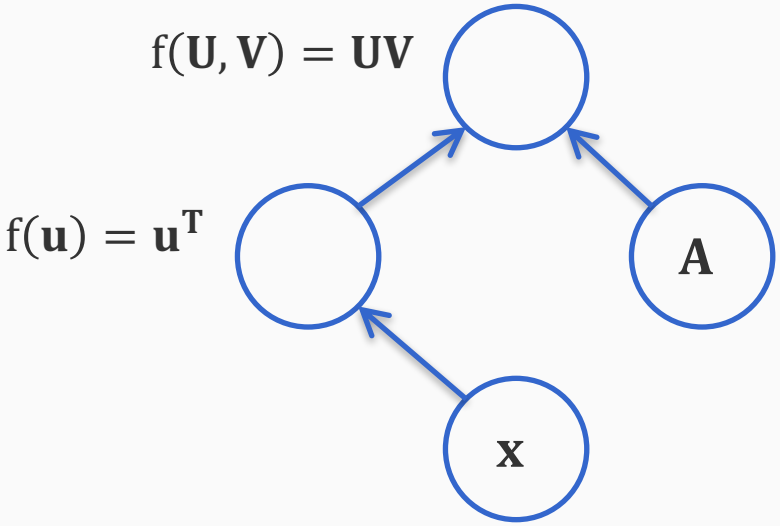
The functions expressed could be

- Nullary, i.e. with no arguments: if a node has no incoming edges
- Unary: if a node has one incoming edge
- Binary: if a node has two incoming edges
- …
- n-ary: if a node has n incoming edges

Let's see some functions as graphs

Expression $\mathbf{x}^{\mathrm{T}}\mathbf{A}$

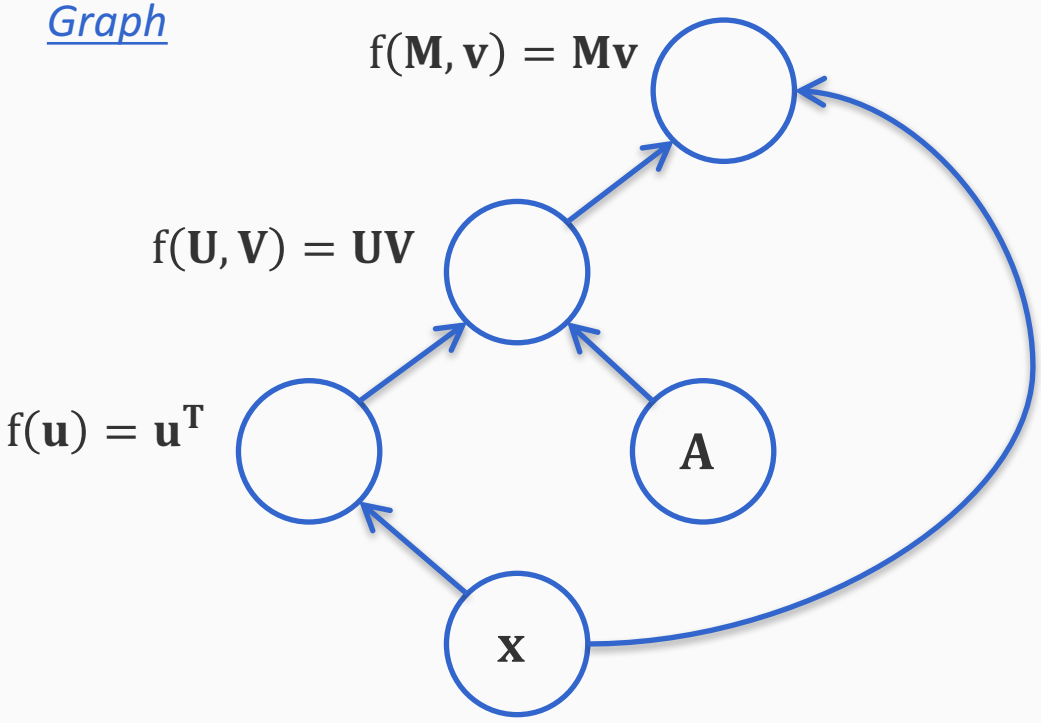
Graph



Let's see some functions as graphs

Expression $\mathbf{x}^T\mathbf{A}\mathbf{x}$

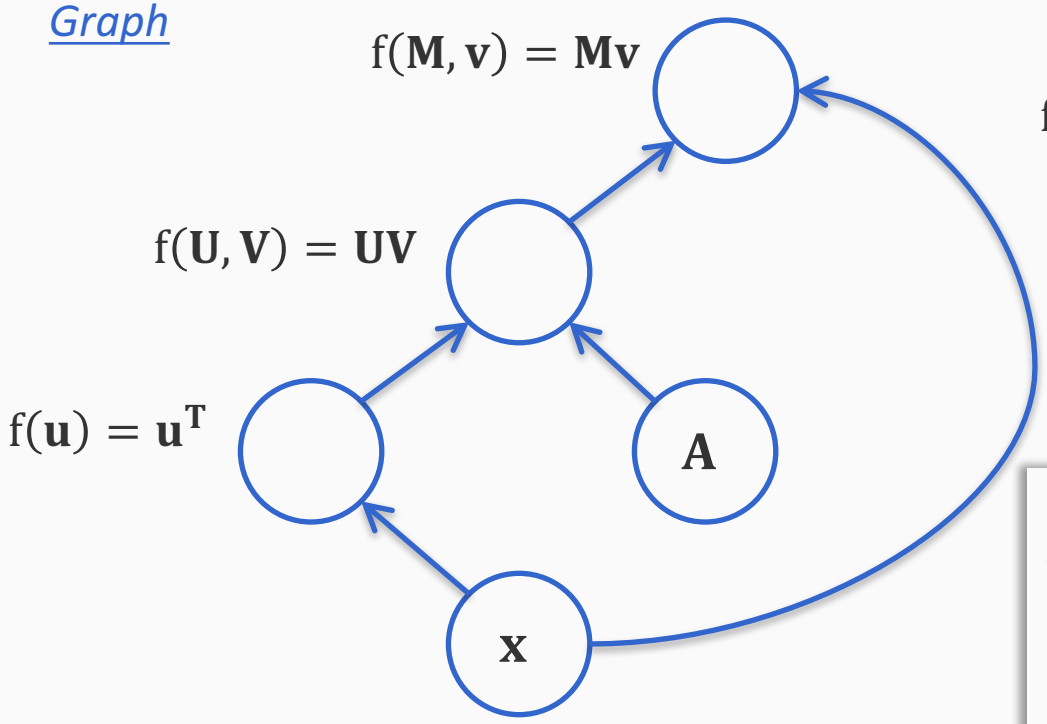
Graph

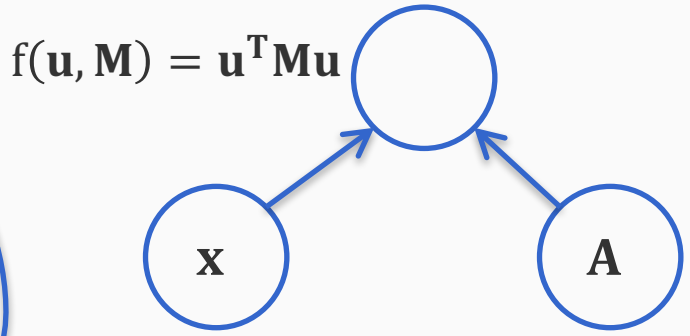


Let's see some functions as graphs

Expression $\mathbf{x}^T\mathbf{A}\mathbf{x}$

Graph





We could have written the same function with a different graph.

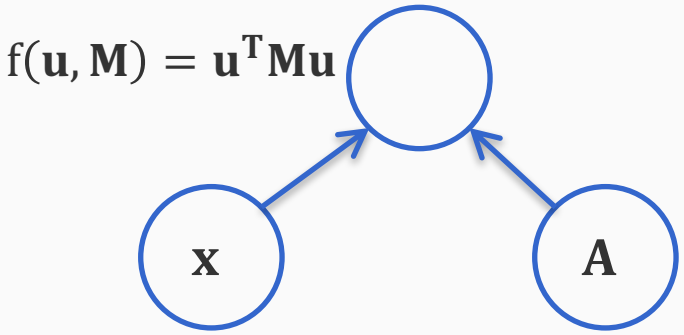
Computation graphs are not necessarily unique for a function

Let's see some functions as graphs

Expression $\mathbf{x}^T\mathbf{A}\mathbf{x}$

Remember: The nodes also know how to compute derivatives with respect to each parent

Graph

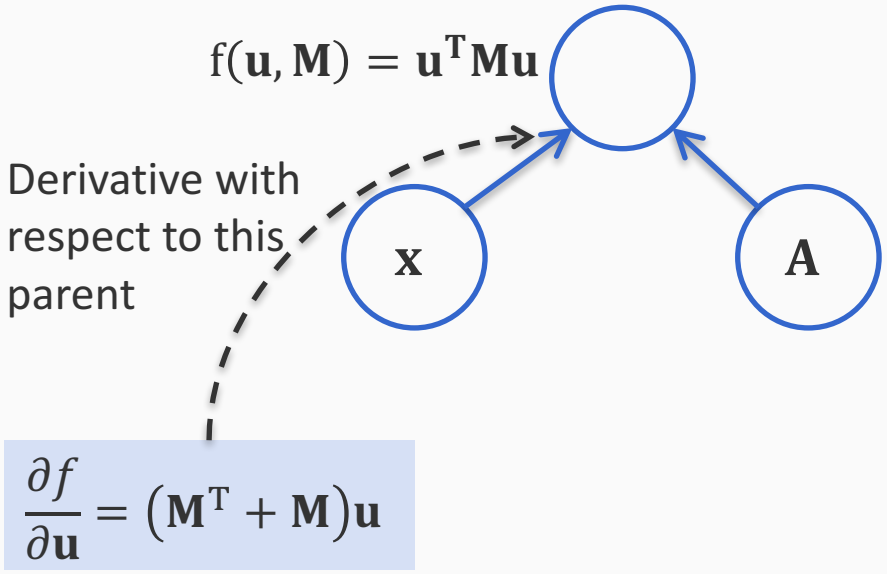


Let's see some functions as graphs

Expression $\mathbf{x}^T\mathbf{A}\mathbf{x}$

Remember: The nodes also know how to compute derivatives with respect to each parent

Graph

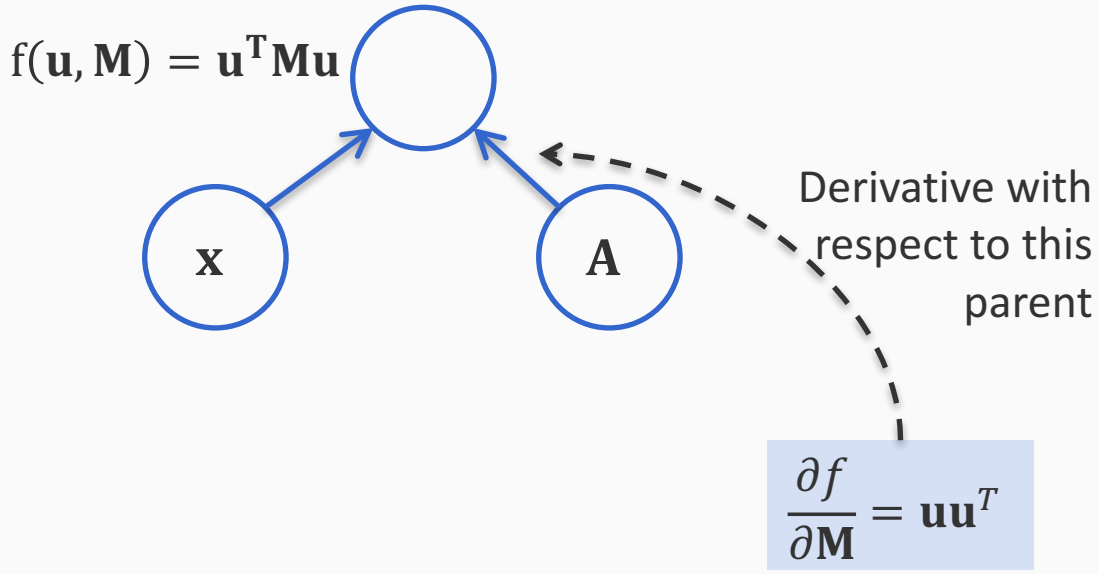


Let's see some functions as graphs

Expression $\mathbf{x}^T\mathbf{A}\mathbf{x}$

Remember: The nodes also know how to compute derivatives with respect to each parent

Graph

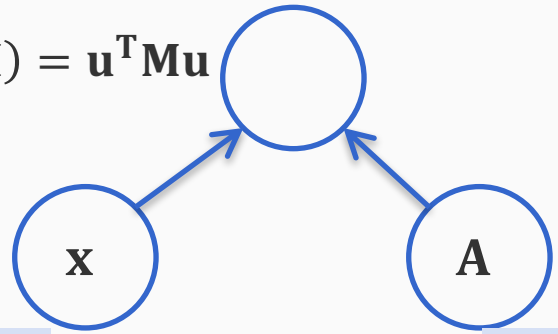


Let's see some functions as graphs

Expression $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}$

$$\frac{\partial f}{\partial \mathbf{x}} = \left(\mathbf{A}^{\mathrm{T}} + \mathbf{A}\right)\mathbf{x} \qquad \frac{\partial f}{\partial \mathbf{A}} = \mathbf{x}\mathbf{x}^{T}$$

Graph



$$\frac{\partial f}{\partial \mathbf{u}} = \left(\mathbf{M}^{\mathrm{T}} + \mathbf{M}\right)\mathbf{u}$$

$$\frac{\partial f}{\partial \mathbf{M}} = \mathbf{u}\mathbf{u}^{T}$$

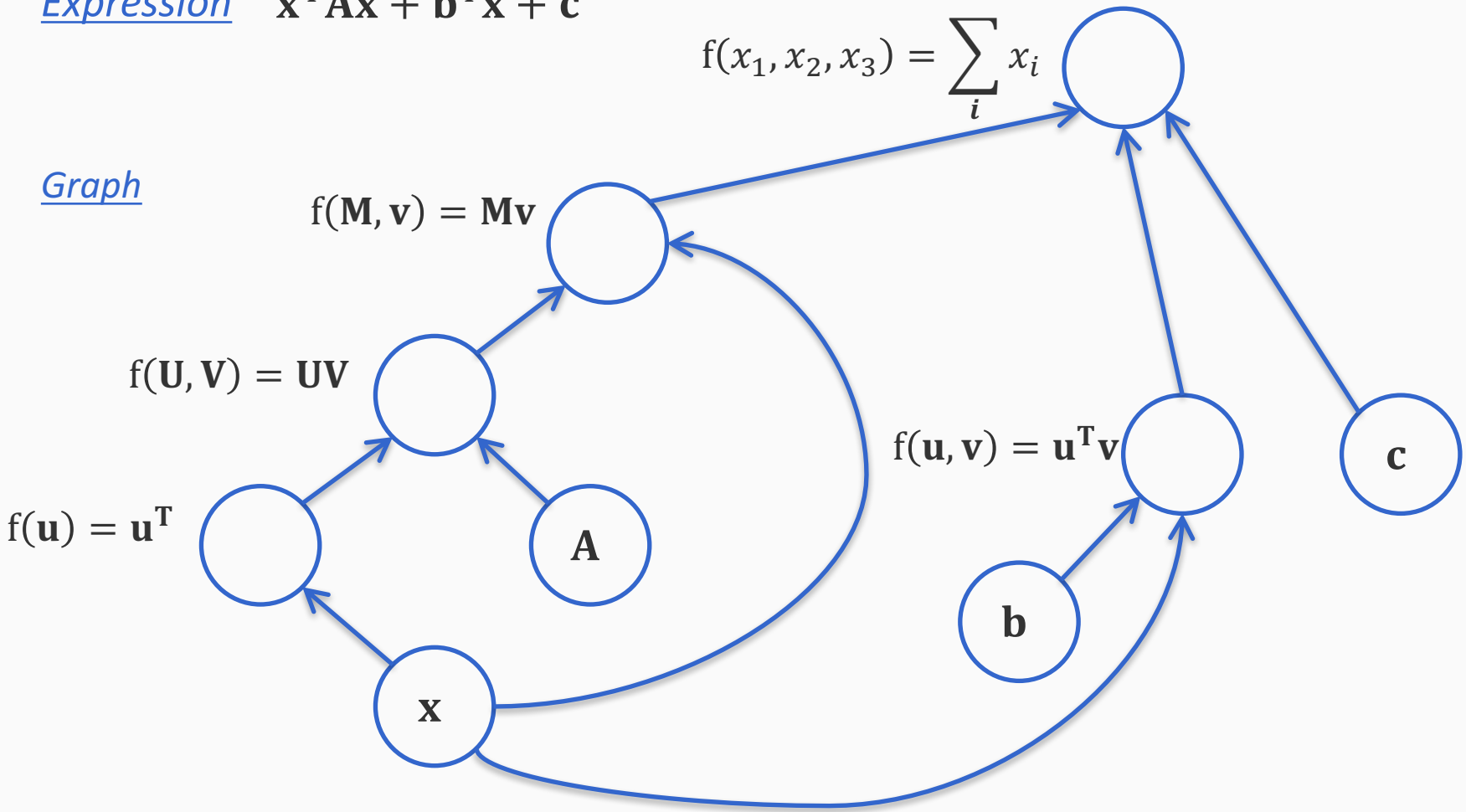
Remember: The nodes also know how to compute derivatives with respect to each parent

Together, we can compute derivatives of any function with respect to all its inputs, for any value of the input

Let's see some functions as graphs

Expression $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b}^{\mathrm{T}}\mathbf{x} + \mathbf{c}$

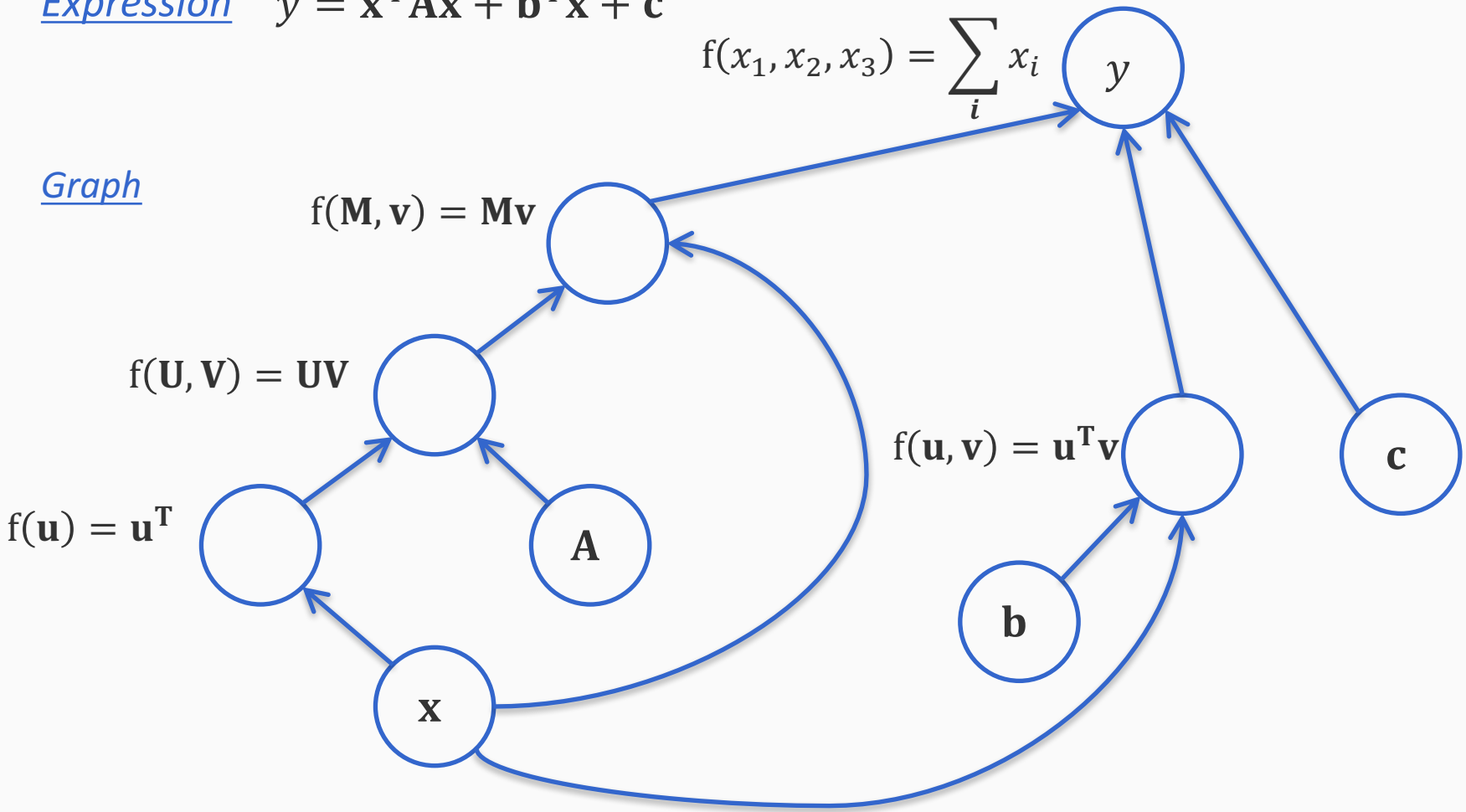
Graph



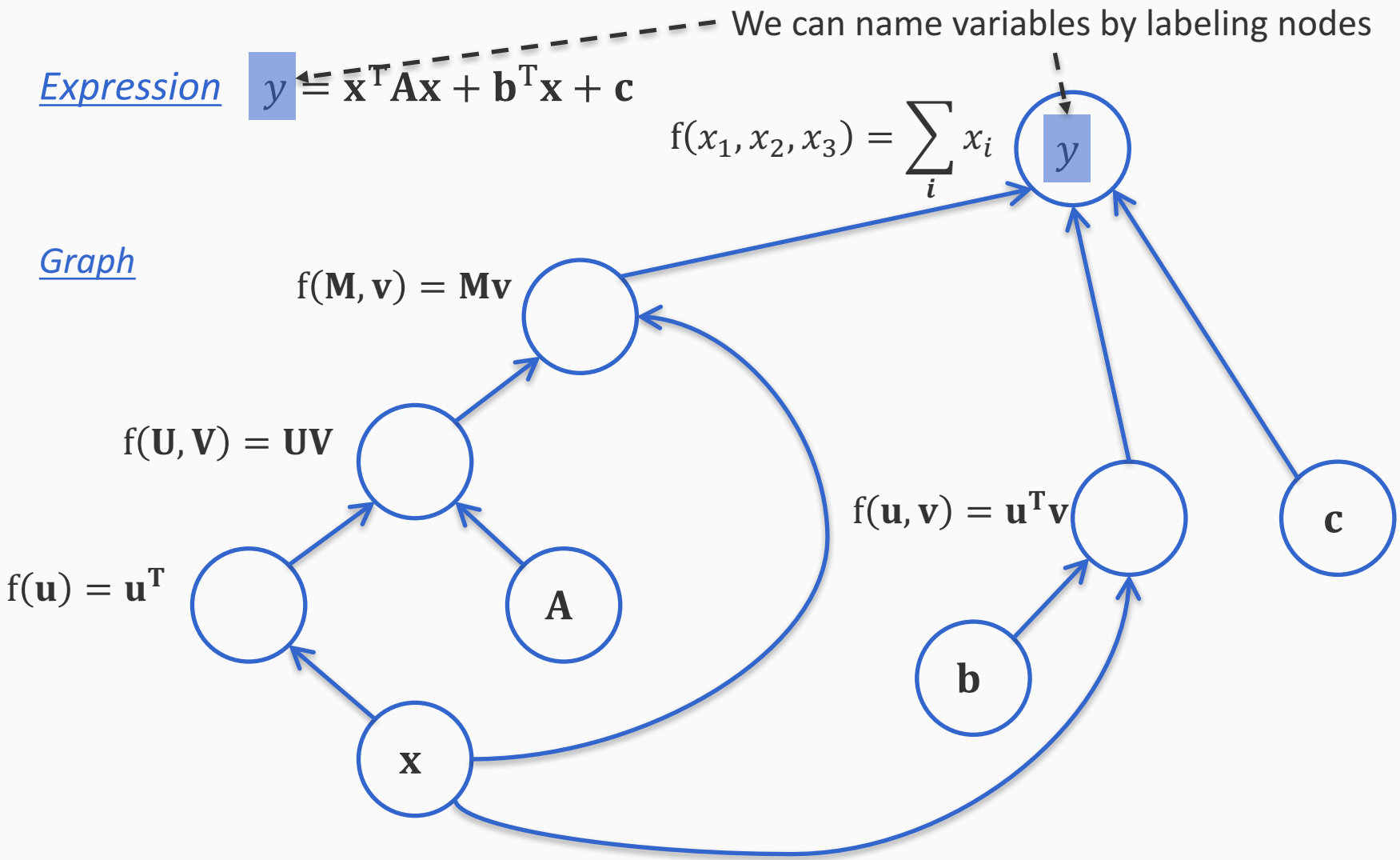
Let's see some functions as graphs

Expression $y = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b}^{\mathrm{T}}\mathbf{x} + \mathbf{c}$

Graph



Let's see some functions as graphs



Why are computation graphs interesting?

1. For starters, we can write neural networks as computation graphs.

2. We can write loss functions as computation graphs.
   Or loss functions within the innermost stochastic gradient descent.

3. They are plug-and-play: We can construct a graph and use it in a program that someone else wrote
   For eg: We can write down a neural network and plug it into a loss function and a minimization function from a library

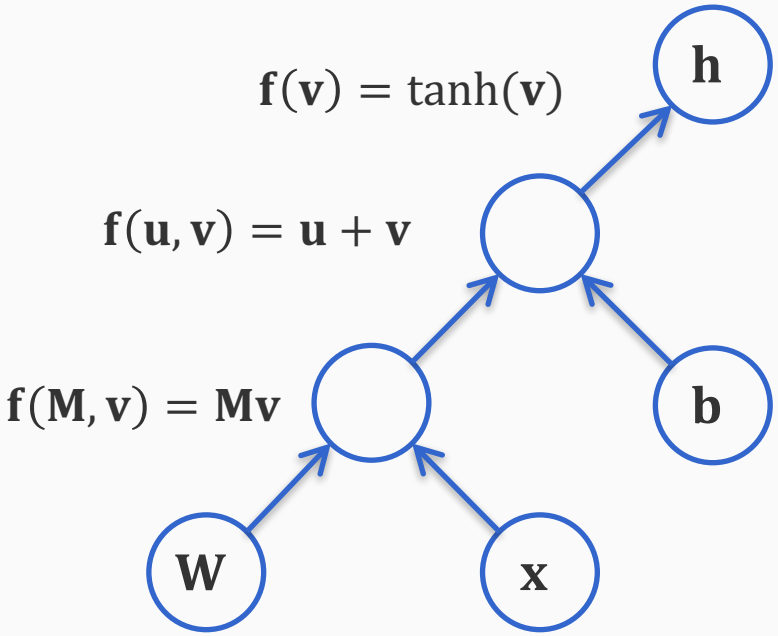
4. They allow efficient gradient computation.

An example two layer neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$

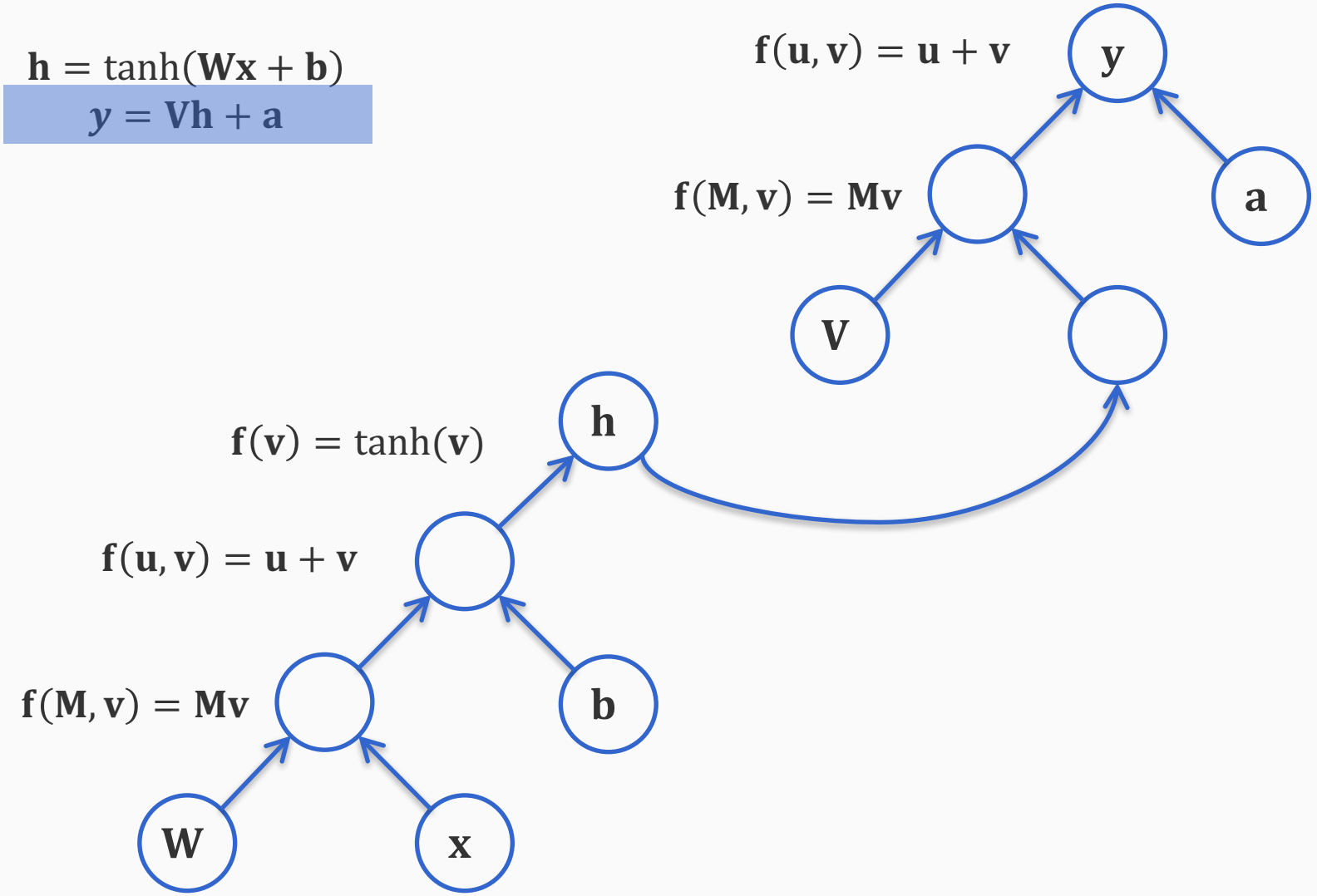
An example two layer neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$



An example two layer neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$



Where are we?

- What is a neural network?
- Computation Graphs
- Algorithms over computation graphs
  - The forward pass
  - The backward pass

Three computational questions

1. Forward propagation
   - Given inputs to the graph, compute the value of the function expressed by the graph
   - Something to think about: Given a node, can we say which nodes are inputs? Which nodes are outputs?

2. Backpropagation
   - After computing the function value for an input, compute the gradient of the function at that input
   - Or equivalently: *How does the output change if I make a small change to the input?*

3. Constructing graphs
   - Need an easy-to-use framework to construct graphs
   - The size of the graph may be input dependent
     - A templating language that creates graphs on the fly
   - Tensorflow, PyTorch are the most popular frameworks today

Forward propagation

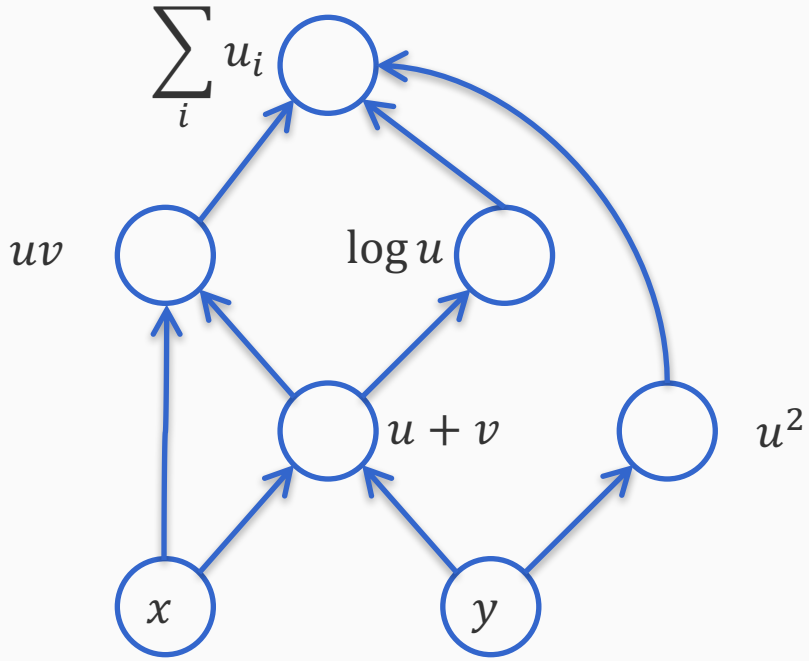
Three computational questions

1. Forward propagation
   - Given inputs to the graph, compute the value of the function expressed by the graph
   - Something to think about: Given a node, can we say which nodes are inputs? Which nodes are outputs?

2. Backpropagation
   - After computing the function value for an input, compute the gradient of the function at that input
   - Or equivalently: *How does the output change if I make a small change to the input?*

3. Constructing graphs
   - Need an easy-to-use framework to construct graphs
   - The size of the graph may be input dependent
     - A templating language that creates graphs on the fly
   - Tensorflow, PyTorch are the most popular frameworks today

Forward pass: An example

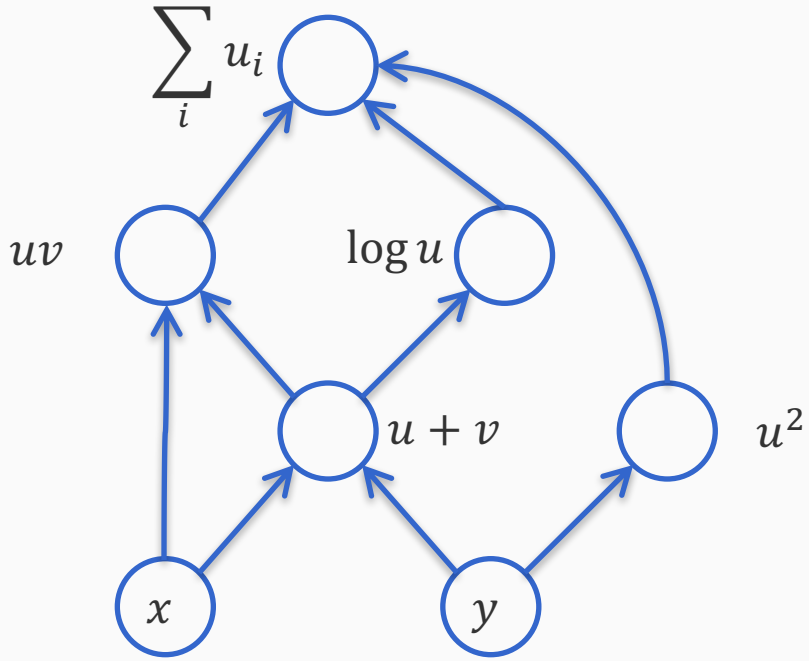


Conventions:

1. Any expression next to a node is the function it computes
2. All the variables in the expression are inputs to the node from left to right.

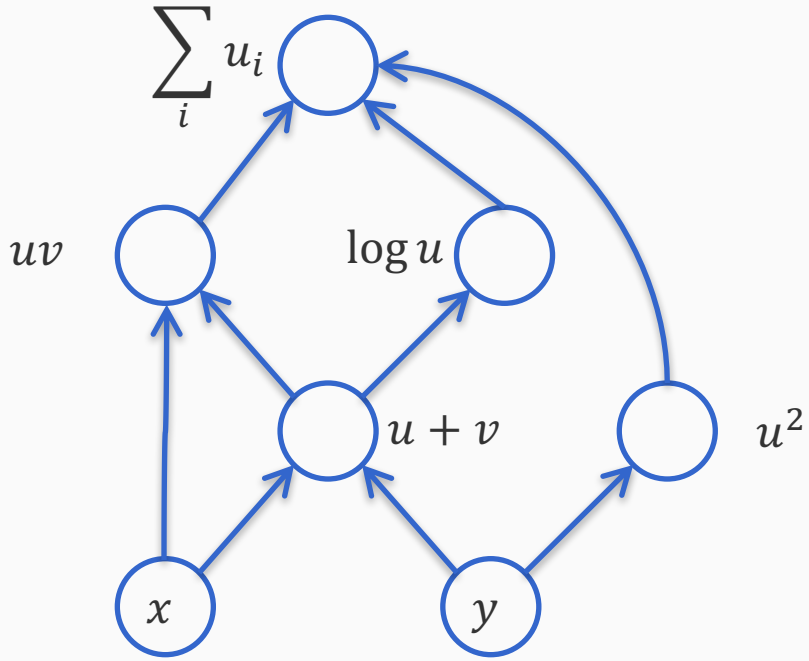
Forward pass

What function does this compute?



Forward pass

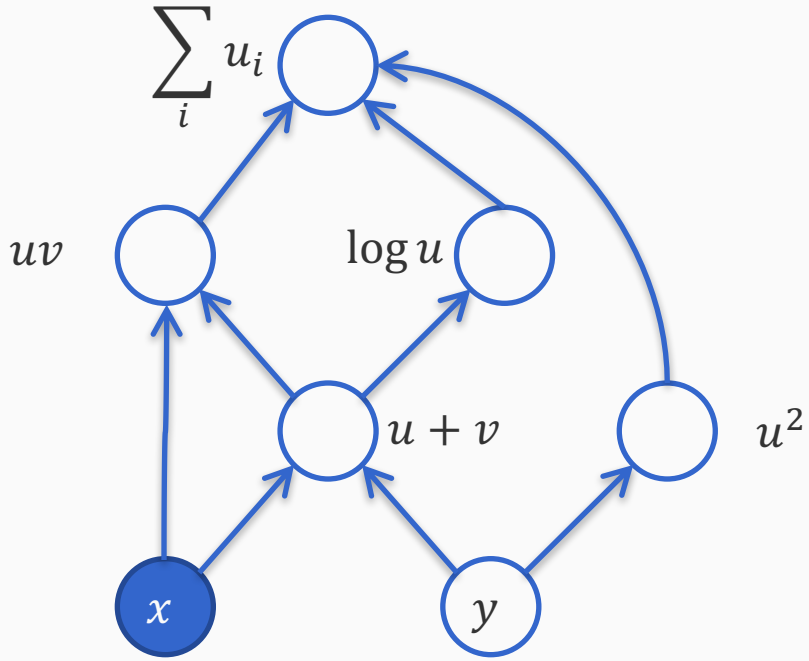
What function does this compute?



Suppose we shade nodes whose values we know (i.e. we have computed).

Forward pass

What function does this compute?



x

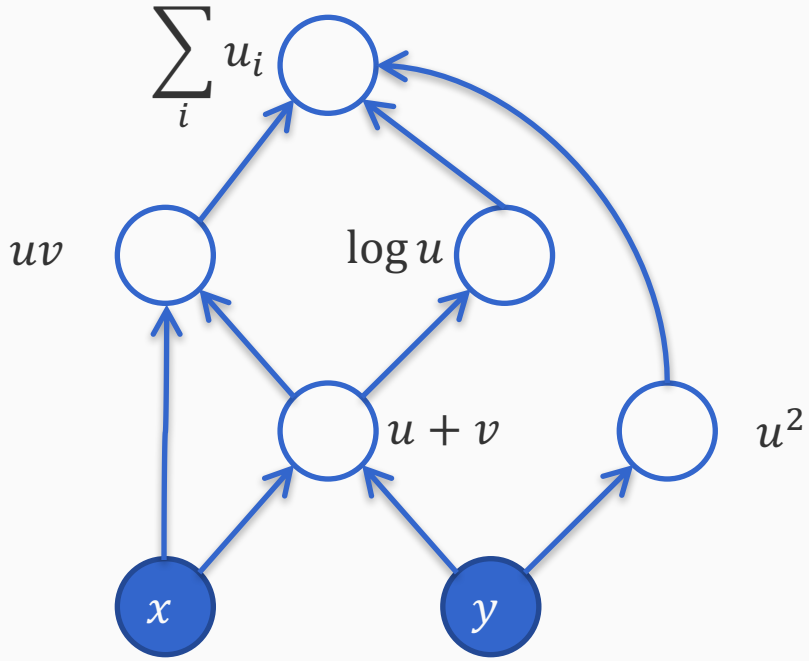
Suppose we shade nodes whose values we know (i.e. we have computed).

Forward pass

What function does this compute?

x

y



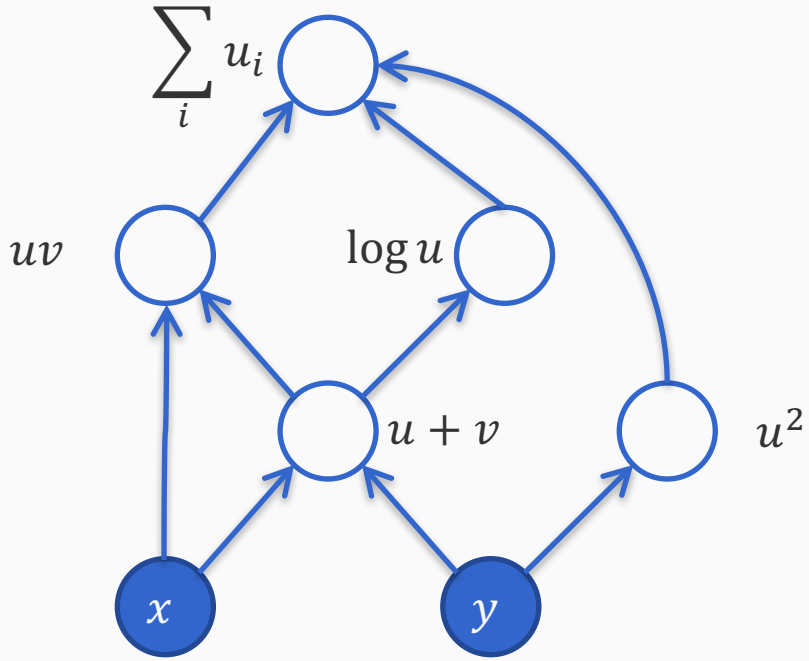
Suppose we shade nodes whose values we know (i.e. we have computed).

Forward pass

What function does this compute?

x

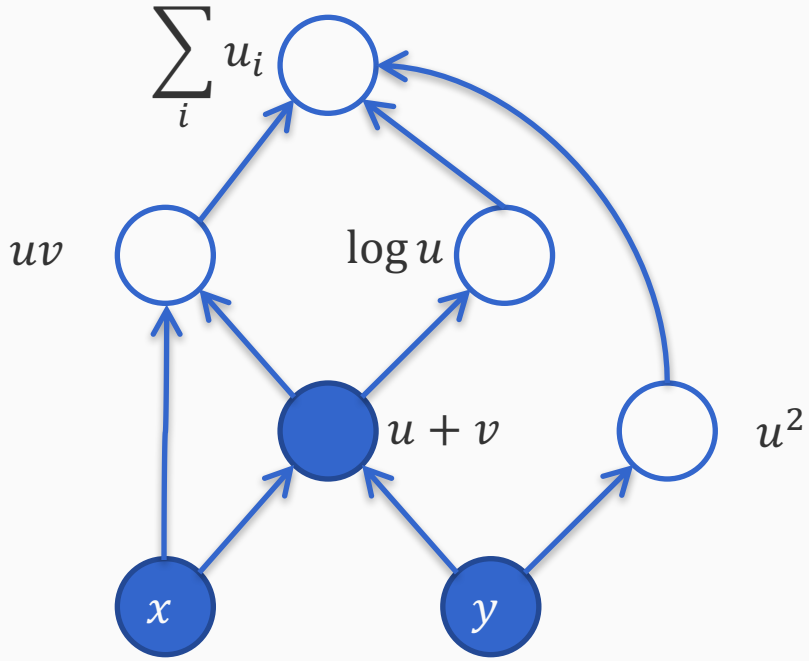
y



Suppose we shade nodes whose values we know (i.e. we have computed).

We can only compute the value of a node if we know the values of all its inputs

Forward pass



What function does this compute?

x

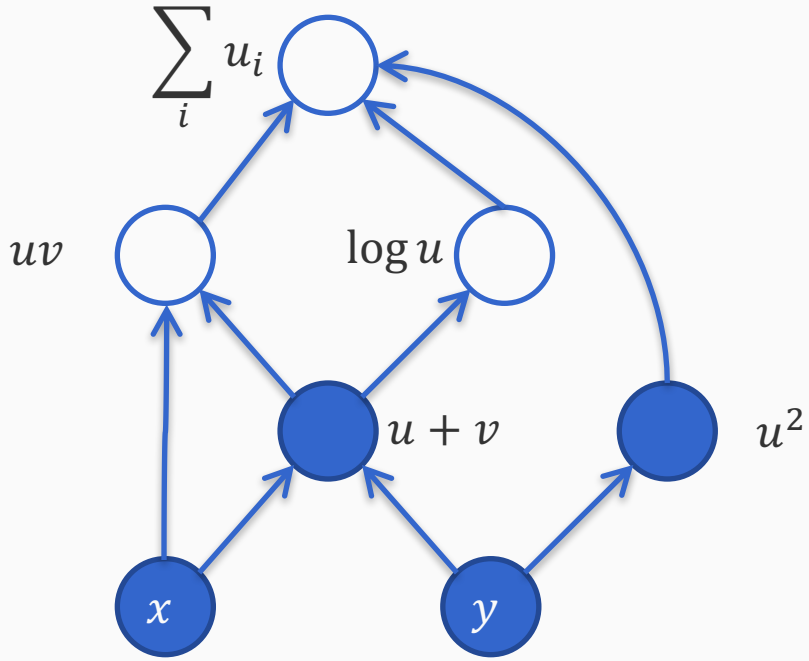
y

$x + y$

Suppose we shade nodes whose values we know (i.e. we have computed).

We can only compute the value of a node if we know the values of all its inputs

Forward pass



What function does this compute?

x

y

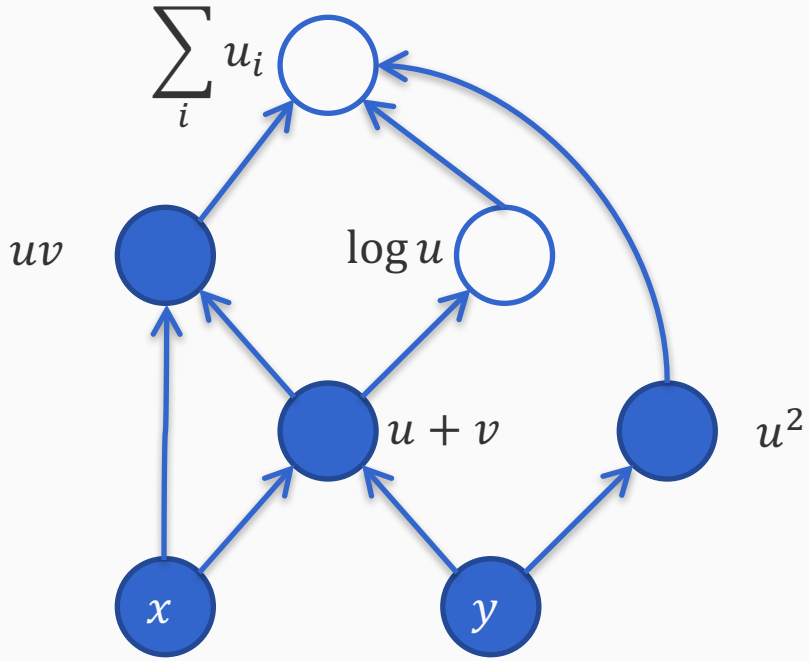
$x + y$

y^2

Suppose we shade nodes whose values we know (i.e. we have computed).

We can only compute the value of a node if we know the values of all its inputs

Forward pass



What function does this compute?

x

y

$x + y$

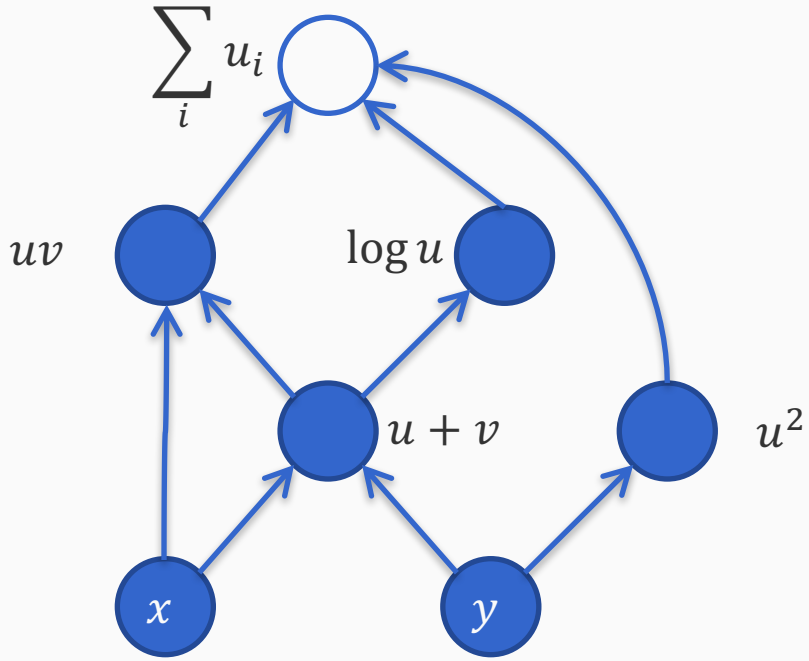
y^2

$x(x + y)$

Suppose we shade nodes whose values we know (i.e. we have computed).

We can only compute the value of a node if we know the values of all its inputs

Forward pass



What function does this compute?

x

y

$x + y$

y^2

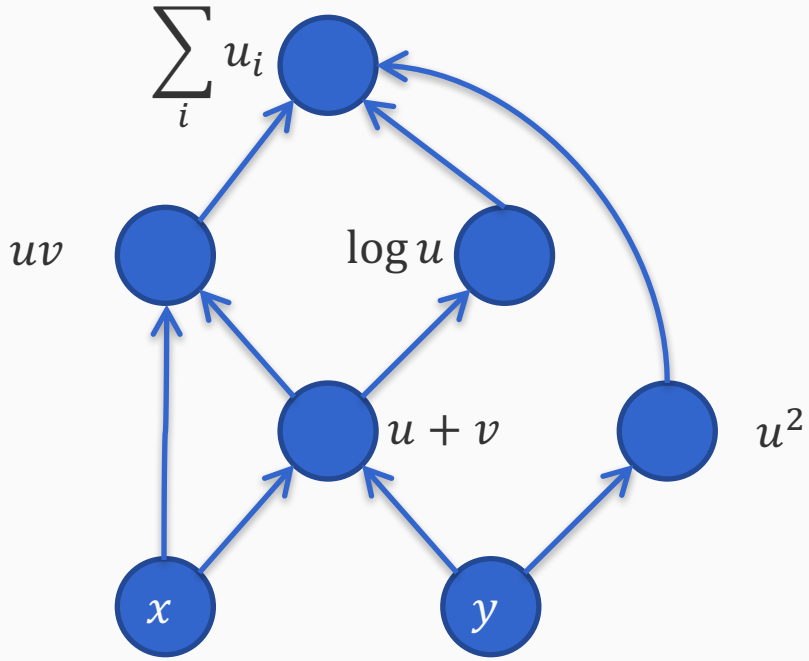
$x(x + y)$

$\log(x + y)$

Suppose we shade nodes whose values we know (i.e. we have computed).

We can only compute the value of a node if we know the values of all its inputs

Forward pass



What function does this compute?

x

y

$x + y$

y^2

$x(x + y)$

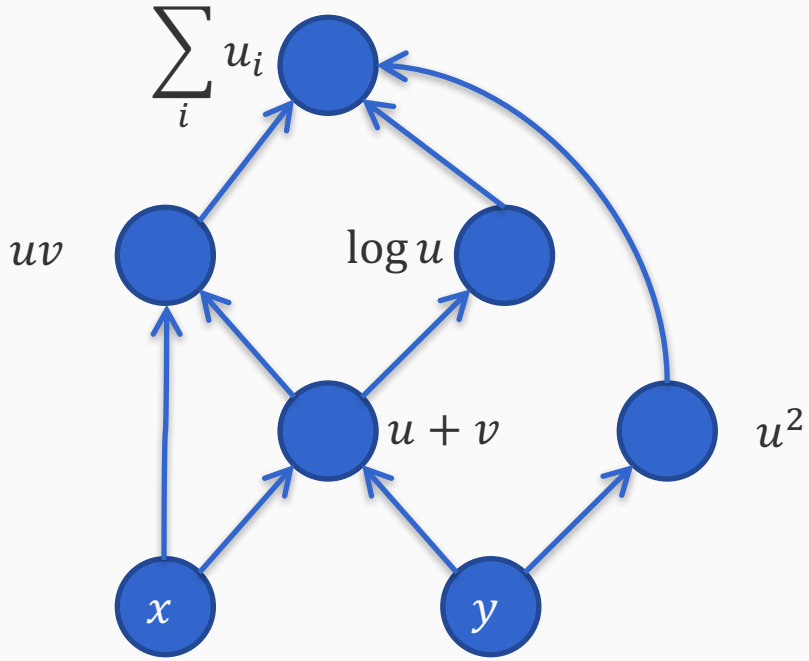
$\log(x + y)$

$\mathrm{x(x + y)} + \log(x + y) + y^2$

Suppose we shade nodes whose values we know (i.e. we have computed).

We can only compute the value of a node if we know the values of all its inputs

Forward pass



What function does this compute?

x

y

$x + y$

y^2

$x(x + y)$

$\log(x + y)$

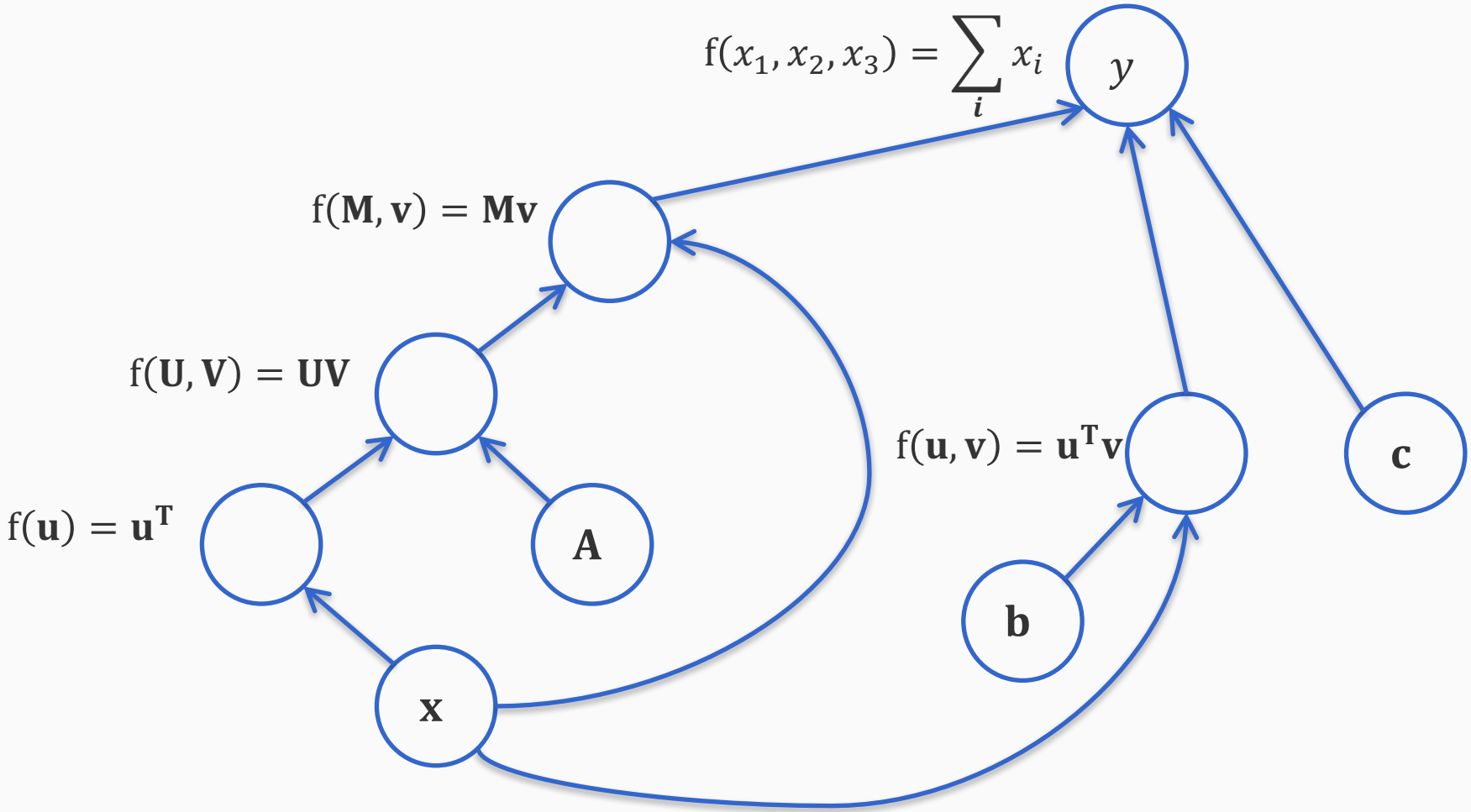
$\mathrm{x(x + y)} + \log(x + y) + y^2$

This gives us the function

Suppose we shade nodes whose values we know (i.e. we have computed).

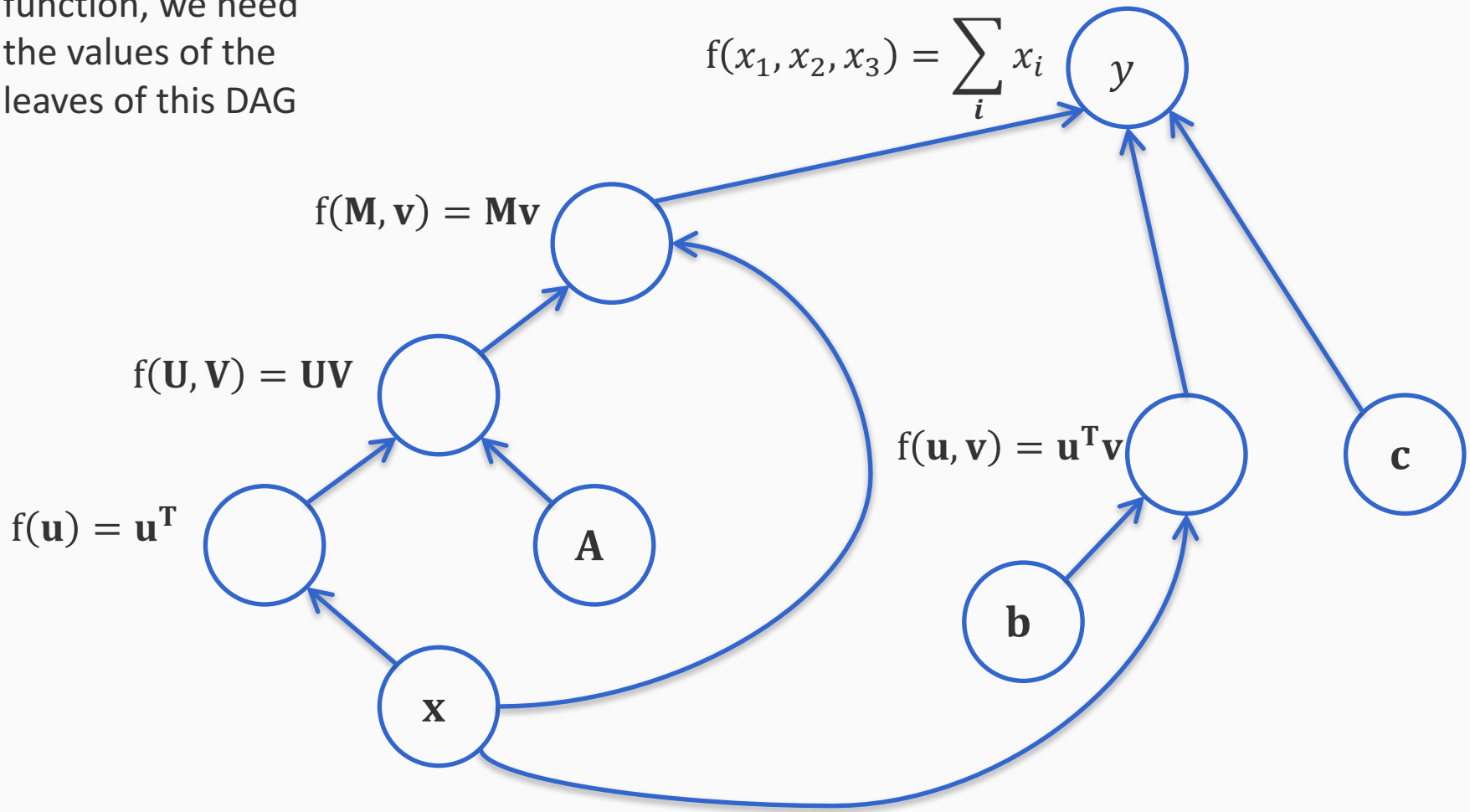
We can only compute the value of a node if we know the values of all its inputs

A second example



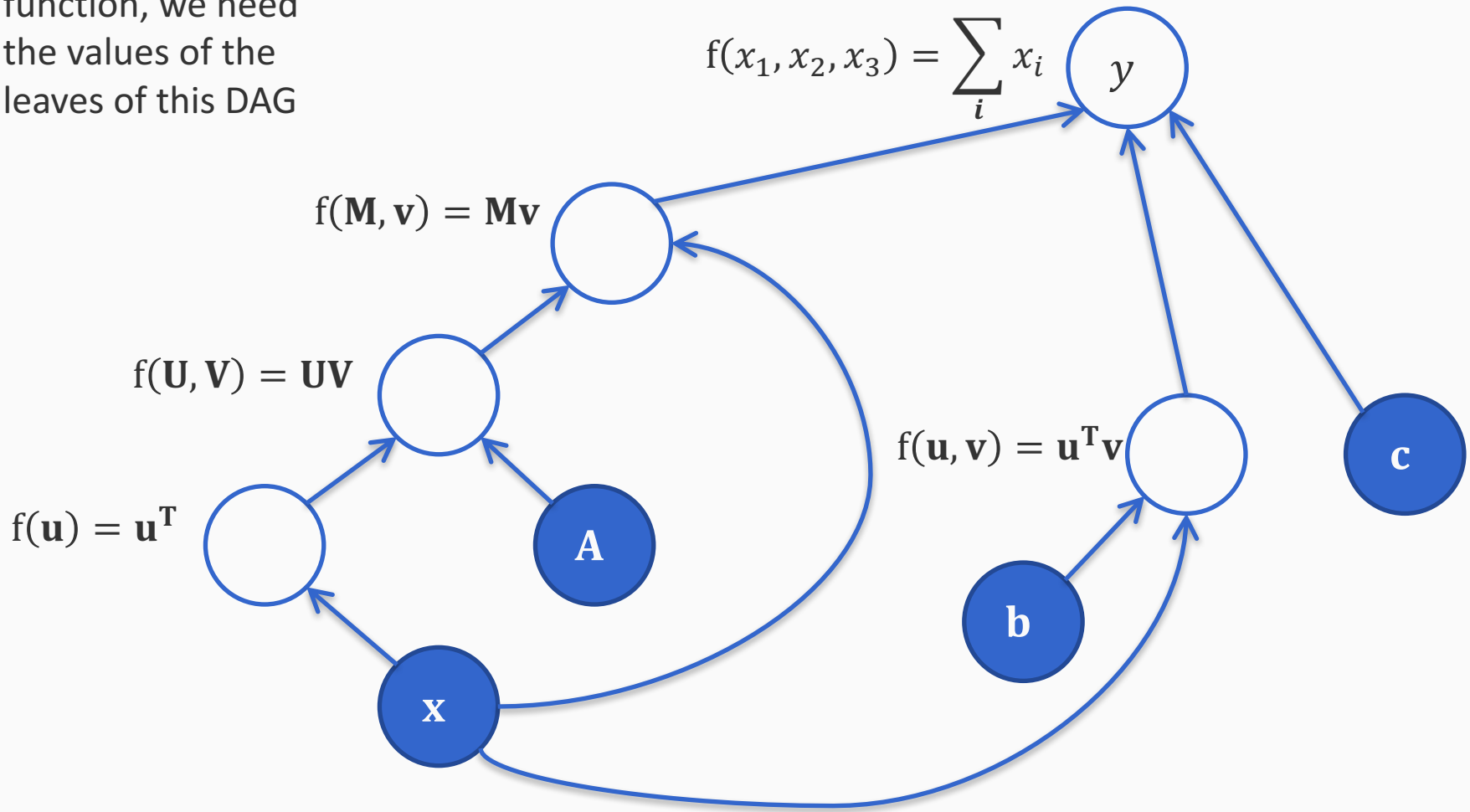
A second example

To compute the function, we need the values of the leaves of this DAG



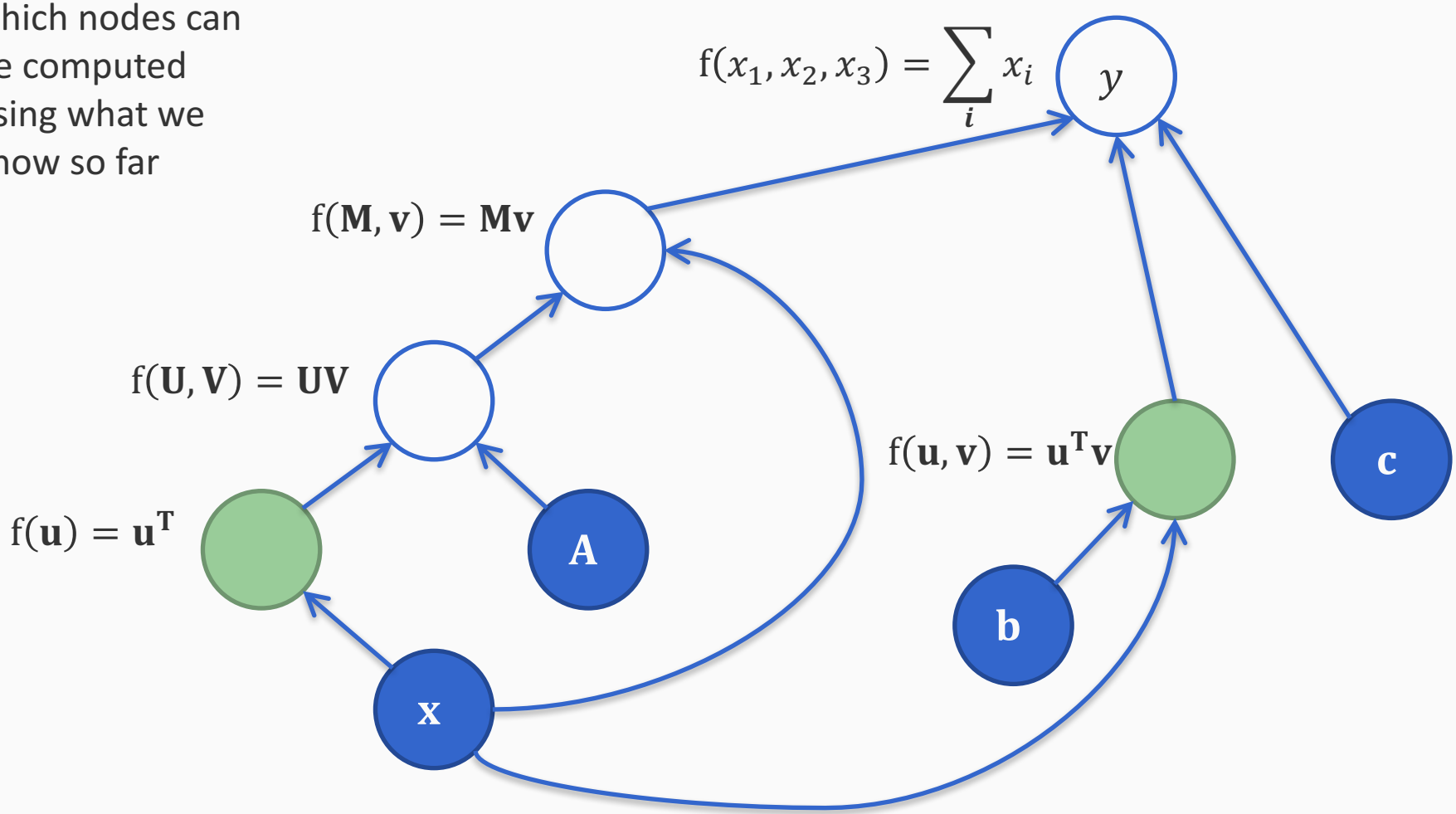
A second example

To compute the function, we need the values of the leaves of this DAG

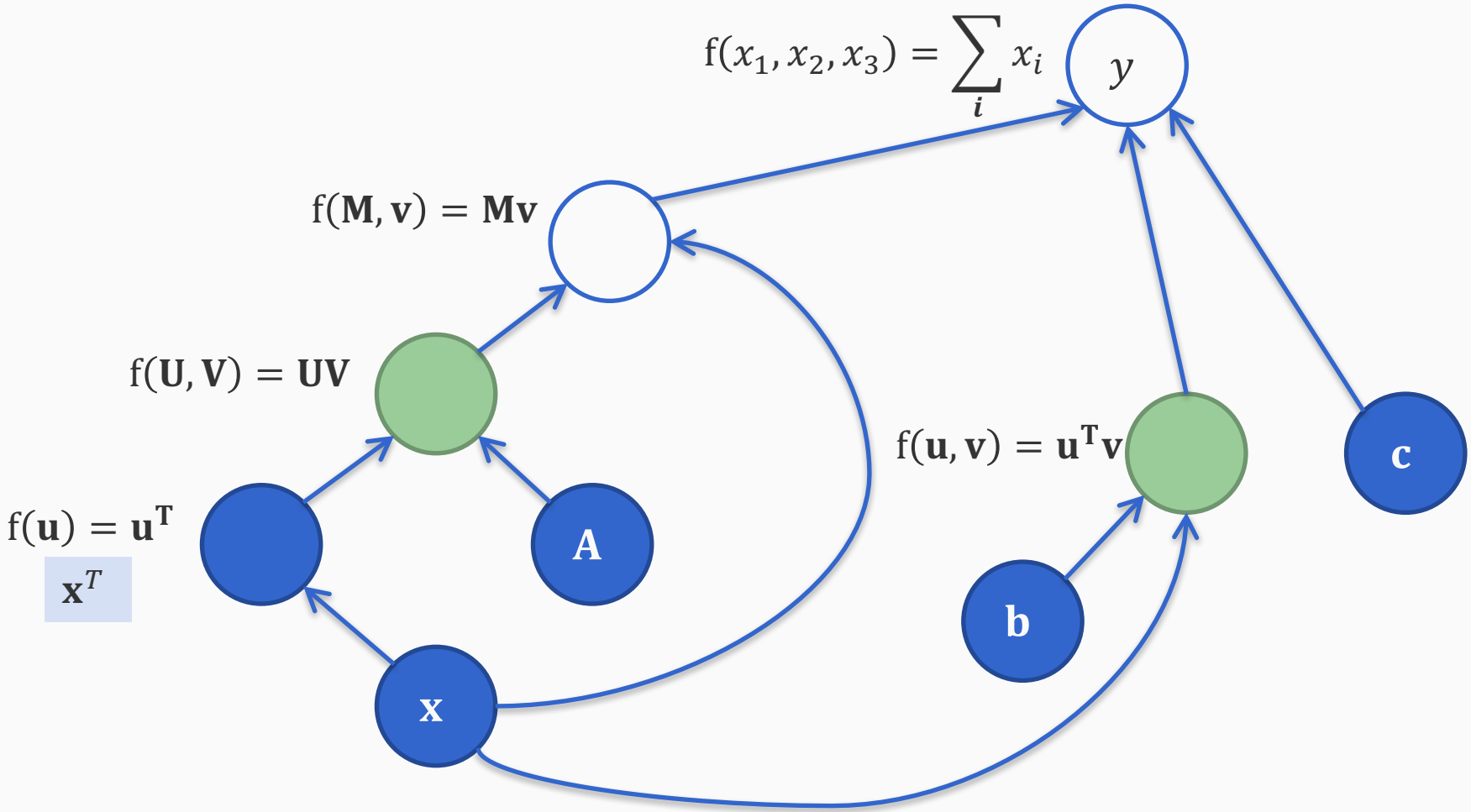


A second example

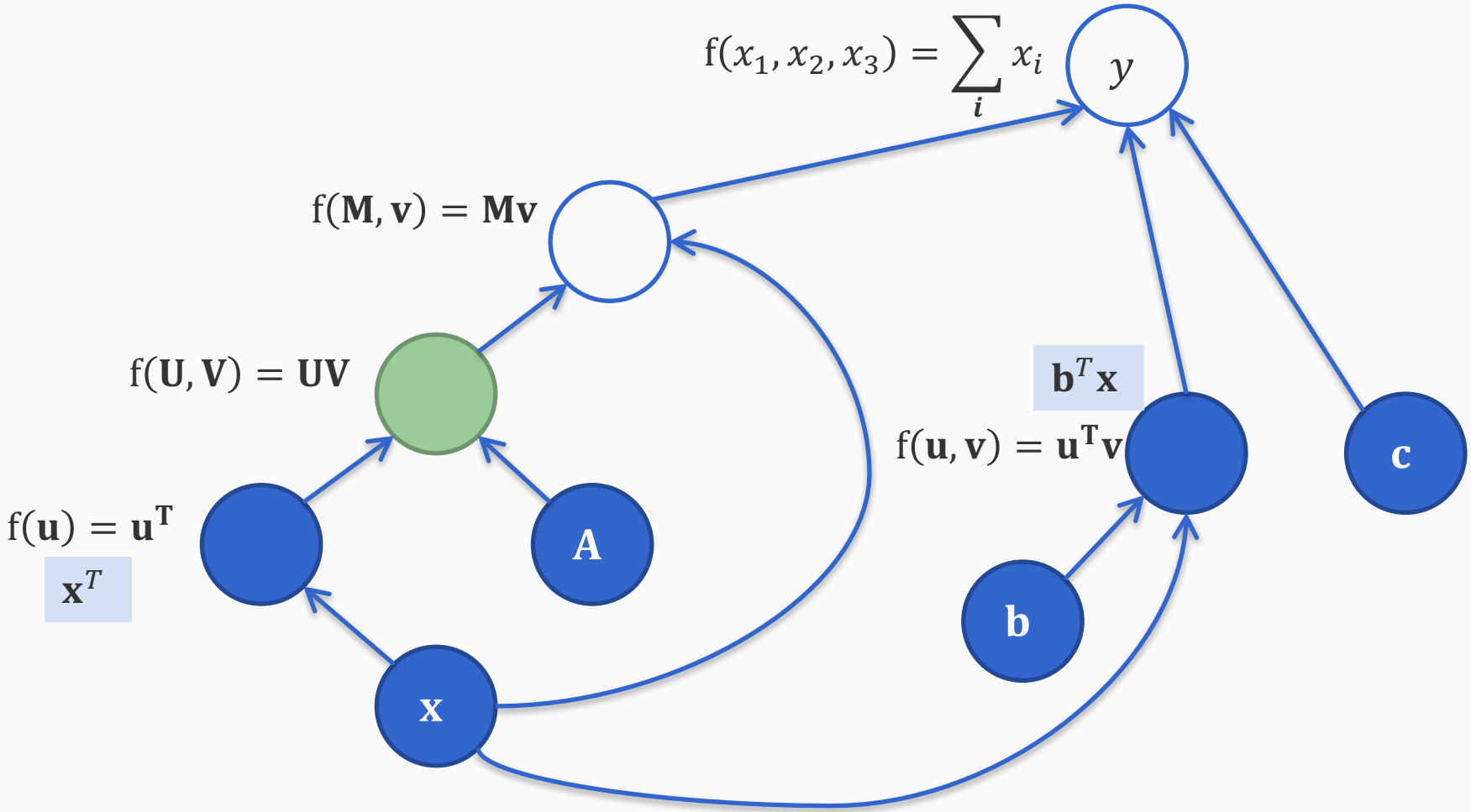
Let's also highlight which nodes can be computed using what we know so far



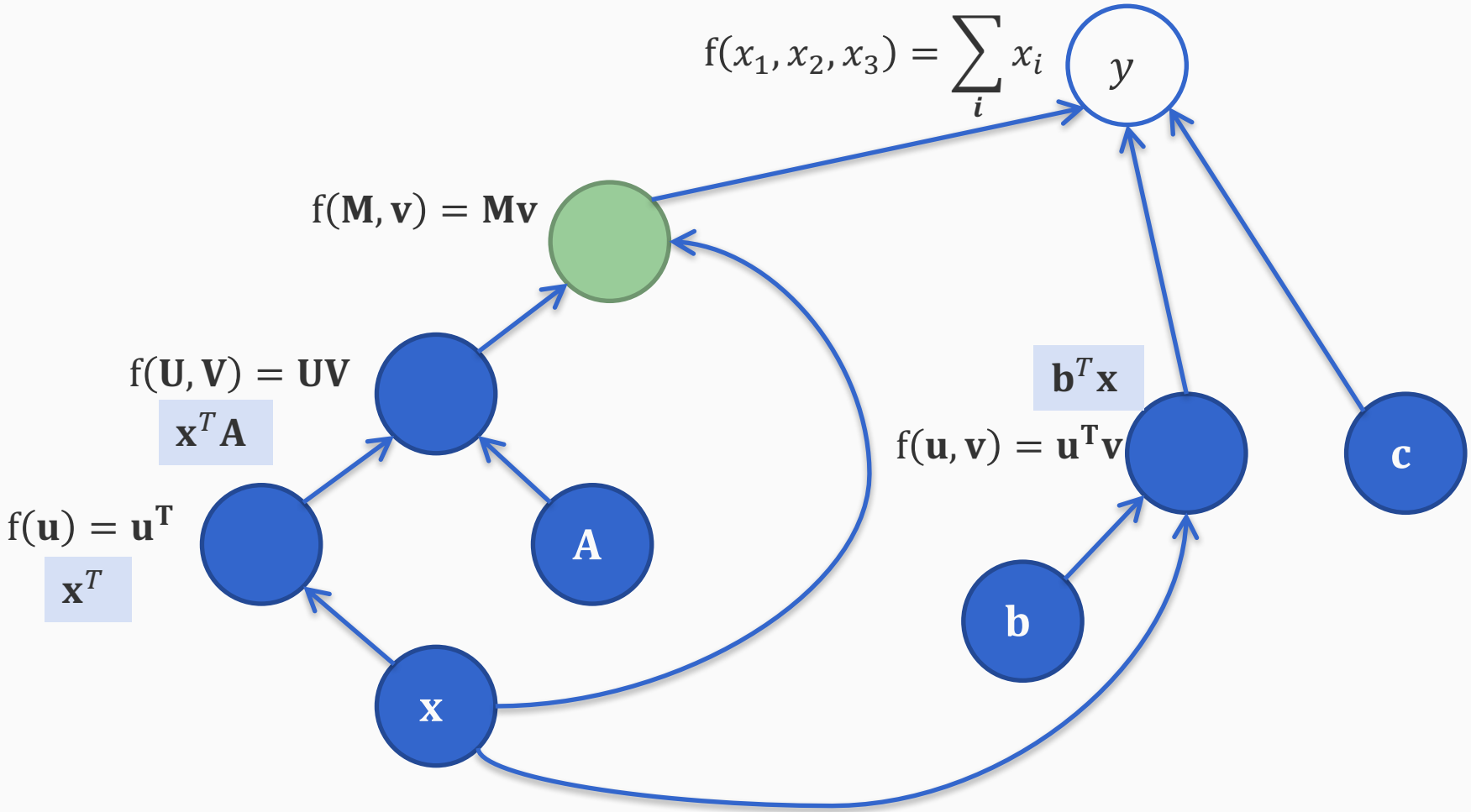
A second example



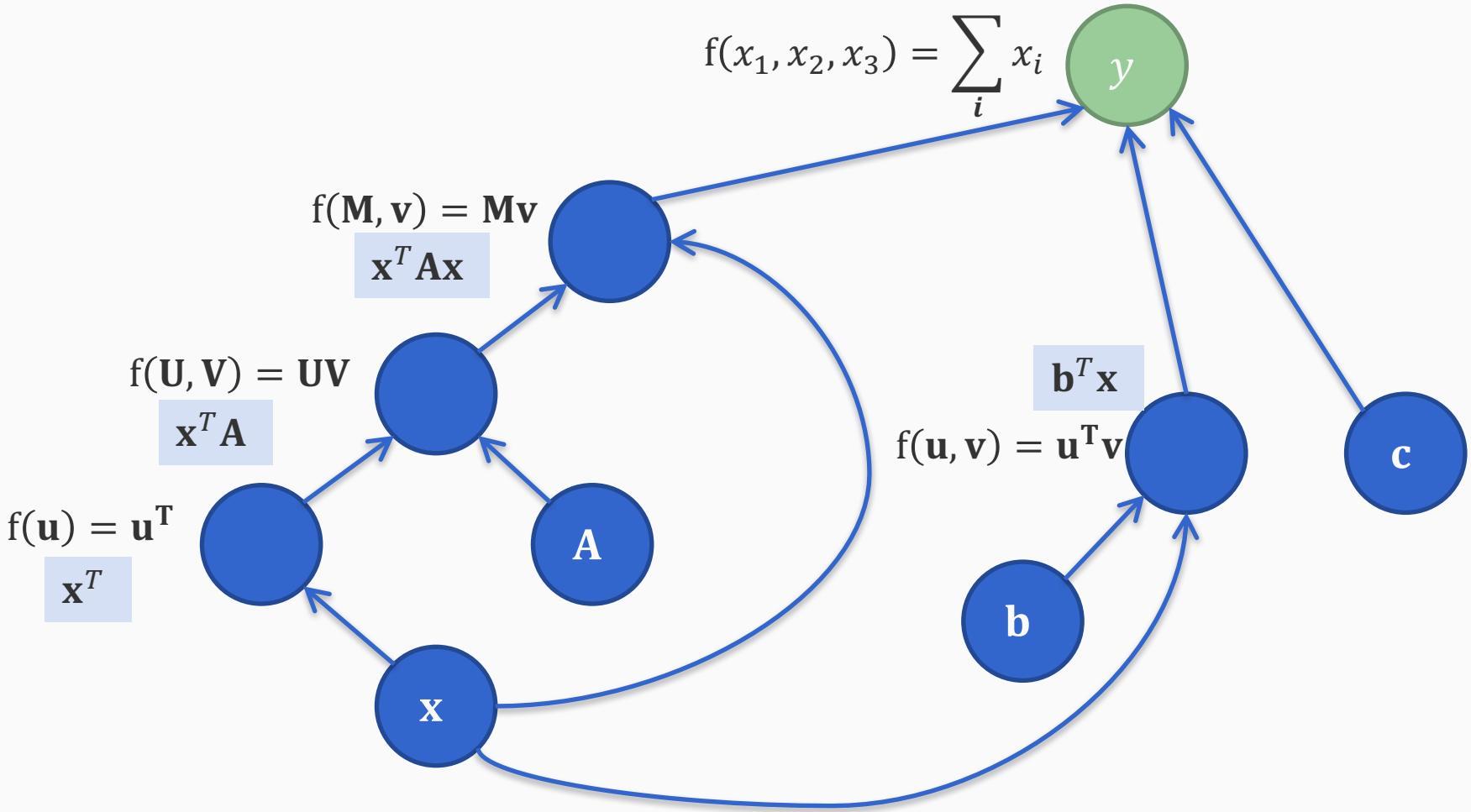
A second example



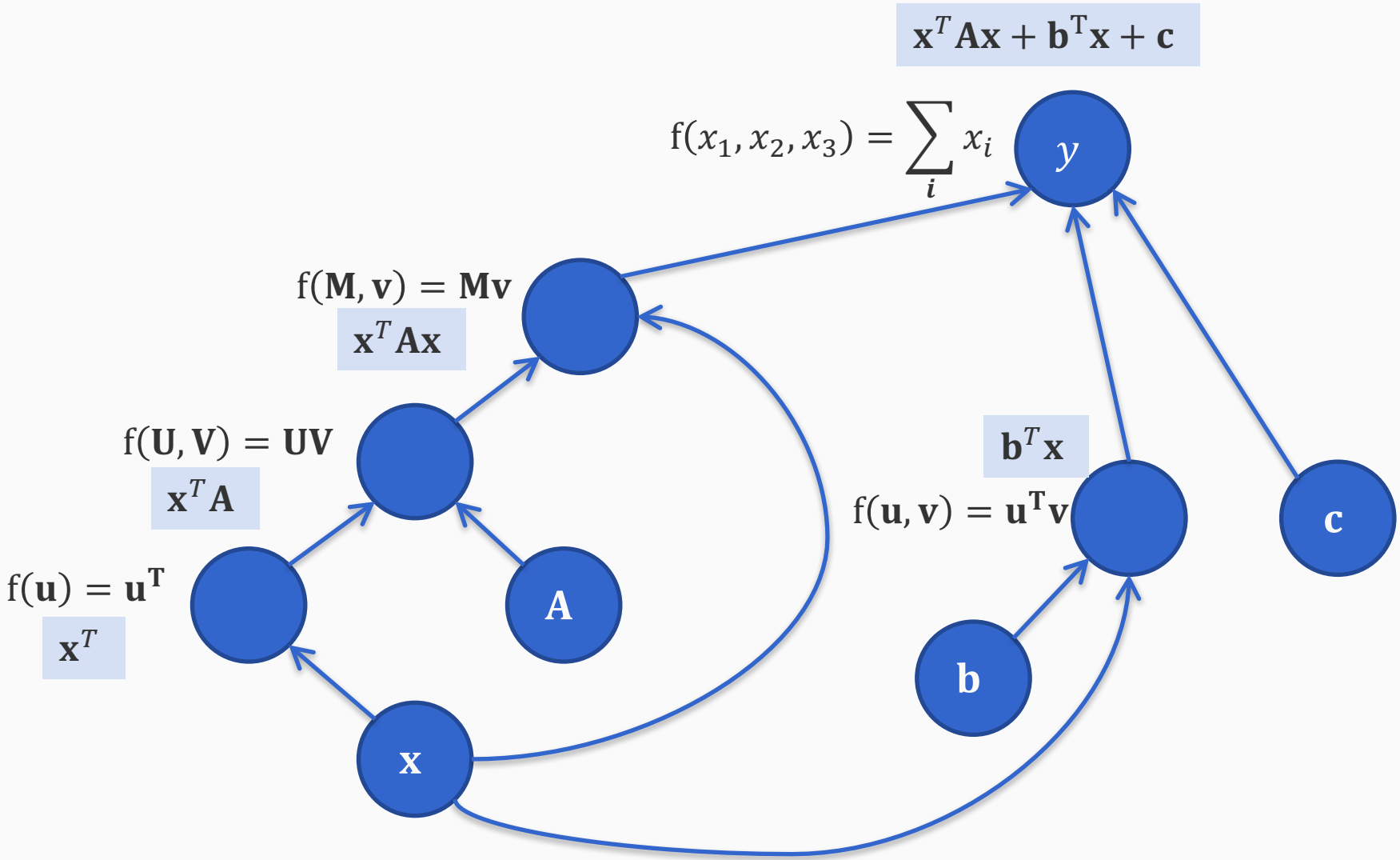
A second example



A second example



A second example



Forward propagation

Given a computation graph G and values of its input nodes:

For each node in the graph, in topological order:

- Compute the value of that node

Forward propagation

Given a computation graph G and values of its input nodes:

For each node in the graph, in **topological order**:

Compute the value of that node

Why topological order: Ensures that children are computed before parents.

Backpropagation with computation graphs

Three computational questions

1. Forward propagation
   - Given inputs to the graph, compute the value of the function expressed by the graph
   - Something to think about: Given a node, can we say which nodes are inputs? Which nodes are outputs?

2. Backpropagation
   - After computing the function value for an input, compute the gradient of the function at that input
   - Or equivalently: *How does the output change if I make a small change to the input?*

3. Constructing graphs
   - Need an easy-to-use framework to construct graphs
   - The size of the graph may be input dependent
     - A templating language that creates graphs on the fly
   - Tensorflow, PyTorch are the most popular frameworks today

Calculus refresher: The chain rule

Suppose we have two functions f and g

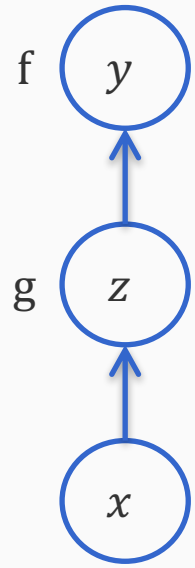
We wish to compute the gradient of $\mathrm{y} = f\big(g(x)\big)$.

We know that $\frac{dy}{dx} = f'\big(g(x)\big) \cdot g'(x)$

Or equivalently: if $z = g(x)$ and $y = f(z)$, then

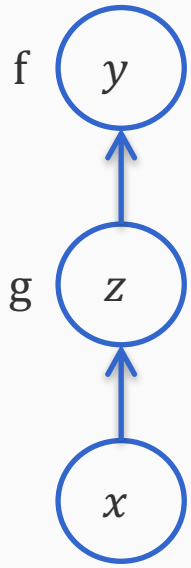
$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dx}$$

Or equivalently: In terms of computation graphs



The forward pass gives us z and y

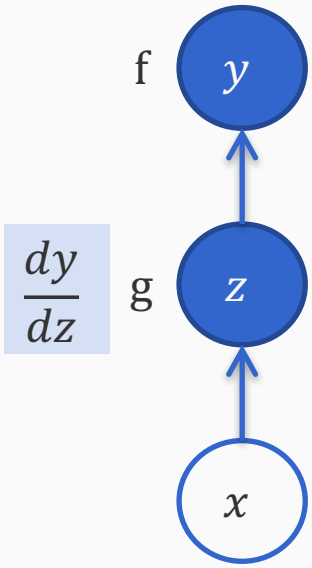
Or equivalently: In terms of computation graphs



The forward pass gives us z and y

Remember that each node knows not only how to compute its value given inputs, but also how to compute gradients

Or equivalently: In terms of computation graphs

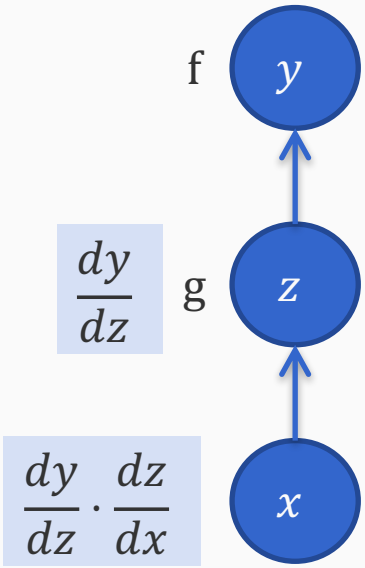


The forward pass gives us z and y

Remember that each node knows not only how to compute its value given inputs, but also how to compute gradients

Start from the root of the graph and work backwards.

Or equivalently: In terms of computation graphs



The forward pass gives us z and y

Remember that each node knows not only how to compute its value given inputs, but also how to compute gradients

Start from the root of the graph and work backwards.

When traversing an edge backwards to a new node:

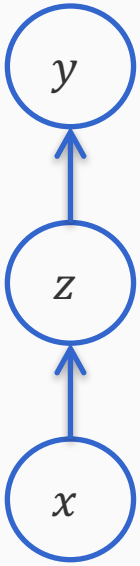
the gradient of the root with respect to that node is the product of the gradient at the parent with the derivative along that edge

A concrete example

$$y = \frac{1}{x^2}$$

$f(u) = \frac{1}{u}$

$\mathrm{g(u)} = \mathrm{u}^2$

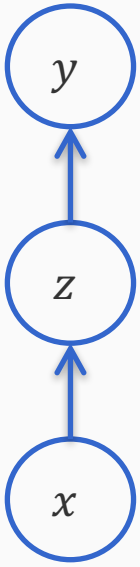


A concrete example

$$y = \frac{1}{x^2}$$

$$\frac{df}{du} = -\frac{1}{u^2} \qquad f(u) = \frac{1}{u}$$

$$\frac{dg}{du} = 2u \qquad g(u) = u^2$$



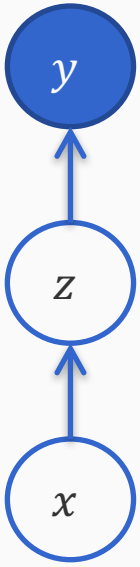
Let's also explicitly write down the derivatives.

A concrete example

$$y = \frac{1}{x^2}$$

$$\frac{df}{du} = -\frac{1}{u^2} \qquad f(u) = \frac{1}{u}$$

$$\frac{dg}{du} = 2u \qquad \mathrm{g(u)} = \mathrm{u}^2$$



$$\frac{dy}{dy} = 1$$

Now, we can proceed backwards from the output

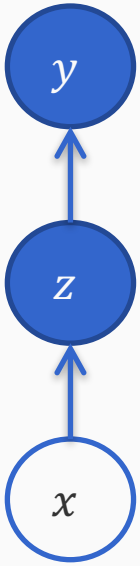
At each step, we compute the gradient of the function represented by the graph with respect to the node that we are at.

A concrete example

$$y = \frac{1}{x^2}$$

$$\frac{df}{du} = -\frac{1}{u^2} \qquad f(u) = \frac{1}{u}$$

$$\frac{dg}{du} = 2u \qquad \mathrm{g(u)} = \mathrm{u}^2$$



$$\frac{dy}{dy} = 1$$

$$\frac{dy}{dz} = \frac{dy}{dy} \cdot \left(\frac{df}{du}\right)_{u=z} = 1 \cdot \left(-\frac{1}{z^2}\right) = -\frac{1}{z^2}$$

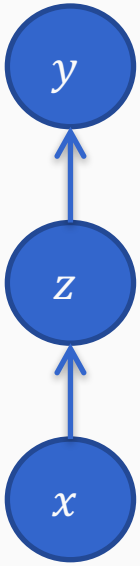
Product of the gradient so far and the derivative computed at this step

A concrete example

$$y = \frac{1}{x^2}$$

$\frac{df}{du} = -\frac{1}{u^2}$ $\quad f(u) = \frac{1}{u}$ $\quad y$ $\quad \frac{dy}{dy} = 1$

$\frac{dg}{du} = 2u$ $\quad \mathrm{g(u)} = \mathrm{u}^2$ $\quad z$ $\quad \frac{dy}{dz} = -\frac{1}{z^2}$



x

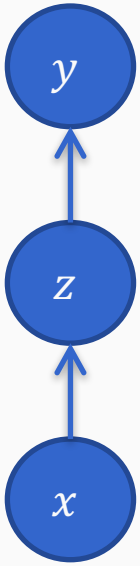
$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \left(\frac{dg}{du}\right)_{u=x} = -\frac{1}{z^2} \cdot 2x = -\frac{2\mathrm{x}}{\mathrm{z}^2}$$

A concrete example

$$y = \frac{1}{x^2}$$

$\frac{df}{du} = -\frac{1}{u^2}$ $\quad f(u) = \frac{1}{u}$ $\quad y$ $\quad \frac{dy}{dy} = 1$

$\frac{dg}{du} = 2u$ $\quad \mathrm{g(u)} = \mathrm{u}^2$ $\quad z$ $\quad \frac{dy}{dz} = -\frac{1}{z^2}$

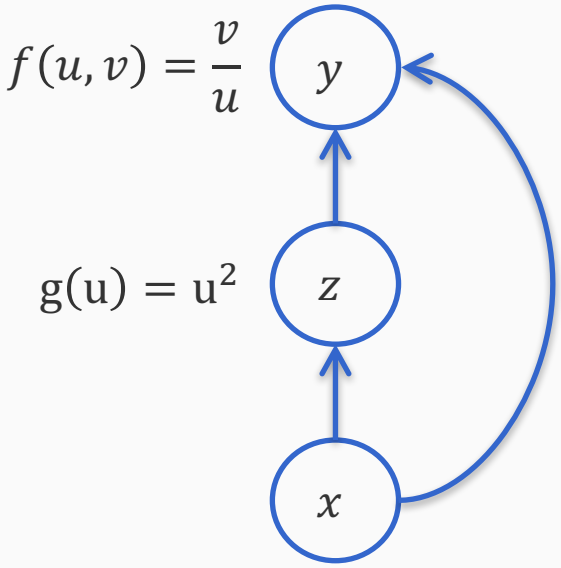


$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \left(\frac{dg}{du}\right)_{u=x} = -\frac{1}{z^2} \cdot 2x = -\frac{2\mathrm{x}}{\mathrm{z}^2}$$

We can simplify this to get $-\frac{2}{x^3}$

A concrete example with multiple outgoing edges

$$y = \frac{1}{x}$$



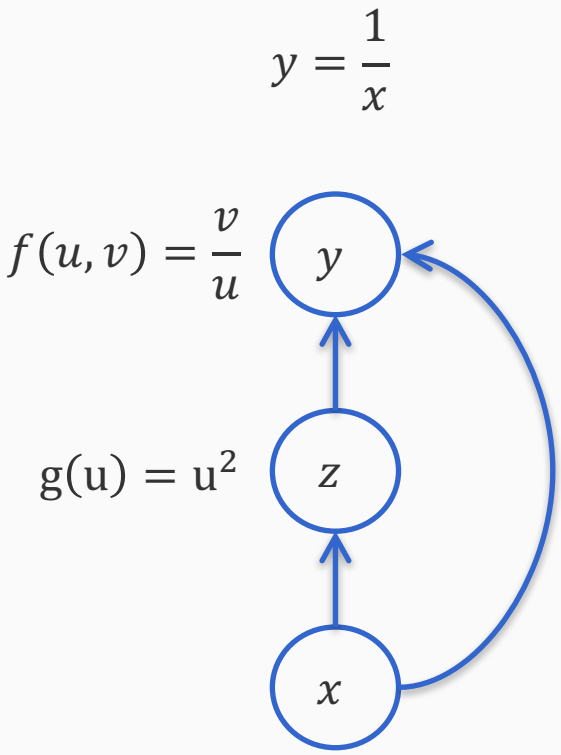
A concrete example with multiple outgoing edges

$$y = \frac{1}{x}$$

$$\frac{df}{du} = -\frac{v}{u^2}$$

$$\frac{df}{dv} = \frac{1}{u}$$

$$\frac{dg}{du} = 2u$$



Let's also explicitly write down the derivatives. Note that f has two derivatives because it has two inputs.

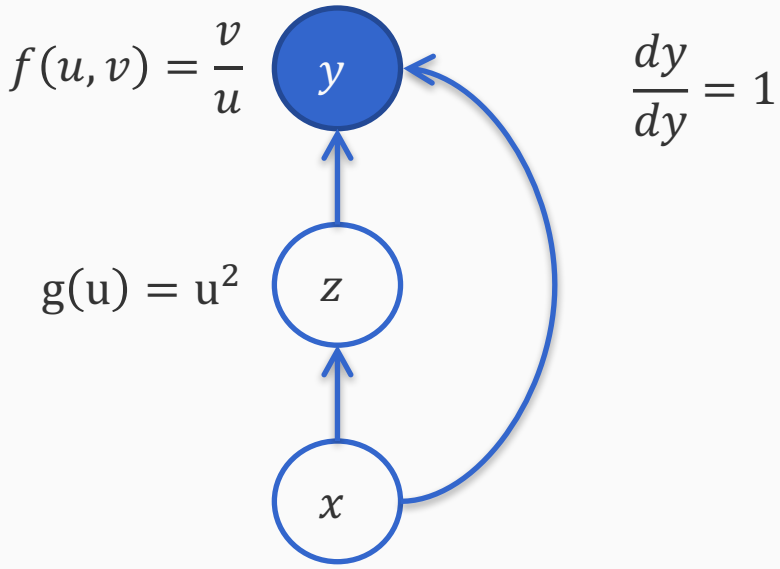
A concrete example with multiple outgoing edges

$$y = \frac{1}{x}$$

$$\frac{df}{du} = -\frac{v}{u^2}$$

$$\frac{df}{dv} = \frac{1}{u}$$

$$\frac{dg}{du} = 2u$$



$$\frac{dy}{dy} = 1$$

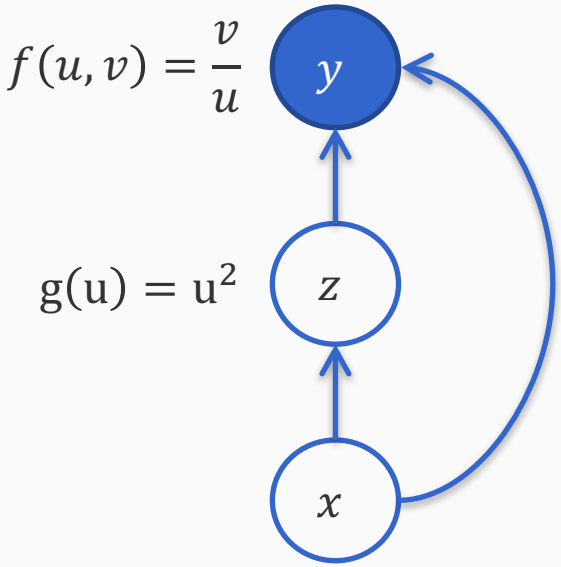
A concrete example with multiple outgoing edges

$$y = \frac{1}{x}$$

$$\frac{df}{du} = -\frac{v}{u^2}$$

$$\frac{df}{dv} = \frac{1}{u}$$

$$\frac{dg}{du} = 2u$$



$$\frac{dy}{dy} = 1$$

At this point, we can compute the gradient of y with respect to z by following the edge from y to z.

But we can not follow the edge from y to x because all of x's descendants are not marked as done.

A concrete example with multiple outgoing edges

$$y = \frac{1}{x}$$

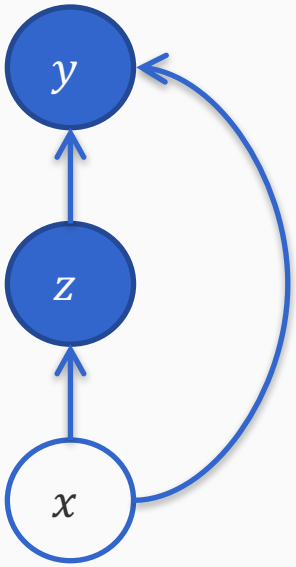
$$\frac{df}{du} = -\frac{v}{u^2}$$

$$\frac{df}{dv} = \frac{1}{u}$$

$$f(u, v) = \frac{v}{u}$$

$$\frac{dg}{du} = 2u$$

$$\mathrm{g(u)} = \mathrm{u}^2$$



$$\frac{dy}{dy} = 1$$

$$\frac{dy}{dz} = \frac{dy}{dy} \cdot \left(\frac{df}{du}\right)_{u=z} = 1 \cdot \left(-\frac{x}{z^2}\right) = -\frac{x}{z^2}$$

Product of the gradient so far and the derivative computed at this step

A concrete example with multiple outgoing edges

$$y = \frac{1}{x}$$

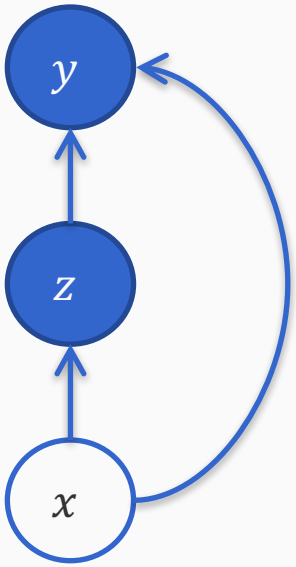
$$\frac{df}{du} = -\frac{v}{u^2}$$

$$\frac{df}{dv} = \frac{1}{u}$$

$$f(u,v) = \frac{v}{u}$$

$$\frac{dg}{du} = 2u$$

$$\mathrm{g(u)} = \mathrm{u}^2$$



$$\frac{dy}{dy} = 1$$

$$\frac{dy}{dz} = \frac{dy}{dy} \cdot \left(\frac{df}{du}\right)_{u=z} = 1 \cdot \left(-\frac{x}{z^2}\right) = -\frac{x}{z^2}$$

Now we can get to x

There are multiple backward paths into x.
The general rule: Add the gradients along all the paths.

A concrete example with multiple outgoing edges

$$y = \frac{1}{x}$$

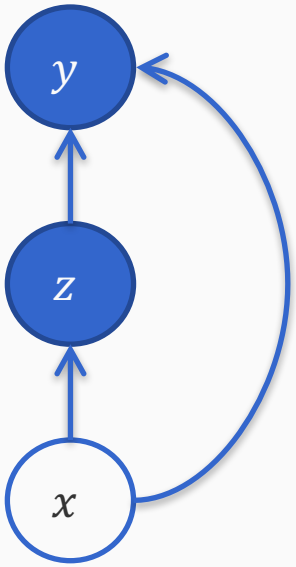
$$\frac{df}{du} = -\frac{v}{u^2}$$

$$\frac{df}{dv} = \frac{1}{u}$$

$$f(u, v) = \frac{v}{u}$$

$$\frac{dg}{du} = 2u$$

$$\mathrm{g(u)} = \mathrm{u}^2$$



$$\frac{dy}{dy} = 1$$

$$\frac{dy}{dz} = -\frac{x}{z^2}$$

$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \left(\frac{dg}{du}\right)_{u=x} + \frac{dy}{dy} \cdot \left(\frac{df}{dv}\right)_{v=x}$$

There are multiple backward paths into x.
The general rule: Add the gradients along all the paths.

A concrete example with multiple outgoing edges

$$y = \frac{1}{x}$$

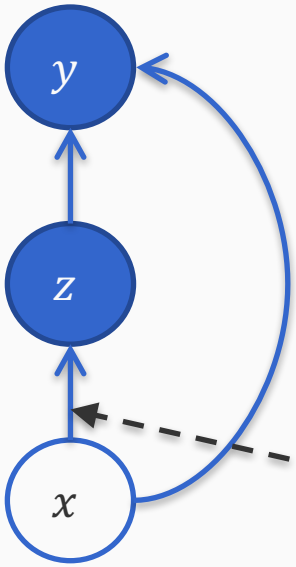
$$\frac{df}{du} = -\frac{v}{u^2}$$

$$\frac{df}{dv} = \frac{1}{u}$$

$$f(u, v) = \frac{v}{u}$$

$$\frac{dg}{du} = 2u$$

$$\mathrm{g(u)} = \mathrm{u}^2$$



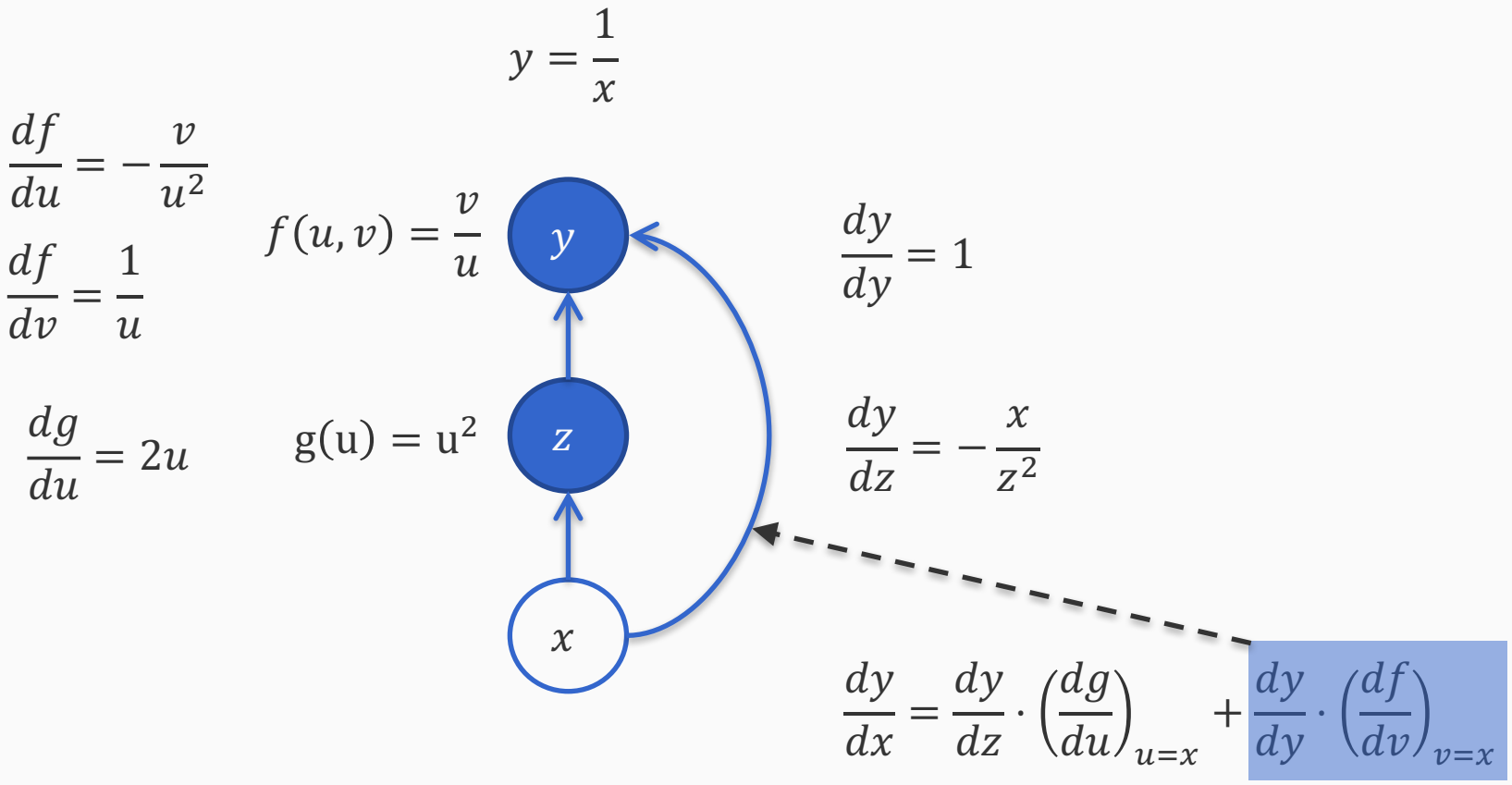
$$\frac{dy}{dy} = 1$$

$$\frac{dy}{dz} = -\frac{x}{z^2}$$

$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \left(\frac{dg}{du}\right)_{u=x} + \frac{dy}{dy} \cdot \left(\frac{df}{dv}\right)_{v=x}$$

There are multiple backward paths into x.
The general rule: Add the gradients along all the paths.

A concrete example with multiple outgoing edges



There are multiple backward paths into x.
The general rule: Add the gradients along all the paths.

A concrete example with multiple outgoing edges

$$y = \frac{1}{x}$$

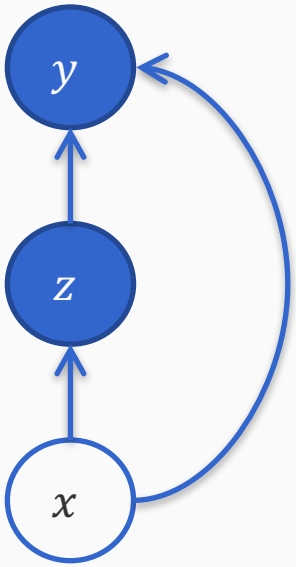
$$\frac{df}{du} = -\frac{v}{u^2}$$

$$\frac{df}{dv} = \frac{1}{u}$$

$$f(u, v) = \frac{v}{u}$$

$$\frac{dg}{du} = 2u$$

$$\mathrm{g(u)} = \mathrm{u}^2$$



$$\frac{dy}{dy} = 1$$

$$\frac{dy}{dz} = -\frac{x}{z^2}$$

$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \left(\frac{dg}{du}\right)_{u=x} + \frac{dy}{dy} \cdot \left(\frac{df}{dv}\right)_{v=x}$$

$$\frac{dy}{dx} = -\frac{x}{z^2} \cdot 2x + 1 \cdot \frac{1}{z} = -\frac{2x^2}{z^2} + \frac{1}{z} = -\frac{1}{x^2}$$

A neural network example

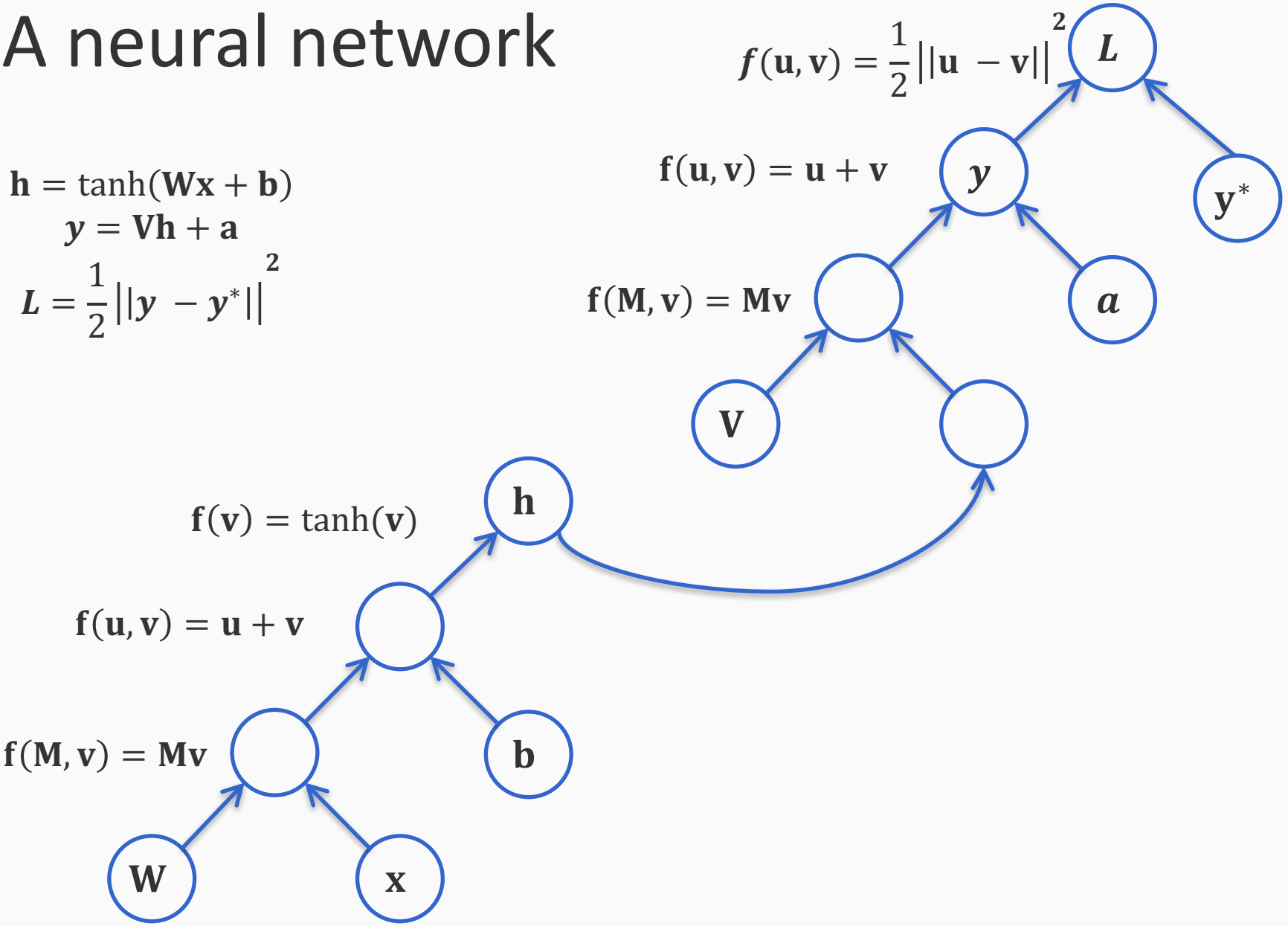
$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^{\mathbf{2}}$$

This is the same two-layer network we saw before. But this time we have added a new loss term at the end.

Suppose our goal is to compute the derivative of the loss with respect to $\mathbf{W}, \mathbf{V}, \mathbf{a}, \mathbf{b}$

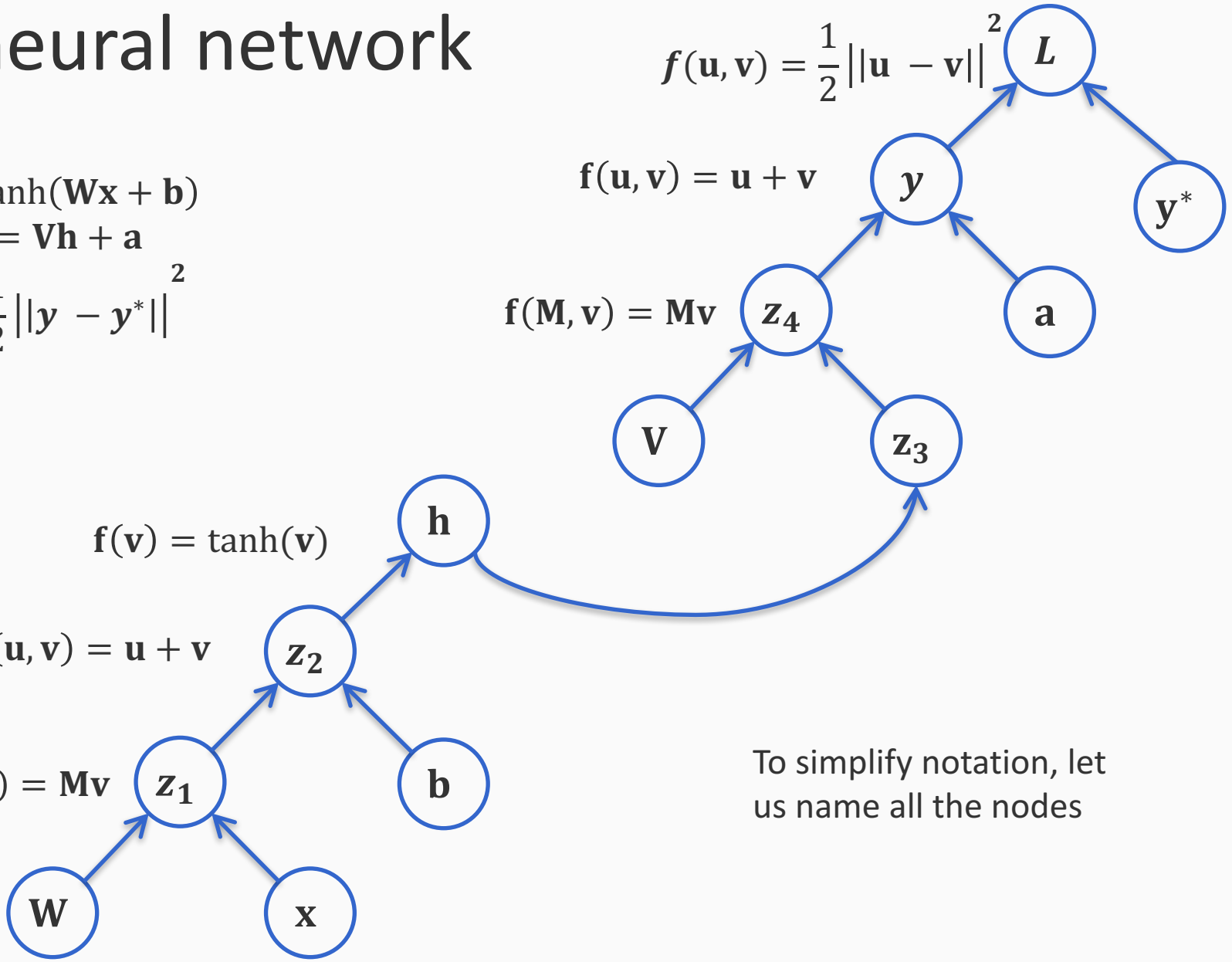
A neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^2$$



A neural network

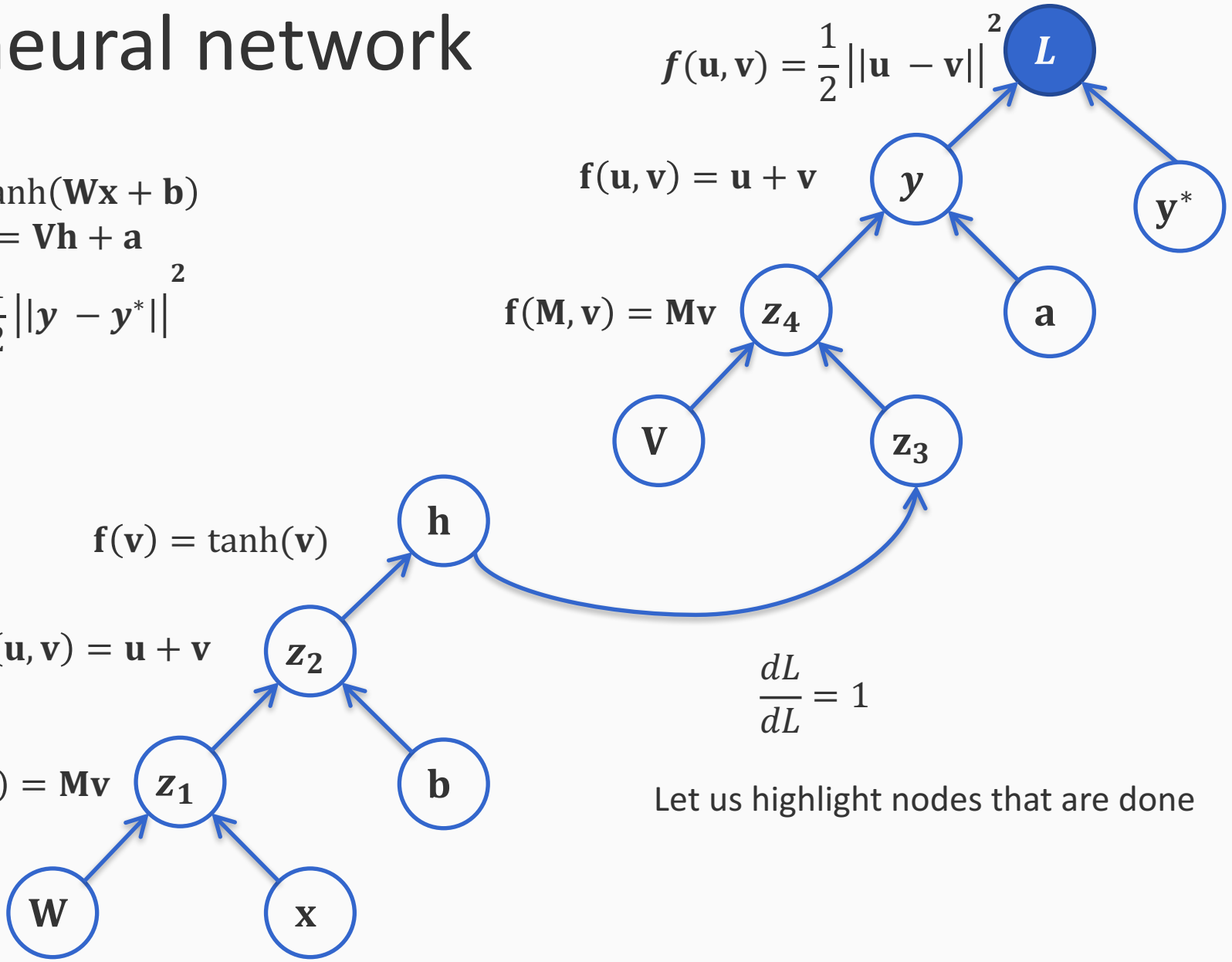
$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^2$$



To simplify notation, let us name all the nodes

A neural network

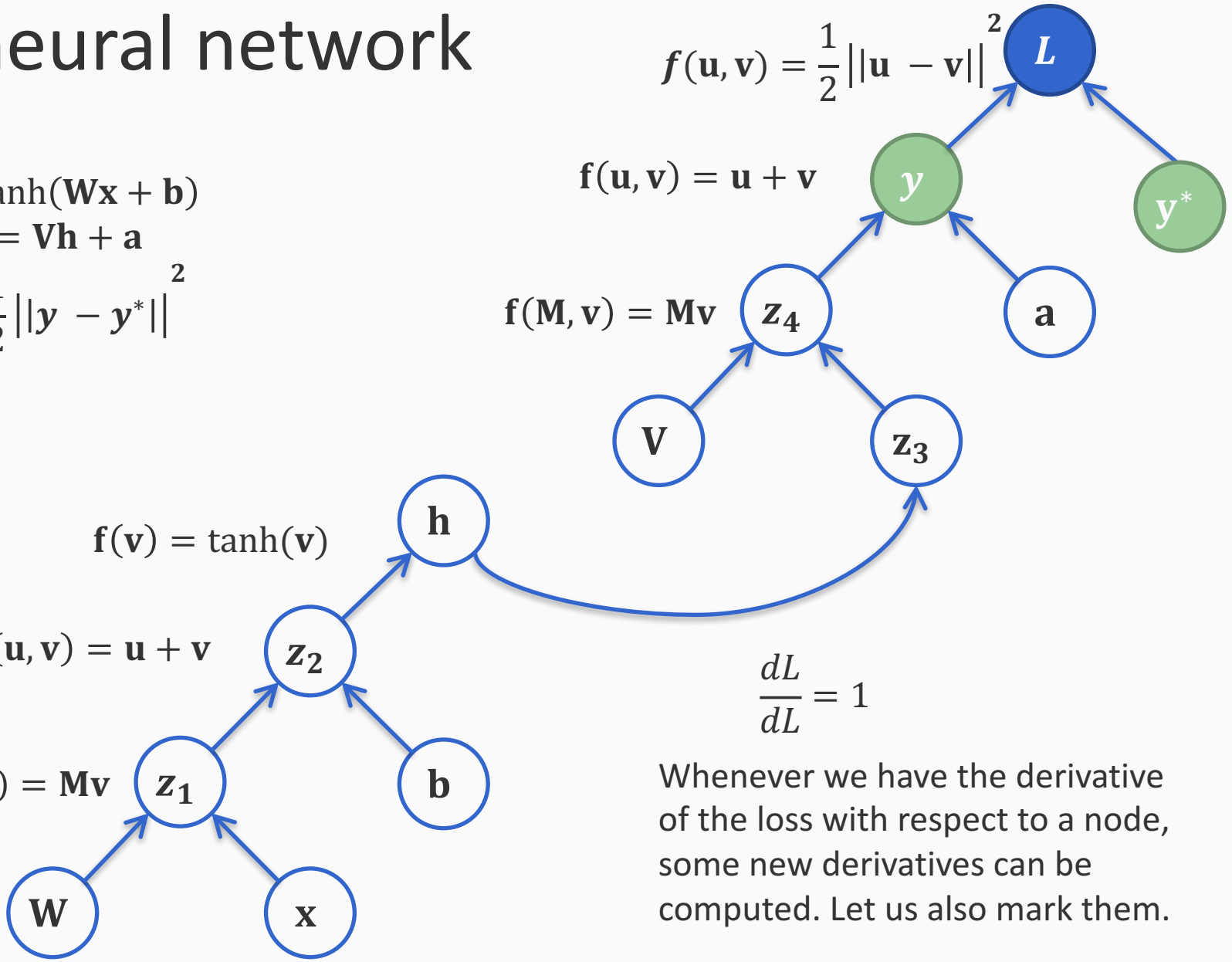
$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^2$$



Let us highlight nodes that are done

A neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^2$$

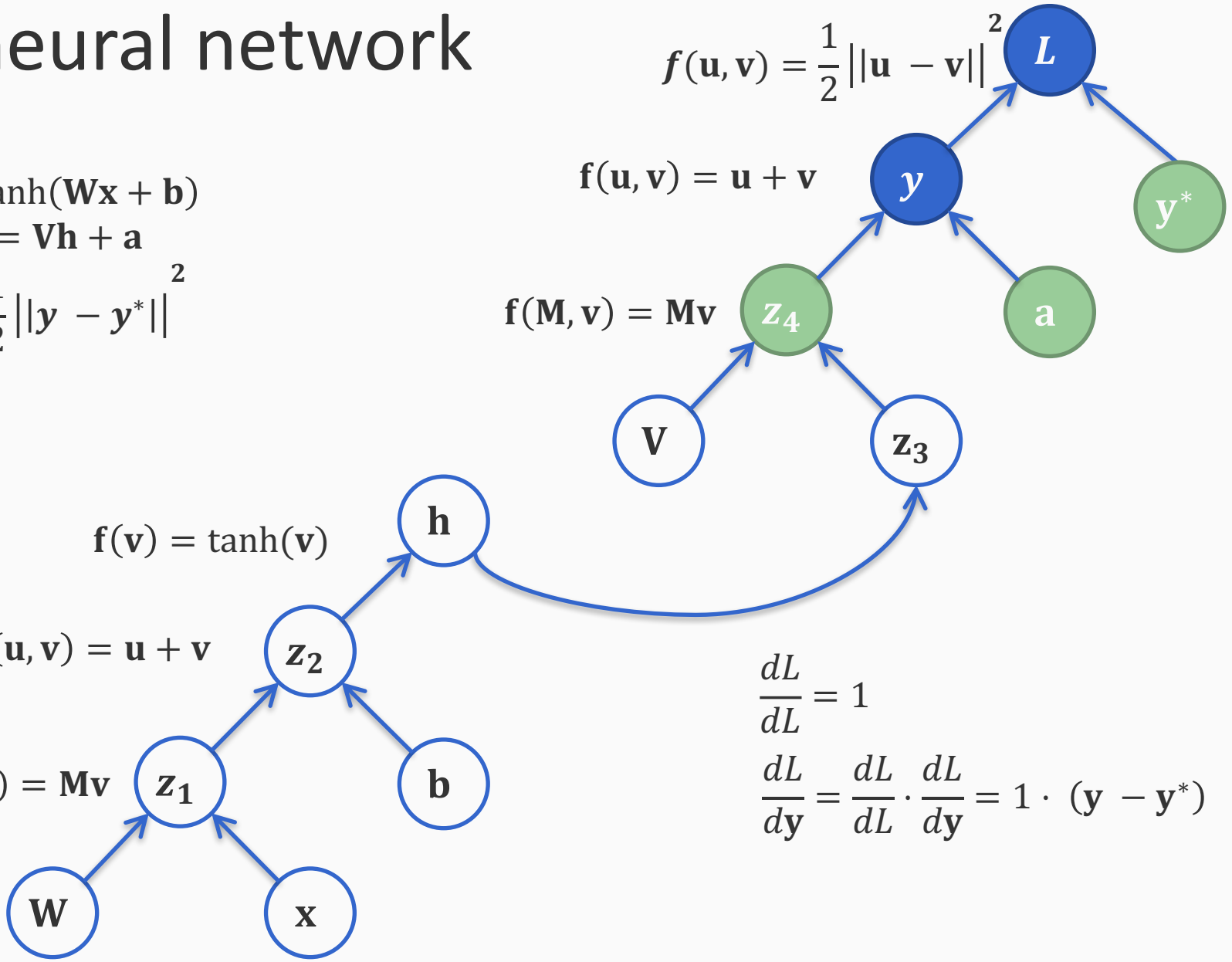


$$\frac{dL}{dL} = 1$$

Whenever we have the derivative of the loss with respect to a node, some new derivatives can be computed. Let us also mark them.

A neural network

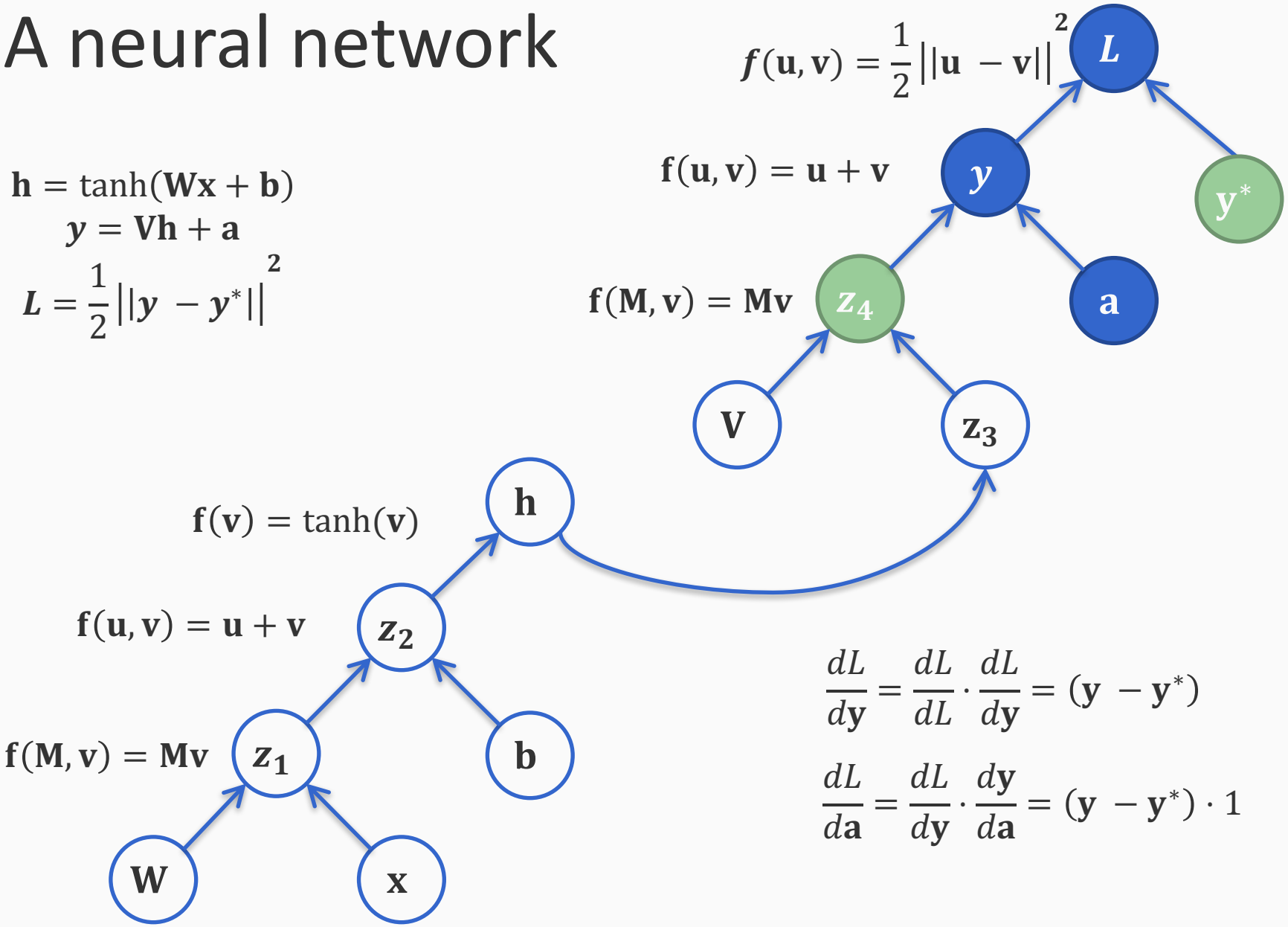
$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^2$$



$$\frac{dL}{dL} = 1$$
$$\frac{dL}{d\mathbf{y}} = \frac{dL}{dL} \cdot \frac{dL}{d\mathbf{y}} = 1 \cdot (\mathbf{y} - \mathbf{y}^*)$$

A neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^2$$

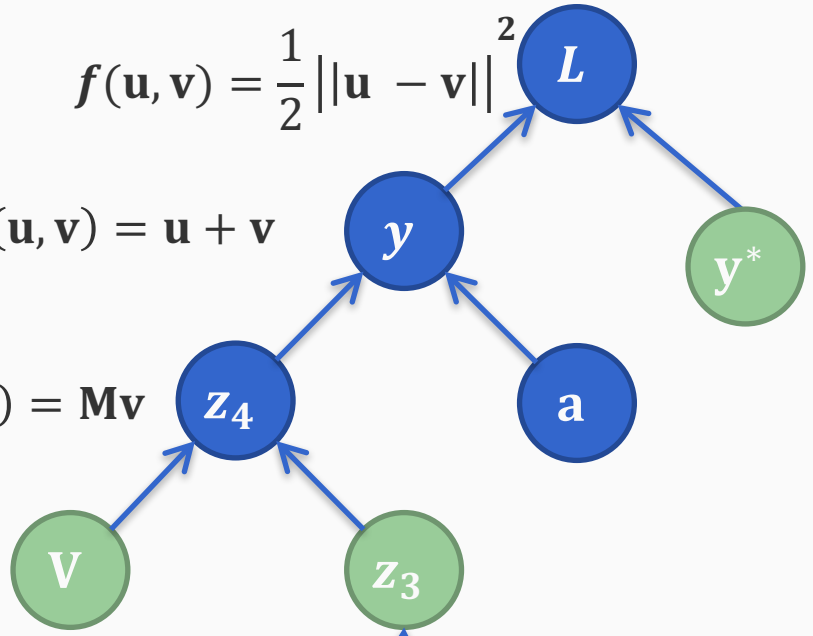


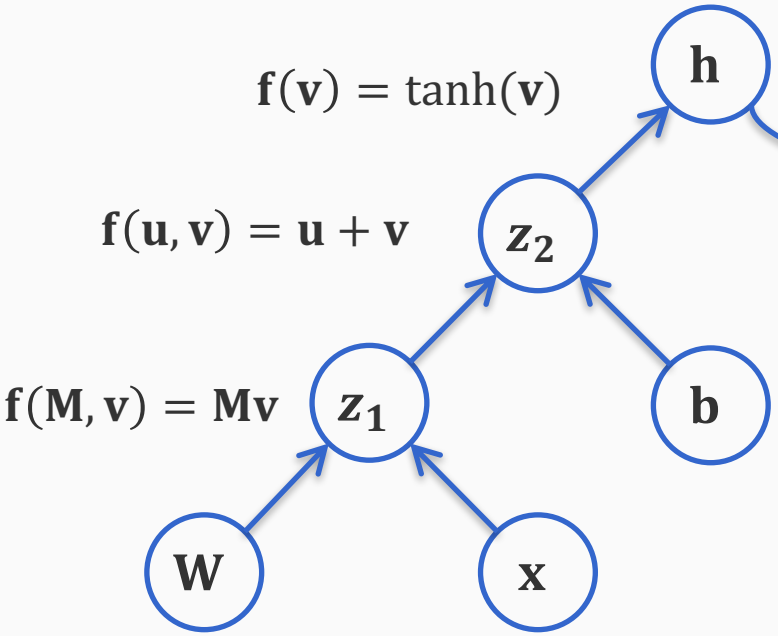
$$\frac{dL}{d\mathbf{y}} = \frac{dL}{dL} \cdot \frac{dL}{d\mathbf{y}} = (\mathbf{y} - \mathbf{y}^*)$$

$$\frac{dL}{d\mathbf{a}} = \frac{dL}{d\mathbf{y}} \cdot \frac{d\mathbf{y}}{d\mathbf{a}} = (\mathbf{y} - \mathbf{y}^*) \cdot 1$$

A neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$L = \frac{1}{2}\left|\left|\mathbf{y} - \mathbf{y}^*\right|\right|^2$$



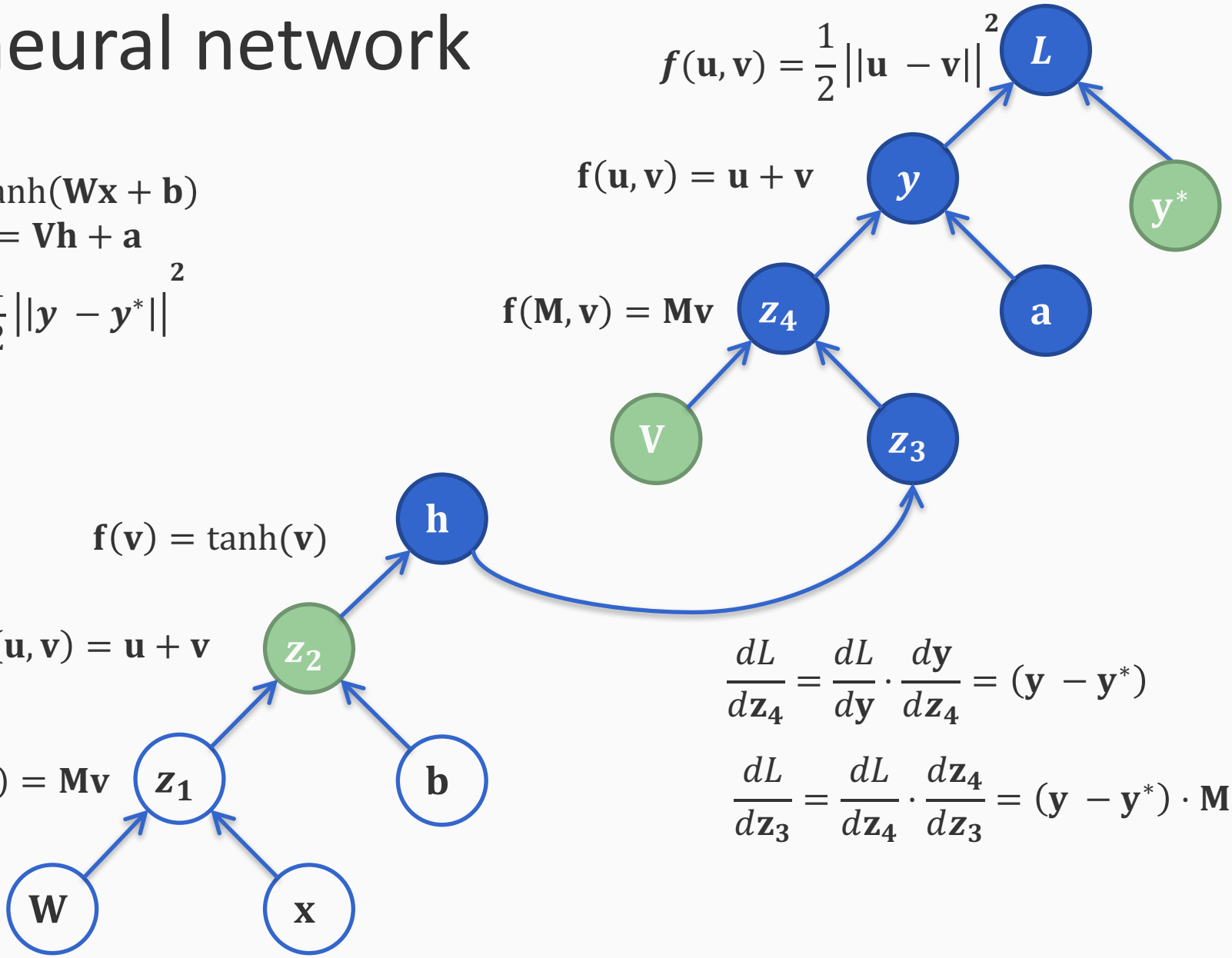


$$\frac{dL}{d\mathbf{y}} = \frac{dL}{dL} \cdot \frac{dL}{d\mathbf{y}} = (\mathbf{y} - \mathbf{y}^*)$$

$$\frac{dL}{dz_4} = \frac{dL}{d\mathbf{y}} \cdot \frac{d\mathbf{y}}{dz_4} = (\mathbf{y} - \mathbf{y}^*) \cdot 1$$

A neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$L = \frac{1}{2}\left|\left|\mathbf{y} - \mathbf{y}^*\right|\right|^2$$

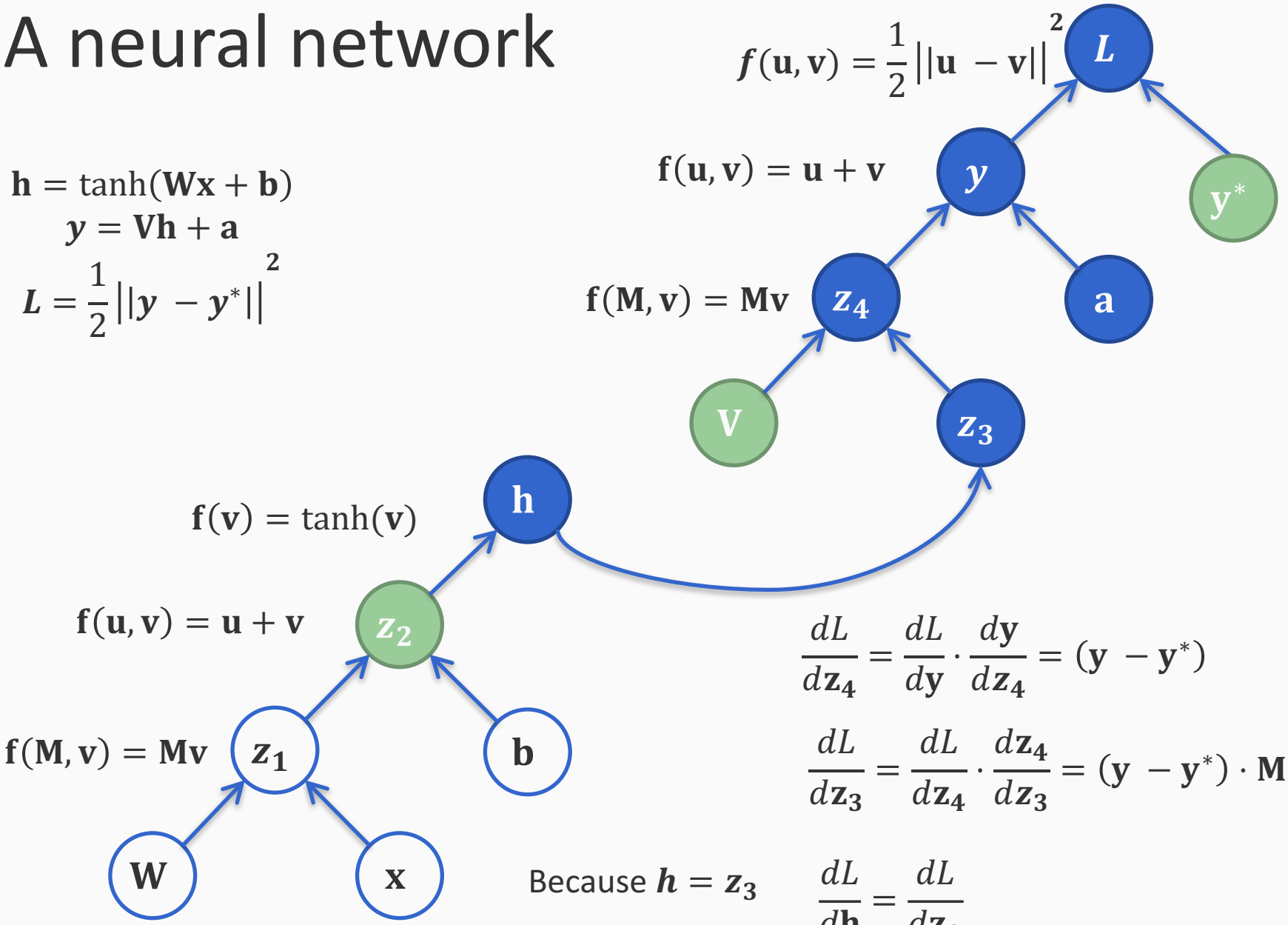


$$\frac{dL}{d\mathbf{z_4}} = \frac{dL}{d\mathbf{y}} \cdot \frac{d\mathbf{y}}{d\mathbf{z_4}} = (\mathbf{y} - \mathbf{y}^*)$$

$$\frac{dL}{d\mathbf{z_3}} = \frac{dL}{d\mathbf{z_4}} \cdot \frac{d\mathbf{z_4}}{d\mathbf{z_3}} = (\mathbf{y} - \mathbf{y}^*) \cdot \mathbf{M}$$

A neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^2$$



$$\frac{dL}{d\mathbf{z_4}} = \frac{dL}{d\mathbf{y}} \cdot \frac{d\mathbf{y}}{d\mathbf{z_4}} = (\mathbf{y} - \mathbf{y}^*)$$

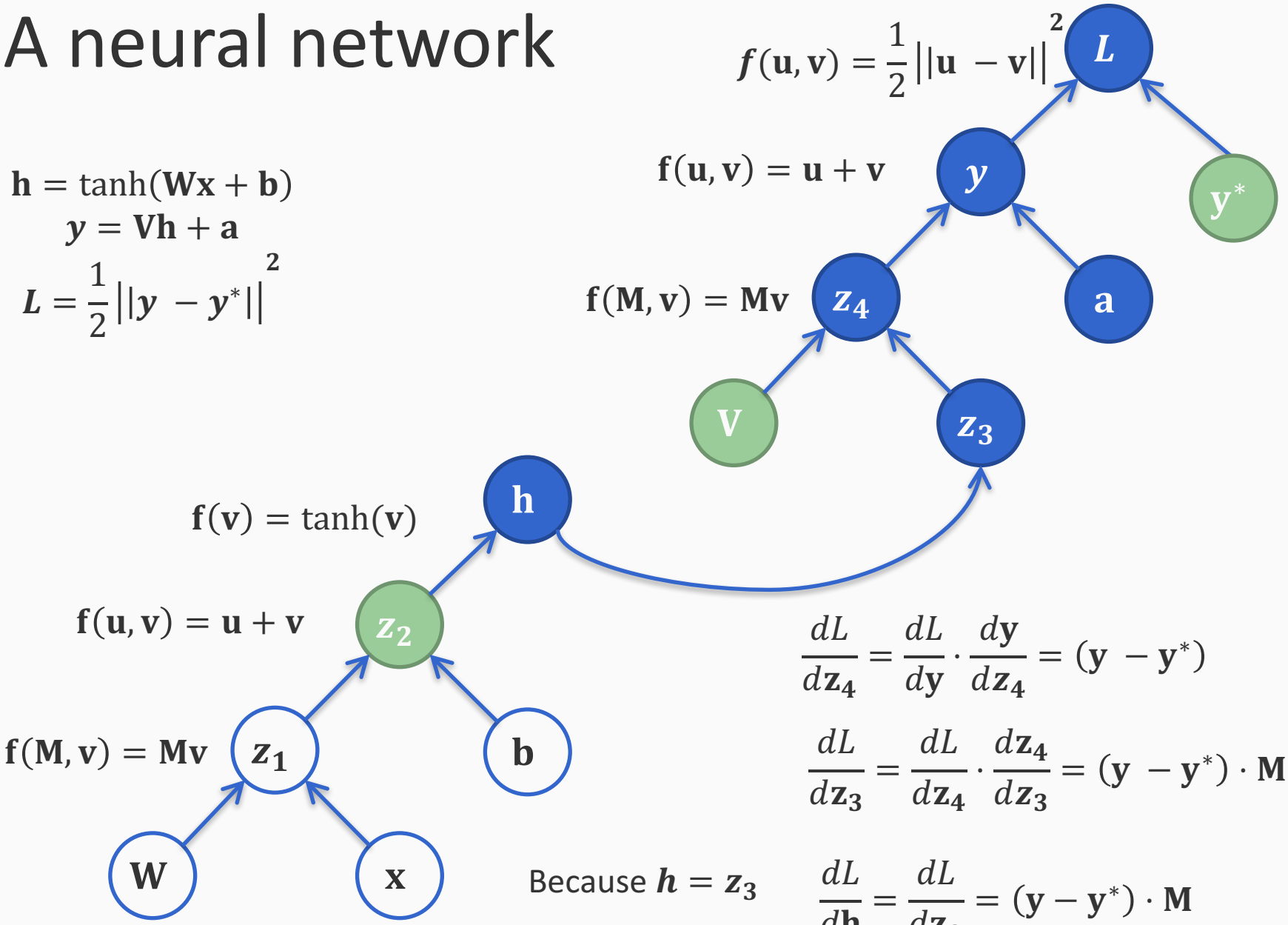
$$\frac{dL}{d\mathbf{z_3}} = \frac{dL}{d\mathbf{z_4}} \cdot \frac{d\mathbf{z_4}}{d\mathbf{z_3}} = (\mathbf{y} - \mathbf{y}^*) \cdot \mathbf{M}$$

Because $\boldsymbol{h} = \boldsymbol{z_3}$

$$\frac{dL}{d\mathbf{h}} = \frac{dL}{d\mathbf{z_3}}$$

A neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^2$$



$$\frac{dL}{d\mathbf{z_4}} = \frac{dL}{d\mathbf{y}} \cdot \frac{d\mathbf{y}}{d\mathbf{z_4}} = (\mathbf{y} - \mathbf{y}^*)$$

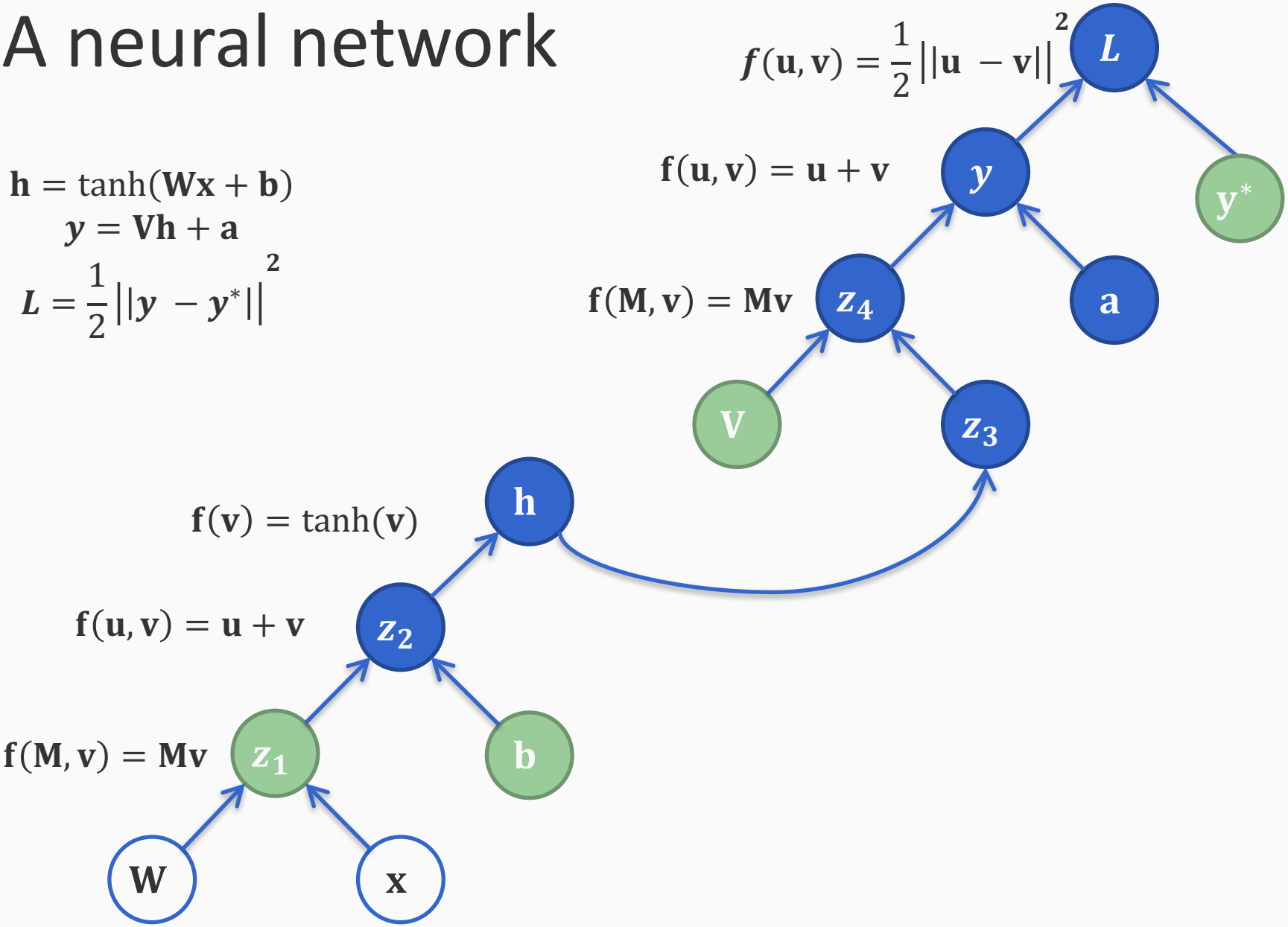
$$\frac{dL}{d\mathbf{z_3}} = \frac{dL}{d\mathbf{z_4}} \cdot \frac{d\mathbf{z_4}}{d\mathbf{z_3}} = (\mathbf{y} - \mathbf{y}^*) \cdot \mathbf{M}$$

Because $\boldsymbol{h} = \boldsymbol{z_3}$

$$\frac{dL}{d\mathbf{h}} = \frac{dL}{d\mathbf{z_3}} = (\mathbf{y} - \mathbf{y}^*) \cdot \mathbf{M}$$

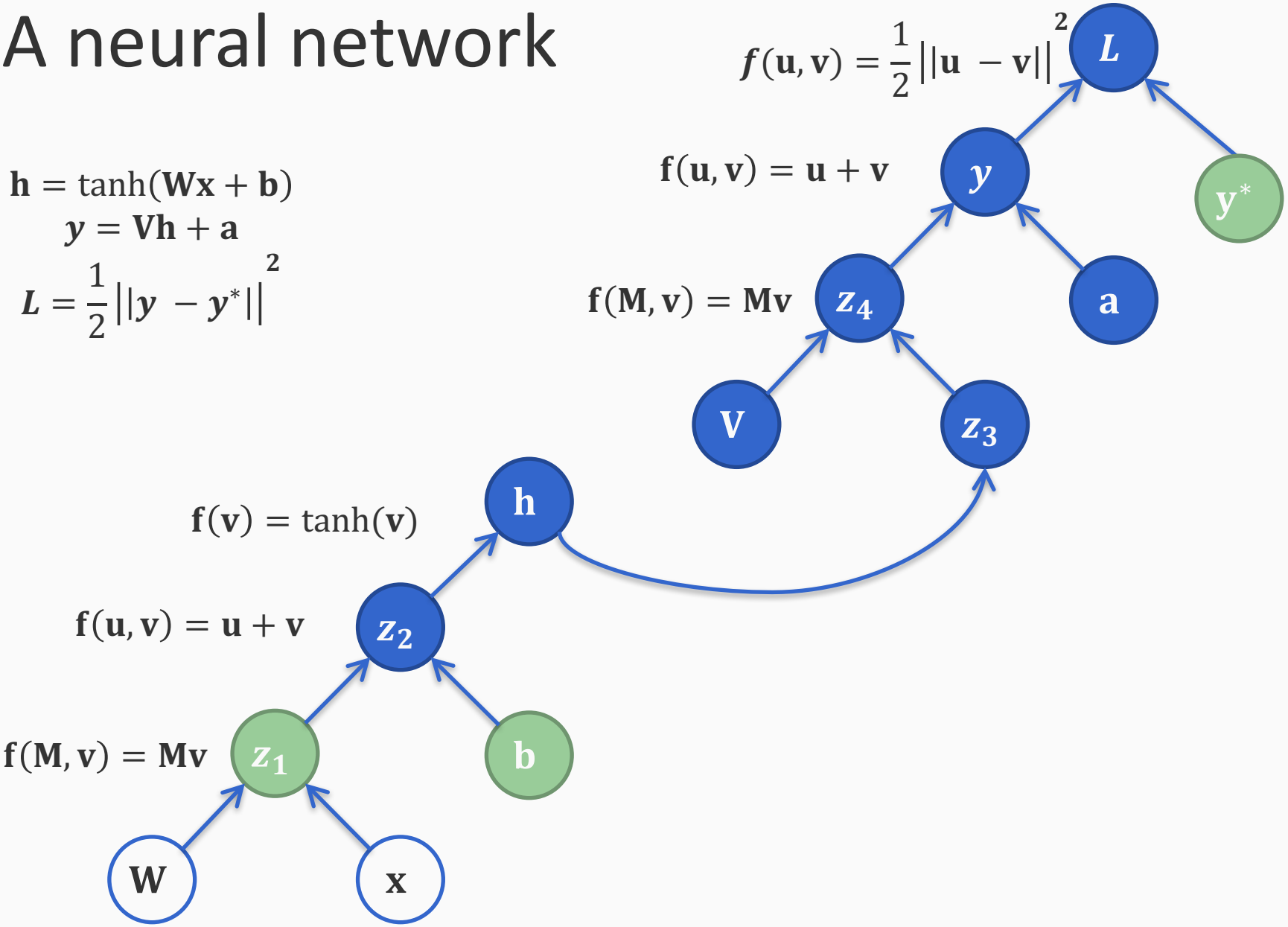
A neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^2$$



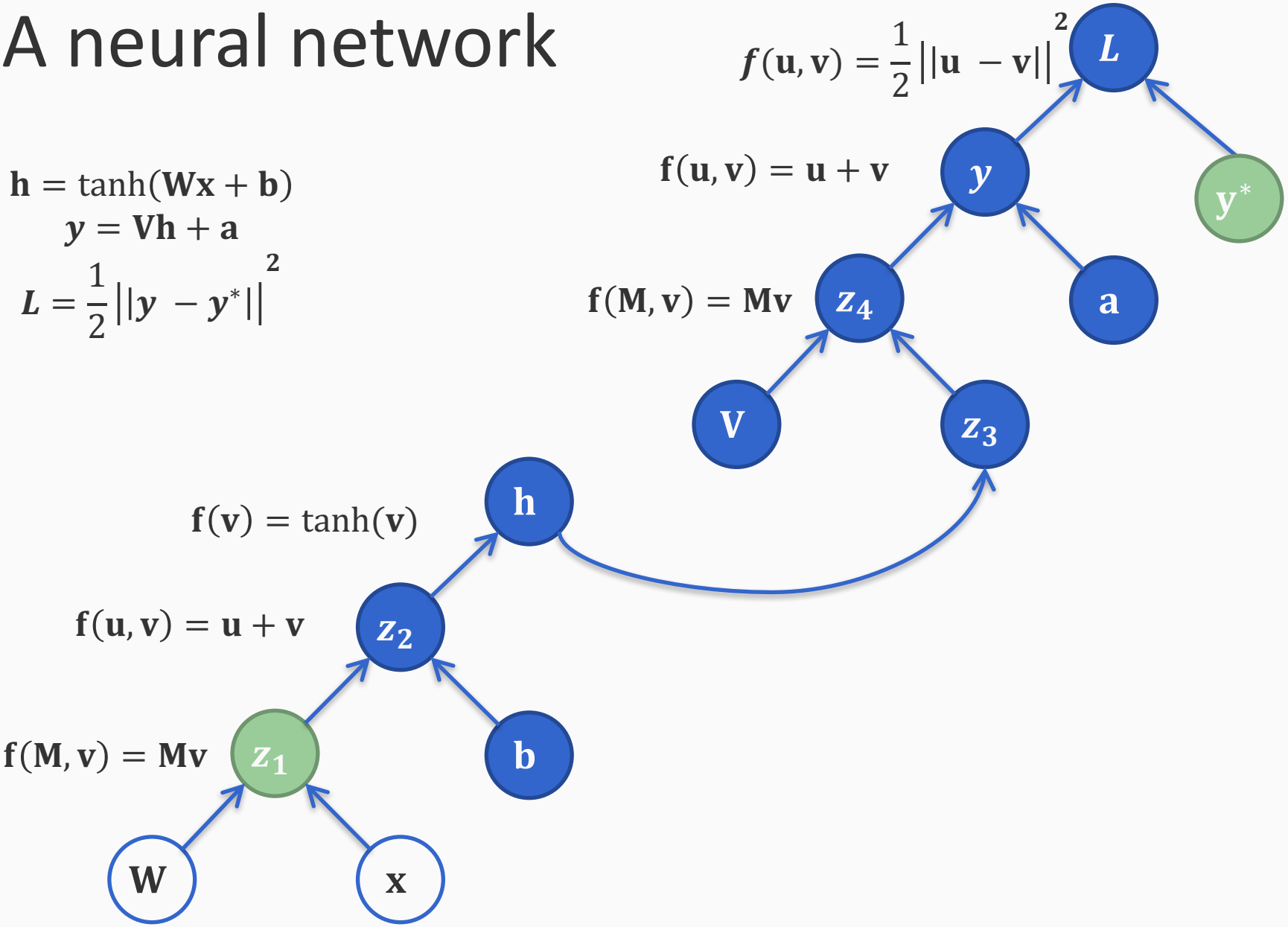
A neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^2$$



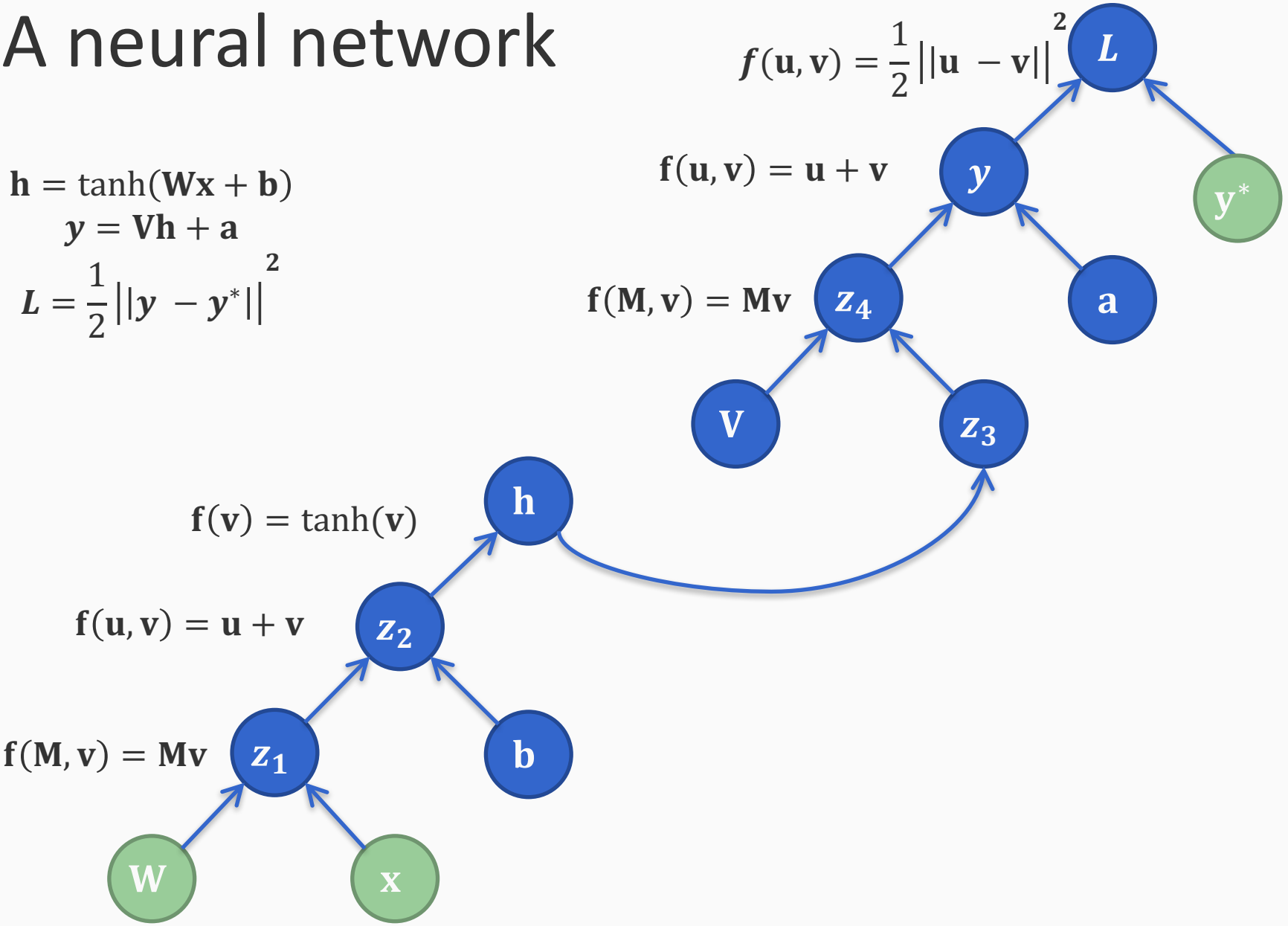
A neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^2$$



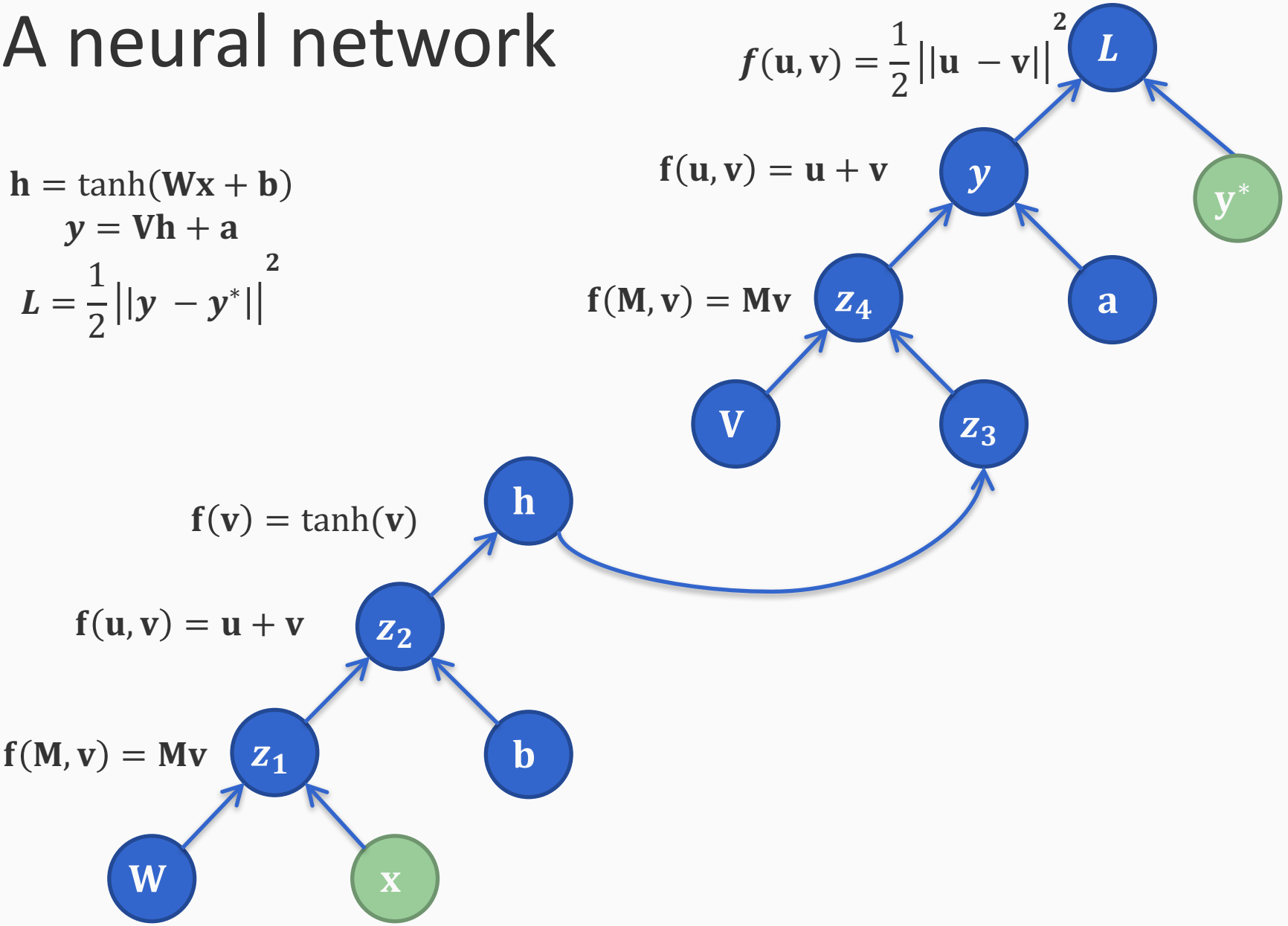
A neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^2$$



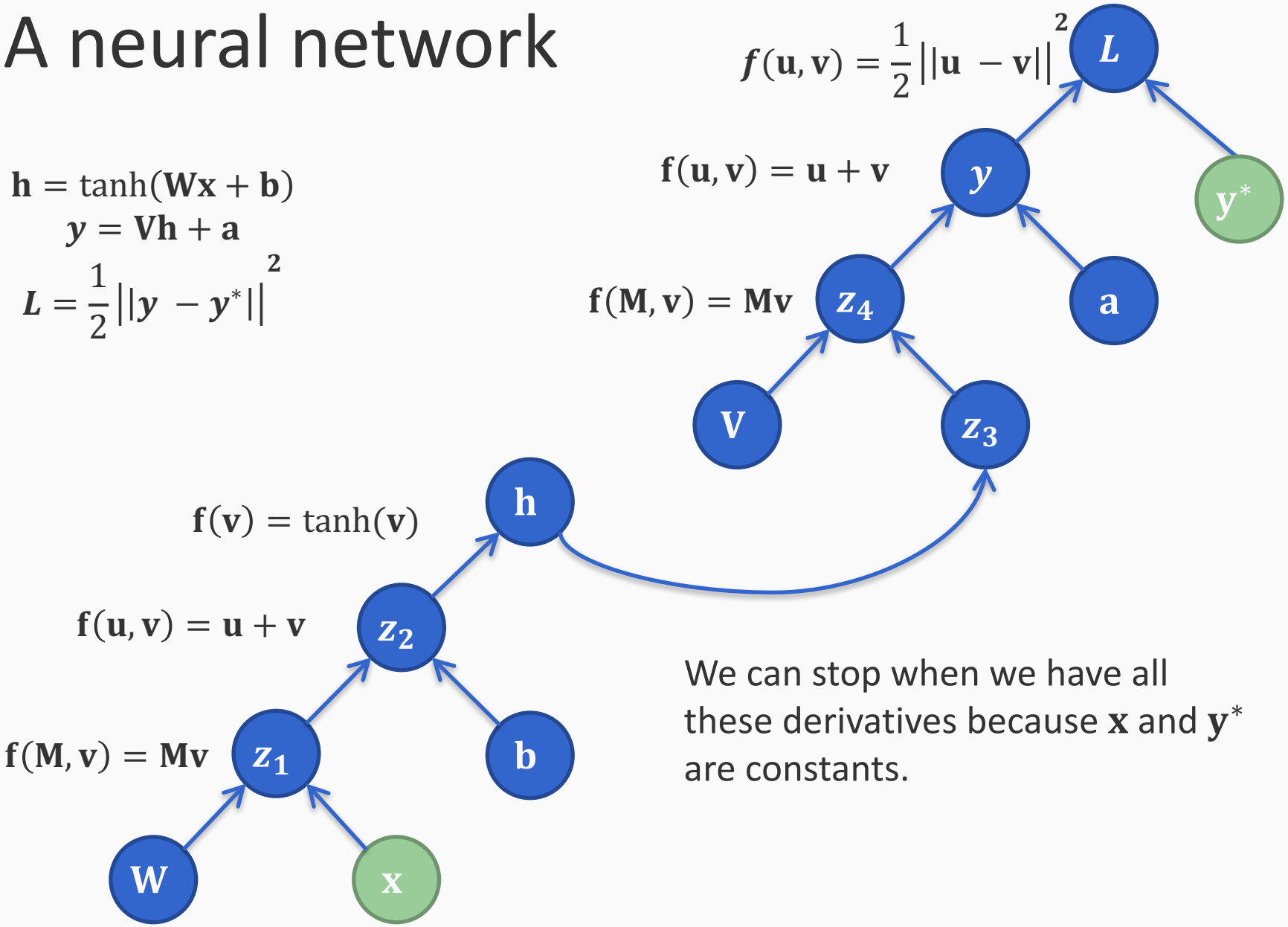
A neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^2$$



A neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\boldsymbol{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$
$$\boldsymbol{L} = \frac{1}{2}\left|\left|\boldsymbol{y} - \boldsymbol{y}^*\right|\right|^2$$



We can stop when we have all these derivatives because $\mathbf{x}$ and $\mathbf{y}^*$ are constants.

Backpropagation, in general

After we have done the forward propagation,

Loop over the nodes in **reverse topological order** starting with a final goal node

- Compute derivatives of final goal node value with respect to each edge's tail node
  - If there are multiple outgoing edges from a node, sum up all the derivatives for the edges

Constructing computation graphs

Three computational questions

1. Forward propagation
   - Given inputs to the graph, compute the value of the function expressed by the graph
   - Something to think about: Given a node, can we say which nodes are inputs? Which nodes are outputs?

2. Backpropagation
   - After computing the function value for an input, compute the gradient of the function at that input
   - Or equivalently: *How does the output change if I make a small change to the input?*

3. Constructing graphs
   - Need an easy-to-use framework to construct graphs
   - The size of the graph may be input dependent
     - A templating language that creates graphs on the fly
   - Tensorflow, PyTorch are the most popular frameworks today

Two methods for constructing graphs

We may require different sized computation graphs for different inputs

- Eg: different sentences have different lengths. We may have a neural network whose size depends on sentence length.
- How could we statically declare a computation graph of a fixed size?

- **One option**: Assume a size that is big enough and for smaller examples, pad it with dummy values

- **Another option**: Dynamically create a computation graph on the fly when we need to.

Two methods for constructing graphs

- Static declaration
  - Phase 1: Define an architecture
    - Maybe using standard control flow operations like loops, conditionals, etc to simplify repeated code
  - Phase 2: Run a bunch of data through the graph to train and make predictions

- Dynamic declaration
  - Graph is constructed implicitly (perhaps via operator overloading) at the same time as the forward propagation

Static declaration

- Pros
  - Offline optimization/scheduling of graphs is powerful
  - Limits on operations mean better hardware support
- Cons
  - Structured data (even simple stuff like sequences), even variable-sized data, is ugly
  - You effectively learn a new programming language (“the Graph Language”) and you write programs in that language to process data.
- Examples: Torch/PyTorch, Theano, TensorFlow

Dynamic declaration

- Pros
  - The library is less invasive, no need to learn a new syntax
  - Forward computation is written in your favorite programming language with all its features, using your favorite algorithms
  - Interleave construction and evaluation of the graph
- Cons
  - We can't do offline graph optimization because there is little time
  - If the graph is static, the effort can be wasted
- Examples: Chainer, most automatic differentiation libraries, DyNet

Summary: Computation graphs

An abstraction that allows us to write any differentiable (or sub-differentiable) functions as a directed acyclic graph

- Building blocks for modern neural networks
- This will allow us to think about differentiable programs

Two algorithms:

- Forward propagation: process nodes in topological order to compute function value
- Backpropagation: process nodes in reverse topological order to compute derivative

Two methods for constructing graphs: Static vs dynamic