Neural Networks and Computation Graphs

THE UNIVERSITY OF UTAH

Based on slides and material from Geoffrey Hinton, Richard Socher, Yoav Goldberg, Chris Dyer, Graham Neubig and others.

This lecture

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

This lecture

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

This section heavily draws upon the “Practical Neural Networks for NLP” by Chris Dyer, Yoav Goldberg, Graham Neubig in EMNLP 2016

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. Tensors

2. What is the semantics of a computation graph?
   – That is, what the nodes and edges mean

3. How to construct them

4. How do perform computations with them

What we will see

1. Tensors

2. What is the semantics of a computation graph?
   – That is, what the nodes and edges mean

3. How to construct them

4. How do perform computations with them

Tensors: A quick primer

Tensors generalize vectors and matrices

- For the most part (in what we will see), whenever you see tensor, you can think “multi-dimensional arrays”

Tensors: A quick primer

Tensors generalize vectors and matrices

- For the most part (in what we will see), whenever you see tensor, you can think "multi-dimensional arrays"

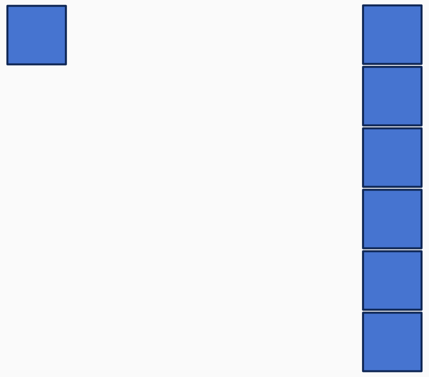
Scalars (i.e., numbers) are tensors with zero dimensions. 3, -1, 1.1, …

Why zero dimensions? Because we need zero indices to find only element contained in it

Tensors: A quick primer

Tensors generalize vectors and matrices

- For the most part (in what we will see), whenever you see tensor, you can think "multi-dimensional arrays"



Vectors are one dimensional tensors:
[1,2,3], [-11.3, 0], …

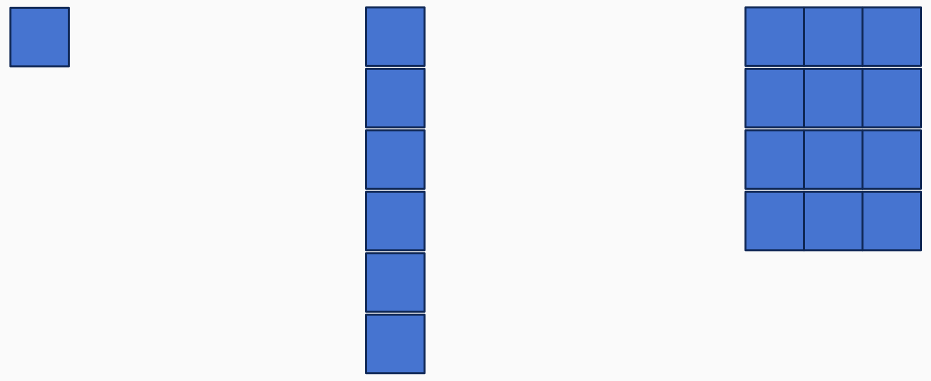
Why one dimensional? Because we need one index to address any element in the vector

The shape of this tensor is 6.
It is a six dimensional vector.

Tensors: A quick primer

Tensors generalize vectors and matrices

– For the most part (in what we will see), whenever you see tensor, you can think "multi-dimensional arrays"



Matrices are two dimensional tensors

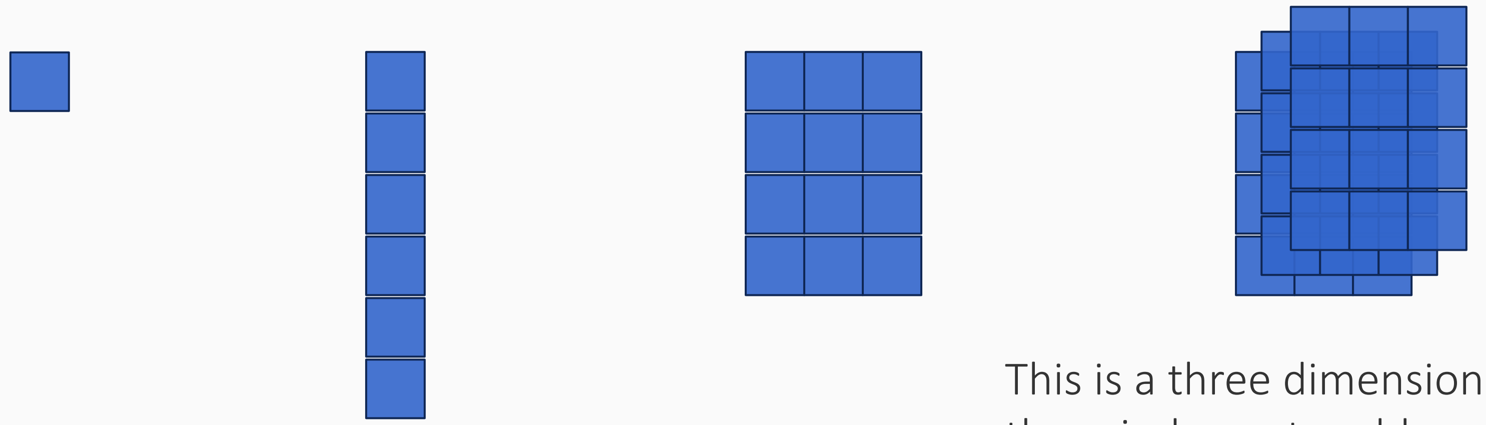
Why two dimensional? Because we need two indexes to address any element in it

The shape of this tensor is (4, 3). It is a 4×3 matrix.

Tensors: A quick primer

Tensors generalize vectors and matrices

- For the most part (in what we will see), whenever you see tensor, you can think "multi-dimensional arrays"



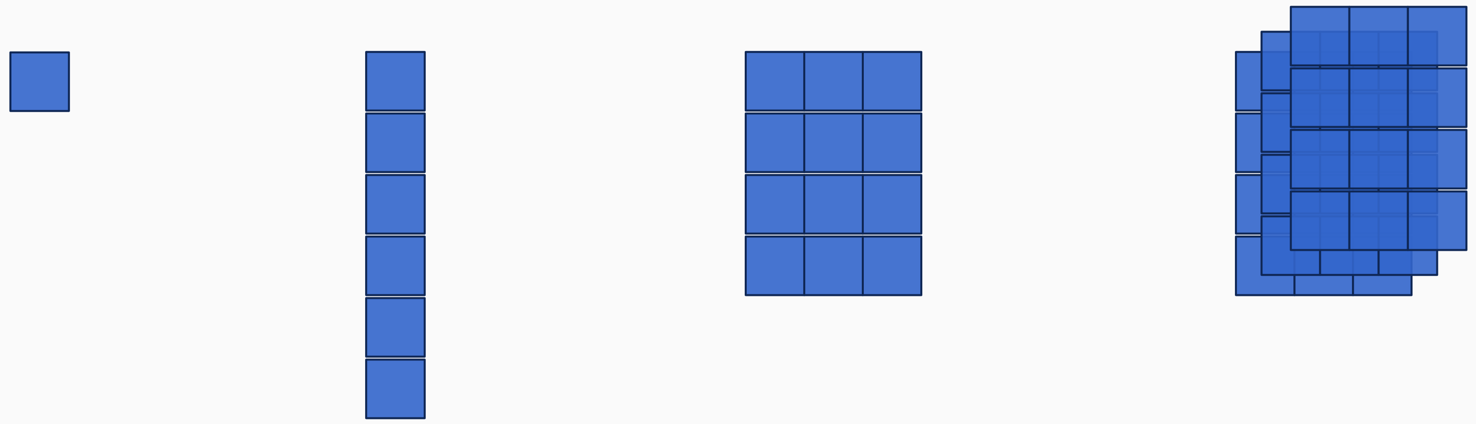
This is a three dimensional tensor. We need three indexes to address any element in it.

Its shape is (4, 3, 3)

Tensors: A quick primer

Tensors generalize vectors and matrices

- For the most part (in what we will see), whenever you see tensor, you can think "multi-dimensional arrays"



And so on…

Operations on tensors

Indexing to obtain sub-tensors (or scalars). Examples:

- $x[i], M[i,j], A[i,j,k], \ldots$ (sometimes written using subscripts): Look up an entry in a vector x or a matrix M or a 3-dimensional tensor A
- $M[i,:]$ (using `numpy` notation): Lookup the i^{th} row of the matrix M
- $A[i,:,:]$ (using `numpy` notation): Lookup the i^{th} slice of tensor A to produce a matrix
- $T[:,:,:,i]$ (using `numpy` notation): Lookup the i^{th} sub-tensor of a 4-dimensional T to produce a 3-dimensional tensor

Operations on tensors

Tensors of the same shape can be:

- Added: Add the corresponding elements
- Multiplied element-wise: Multiply corresponding elements
- …any binary operation on numbers can be applied elementwise

Operations on tensors

Tensors can be multiplied using a generalization of matrix multiplication

Sometimes this is called **Tensor Mode-n Multiplication**

Operations on tensors

Tensors can be multiplied using a generalization of matrix multiplication

Sometimes this is called **Tensor Mode-n Multiplication**

Suppose we have $A \in \Re^{M \times N}, B \in \Re^{N \times K}$
We can define the product of A and B to produce a tensor C as follows:

$$C[m,k] = \sum_{n} A[m,n]B[n,k]$$

This is just matrix-matrix multiplication

Operations on tensors

Tensors can be multiplied using a generalization of matrix multiplication

Sometimes this is called **Tensor Mode-n Multiplication**

Suppose we have $A \in \Re^{M\times N}, B \in \Re^{N\times K}$
We can define the product of A and B to produce a tensor C as follows:

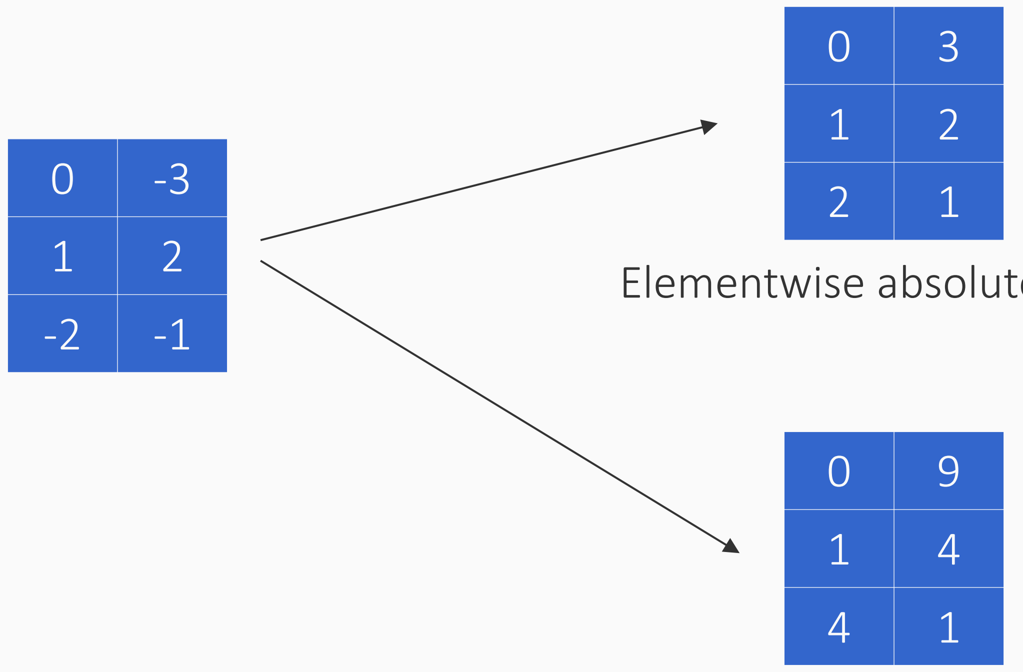
$$C[m,k] = \sum_{n} A[m,n]B[n,k]$$

Suppose we have $A \in \Re^{M\times N\times R}, B \in \Re^{N\times K}$
We can define the product of A and B to produce a tensor C as follows:

$$C[m,r,k] = \sum_{n} A[m,n,r]B[n,k]$$

Operations on tensors

Elementwise operations: Apply some function to each element of the tensor



Elementwise absolute value

Elementwise square

Operations on tensors

Reshape: Re-organize the numbers in the tensor to produce a tensor of a different shape and/or dimensionality

4
3.5
1.2
-3
9
-1

4	-3
3.5	9
1.2	-1

A 6 dimensional tensor reshaped to a 3×2 matrix

There is a lot more about tensors that you can learn by doing

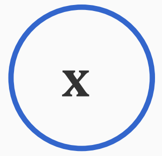
What we will see

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

Nodes represent values

Expression $\mathbf{x}$

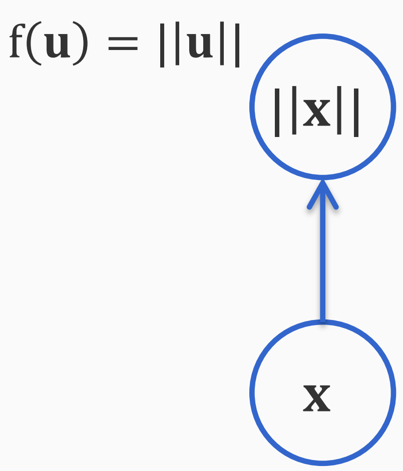
Graph

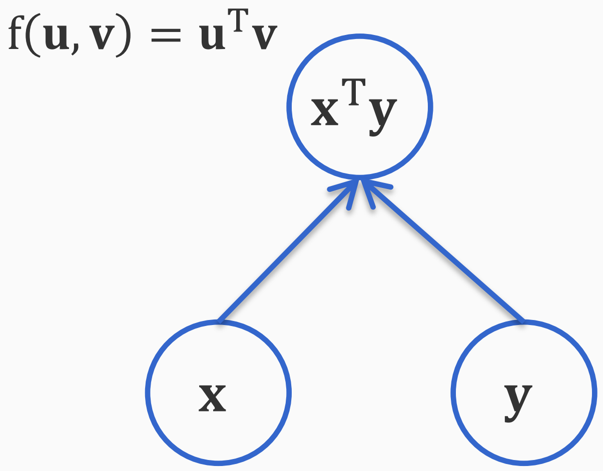


The value is implicitly or explicitly typed.

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

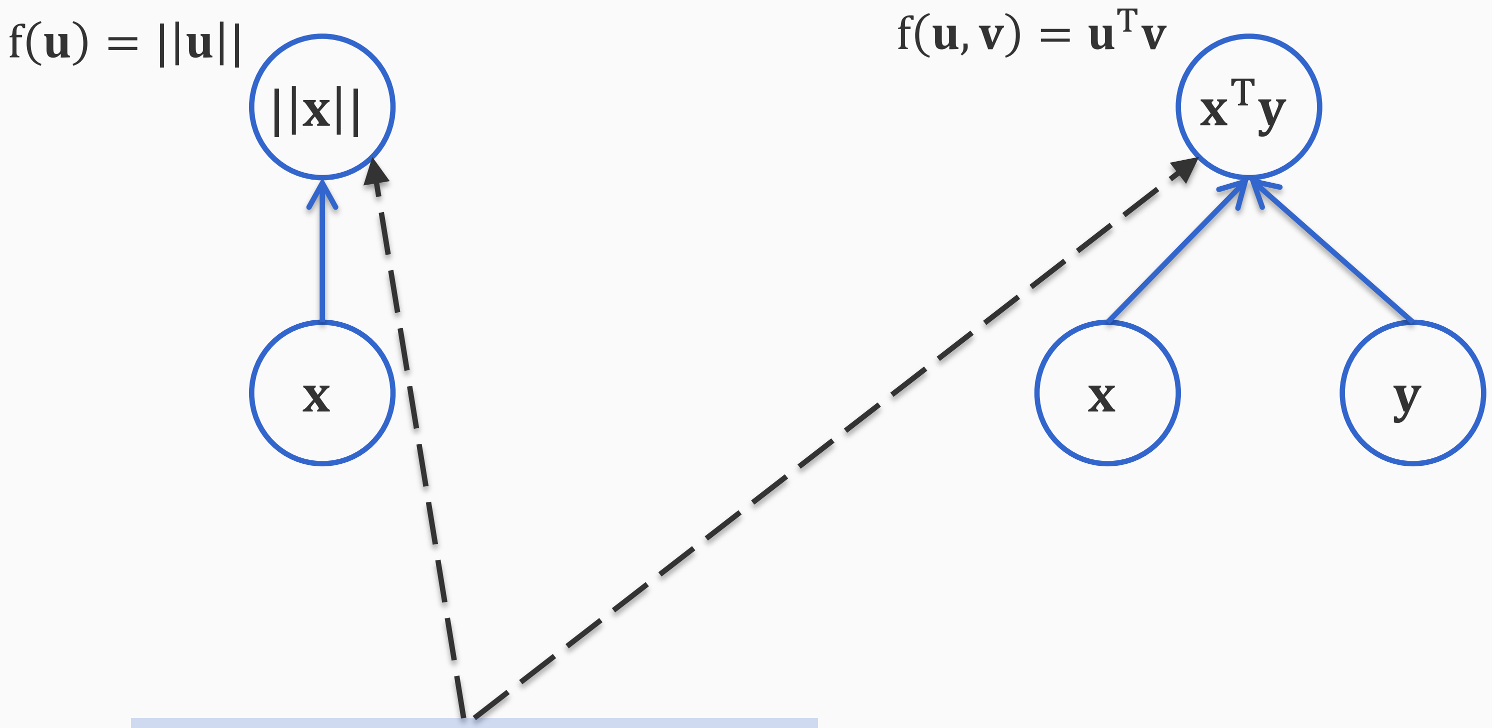
Edges represent function arguments





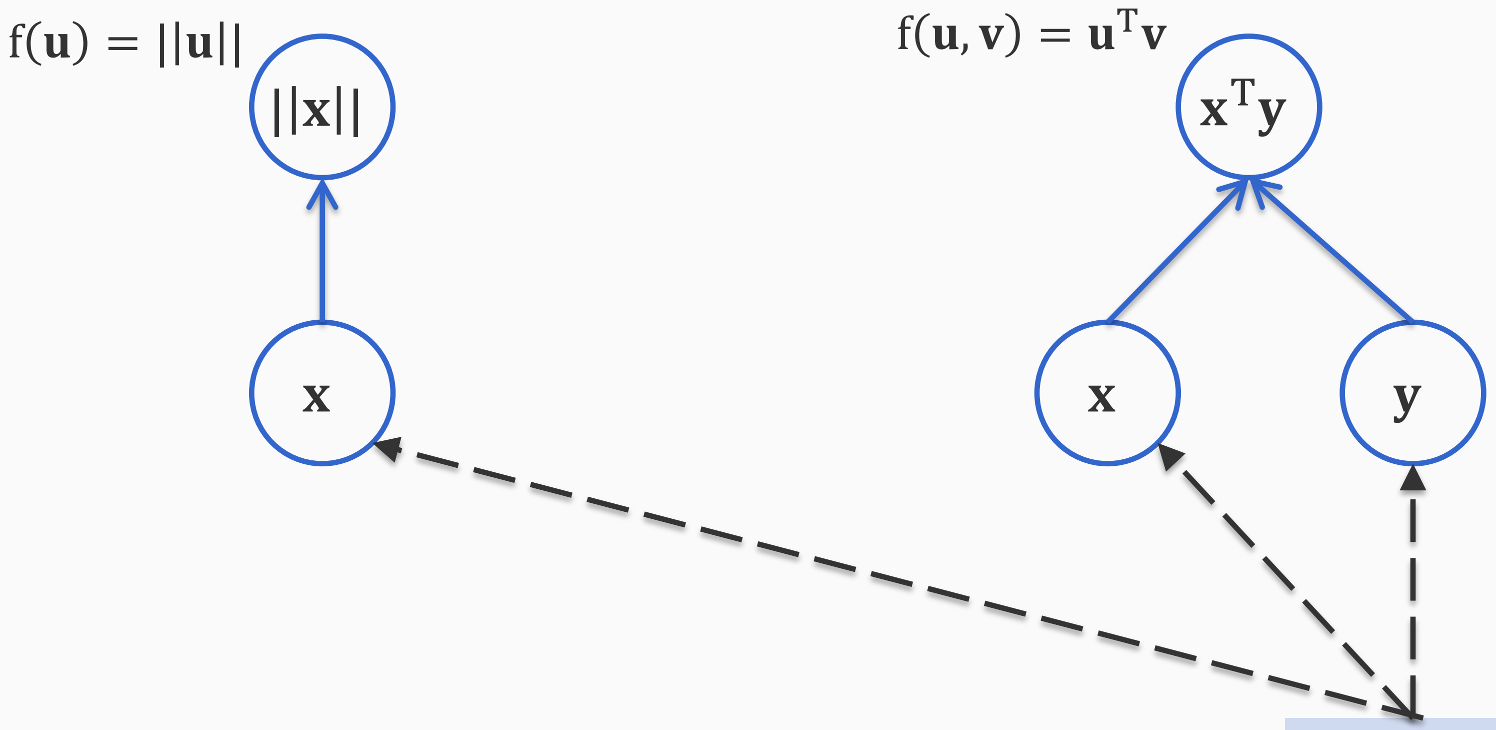
A node with an incoming edge is a function of the the parent node

Edges represent function arguments


$f(\mathbf{u}) = ||\mathbf{u}||$
$||\mathbf{x}||$
$\mathbf{x}$
$f(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T\mathbf{v}$
$\mathbf{x}^T\mathbf{y}$
$\mathbf{x}$
$\mathbf{y}$

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

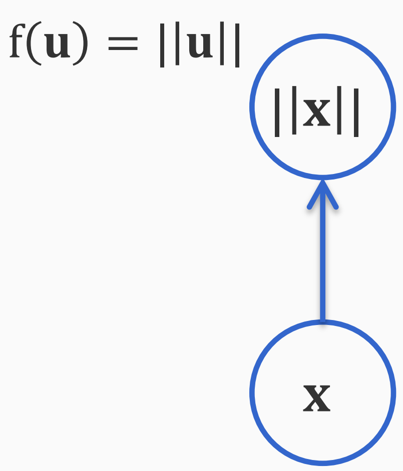
Edges represent function arguments

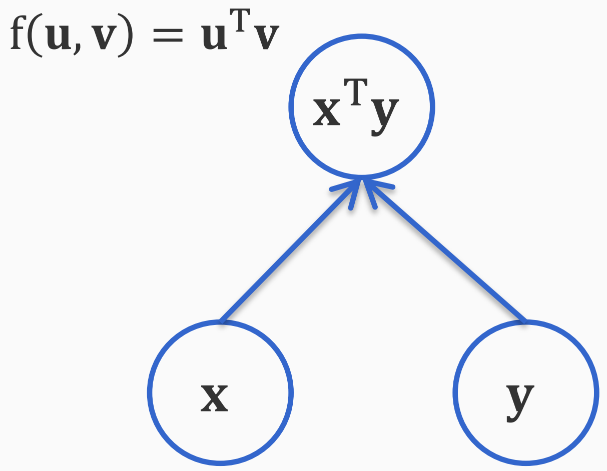


$f(\mathbf{u}) = ||\mathbf{u}||$
$||\mathbf{x}||$
$\mathbf{x}$
$f(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T\mathbf{v}$
$\mathbf{x}^T\mathbf{y}$
$\mathbf{x}$
$\mathbf{y}$

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

$\mathrm{f}(\mathbf{u}) = ||\mathbf{u}||$

$$\begin{tikzcd}
{||\mathbf{x}||} \\
\mathbf{x} \arrow[u]
\end{tikzcd}$$

$\mathrm{f}(\mathbf{u}, \mathbf{v}) = \mathbf{u}^{\mathrm{T}}\mathbf{v}$

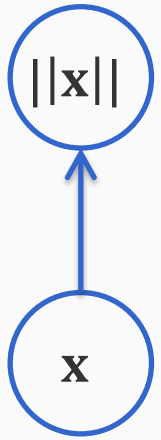
$$\begin{tikzcd}
& \mathbf{x}^{\mathrm{T}}\mathbf{y} & \\
\mathbf{x} \arrow[ur] & & \mathbf{y} \arrow[ul]
\end{tikzcd}$$

Each node knows how to compute two things:

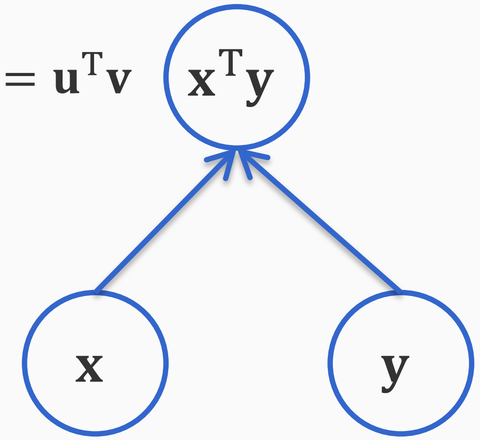
1. Its own value using its inputs
   - In these examples, the nodes on top compute $||\mathbf{x}||$ and $\mathbf{x}^{\mathrm{T}}\mathbf{y}$
2. The value of its partial derivative with respect to each input
   - Left graph: the node on top knows to compute $\frac{\partial f}{\partial \mathbf{u}}$
   -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

$f(\mathbf{u}) = ||\mathbf{u}||$



$f(\mathbf{u}, \mathbf{v}) = \mathbf{u}^\mathrm{T}\mathbf{v}$



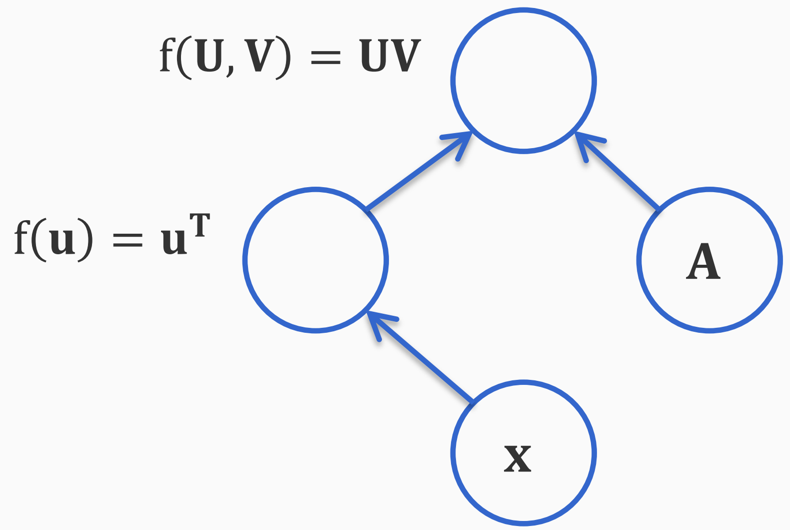
The functions expressed could be

- 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

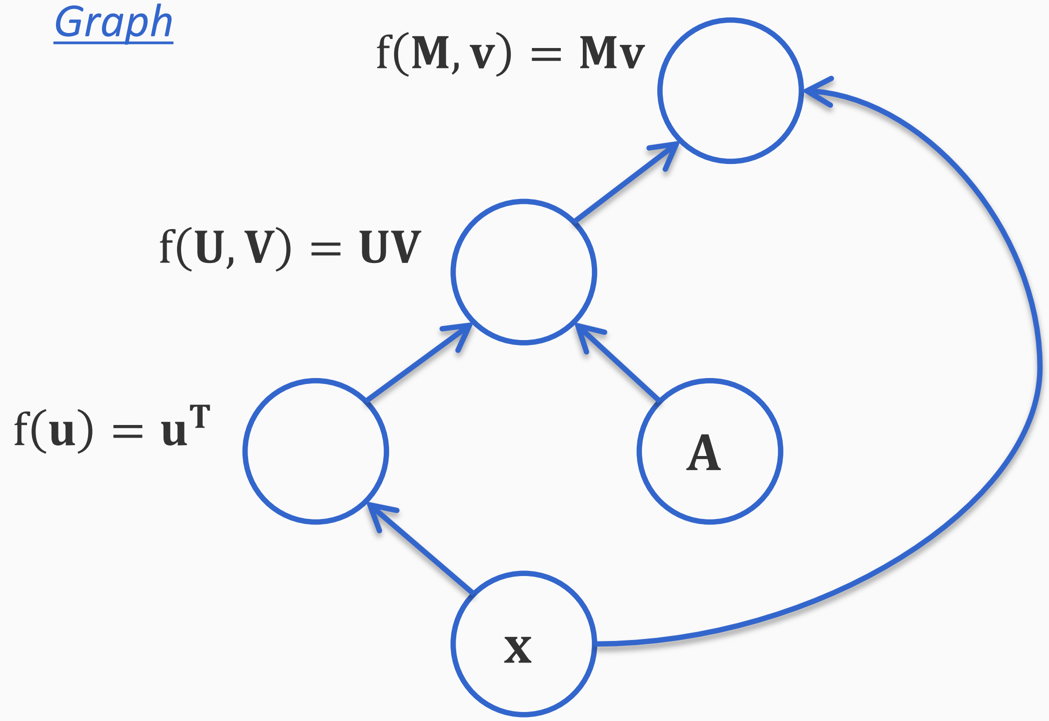


$f(\mathbf{U},\mathbf{V}) = \mathbf{U}\mathbf{V}$
$f(\mathbf{u}) = \mathbf{u}^{\mathrm{T}}$
$\mathbf{A}$
$\mathbf{x}$

Let’s see some functions as graphs

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

Graph

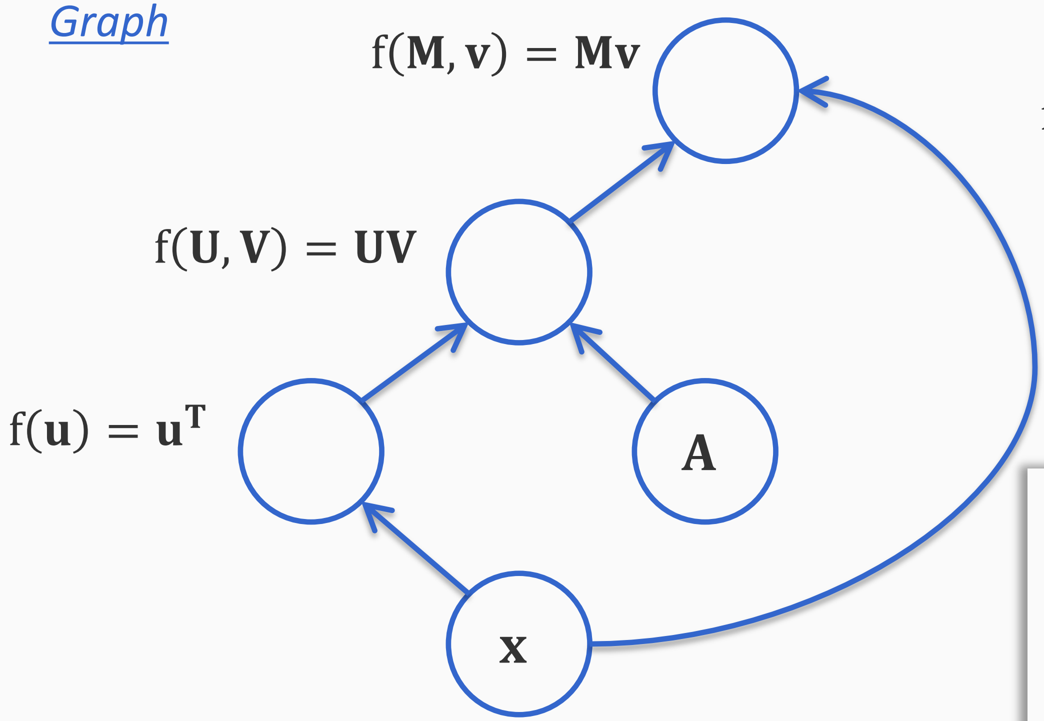


$f(\mathbf{M}, \mathbf{v}) = \mathbf{M}\mathbf{v}$
$f(\mathbf{U}, \mathbf{V}) = \mathbf{U}\mathbf{V}$
$f(\mathbf{u}) = \mathbf{u}^{\mathrm{T}}$
$\mathbf{A}$
$\mathbf{x}$

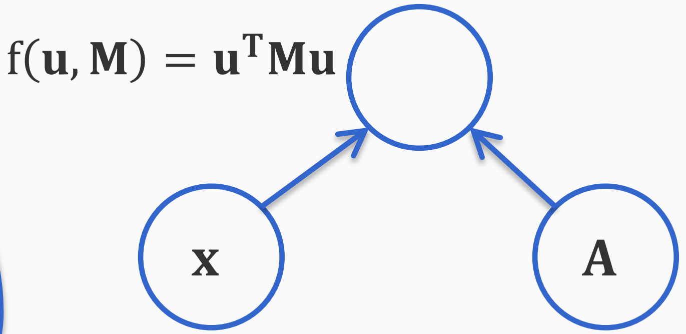
Let’s see some functions as graphs

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

Graph



$f(\mathbf{M},\mathbf{v}) = \mathbf{M}\mathbf{v}$
$f(\mathbf{U},\mathbf{V}) = \mathbf{U}\mathbf{V}$
$f(\mathbf{u}) = \mathbf{u}^{\mathrm{T}}$
$\mathbf{A}$
$\mathbf{x}$



$f(\mathbf{u},\mathbf{M}) = \mathbf{u}^{\mathrm{T}}\mathbf{M}\mathbf{u}$
$\mathbf{x}$
$\mathbf{A}$

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

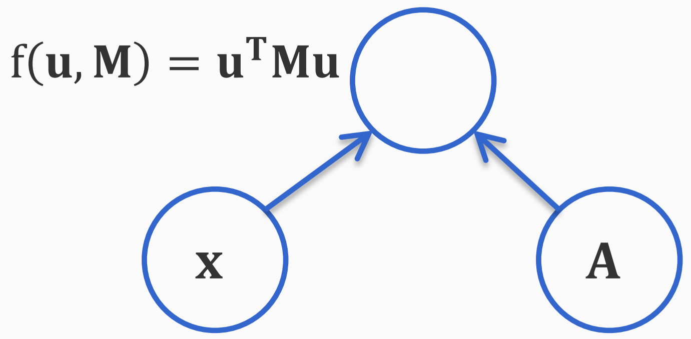
Computation graphs are not necessarily unique for a function

Let’s see some functions as graphs

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

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

Graph

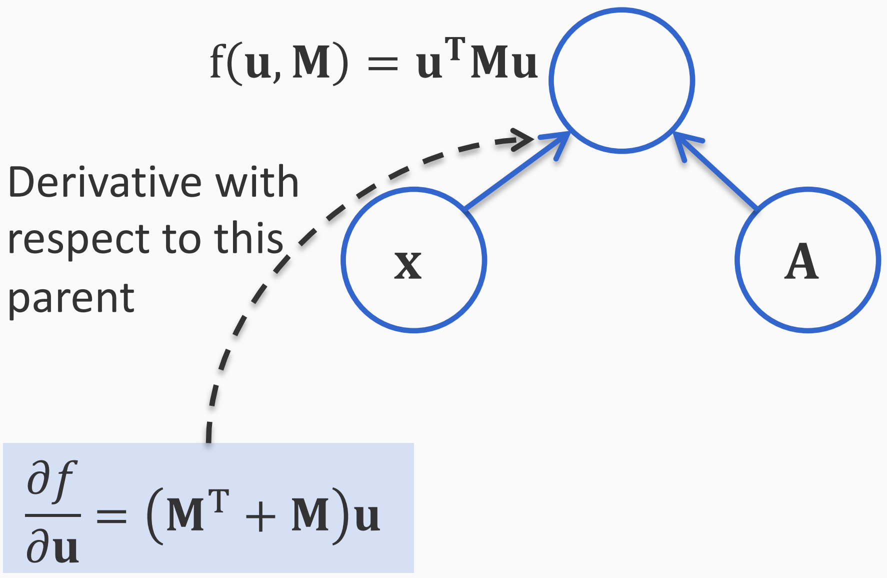

$f(\mathbf{u}, \mathbf{M}) = \mathbf{u}^{\mathrm{T}}\mathbf{M}\mathbf{u}$
$\mathbf{x}$
$\mathbf{A}$

Let’s see some functions as graphs

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

Graph

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



$\mathrm{f}(\mathbf{u}, \mathbf{M}) = \mathbf{u}^{\mathrm{T}}\mathbf{M}\mathbf{u}$
Derivative with respect to this parent
$\mathbf{x}$
$\mathbf{A}$

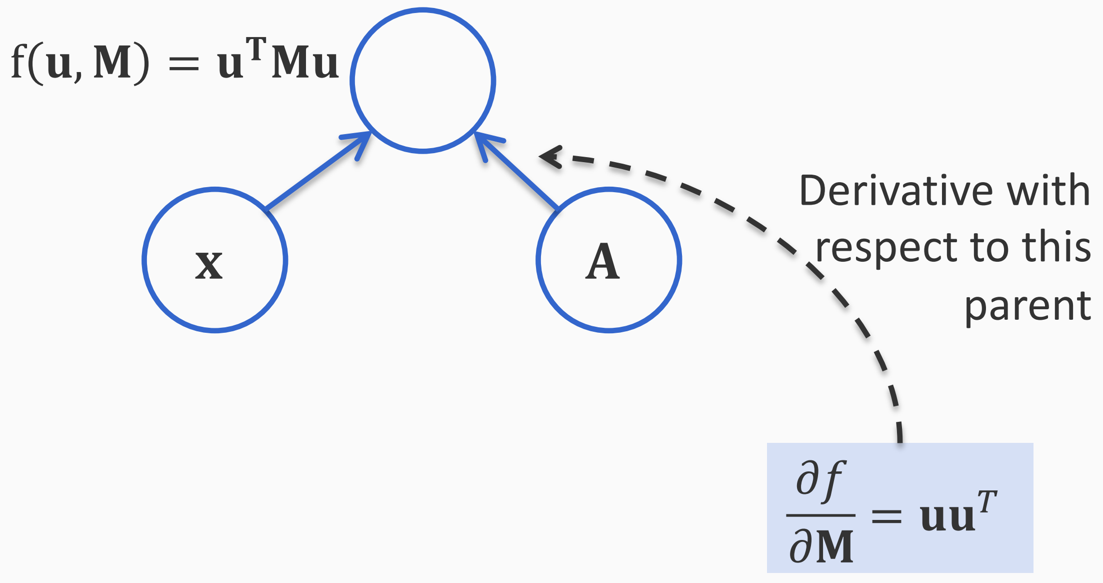
$$\frac{\partial f}{\partial \mathbf{u}} = \left(\mathbf{M}^{\mathrm{T}} + \mathbf{M}\right)\mathbf{u}$$

Let’s see some functions as graphs

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

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

Graph



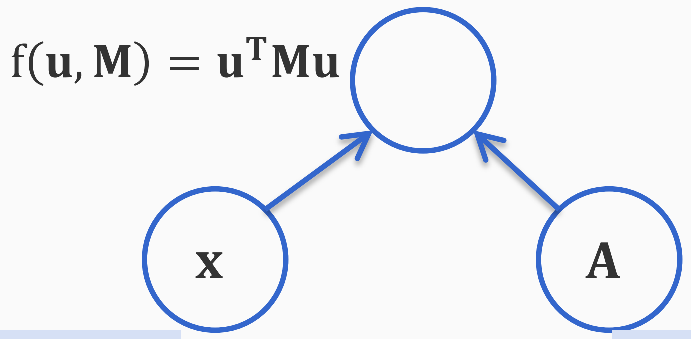
$f(\mathbf{u}, \mathbf{M}) = \mathbf{u}^{\mathrm{T}}\mathbf{M}\mathbf{u}$
$\mathbf{x}$
$\mathbf{A}$
Derivative with respect to this parent
$\frac{\partial f}{\partial \mathbf{M}} = \mathbf{u}\mathbf{u}^T$

Let’s see some functions as graphs

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

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

Graph



$f(\mathbf{u}, \mathbf{M}) = \mathbf{u}^{\mathrm{T}}\mathbf{M}\mathbf{u}$
$\mathbf{x}$
$\mathbf{A}$

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

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

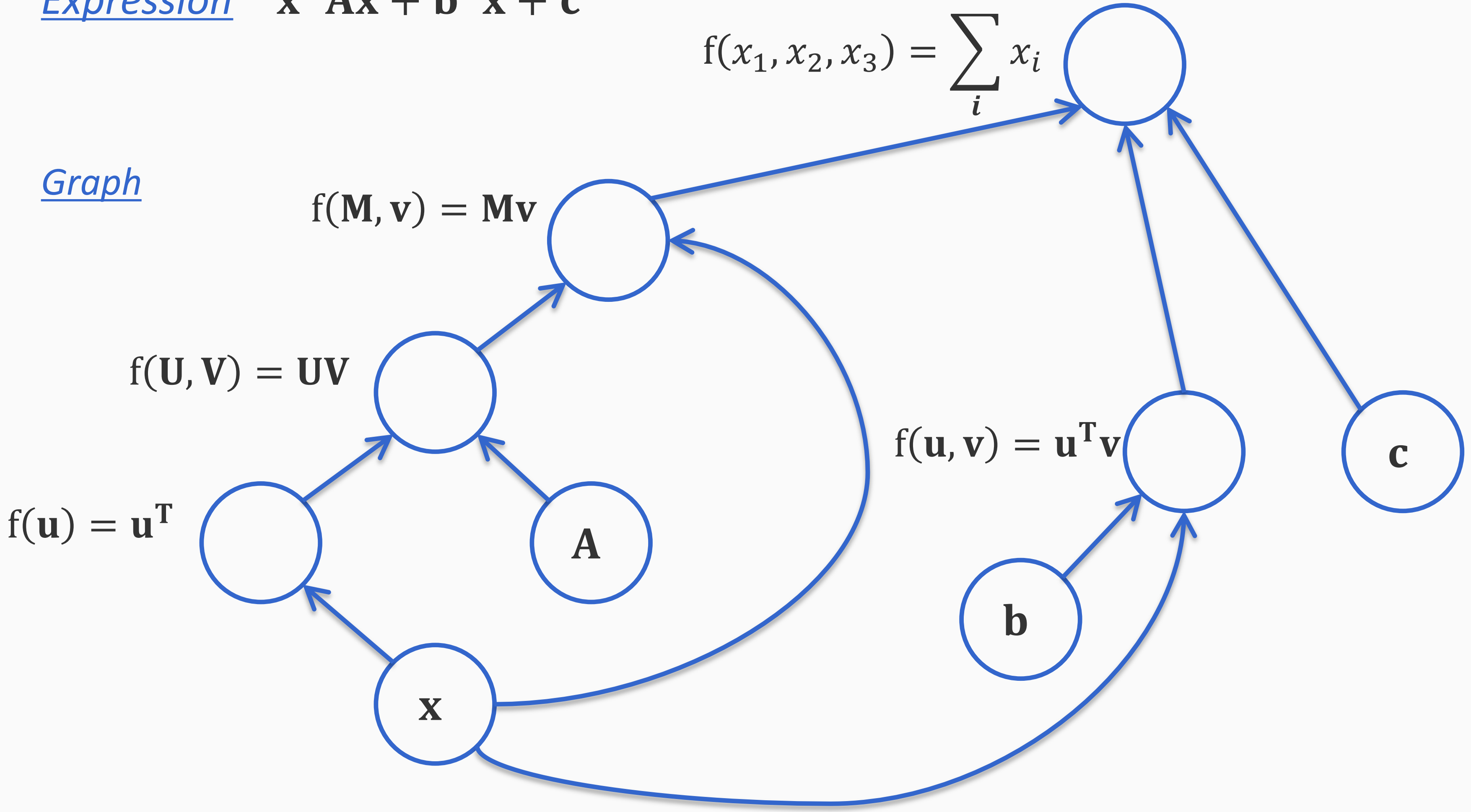
Remember: The nodes also know how to compute derivatives with respect to each parent

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

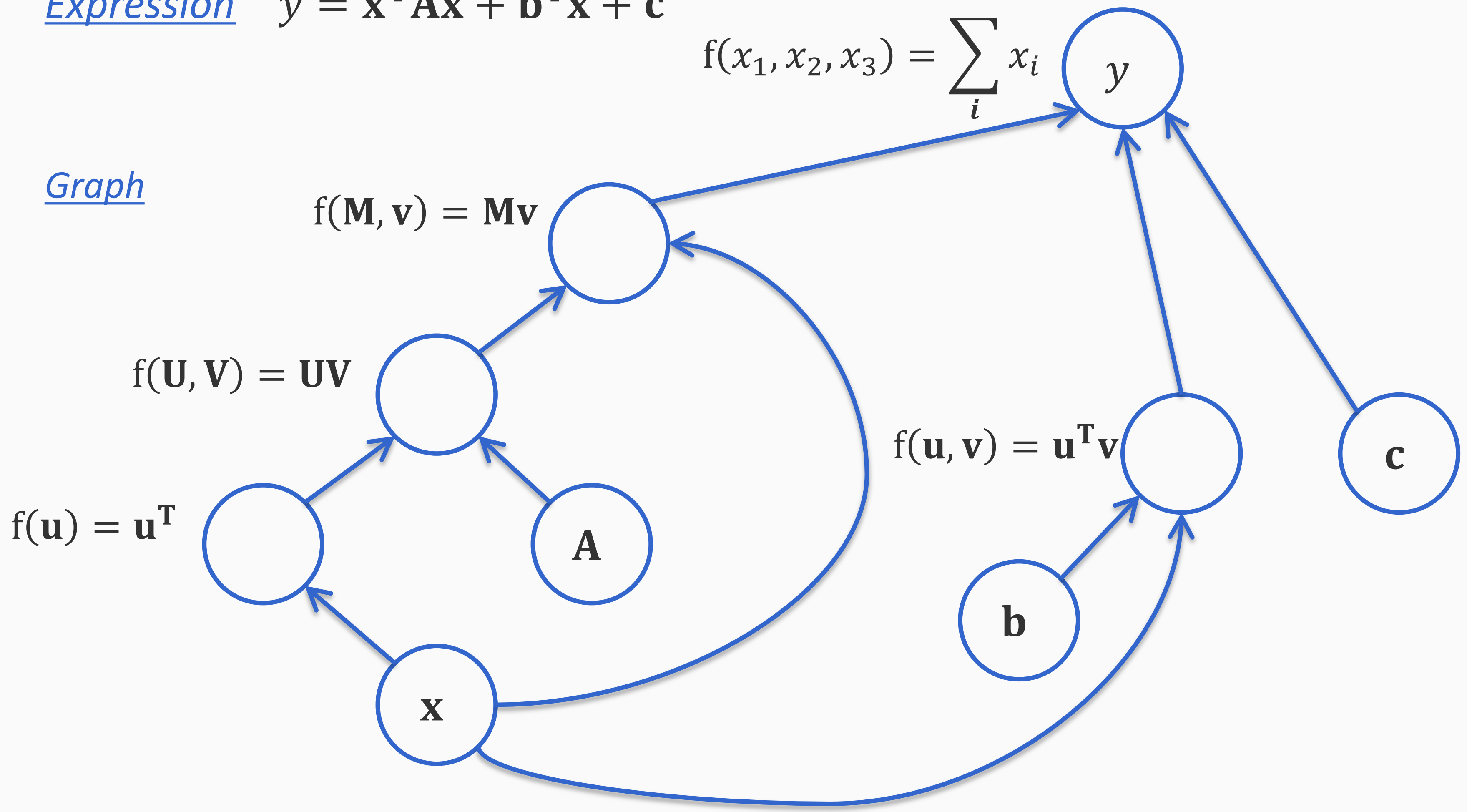


$f(x_1, x_2, x_3) = \sum_i x_i$
$f(\mathbf{M}, \mathbf{v}) = \mathbf{M}\mathbf{v}$
$f(\mathbf{U}, \mathbf{V}) = \mathbf{U}\mathbf{V}$
$f(\mathbf{u}) = \mathbf{u}^{\mathrm{T}}$
$f(\mathbf{u}, \mathbf{v}) = \mathbf{u}^{\mathrm{T}}\mathbf{v}$
$\mathbf{A}$
$\mathbf{x}$
$\mathbf{b}$
$\mathbf{c}$

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



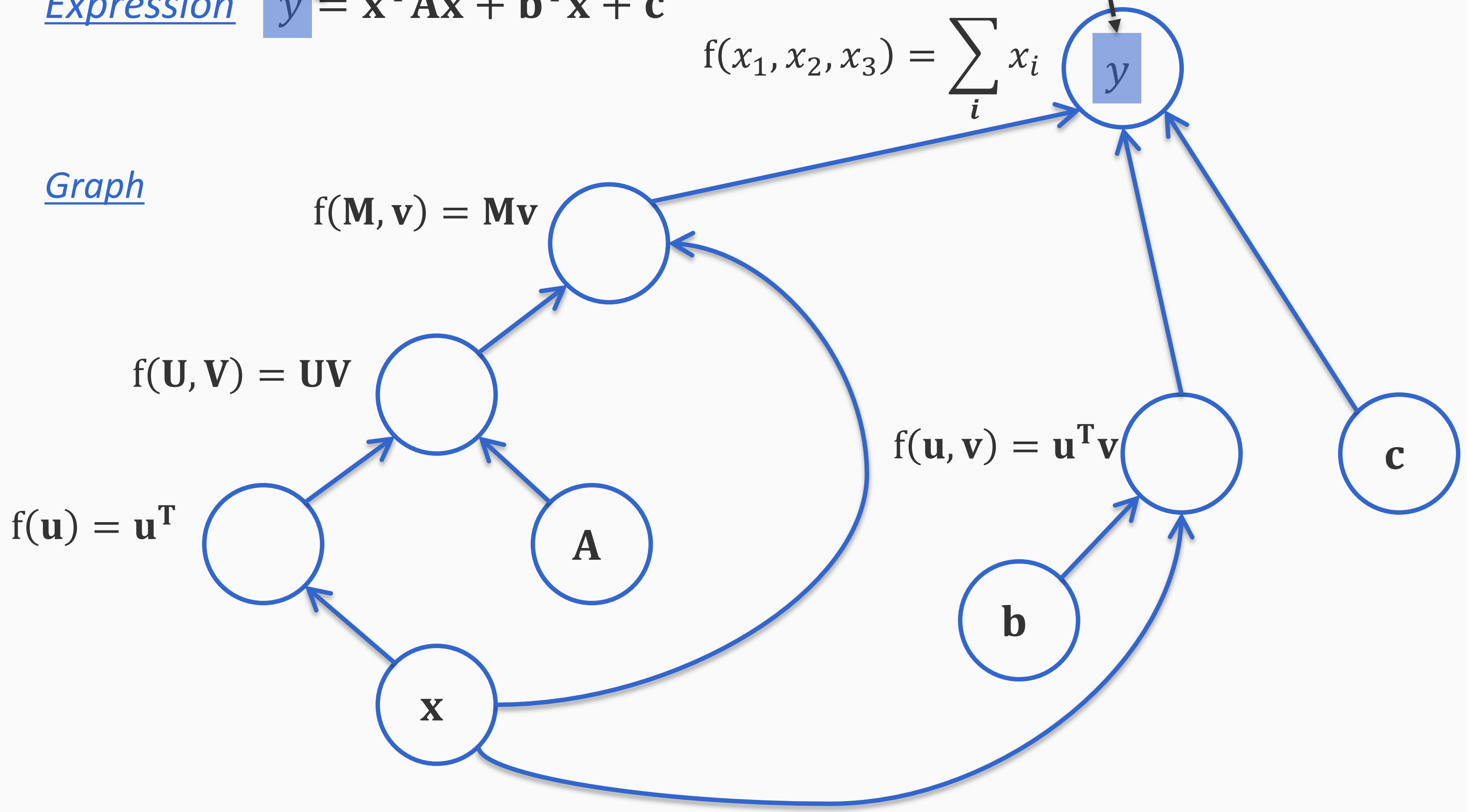
$\mathrm{f}(x_1, x_2, x_3) = \sum_i x_i$
y
$\mathrm{f}(\mathbf{M}, \mathbf{v}) = \mathbf{M}\mathbf{v}$
$\mathrm{f}(\mathbf{U}, \mathbf{V}) = \mathbf{U}\mathbf{V}$
$\mathrm{f}(\mathbf{u}, \mathbf{v}) = \mathbf{u}^{\mathrm{T}}\mathbf{v}$
$\mathbf{c}$
$\mathrm{f}(\mathbf{u}) = \mathbf{u}^{\mathrm{T}}$
$\mathbf{A}$
$\mathbf{b}$
$\mathbf{x}$

Let’s see some functions as graphs

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

We can name variables by labeling nodes

Graph



$f(x_1, x_2, x_3) = \sum_i x_i$
y
$f(\mathbf{M}, \mathbf{v}) = \mathbf{M}\mathbf{v}$
$f(\mathbf{U}, \mathbf{V}) = \mathbf{U}\mathbf{V}$
$f(\mathbf{u}, \mathbf{v}) = \mathbf{u}^{\mathrm{T}}\mathbf{v}$
$\mathbf{c}$
$f(\mathbf{u}) = \mathbf{u}^{\mathrm{T}}$
$\mathbf{A}$
$\mathbf{b}$
$\mathbf{x}$

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.

Libraries like PyTorch and TensorFlow help construct computation graphs

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

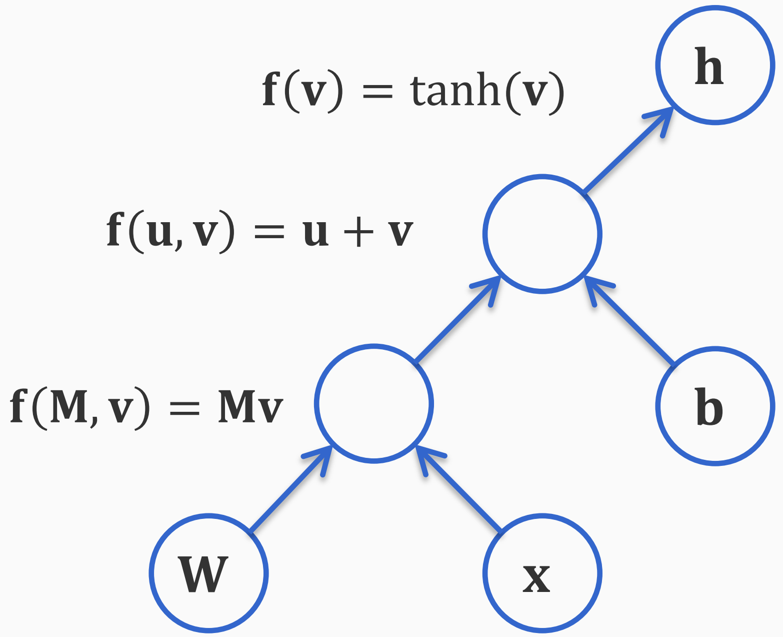
4. They allow efficient gradient computation.

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

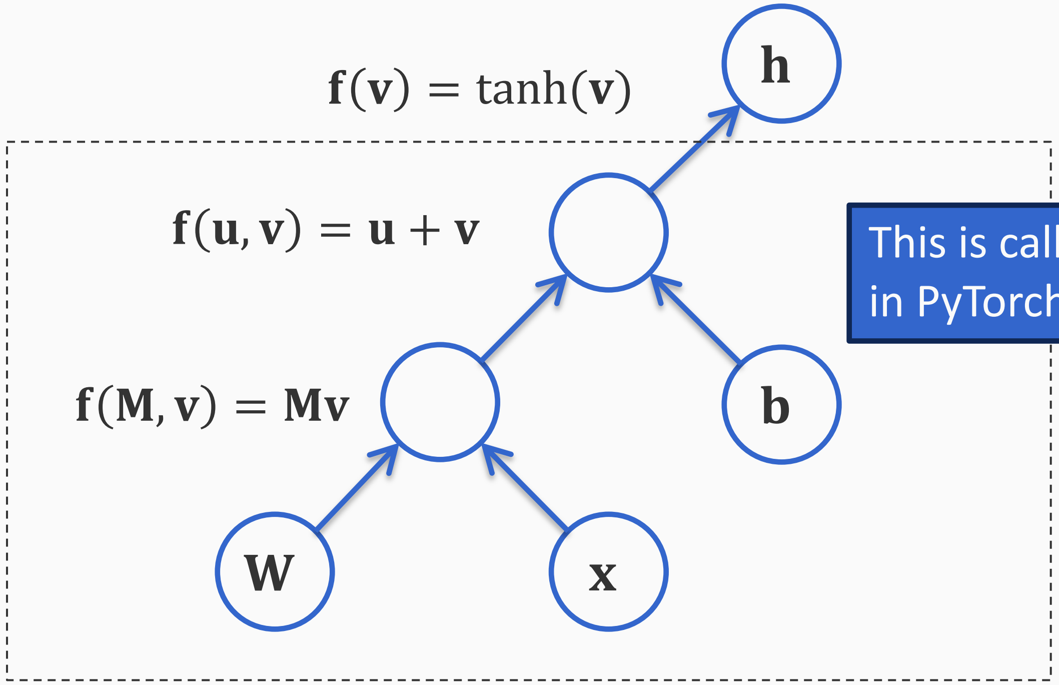
$$\mathbf{h} = \tanh(\mathbf{Wx} + \mathbf{b})$$
$$y = \mathbf{Vh} + \mathbf{a}$$



$\mathbf{f}(\mathbf{v}) = \tanh(\mathbf{v})$
$\mathbf{f}(\mathbf{u}, \mathbf{v}) = \mathbf{u} + \mathbf{v}$
$\mathbf{f}(\mathbf{M}, \mathbf{v}) = \mathbf{Mv}$
$\mathbf{h}$
$\mathbf{b}$
$\mathbf{W}$
$\mathbf{x}$

An example two layer neural network

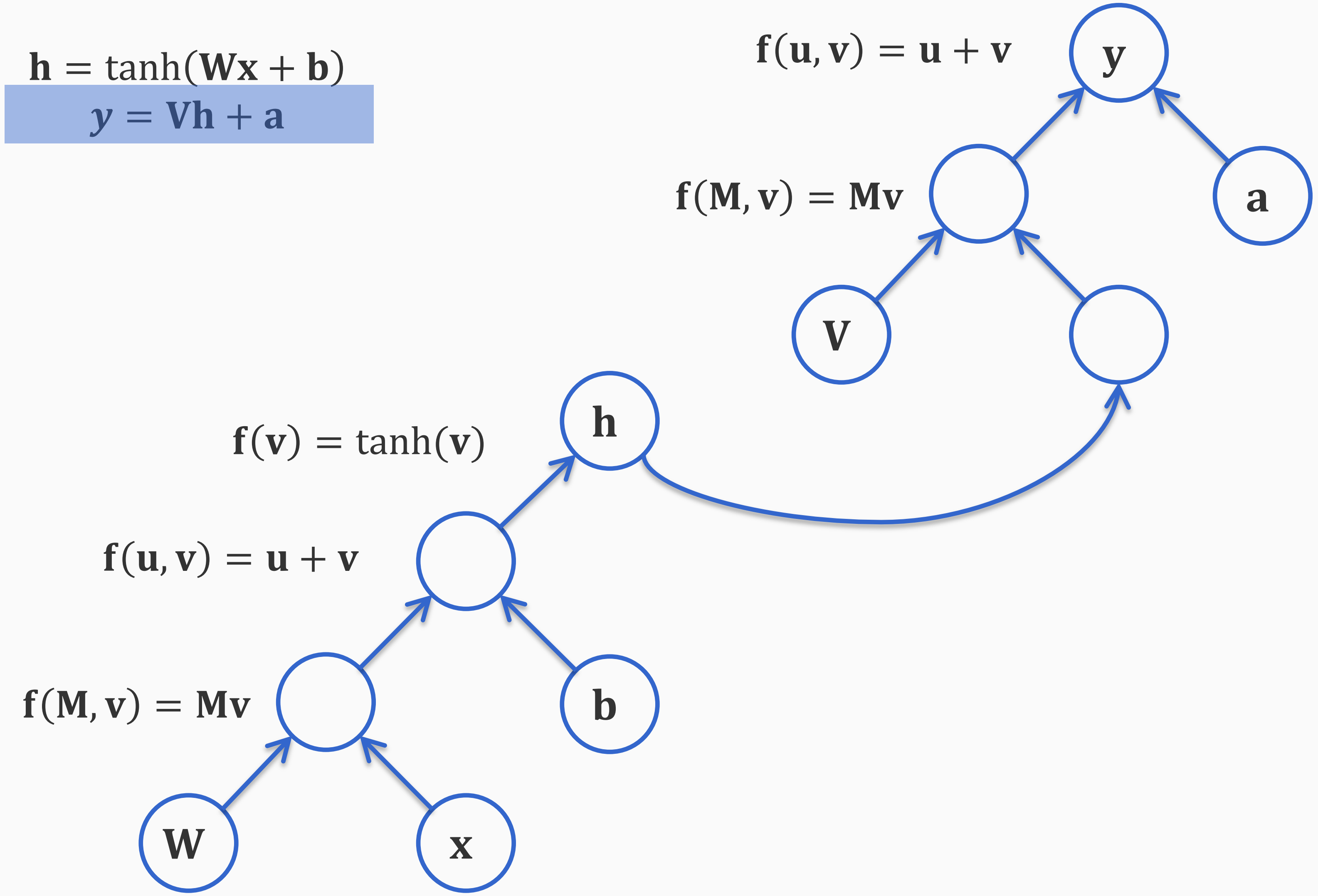
$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$y = \mathbf{V}\mathbf{h} + \mathbf{a}$$


$\mathbf{f}(\mathbf{v}) = \tanh(\mathbf{v})$
$\mathbf{f}(\mathbf{u}, \mathbf{v}) = \mathbf{u} + \mathbf{v}$
$\mathbf{f}(\mathbf{M}, \mathbf{v}) = \mathbf{M}\mathbf{v}$
h
b
W
x

This is called `nn.Linear` in PyTorch

An example two layer neural network

$$\mathbf{h} = \tanh(\mathbf{Wx} + \mathbf{b})$$
$$y = \mathbf{Vh} + \mathbf{a}$$



Exercises

Write the following functions as computation graphs:

- $f(x) = x^3 - \log(x)$
- $f(x) = \frac{1}{1+\exp(-x)}$
- $f(\mathrm{w}, \mathrm{x}, y) = \max(0, 1 - y\mathrm{w}^{\mathrm{T}}\mathrm{x})$
- $\min_{\mathrm{w}} \frac{1}{2}\mathrm{w}^T\mathrm{w} + C\sum_i \max(0, 1 - y_i\mathrm{w}^{\mathrm{T}}x_i)$