ANTONIS ANASTASOPOULOS

CS499 INTRODUCTION TO NLP

PRELIMINARIES

https://cs.gmu.edu/~antonis/course/cs499-spring21/

STRUCTURE OF THIS LECTURE

1. Probability Refresher
2. Working with text
3. Regular Expressions
4. Neural Nets Primer

PROBABILITIES

RANDOM VARIABLES

A random variable is a variable with a different random value in each “experiment”

Random Variable: X
P(X) is the distribution of X.

If $x \in X$, we write $P(X = x)$ for the probability that X has value x

$\Sigma_{x \in X} P(X = x) = 1$

If P(W) is the distribution of English words, we might have:
$P(W = \text{the}) = 0.1$
$P(W = \text{syzygy}) = 10^{-10}, \ldots$

JOINT AND MARGINAL PROBABILITIES

Random Variables W (words) and S (speaker)

Joint distribution $P(S, W)$ such that:

$\Sigma_{s,w} P(S = s, W = w) = 1$

$P(S = Trump, W = bigly) = 0.2$

$P(S = Trump, W = huge) = 0.4$

$P(S = Biden, W = people) = 0.3$

$P(S = Biden, W = fellas) = 0.1$

JOINT AND MARGINAL PROBABILITIES

Marginal distributions

$P(S = s) = \Sigma_w \ P(S = s, W = w)$

$P(W = w) = \Sigma_s \ P(S = s, W = w)$

For our made up numbers:

$P(S = Trump) = 0.2 + 0.4 = 0.6$

$P(S = Biden) = 0.3 + 0.1 = 0.4$

CONDITIONAL DISTRIBUTIONS

$$P(s \mid w) = \frac{P(s, w)}{P(w)}$$

Note that $\Sigma_s \; P(s \mid w) = 1$.

You know this already, but do not confuse $p(w \mid s)$ and $p(s \mid w)$:

For our made up numbers:

$$P(Trump \mid bigly) = \frac{0.2}{0.2} = 1$$

$$P(bigly \mid Trump) = \frac{0.2}{0.6} \approx 0.33$$

EXPECTED VALUES

$c_e(w)$: number of occurrences of letter e in a word

The expectation of $c_e(w)$ is

$$E[c_e] = \Sigma_w \, P(W = w)c_e(w)$$

For our made up numbers:

$$E[c_e] = 0.2 \cdot 0 + 0.4 \cdot 1 + 0.3 \cdot 2 + 0.1 \cdot 1 = 1.1$$

LOGARITHMS

Some identities that will be useful

$$\log \exp x = x \qquad \exp \log x = x$$

$$\log xy = \log x + \log y \qquad \exp(x+y) = \exp x \exp y$$

$$\log \prod_i x_i = \sum_i \log x_i \qquad \exp \sum_i x_i = \prod_i \exp x_i$$

$$\log x^n = n \log x \qquad \exp nx = (\exp x)^n$$

$$\log 1 = 0 \qquad \exp 0 = 1$$

LOGARITHMS

Used to simplify expressions like a product of probabilities:

$$p(x_1, \ldots, x_n) = \Pi_i p(x_i)$$

Take the log of everything, and now you have a sum:

$$\log p(x_1, \ldots, x_n) = \Sigma_i \log p(x_i)$$

LOGARITHMS

Used to simplify expressions like a product of probabilities:

$$p(x_1, \ldots, x_n) = \Pi_i p(x_i)$$

Take the log of everything, and now you have a sum:

$$\log p(x_1, \ldots, x_n) = \Sigma_i \log p(x_i)$$

For two probabilities p, q comparing $\log p$ and $\log q$ is equivalent.

Instead of multiplying two probabilities $p \cdot q$ we can just add $\log p + \log q$

SOFTMAX

Let $x = [x_1, x_2, \ldots, x_n]$ be a vector of real numbers

We define $[\text{softmax } \mathbf{x}]_i = \dfrac{\exp x_i}{\Sigma_{i'=1}^{n} \exp x_{i'}}$.

WORKING WITH TEXT

SOME TERMINOLOGY

A **word** is an ill-defined concept:

do — do not — don’t

Lebensversicherungsgesellschaftsangestellter (life insurance company employee)

莎拉波娃现在居住在美国东南部的佛罗里达。 (Sharapova now lives in Us southeastern Florida)

Type: a *class* of tokens that use the same character sequence

Token: an individual occurrence of a type in speech or writing

Vocabulary: the set of types

https://en.wikipedia.org/wiki/Type%E2%80%93token_distinction

SOME TERMINOLOGY

A rose is a rose is a rose.

#Types: 4

Vocabulary: {a, rose, is, .}

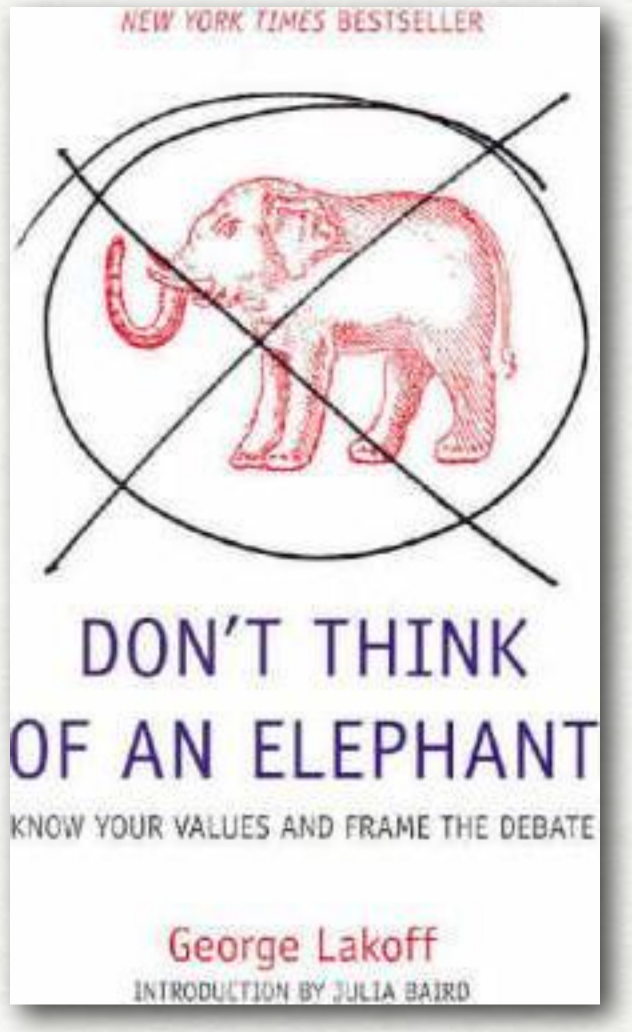
#Tokens: 9

TEXT NORMALIZATION

"Don't think of an elephant!," says George.

Elephants are not something you should be thinking, according to Lakoff.

Dr. Lakoff asks that you do not think of an elephant.

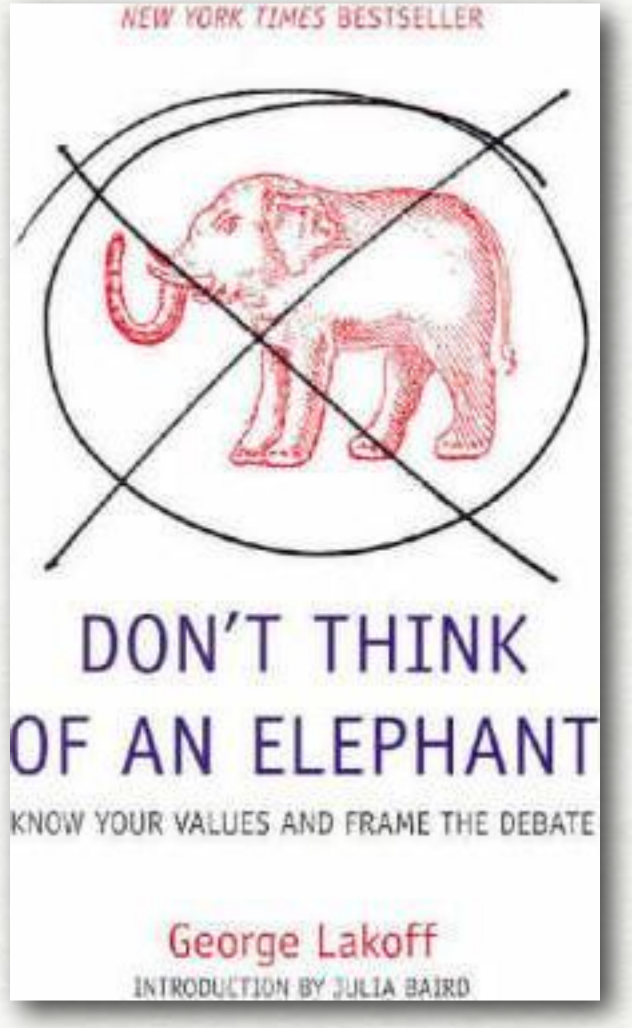


SEGMENTATION

" Don't think of an elephant ! , " says George .

Elephants are not something you should be thinking , according to Lakoff .

Dr. Lakoff asks that you do not think of an elephant .

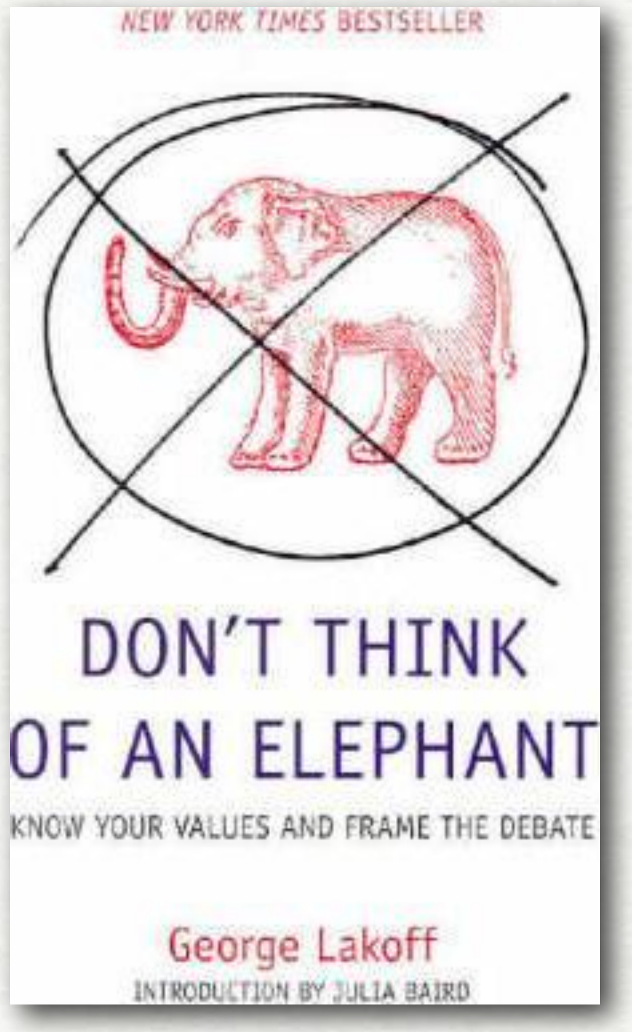


TRUE CASING

" do n't think of an elephant ! , " says George .

elephants are not something you should be thinking , according to Lakoff .

dr. Lakoff asks that you do not think of an elephant .



Tools:

- NLTK (https://www.nltk.org/)
- spacy (https://spacy.io/)
- Moses tools (http://www.statmt.org/moses/?n=Moses.SupportTools)

MORE READINGS

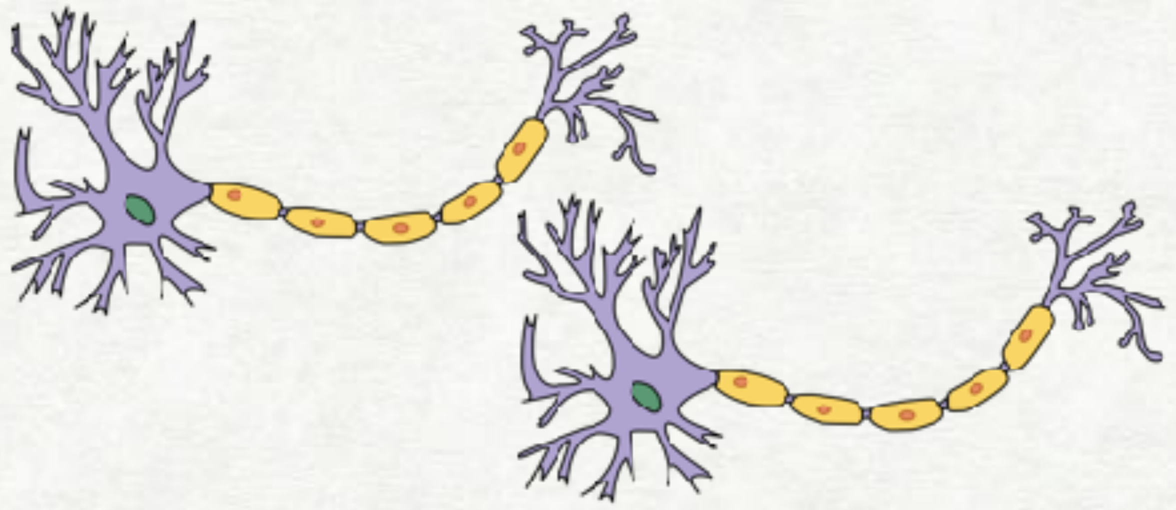
RegExes: https://web.stanford.edu/~jurafsky/slp3/2.pdf

Working with text: https://web.stanford.edu/~jurafsky/slp3/slides/2_TextProc_Jan_06_2021.pdf

NEURAL NETS

"NEURAL" NETS

Original Motivation: The Brain



Current Implementation: Computation Graphs

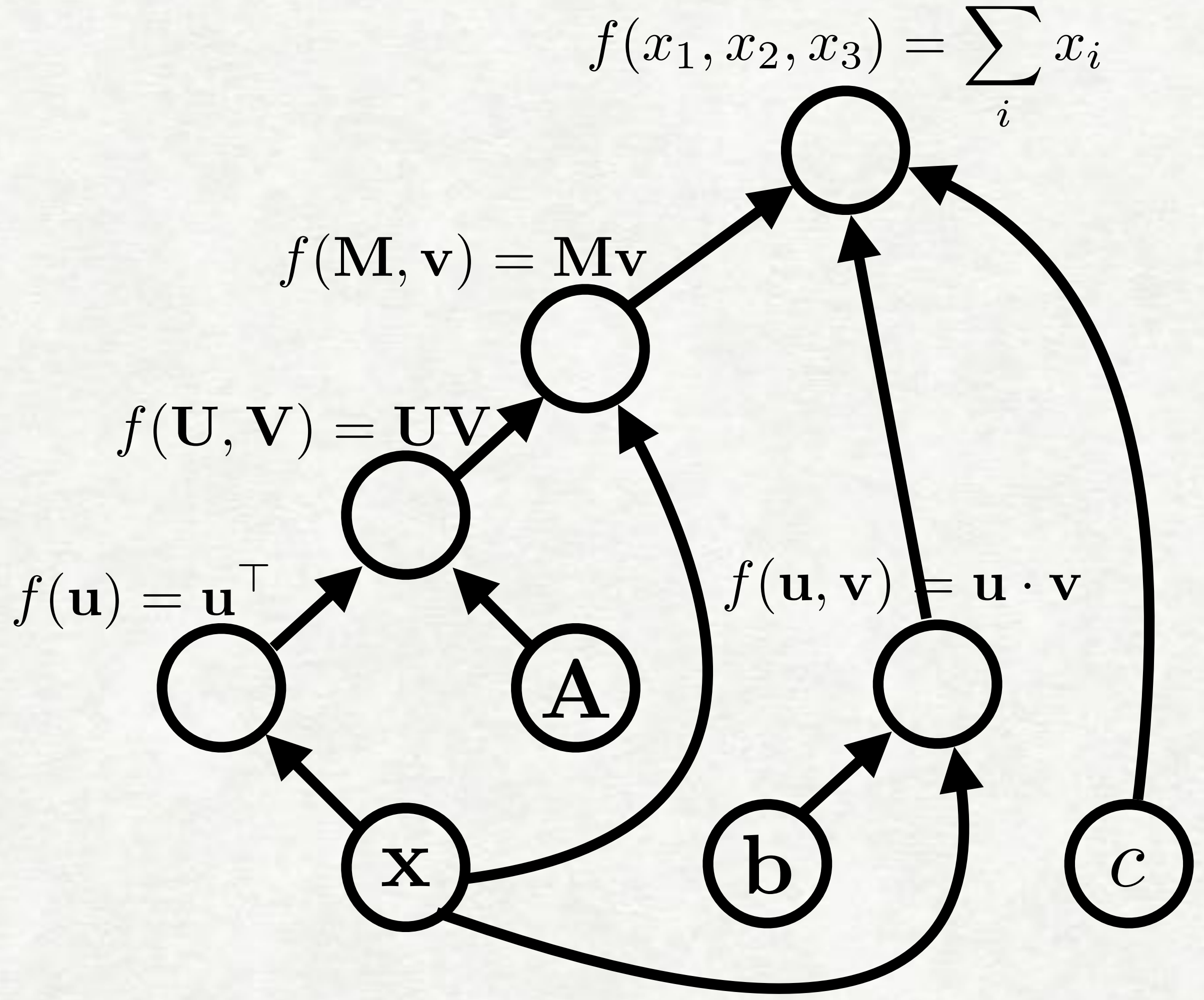


Image credit: Wikipedia

COMPOSITE FUNCTIONS

We will build computation graphs using an "ordered series of equations".

Each equation is only a function of the preceding equations

$$f(x, y, z) = z + \sin(x^2 + y \times \exp(z))$$

We can represent the above equation using intermediate variables:

$$a = x^2$$
$$b = exp(z)$$
$$c = y \times b$$
$$d = a + c$$
$$e = \sin(d)$$
$$f = e + z$$

AUTODIFF

Main idea behind AD: as long as we have access to the derivatives of a set of primitives, then we can stick these together to get the derivative of any composite function

Saving the values of intermediate variables (dynamic programming!) allows for low computational complexity (exponential —> linear).

GENERAL AUTODIFF FRAMEWORK

Primitives	Their Derivatives
$f(y,z) = y + z$	$\frac{\partial}{\partial x} f(y,z) = \frac{\partial y}{\partial x} + \frac{\partial z}{\partial x}$
$f(y,z) = y \times z$	$\frac{\partial}{\partial x} f(y,z) = y\frac{\partial z}{\partial x} + z\frac{\partial y}{\partial x}$
$f(y,z) = y^3$	$\frac{\partial}{\partial x} f(y,z) = 3y^2\frac{\partial y}{\partial x}$
$f(y,z) = \log(y)$	$\frac{\partial}{\partial x} f(y,z) = \frac{1}{y}\frac{\partial y}{\partial x}$

1. All edges (hyperedges) are made of primitives
2. Perform the forward pass, to compute the functions' value
3. Run back-propagation using the stored forward values, using the derivatives

EXAMPLE GRADIENT CALCULATION

$$f(x, y, z) = z + \sin(x^2 + y \times \exp(z))$$

forward

$a = x^2$

$b = exp(z)$

$c = y \times b$

$d = a + c$

$e = \sin(d)$

$f = e + z$

backward

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial b}\frac{\partial b}{\partial z} + 1 = \frac{\partial f}{\partial b}\exp(z) + 1$$

$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c}\frac{\partial c}{\partial b} = \frac{\partial f}{\partial c}y$$

$$\frac{\partial f}{\partial c} = \frac{\partial f}{\partial d}\frac{\partial d}{\partial c} = \frac{\partial f}{\partial d}1 \qquad \frac{\partial f}{\partial a} = \frac{\partial f}{\partial d}\frac{\partial d}{\partial a} = \frac{\partial f}{\partial d}1$$

$$\frac{\partial f}{\partial d} = \frac{\partial f}{\partial e}\frac{\partial e}{\partial d} = \frac{\partial f}{\partial e}\cos(d)$$

$$\frac{\partial f}{\partial e} = 1$$

We can easily write the derivatives of individual terms in the graph.

Given all these, we can work backwards to compute the derivative of $f(x, y, z)$ with respect to each variable.

COMPUTATION GRAPHS

expression:

$\mathbf{x}$

graph:

A **node** is a {tensor, matrix, vector, scalar} value

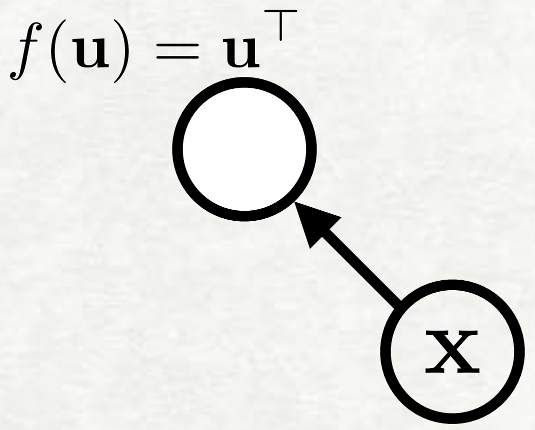
$\mathbf{x}$

COMPUTATION GRAPHS

expression:

$$\mathbf{x}^\top$$

graph:



A **node** with an incoming **edge** is a **function** of that edge's tail node.

A **node** knows how to compute its value and the *value of its derivative w.r.t each argument (edge) times a derivative of an arbitrary input* $\frac{\partial \mathcal{F}}{\partial f(\mathbf{u})}$.

$$\frac{\partial f(\mathbf{u})}{\partial \mathbf{u}} \frac{\partial \mathcal{F}}{\partial f(\mathbf{u})} = \left(\frac{\partial \mathcal{F}}{\partial f(\mathbf{u})}\right)^\top$$

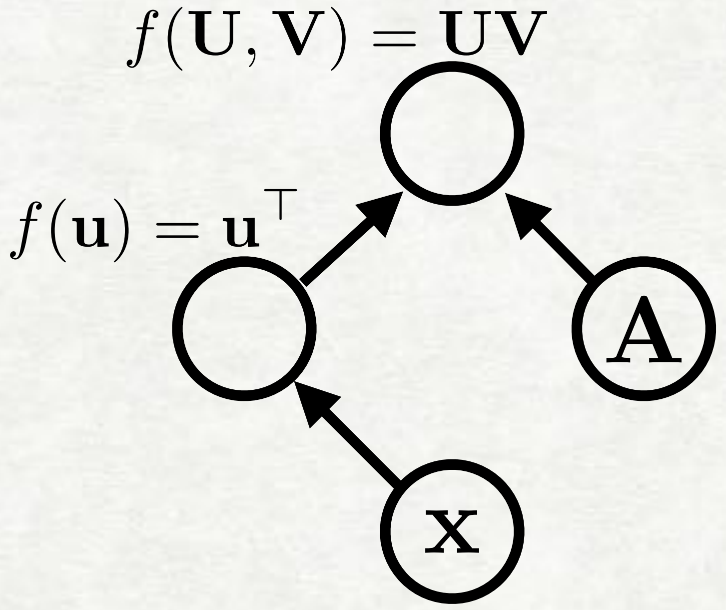
An **edge** represents a function argument (and also an data dependency). They are just pointers to nodes.

COMPUTATION GRAPHS

expression:

$$\mathbf{x}^\top \mathbf{A}$$

graph:



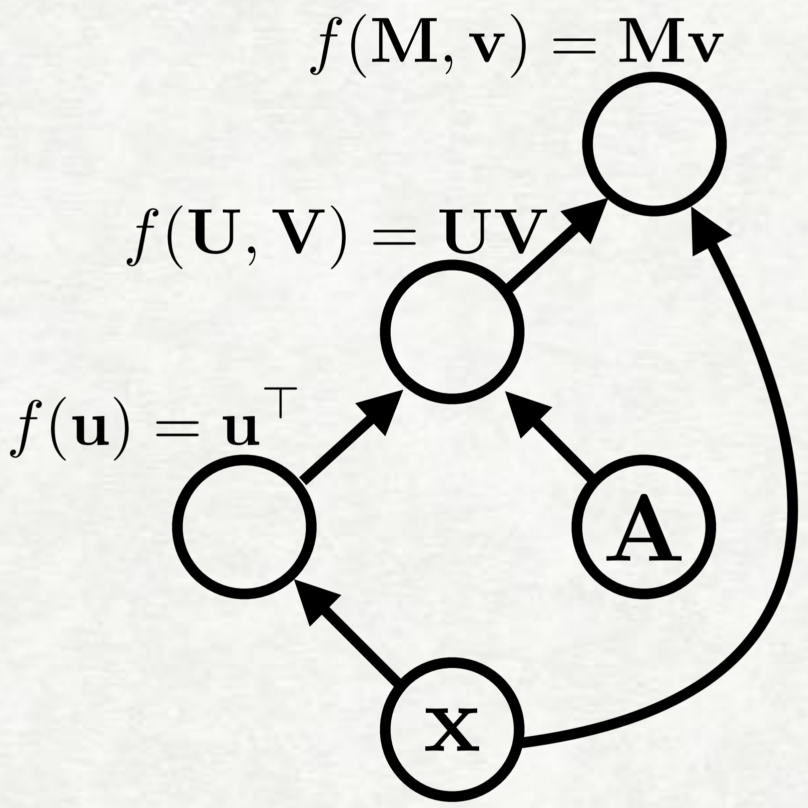
Functions can be nullary, unary, binary, … *n*-ary. Often they are unary or binary.

COMPUTATION GRAPHS

expression:

$$\mathbf{x}^\top \mathbf{A} \mathbf{x}$$

graph:



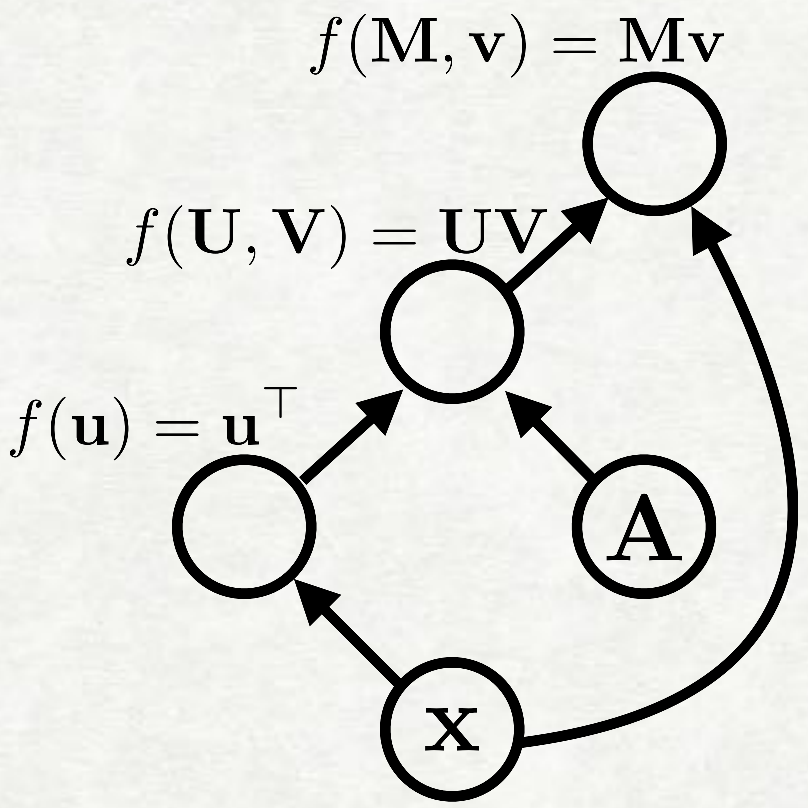
Computation graphs are directed and acyclic.

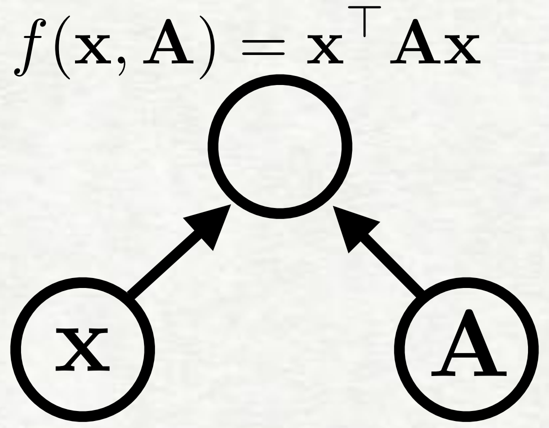
COMPUTATION GRAPHS

expression:

$$\mathbf{x}^\top \mathbf{A}\mathbf{x}$$

graph:





$$\frac{\partial f(\mathbf{x}, \mathbf{A})}{\partial \mathbf{x}} = (\mathbf{A}^\top + \mathbf{A})\mathbf{x}$$

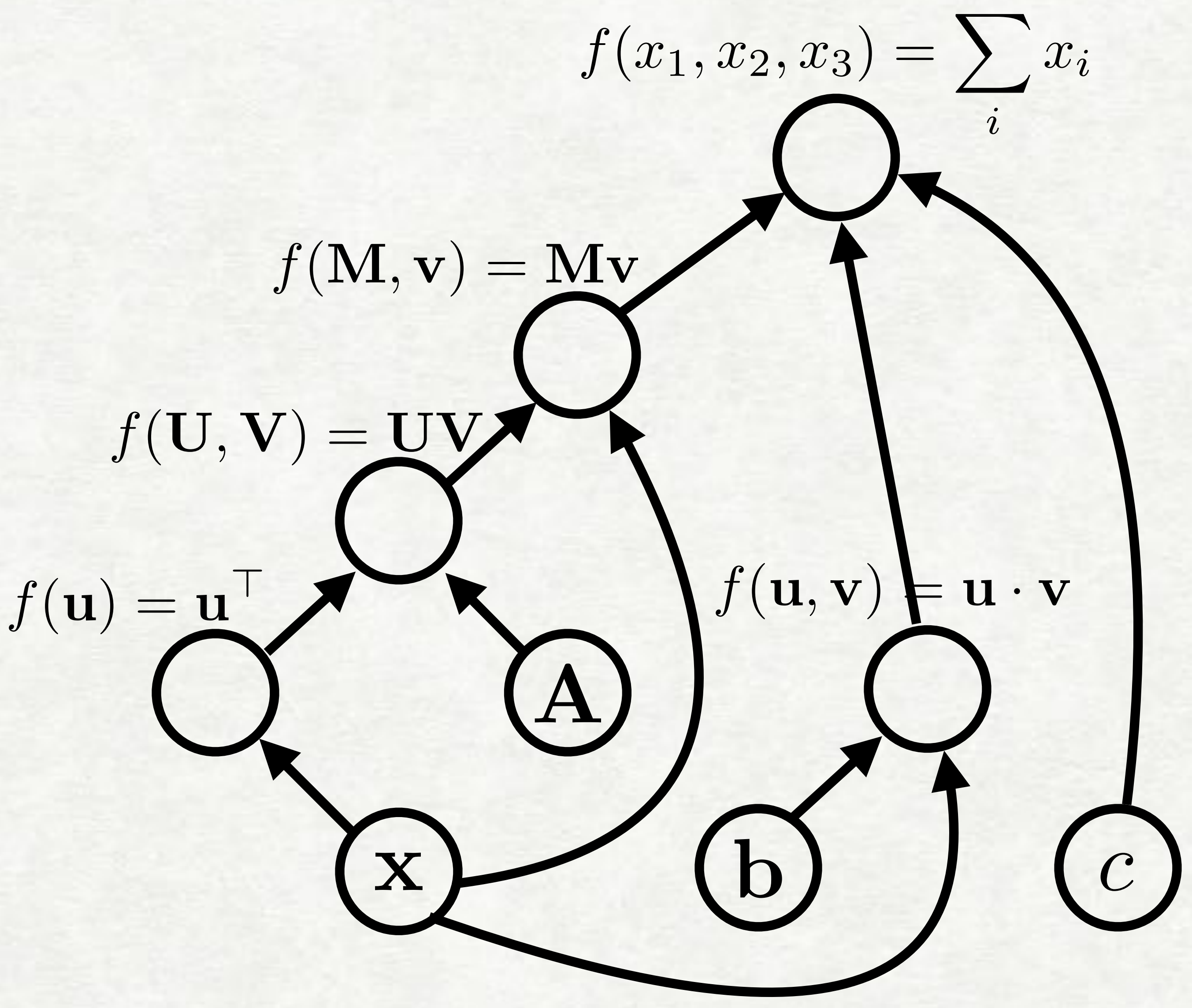
$$\frac{\partial f(\mathbf{x}, \mathbf{A})}{\partial \mathbf{A}} = \mathbf{x}\mathbf{x}^\top$$

COMPUTATION GRAPHS

expression:

$$\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c$$

graph:

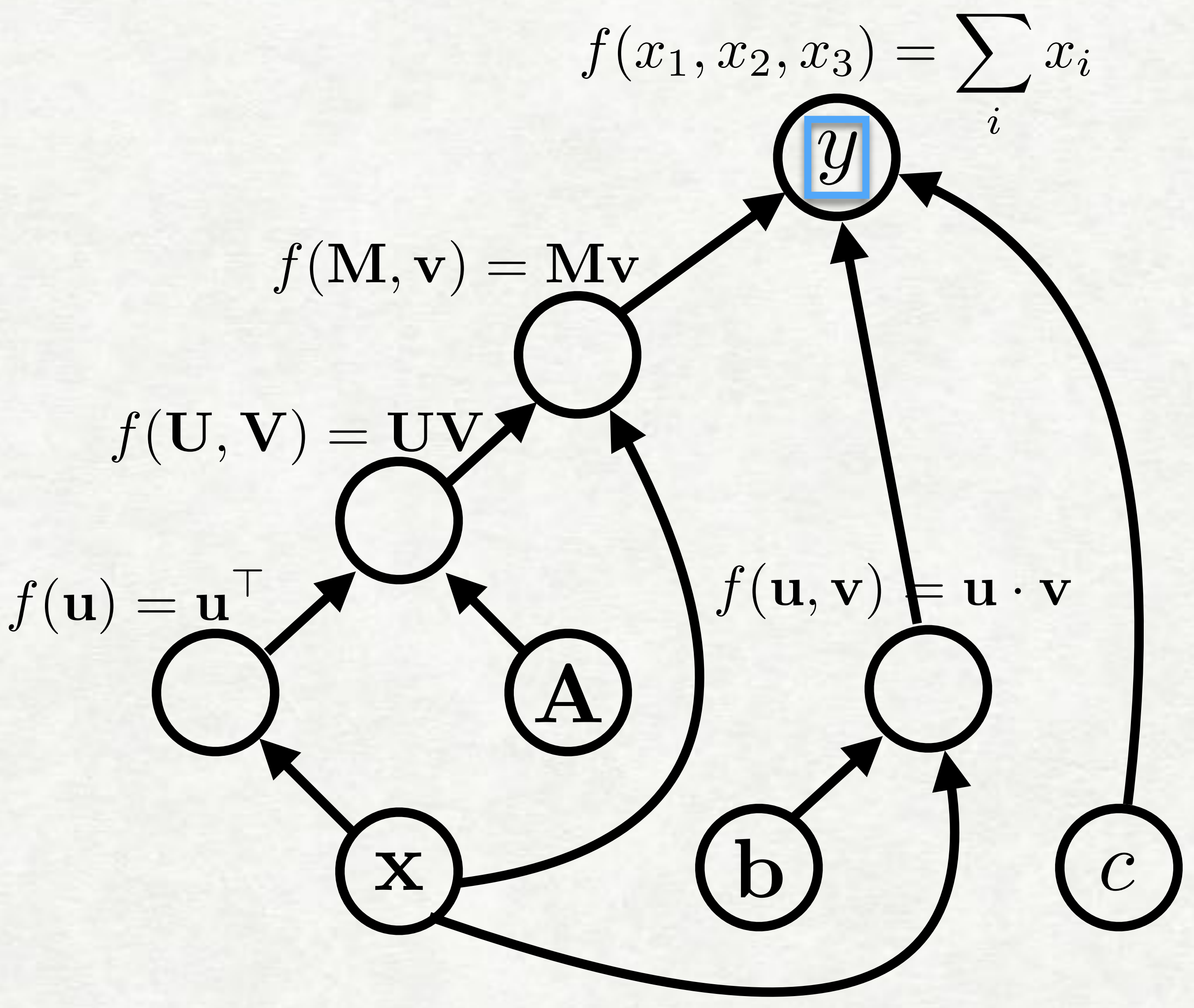


COMPUTATION GRAPHS

expression:

$$\boxed{y =} \mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c$$

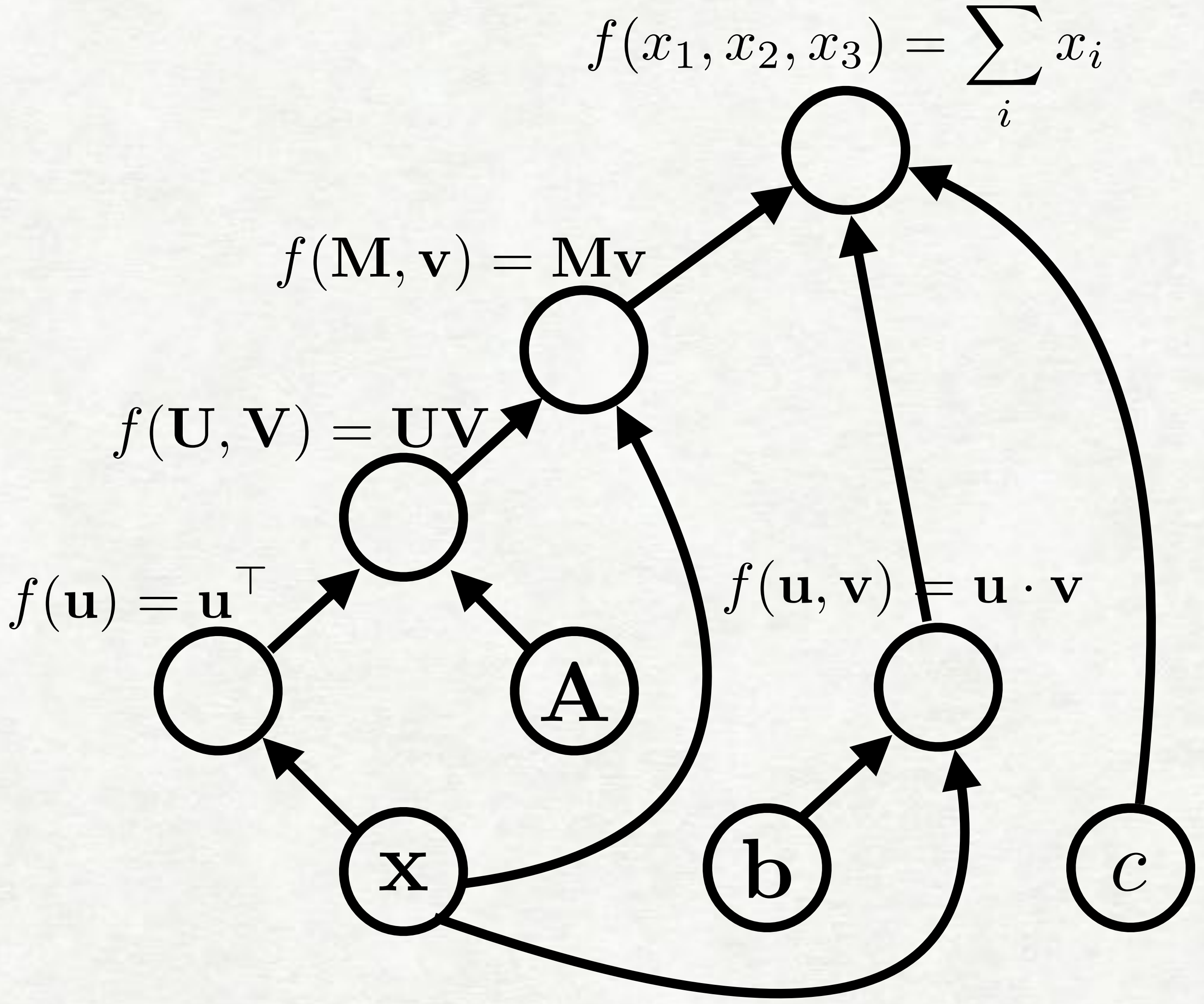
graph:



variable names are just labelings of nodes.

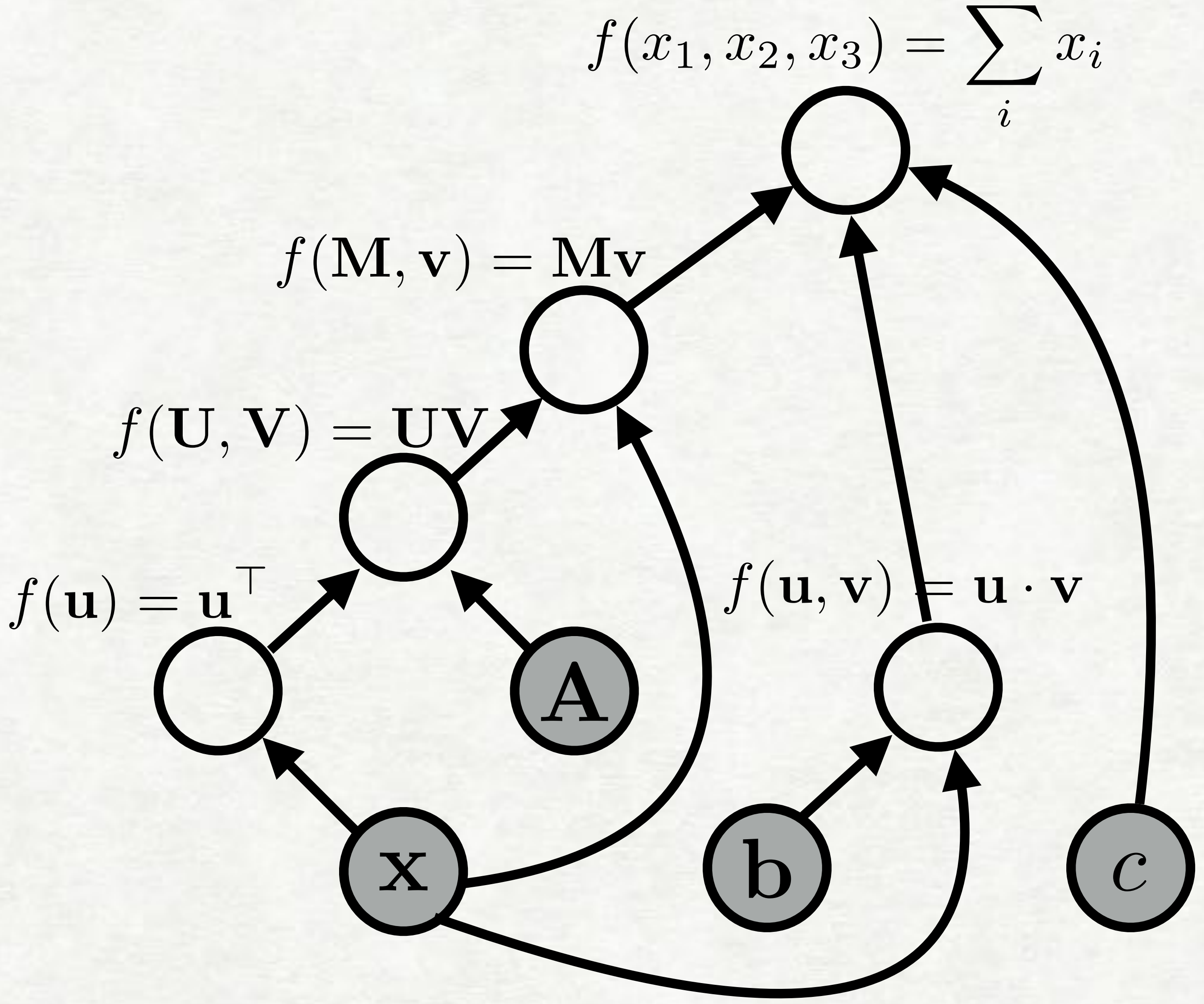
FORWARD PROPAGATION

graph:



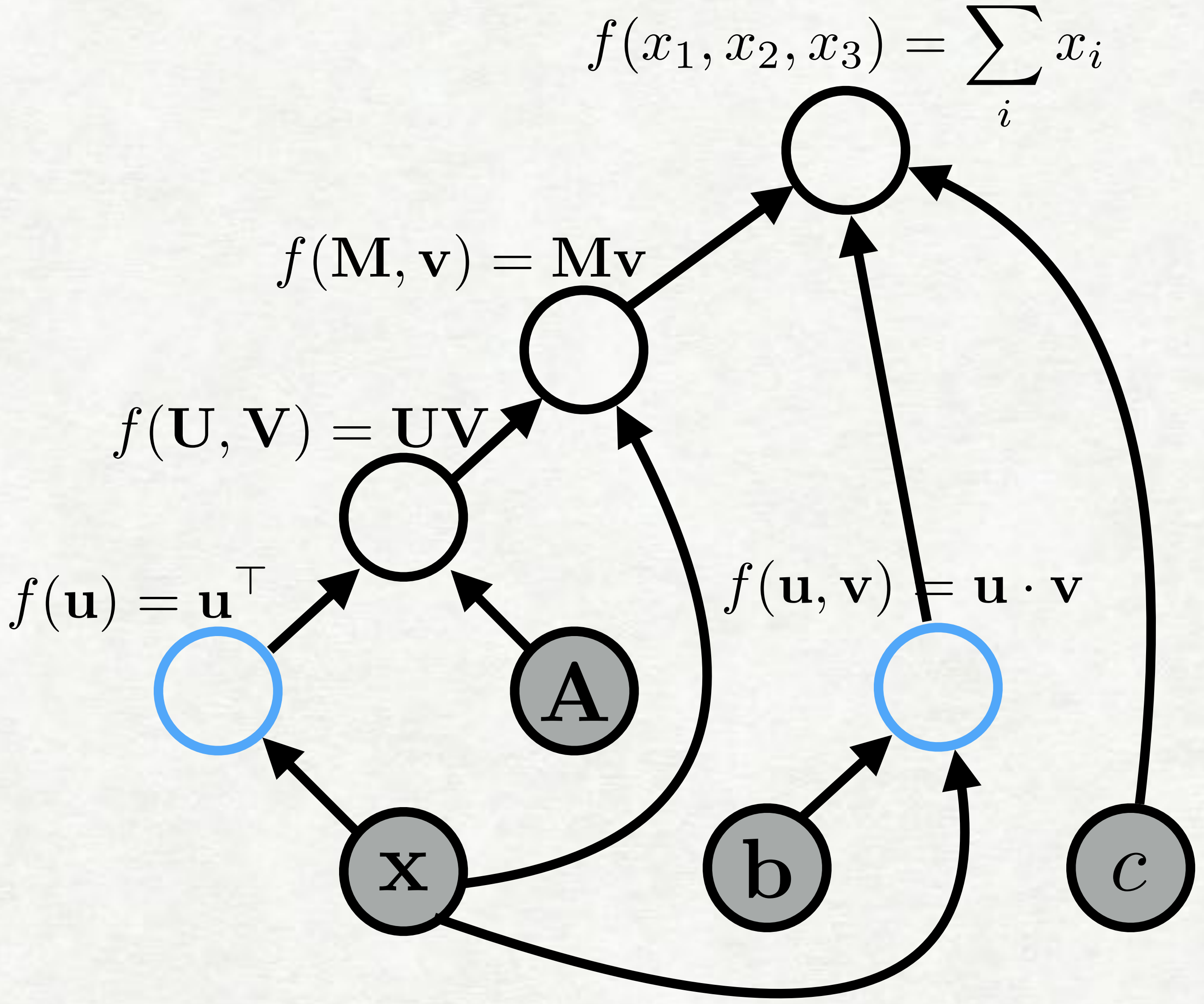
FORWARD PROPAGATION

graph:



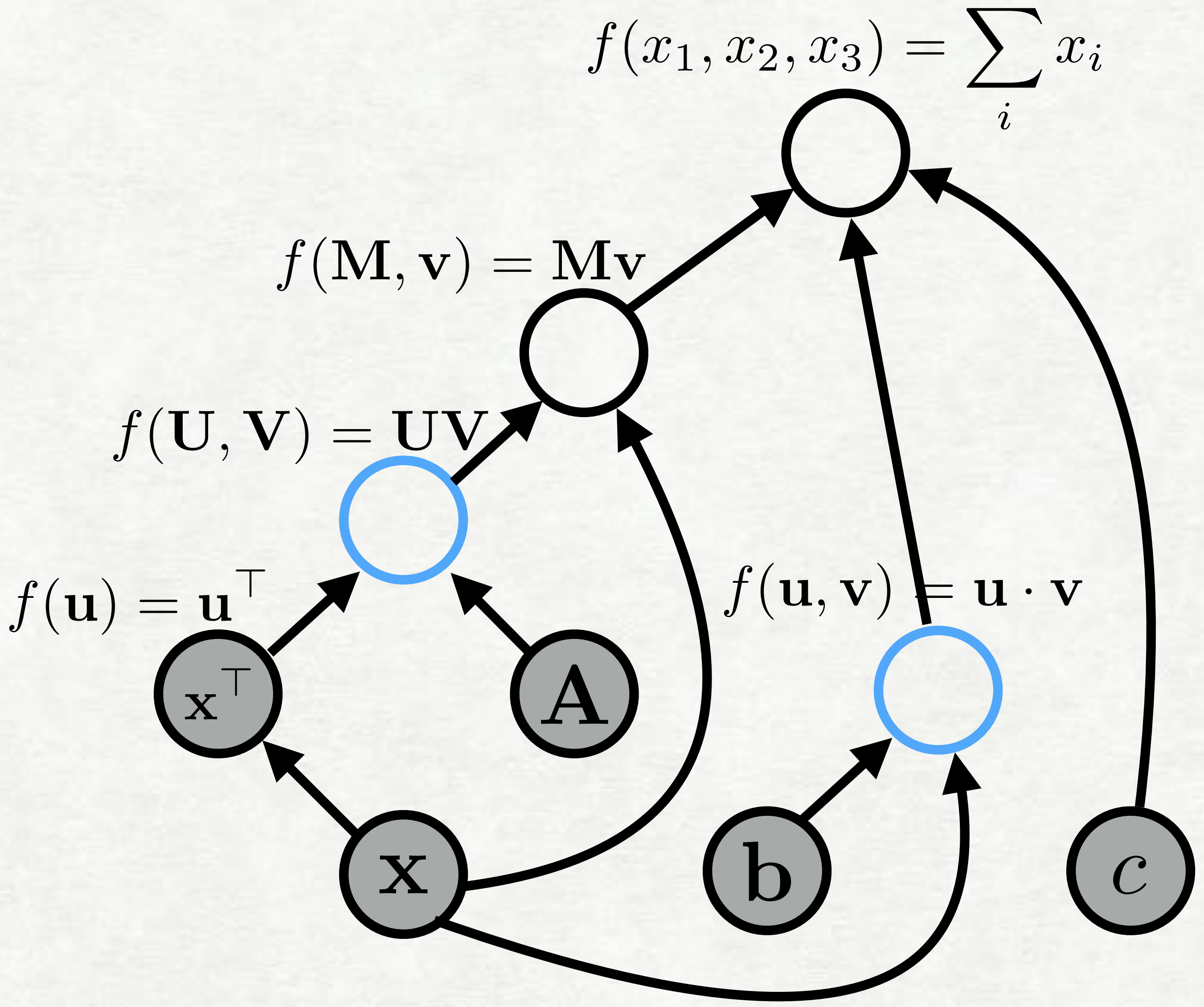
FORWARD PROPAGATION

graph:



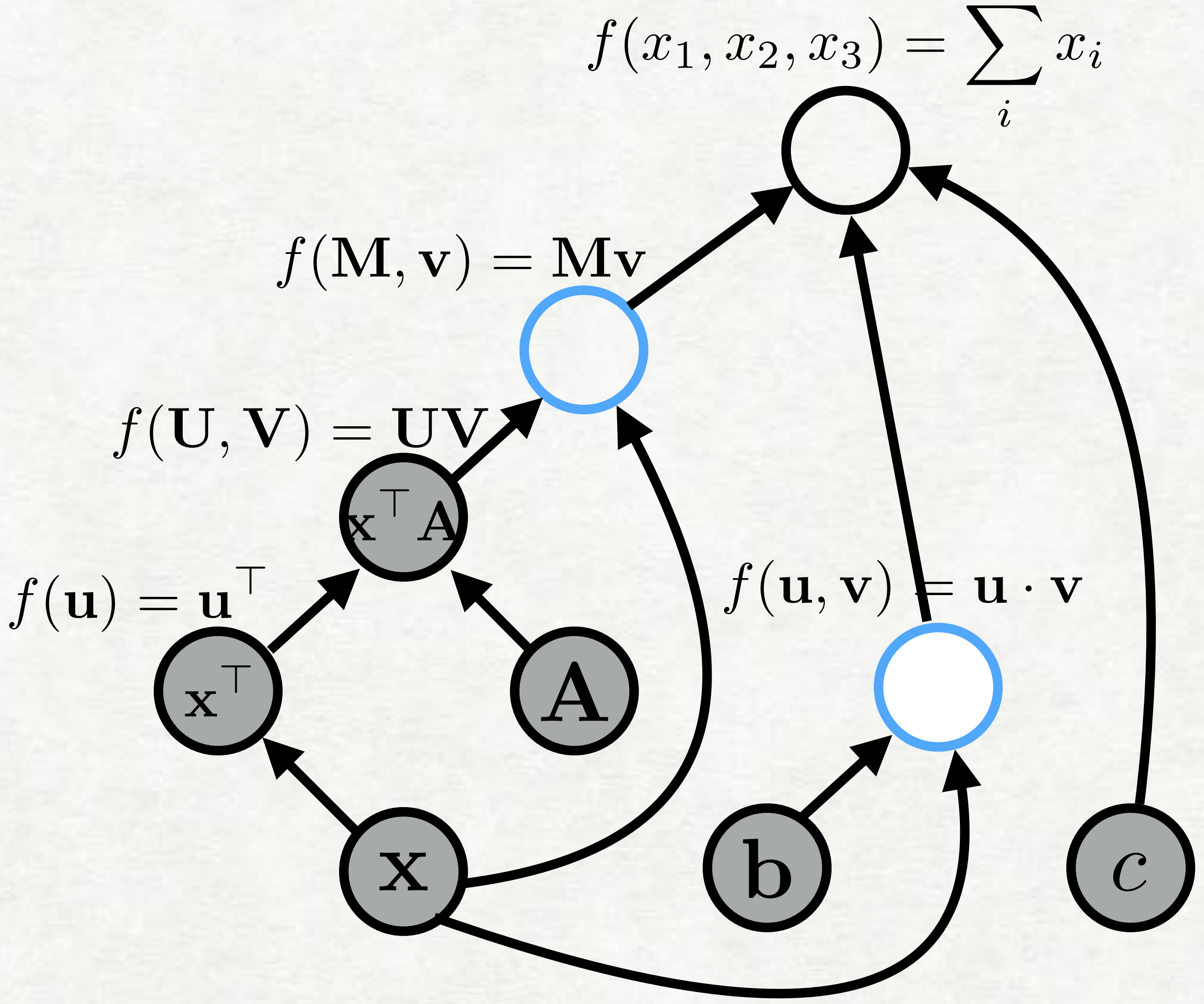
FORWARD PROPAGATION

graph:



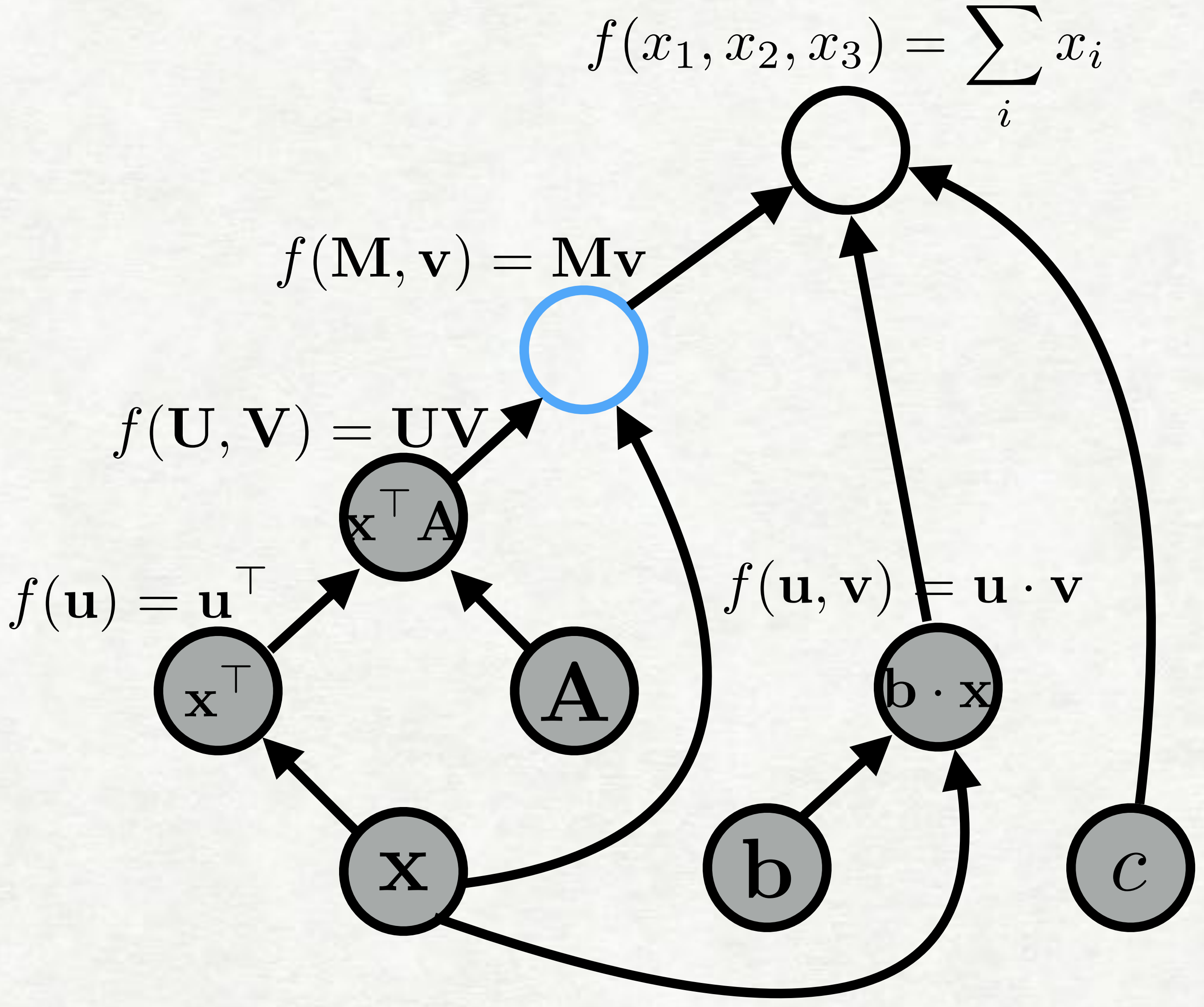
FORWARD PROPAGATION

graph:



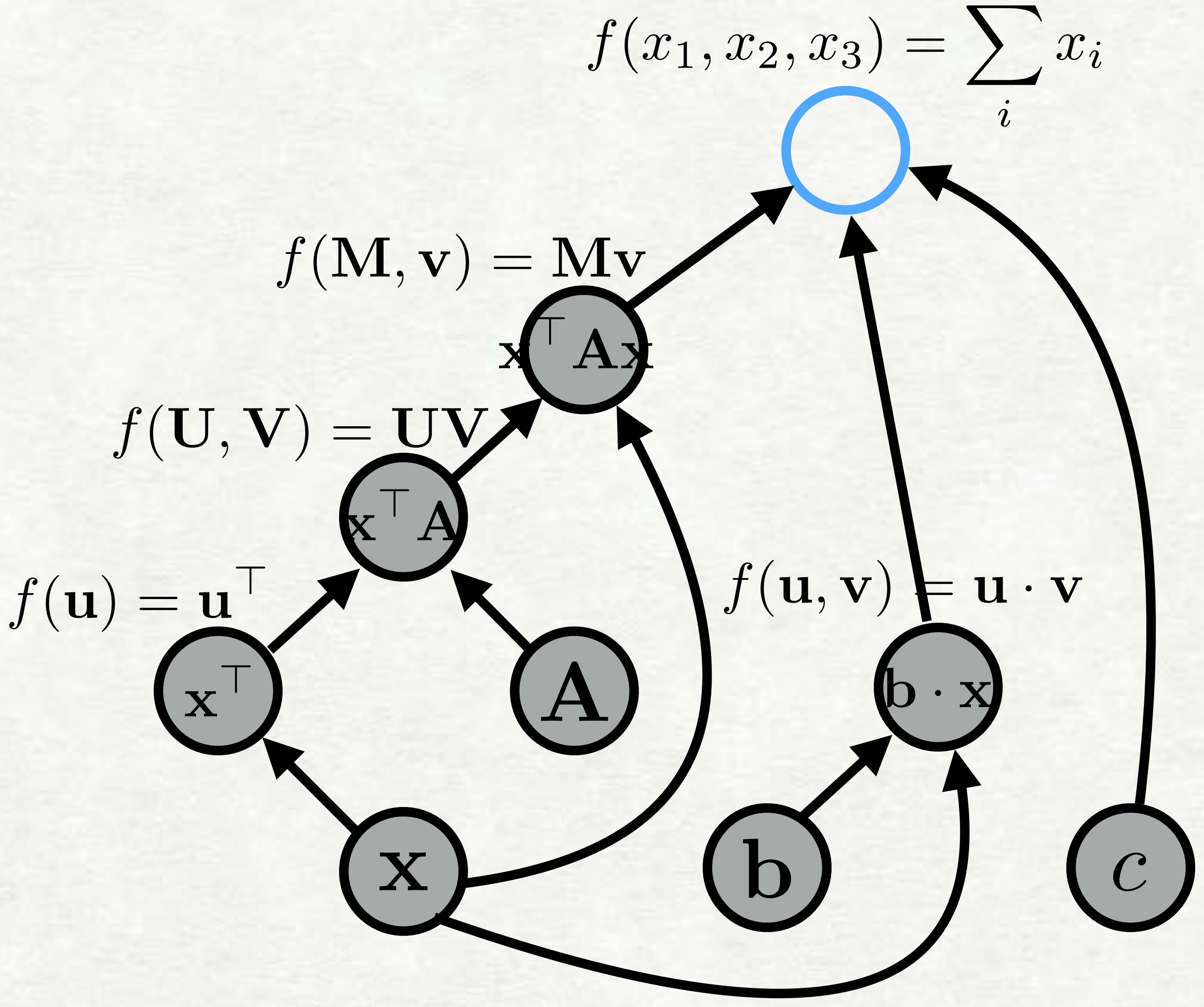
FORWARD PROPAGATION

graph:



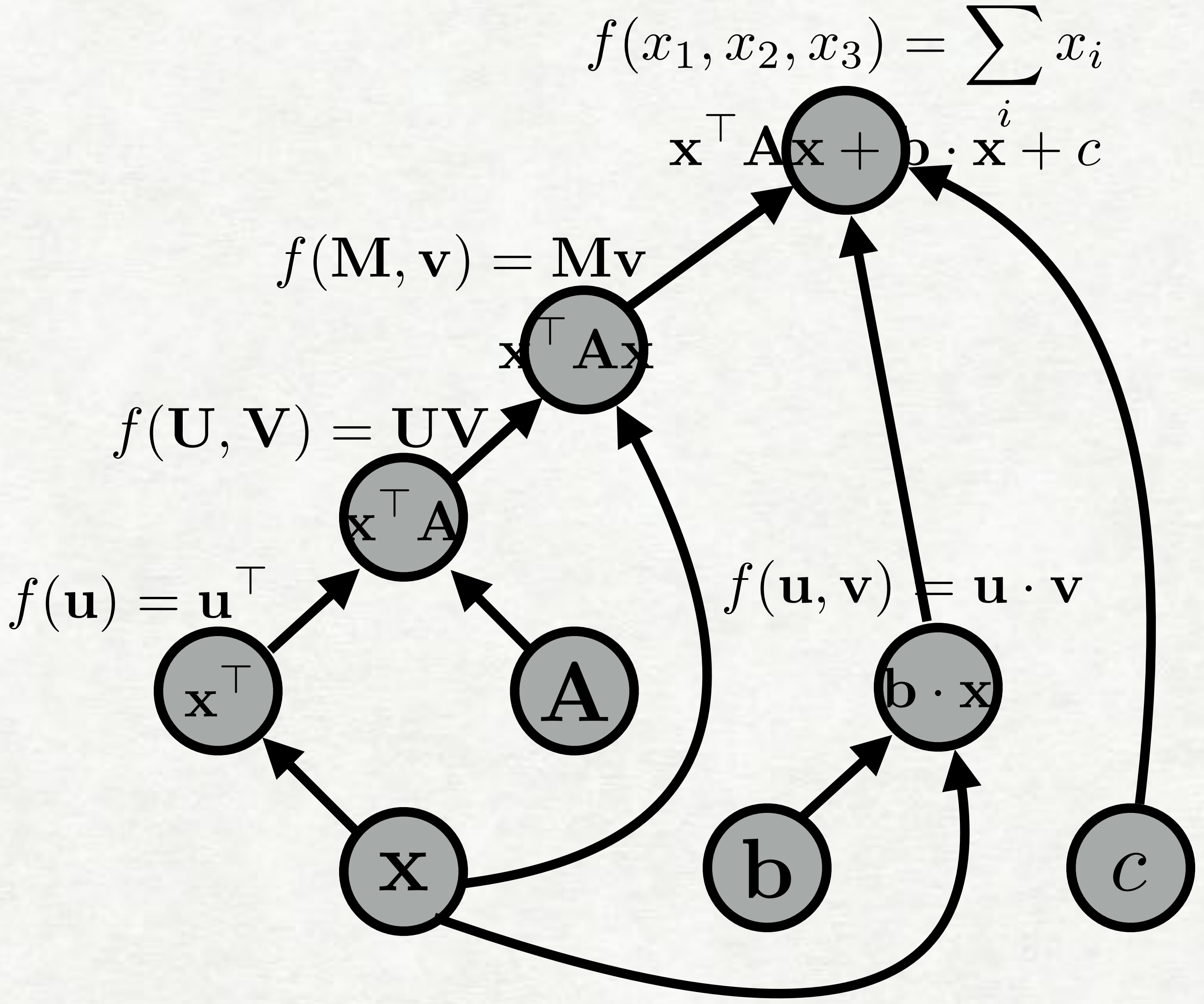
FORWARD PROPAGATION

graph:



FORWARD PROPAGATION

graph:



BACK-PROPAGATION

Back-propagation:

Process examples in reverse topological order

Calculate the derivatives of the parameters with respect to the final value
(This is usually a “loss function”, a value we want to minimize)

Parameter update:

Move the parameters in the direction of this derivative
W -= **α** * dl/dW

NEURAL NETWORK FRAMEWORKS

Examples in this class will be in DyNet or PyTorch:

intuitive, program like you think (c.f. TensorFlow, Theano)

fast for complicated networks on CPU (c.f. autodiff libraries, Chainer, PyTorch)

has **nice features to make efficient implementation easier** (automatic batching)

Static Frameworks

theano

Caffe

mxnet

TensorFlow

+Gluon

+Eager

Dynamic Frameworks (Recommended!)

dy/net

Chainer

PYTORCH

BASIC PROCESS IN DYNAMIC NEURAL NETWORK FRAMEWORKS

Create a model

For each example

- **create a graph** that represents the computation you want
- **calculate the result** of that computation
- if training, perform **back propagation and update**

NEXT CLASS PREVIEW

The building blocks of words

Lexicons

Edit Distance

ASSIGNMENT 1 OUT!