CS5740: Natural Language Processing
Spring 2017

Computation Graphs

Instructor: Yoav Artzi

From Practical Neural Networks for NLP / Chris Dyer, Yoav Goldberg, Graham Neubig / EMNLP 2016

Computation Graphs

- The descriptive language of deep learning models
- Functional description of the required computation
- Can be instantiated to do two types of computation:
  - Forward computation
  - Backward computation

expression:

$$\mathbf{x}$$

graph:

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

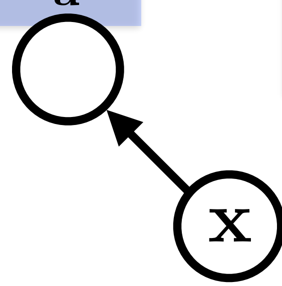
$\mathbf{x}$

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

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* $\frac{\partial \mathcal{F}}{\partial f(\mathbf{u})}$.

$$f(\mathbf{u}) = \mathbf{u}^{\top}$$



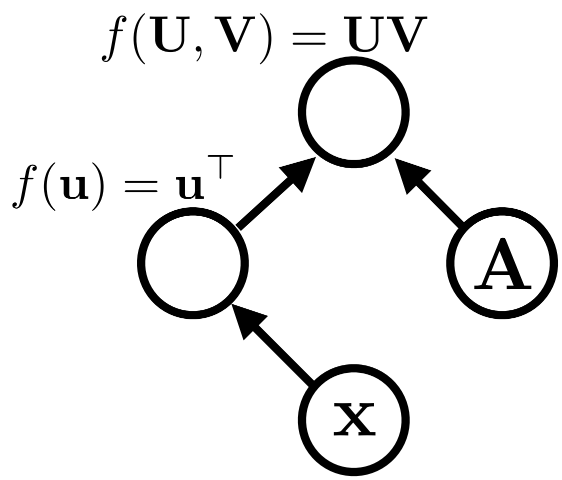
$$\frac{\partial f(\mathbf{u})}{\partial \mathbf{u}} \frac{\partial \mathcal{F}}{\partial f(\mathbf{u})} = \left(\frac{\partial \mathcal{F}}{\partial f(\mathbf{u})} \right)^{\top}$$

expression:

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

graph:

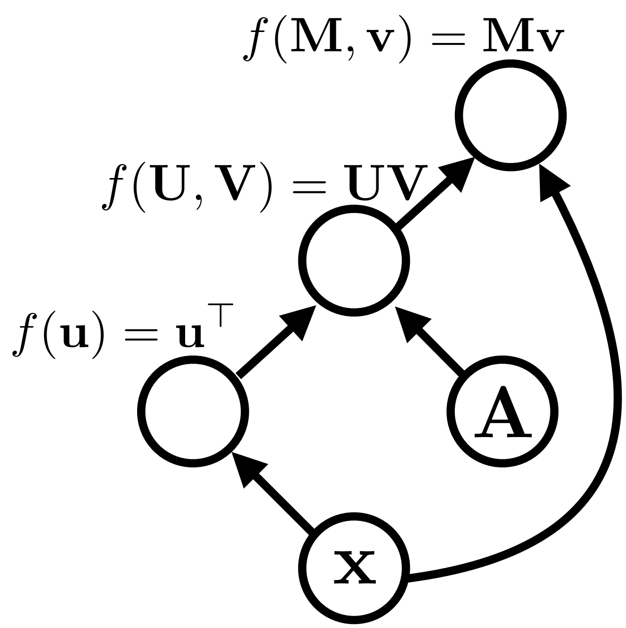
Functions can be nullary, unary, binary, … *n*-ary. Often they are unary or binary.



expression:

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

graph:

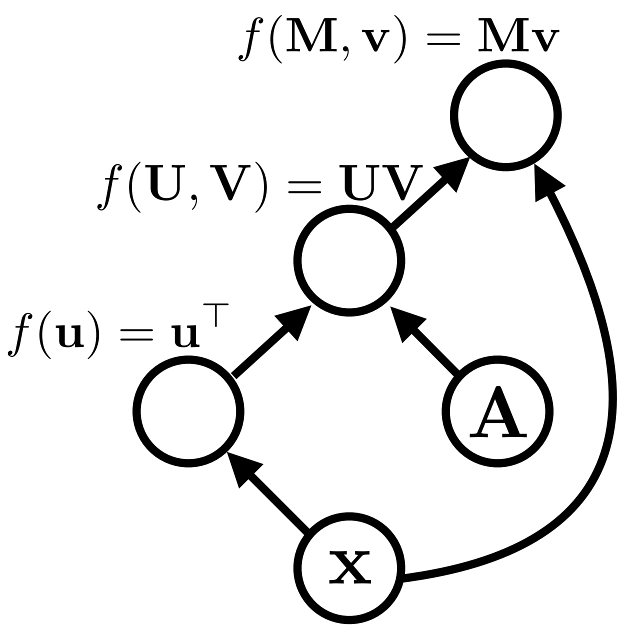


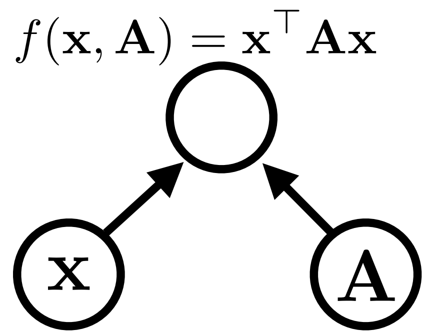
Computation graphs are directed and acyclic (usually)

expression:

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

graph:





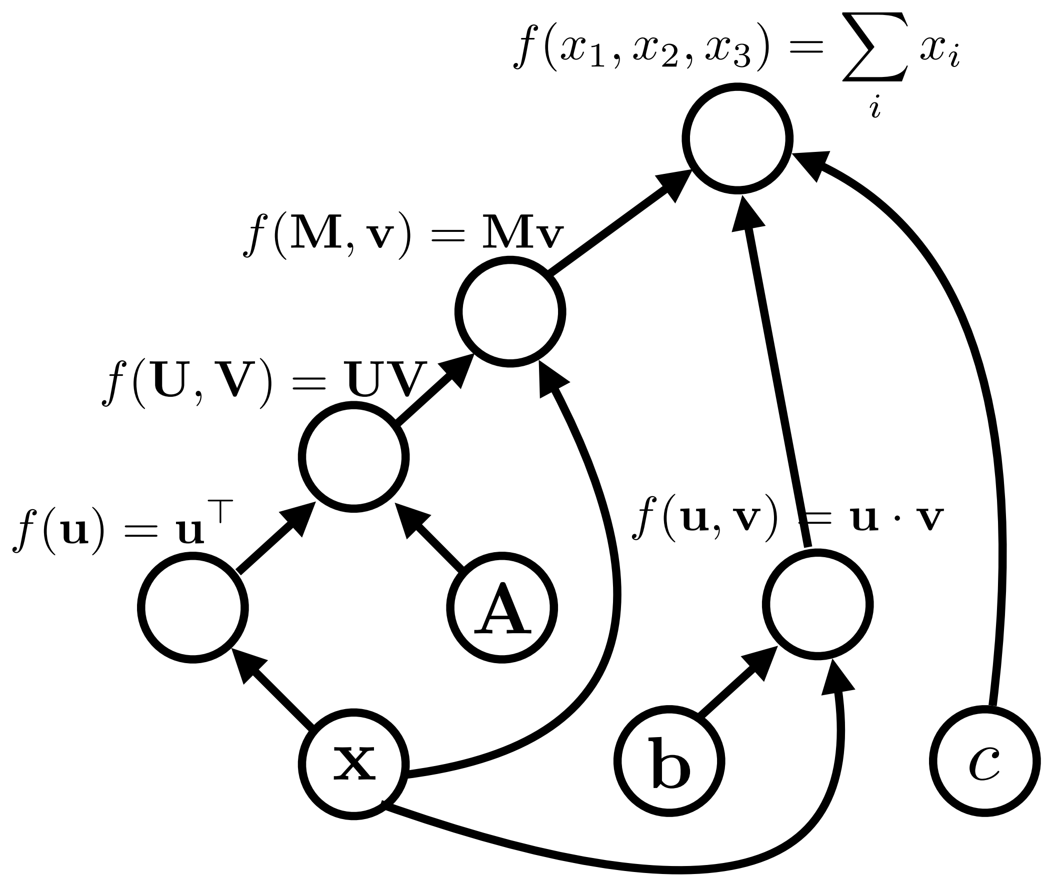
$$\frac{\partial f(\mathbf{x}, \mathbf{A})}{\partial \mathbf{x}} = (\mathbf{A}^\top + \mathbf{A})\mathbf{x}$$

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

expression:

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

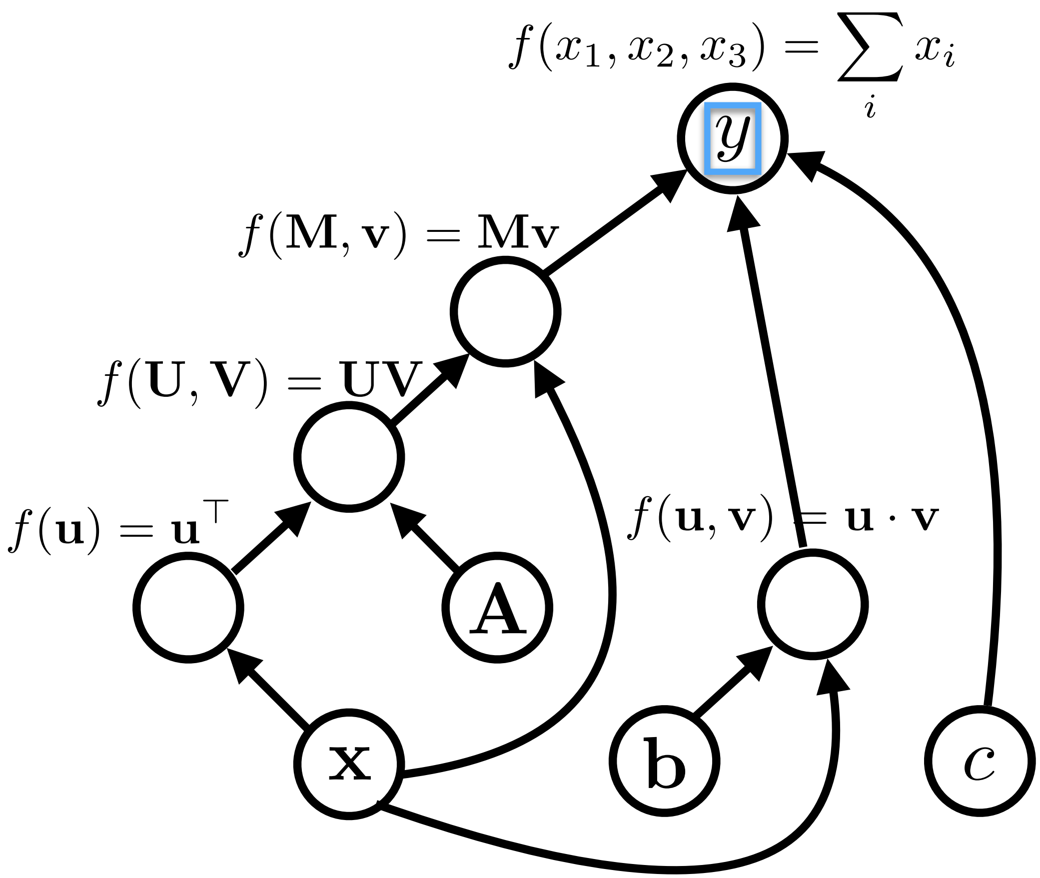
graph:



expression:

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

graph:



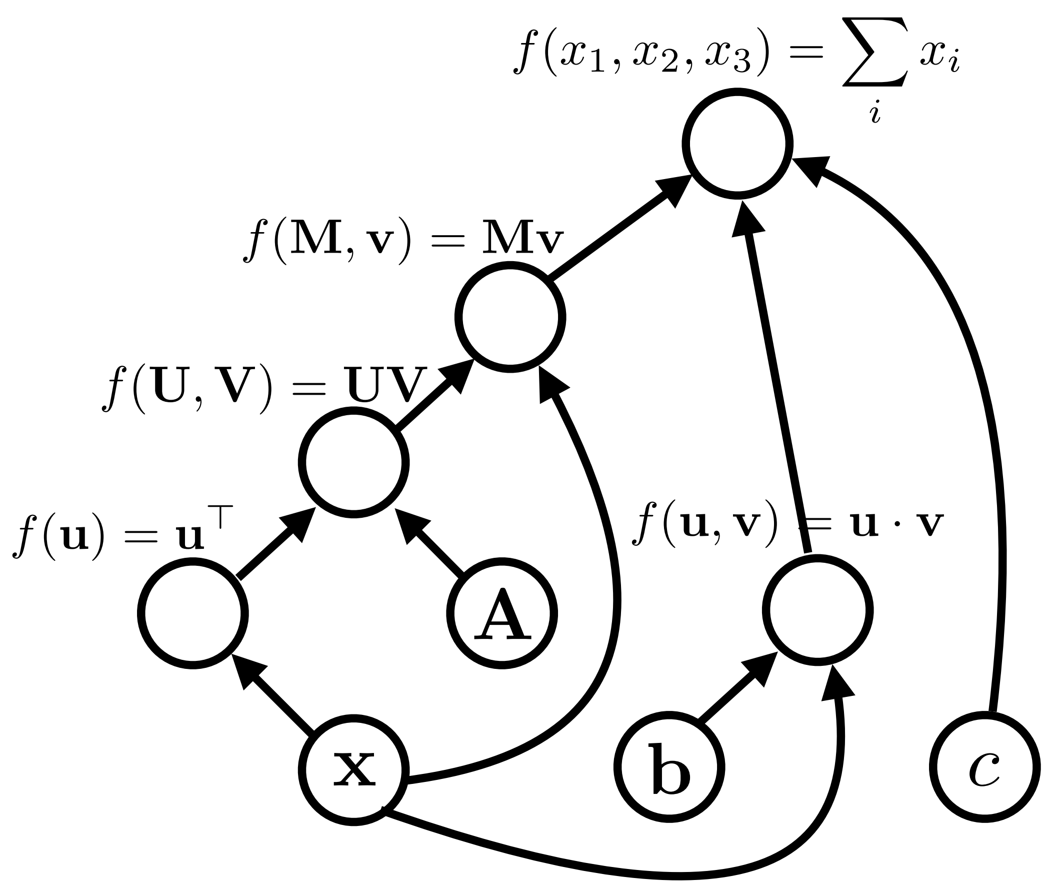
variable names are just labelings of nodes.

Algorithms

- **Graph construction**
- **Forward propagation**
  - Loop over nodes in topological order
    - Compute the value of the node given its inputs
  - *Given my inputs, make a prediction (or compute an "error" with respect to a "target output")*
- **Backward 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
  - *How does the output change if I make a small change to the inputs?*

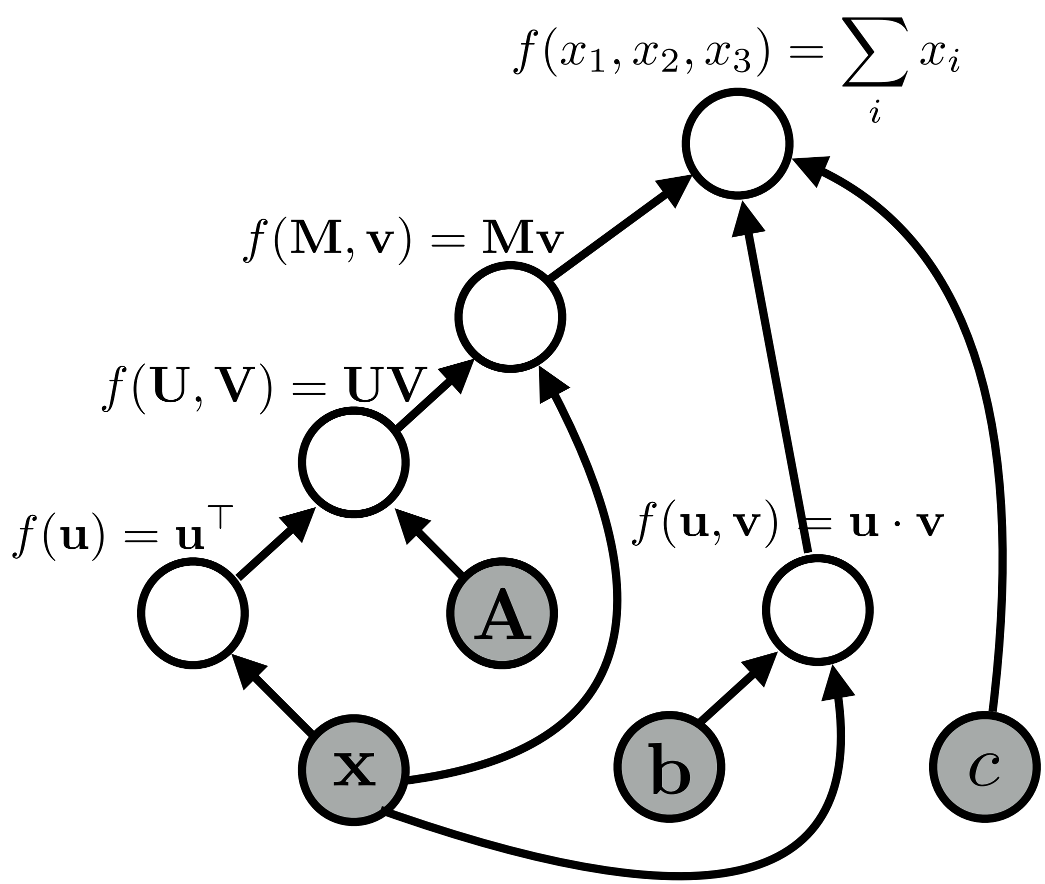
Forward Propagation

graph:



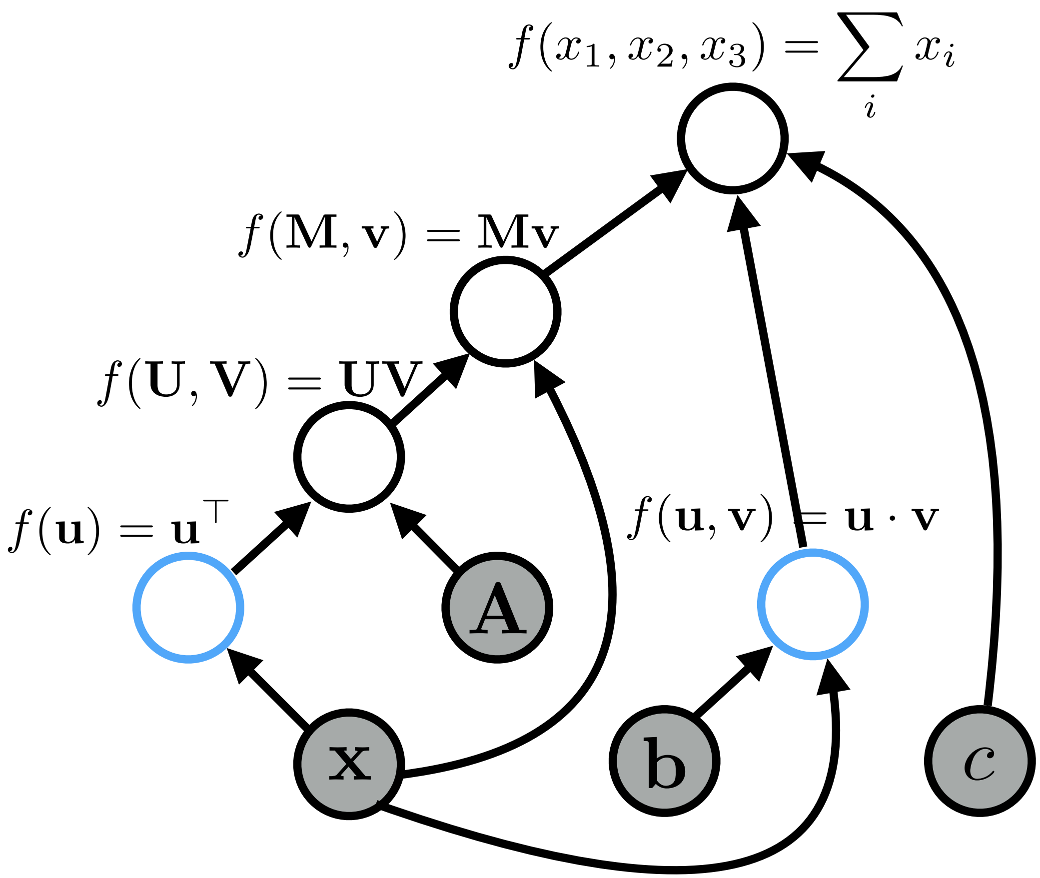
Forward Propagation

graph:



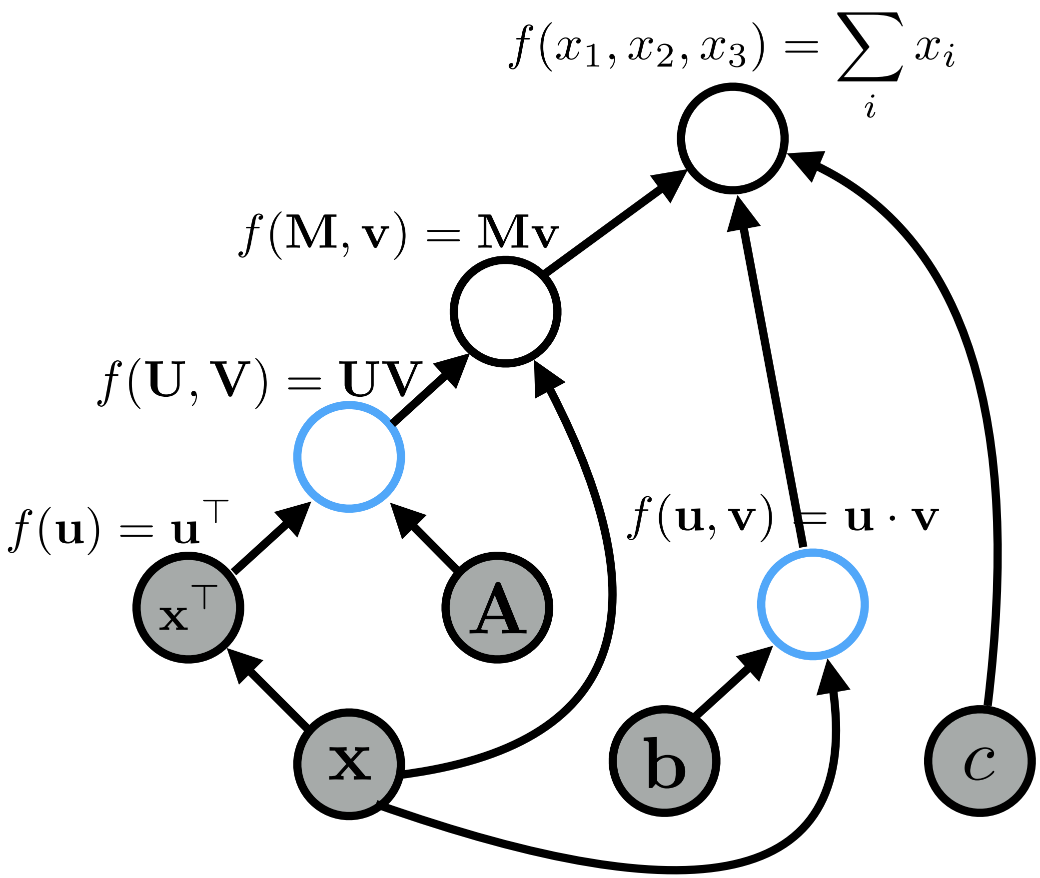
Forward Propagation

graph:



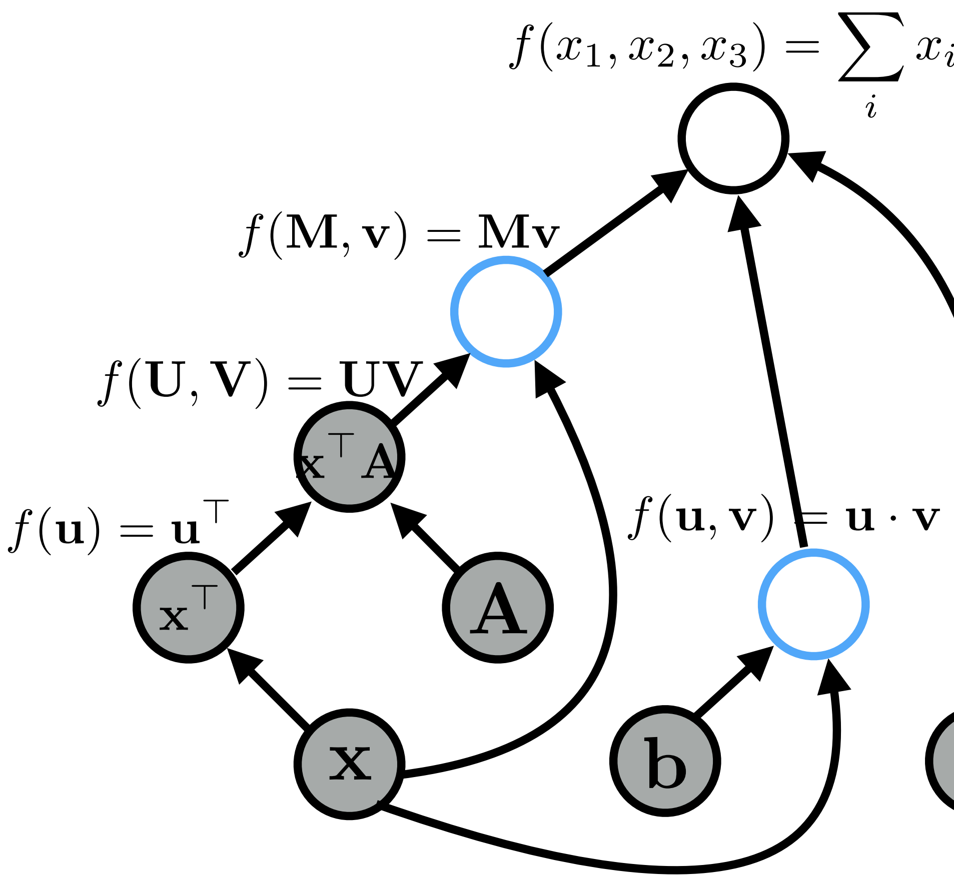
Forward Propagation

graph:



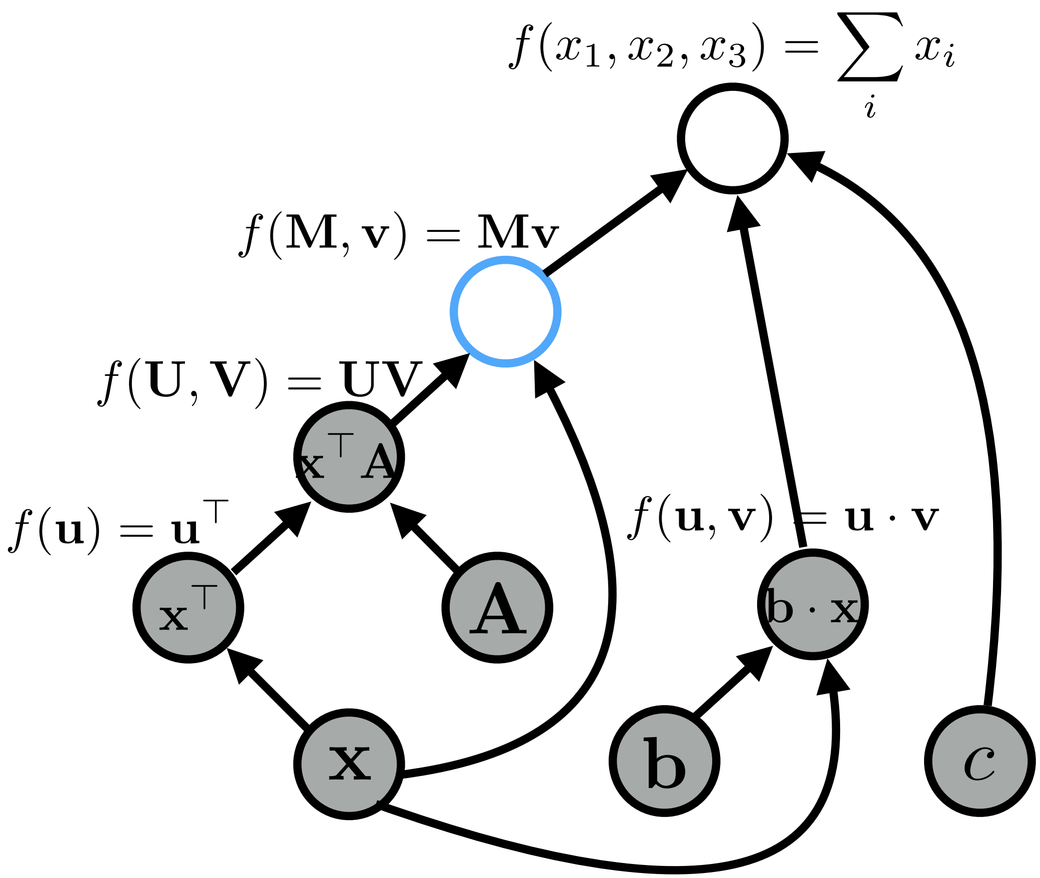
Forward Propagation

graph:



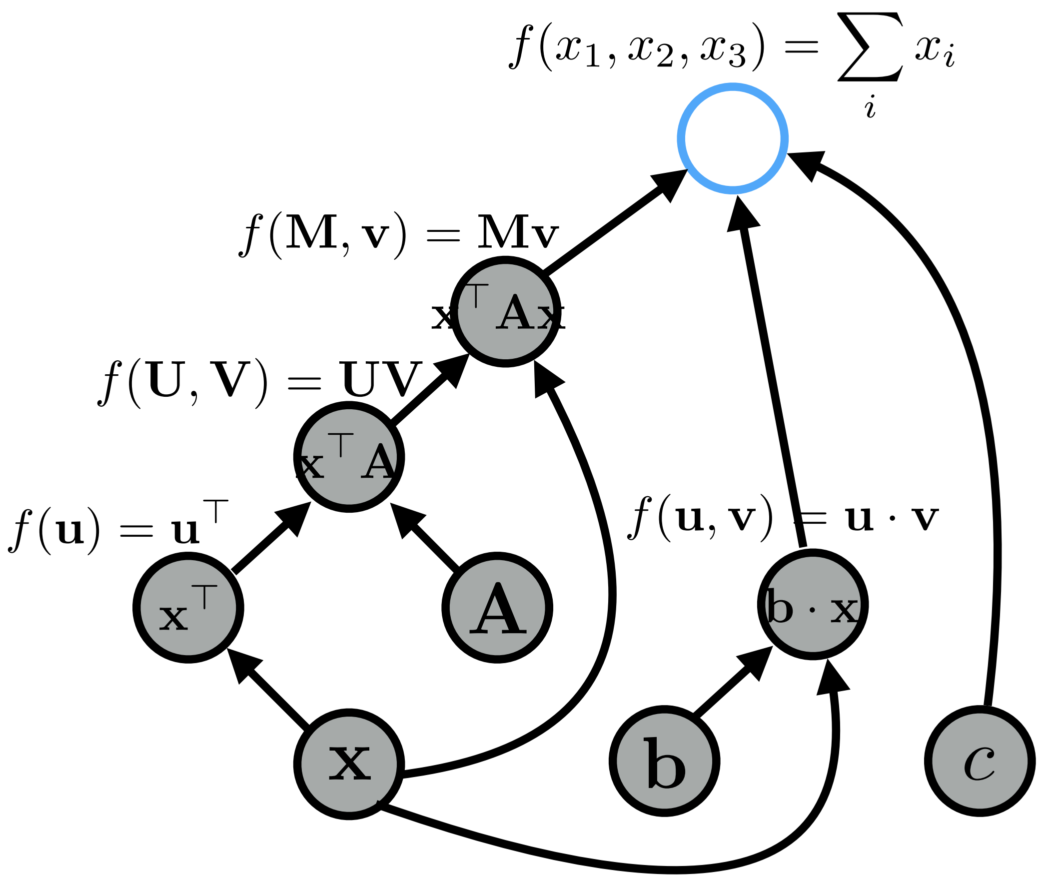
Forward Propagation

graph:



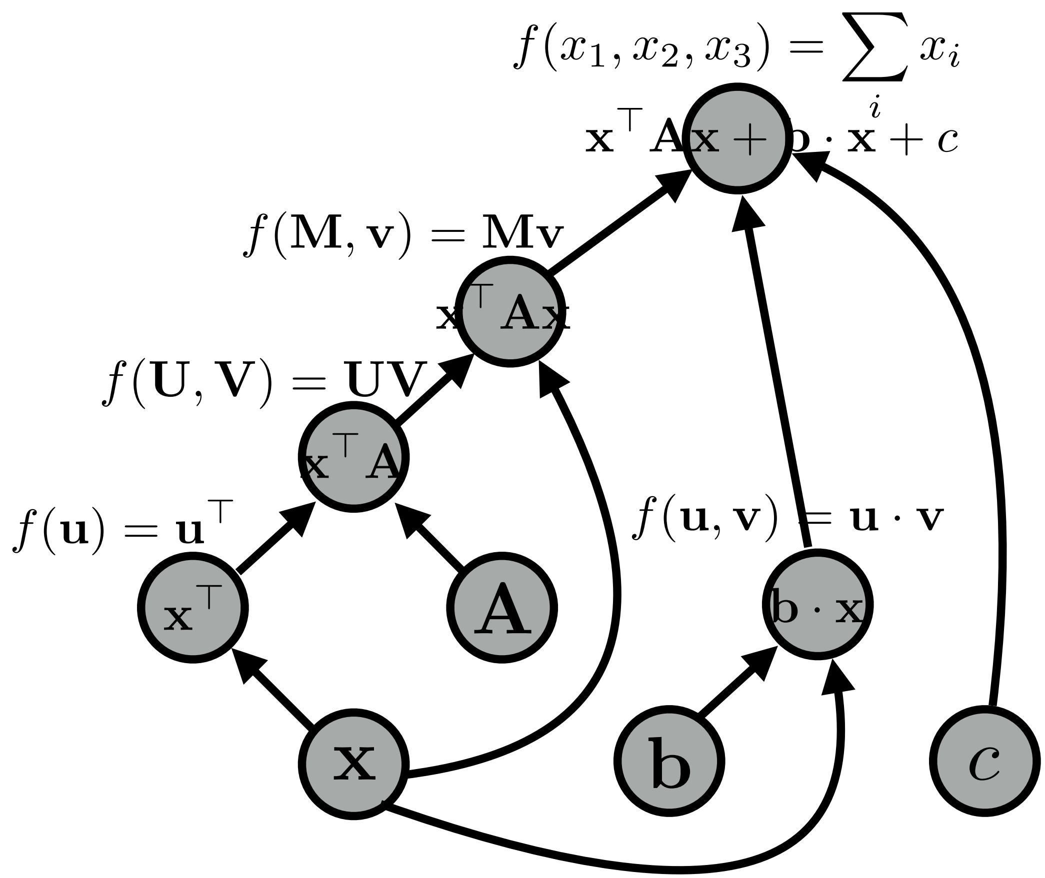
Forward Propagation

graph:



Forward Propagation

graph:



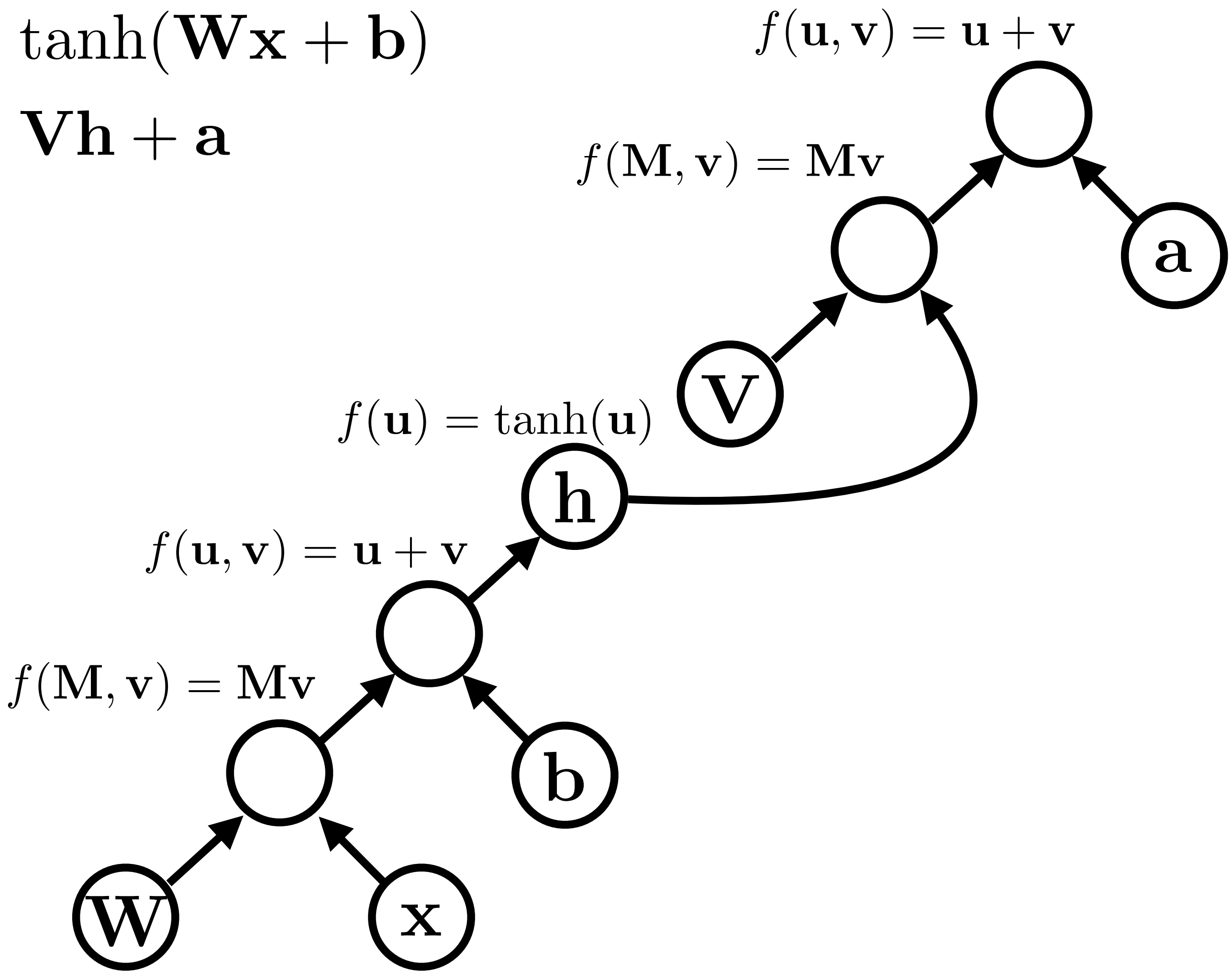
The MLP

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

$$\mathbf{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$

The MLP

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



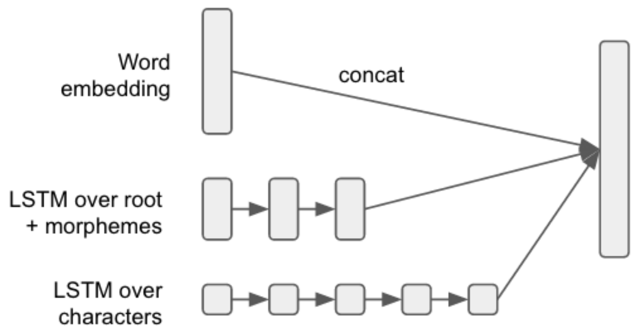
Constructing Graphs

Two Software Models

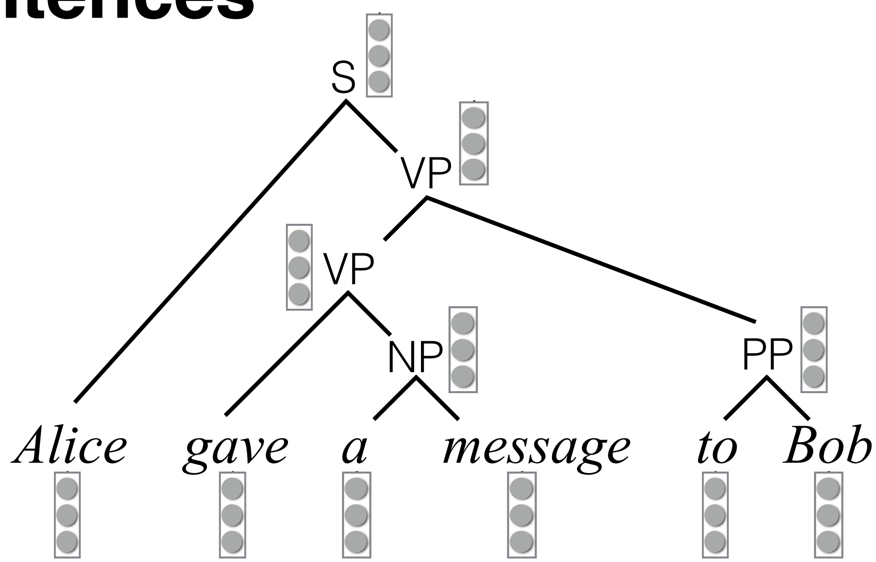
- **Static declaration**
  - Phase 1: define an architecture
    (maybe with some primitive flow control like loops and conditionals)
  - Phase 2: run a bunch of data through it to train the model and/or make predictions
- **Dynamic declaration**
  - Graph is defined implicitly (e.g., using operator overloading) as the forward computation is executed

Hierarchical Structure

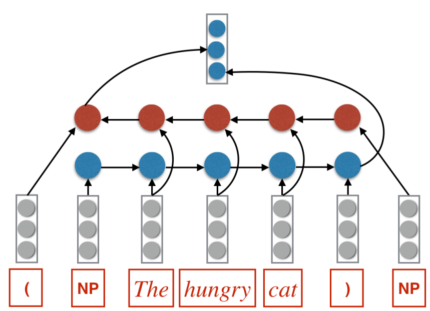
Words



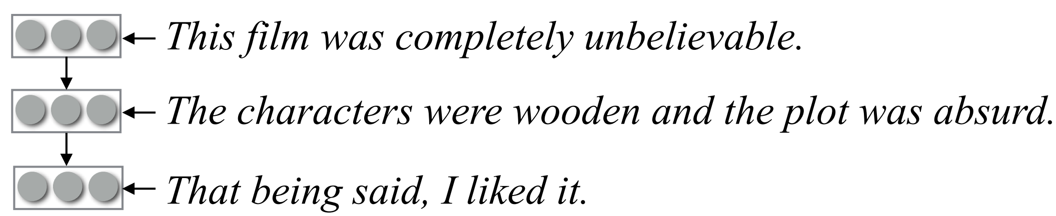
Sentences



Phrases



Documents



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, Theano, TensorFlow

Dynamic Declaration

- **Pros**
  - library is less invasive
  - the 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**
  - little time for graph optimization
  - if the graph is static, effort can be wasted
- examples: Chainer, *most automatic differentiation libraries,* **DyNet**

Dynamic Structure?

- Hierarchical structures exist in language
  - We might want to let the network reflect that hierarchy
  - Hierarchical structure is easiest to process with traditional flow-control mechanisms in your favorite languages
- Combinatorial algorithms (e.g., dynamic programming)
  - Exploit independencies to compute over a large space of operations tractably