CS11-711 Advanced NLP

Language Modeling and Neural Networks

Graham Neubig

Carnegie Mellon University
Language Technologies Institute

Site
https://phontron.com/class/anlp2021/

Are These Sentences OK?

- Jane went to the store.
- store to Jane went the.
- Jane went store.
- Jane goed to the store.
- The store went to Jane.
- The food truck went to Jane.

Engineering Solutions

- Jane went to the store.
- store to Jane went the.
- Jane went store.

Create a grammar of the language

- Jane goed to the store.

Consider morphology and exceptions

- The store went to Jane.

Semantic categories, preferences

- The food truck went to Jane.

And their exceptions

Probabilistic Language Models

$$P(X) = \prod_{i=1}^{I} P(\underbrace{x_i}_{\text{Next Word}} \mid \underbrace{x_1, \ldots, x_{i-1}}_{\text{Context}})$$

The big problem: How do we predict

$$P(x_i \mid x_1, \ldots, x_{i-1})$$

?!?!

What Can we Do w/ LMs?

- Score sentences:

Jane went to the store . → high
store to Jane went the . → low

(same as calculating loss for training)

- Generate sentences:

while didn't choose end-of-sentence symbol:
  calculate probability
  sample a new word from the probability distribution

Count-based Language Models

Review: Count-based Unigram Model

- **Independence assumption:** $P(x_i|x_1,\ldots,x_{i-1}) \approx P(x_i)$

- **Count-based maximum-likelihood estimation:**

$$P_{\mathrm{MLE}}(x_i) = \frac{c_{\mathrm{train}}(x_i)}{\sum_{\tilde{x}} c_{\mathrm{train}}(\tilde{x})}$$

- **Interpolation w/ UNK model:**

$$P(x_i) = (1 - \lambda_{\mathrm{unk}}) * P_{\mathrm{MLE}}(x_i) + \lambda_{\mathrm{unk}} * P_{\mathrm{unk}}(x_i)$$

Higher-order *n*-gram Models

- Limit context length to *n*, count, and divide

$$P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) := \frac{c(x_{i-n+1}, \ldots, x_i)}{c(x_{i-n+1}, \ldots, x_{i-1})}$$

$$\text{P(example | this is a)} = \frac{\text{c(this is an example)}}{\text{c(this is an)}}$$

- Add smoothing, to deal with zero counts:

$$\begin{aligned} P(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) =& \lambda P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) \\ &+ (1-\lambda) P(x_i \mid x_{1-n+2}, \ldots, x_{i-1}) \end{aligned}$$

Smoothing Methods
(e.g. Goodman 1998)

- **Additive/Dirichlet:**

fallback distribution

$$P(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) := \frac{c(x_{i-n+1}, \ldots, x_i) + \boxed{\alpha}\boxed{P(x_i \mid x_{i-n+2}, \ldots, x_{i-1})}}{c(x_{i-n+1}, \ldots, x_{i-1}) + \boxed{\alpha}}$$

interpolation hyperparameter

- **Discounting:**

discount hyperparameter

$$P(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) := \frac{c(x_{i-n+1}, \ldots, x_i) - \boxed{d} + \boxed{\alpha} P(x_i \mid x_{i-n+2}, \ldots, x_{i-1})}{c(x_{i-n+1}, \ldots, x_{i-1})}$$

interpolation calculated by sum of discounts $\boxed{\alpha} = \sum_{\{\tilde{x}; c(x_{i-n+1}, \ldots, \tilde{x}) > 0\}} d$

- **Kneser-Ney:** discounting w/ modification of the lower-order distribution

Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. 1998.

Problems and Solutions?

- Cannot share strength among **similar words**

she bought a car	she bought a bicycle
she purchased a car	she purchased a bicycle

→ solution: class based language models

- Cannot condition on context with **intervening words**

Dr. Jane Smith   Dr. Gertrude Smith

→ solution: skip-gram language models

- Cannot handle **long-distance dependencies**

for tennis class he wanted to buy his own racquet
for programming class he wanted to buy his own computer

→ solution: cache, trigger, topic, syntactic models, etc.

When to Use n-gram Models?

- Neural language models (next) achieve better performance, but
- n-gram models are extremely fast to estimate/apply
- n-gram models can be better at modeling low-frequency phenomena
- **Toolkit:** kenlm

https://github.com/kpu/kenlm

LM Evaluation

Evaluation of LMs

- **Log-likelihood:**

$$LL(\mathcal{E}_{test}) = \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

- **Per-word Log Likelihood:**

$$WLL(\mathcal{E}_{test}) = \frac{1}{\sum_{E \in \mathcal{E}_{test}} |E|} \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

- **Per-word (Cross) Entropy:**

$$H(\mathcal{E}_{test}) = \frac{1}{\sum_{E \in \mathcal{E}_{test}} |E|} \sum_{E \in \mathcal{E}_{test}} -\log_2 P(E)$$

- **Perplexity:**

$$ppl(\mathcal{E}_{test}) = 2^{H(\mathcal{E}_{test})} = e^{-WLL(\mathcal{E}_{test})}$$

Unknown Words

- Necessity for UNK words
  - We won’t have all the words in the world in training data
  - Larger vocabularies require more memory and computation time
- Common ways:
  - Limit vocabulary by frequency threshold (usually UNK <= 1) or rank threshold
  - Model characters or subwords

Evaluation and Vocabulary

- **Important:** the vocabulary must be the same over models you compare
- Or more accurately, all models must be able to generate the test set (it's OK if they can generate *more* than the test set, but not less)
    - e.g. Comparing a character-based model to a word-based model is fair, but not vice-versa

An Alternative:
Featurized Log-Linear Models

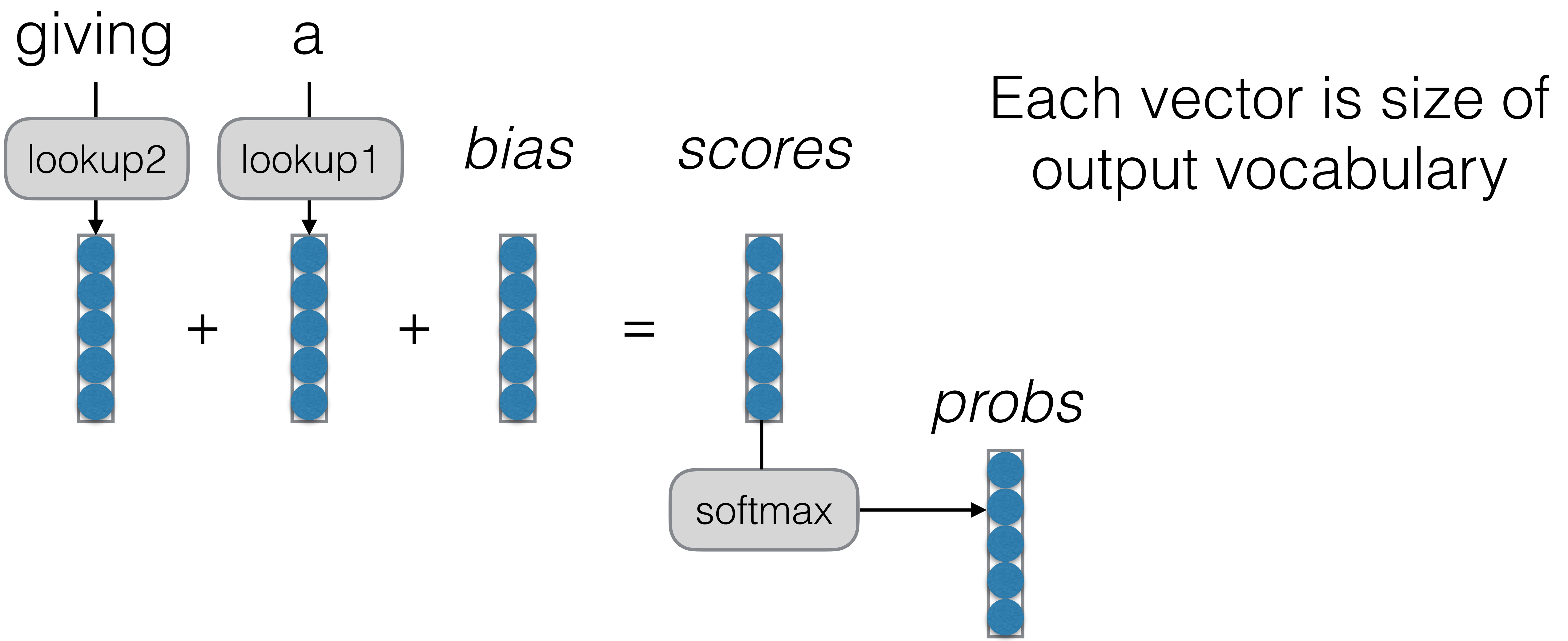
(Rosenfeld 1996)

An Alternative: Featurized Models

- Calculate features of the context
- Based on the features, calculate probabilities
- Optimize feature weights using gradient descent, etc.

An Alternative: Featurized Models

- Calculate features of the context, calculate probabilities



- Feature weights optimized by SGD, etc.
- What are similarities/differences w/ BOW classifier?

Example:

Previous words: "giving a"

$$
\begin{array}{c}
\text{a} \\ \text{the} \\ \text{talk} \\ \text{gift} \\ \text{hat} \\ \ldots
\end{array}
\quad
b=\begin{pmatrix} 3.0 \\ 2.5 \\ -0.2 \\ 0.1 \\ 1.2 \\ \ldots \end{pmatrix}
\quad
w_{1,a}=\begin{pmatrix} -6.0 \\ -5.1 \\ 0.2 \\ 0.1 \\ 0.5 \\ \ldots \end{pmatrix}
\quad
w_{2,giving}=\begin{pmatrix} -0.2 \\ -0.3 \\ 1.0 \\ 2.0 \\ -1.2 \\ \ldots \end{pmatrix}
\quad
s=\begin{pmatrix} -3.2 \\ -2.9 \\ 1.0 \\ 2.2 \\ 0.6 \\ \ldots \end{pmatrix}
$$

Words we're predicting	How likely are they?	How likely are they given prev. word is "a"?	How likely are they given 2nd prev. word is "giving"?	Total score

Reminder: Training Algorithm

- Calculate the **gradient of the loss function** with respect to the parameters

$$\frac{\partial \mathcal{L}_{\mathrm{train}}(\theta)}{\partial \theta}$$

  - How? Use the chain rule / back-propagation. More in a second

- **Update** to move in a direction that decreases the loss

$$\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}_{\mathrm{train}}(\theta)}{\partial \theta}$$

What Problems are Handled?

- Cannot share strength among **similar words**

she bought a car she purchased a car	she bought a bicycle she purchased a bicycle

→ not solved yet 😞

- Cannot condition on context with **intervening words**

Dr. Jane Smith Dr. Gertrude Smith

→ solved! 😀

- Cannot handle **long-distance dependencies**

for tennis class he wanted to buy his own racquet

for programming class he wanted to buy his own computer

→ not solved yet 😞

Beyond Linear Models

Linear Models can't Learn Feature Combinations

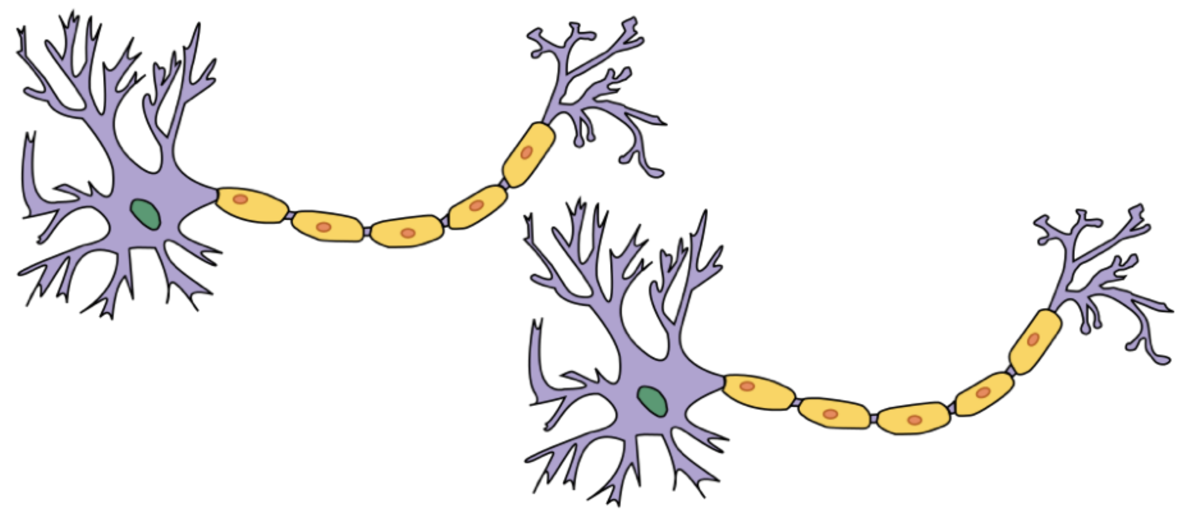
students take tests→ **high** teachers take tests → **low**

students write tests → **low** teachers write tests → **high**

- These can't be expressed by linear features
- What can we do?
  - Remember combinations as features (individual scores for "students take", "teachers write") → Feature space explosion!
  - Neural networks!

“Neural” Nets

Original Motivation: Neurons in the Brain



Current Conception: Computation Graphs

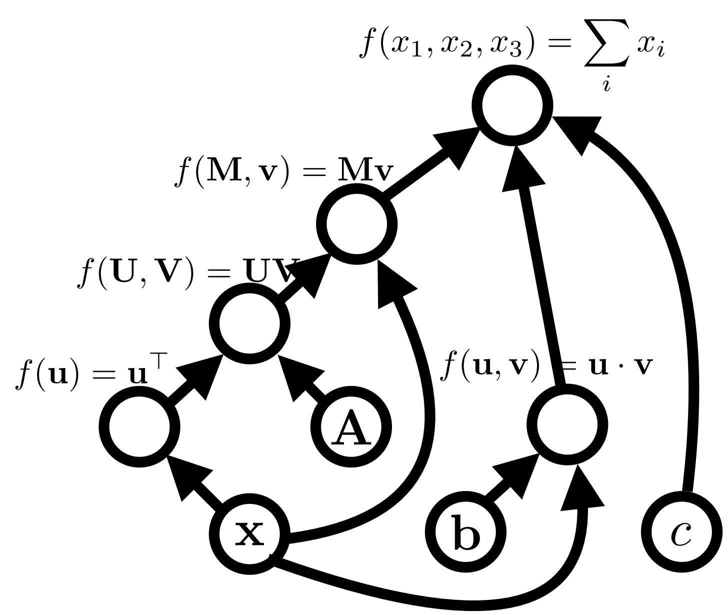


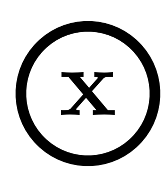
Image credit: Wikipedia

expression:

$$\mathbf{x}$$

graph:

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

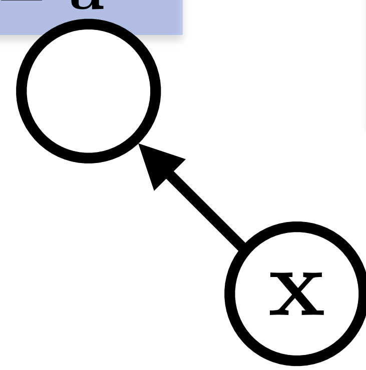


An **edge** represents a function argument (and also an data dependency). They are just pointers to nodes.

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

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



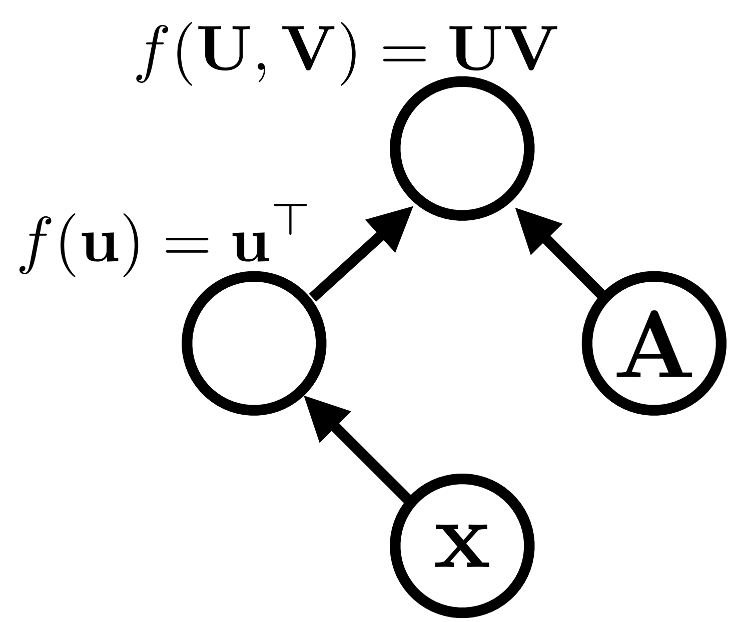
$$\frac{\partial f(\mathbf{u})}{\partial \mathbf{u}} \frac{\partial \mathcal{F}}{\partial f(\mathbf{u})} = \left(\frac{\partial \mathcal{F}}{\partial f(\mathbf{u})} \right)^\top$$

expression:

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

graph:

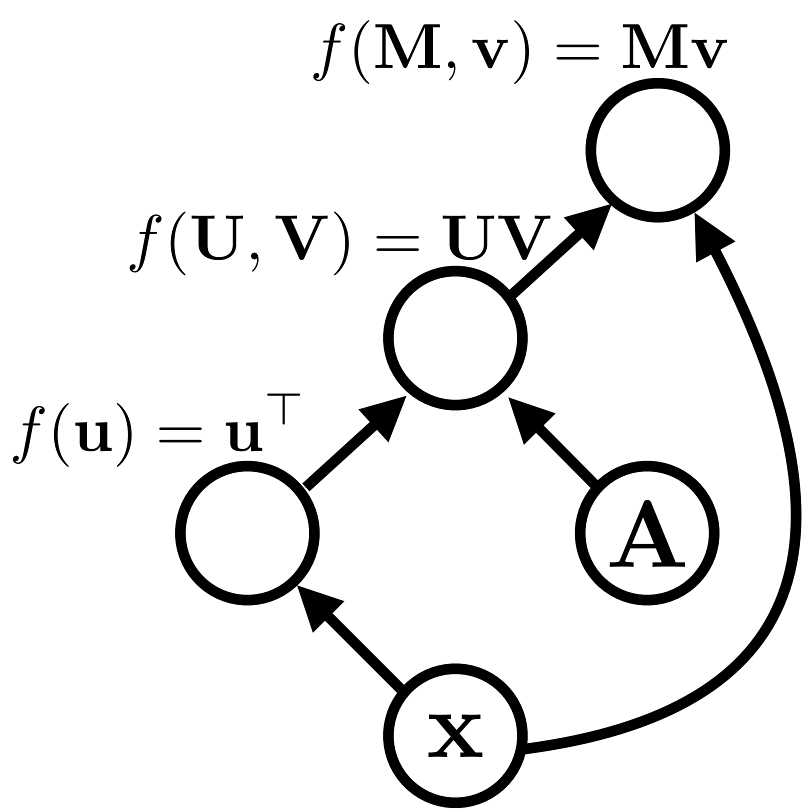
Functions can be nullary, unary, binary, … *n*-ary. Often they are unary or binary.



expression:

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

graph:

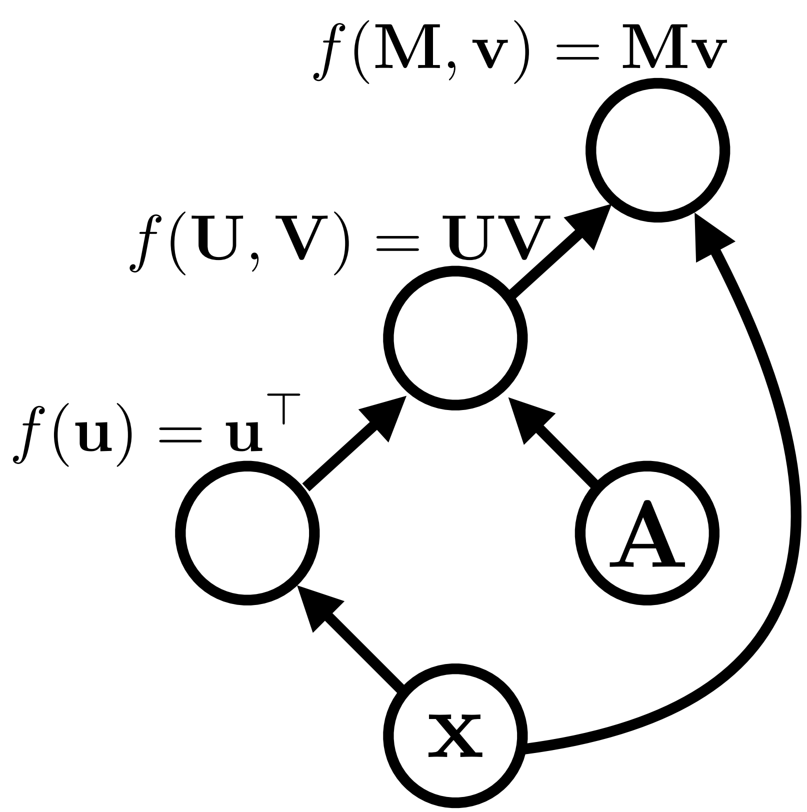


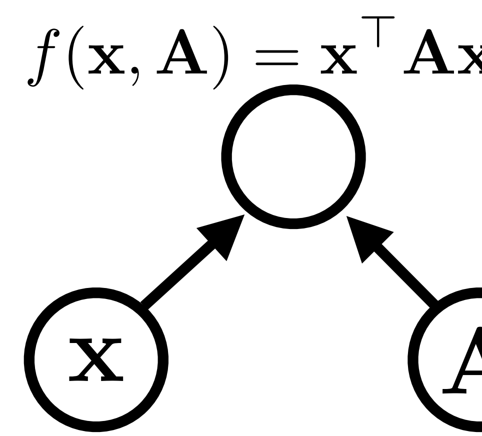
Computation graphs are generally directed and acyclic

expression:

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

graph:





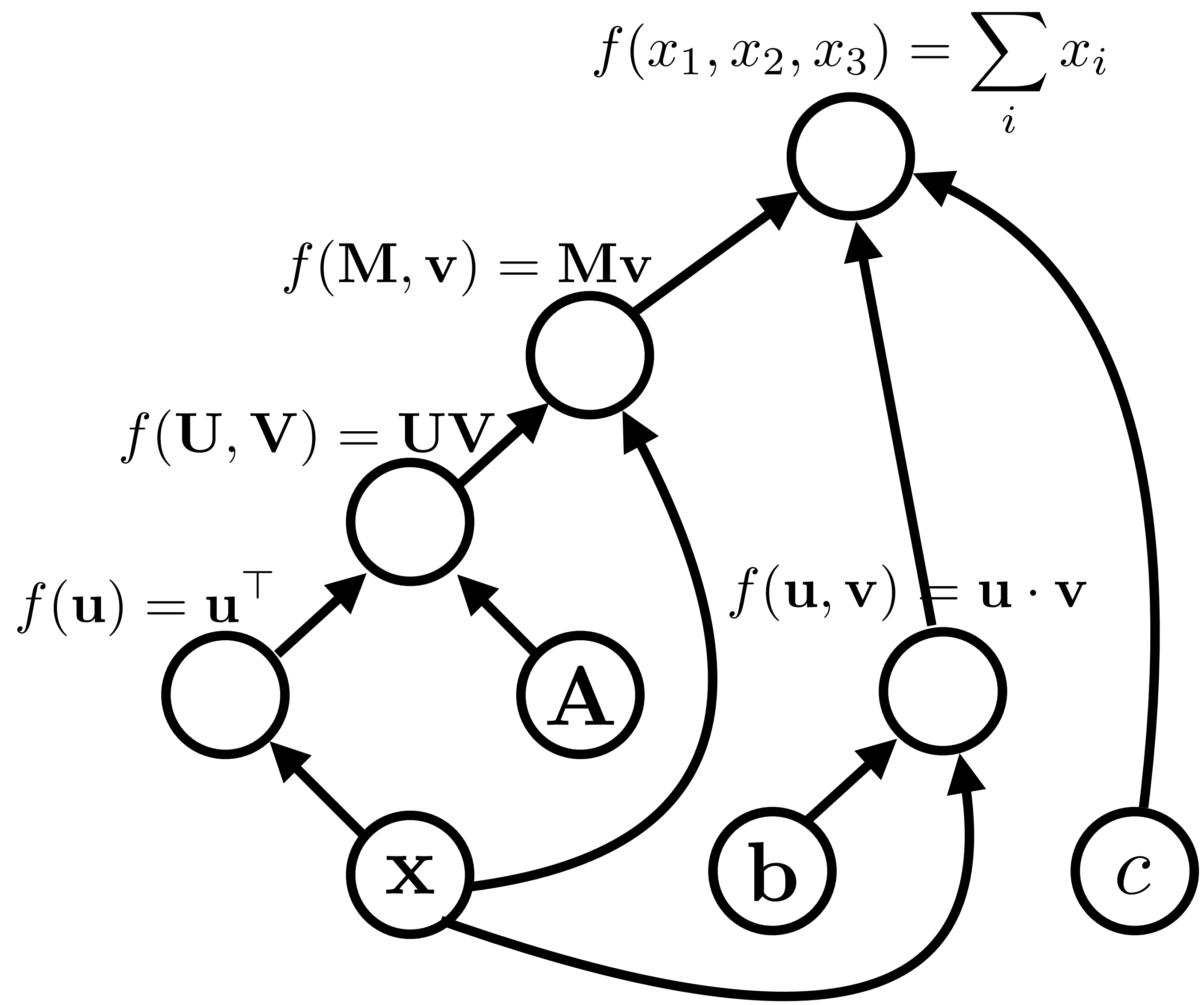
$$\frac{\partial f(\mathbf{x}, \mathbf{A})}{\partial \mathbf{x}} = (\mathbf{A}^\top + \mathbf{A})\mathbf{x}$$

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

expression:

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

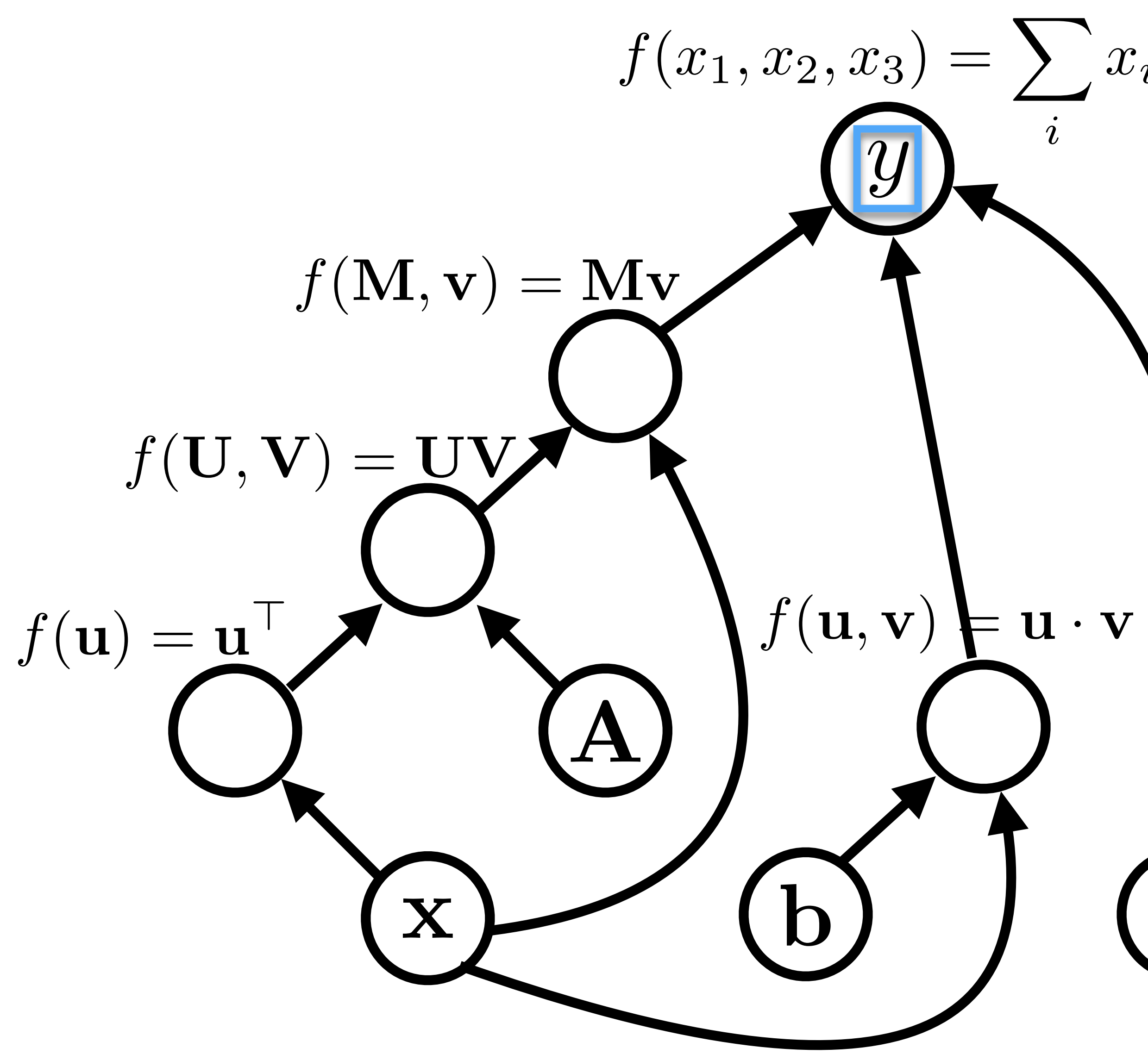
graph:



expression:

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

graph:



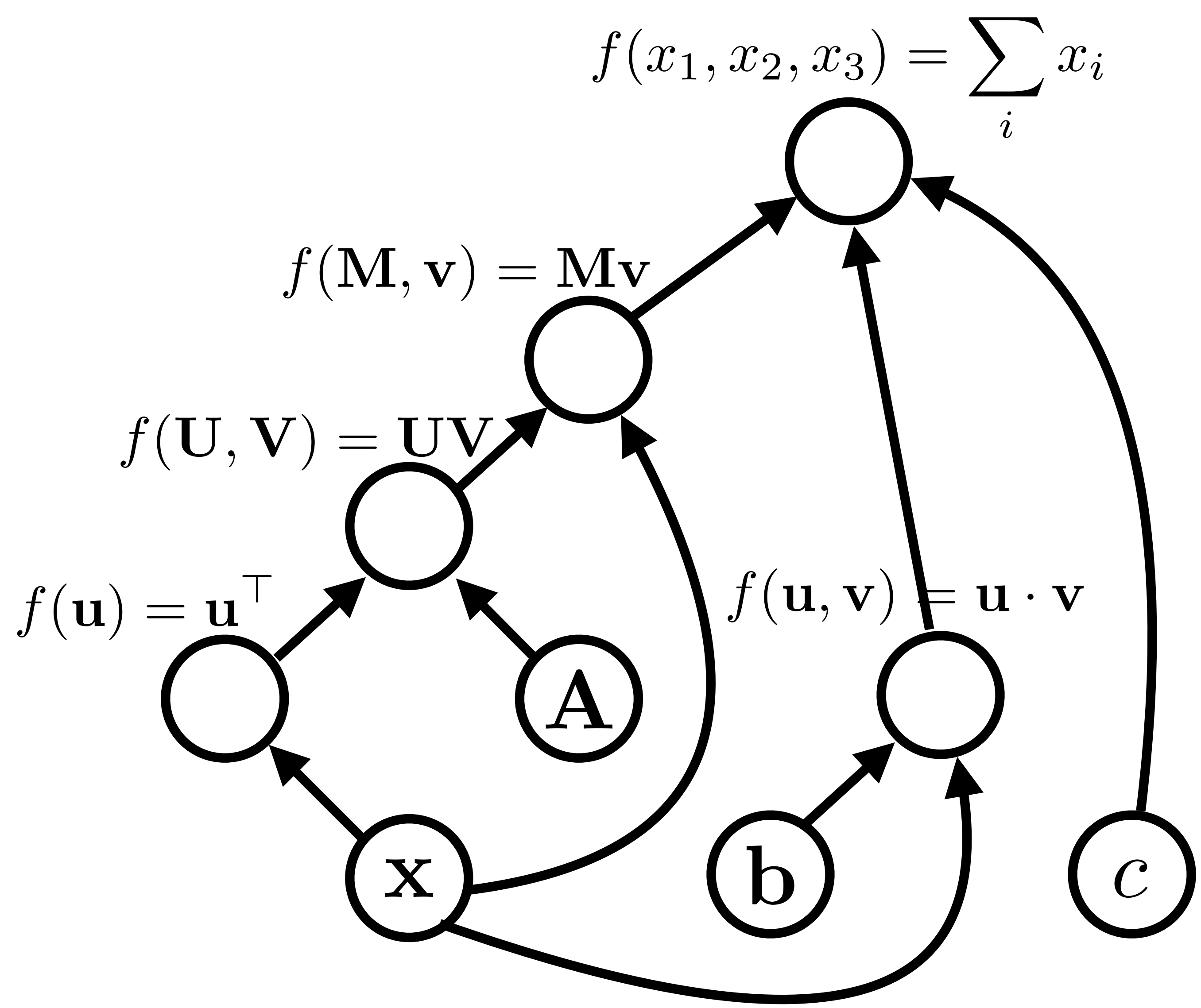
variable names are just labelings of nodes.

Algorithms (1)

- **Graph construction**
- **Forward propagation**
    - In topological order, compute the **value** of the node given its inputs

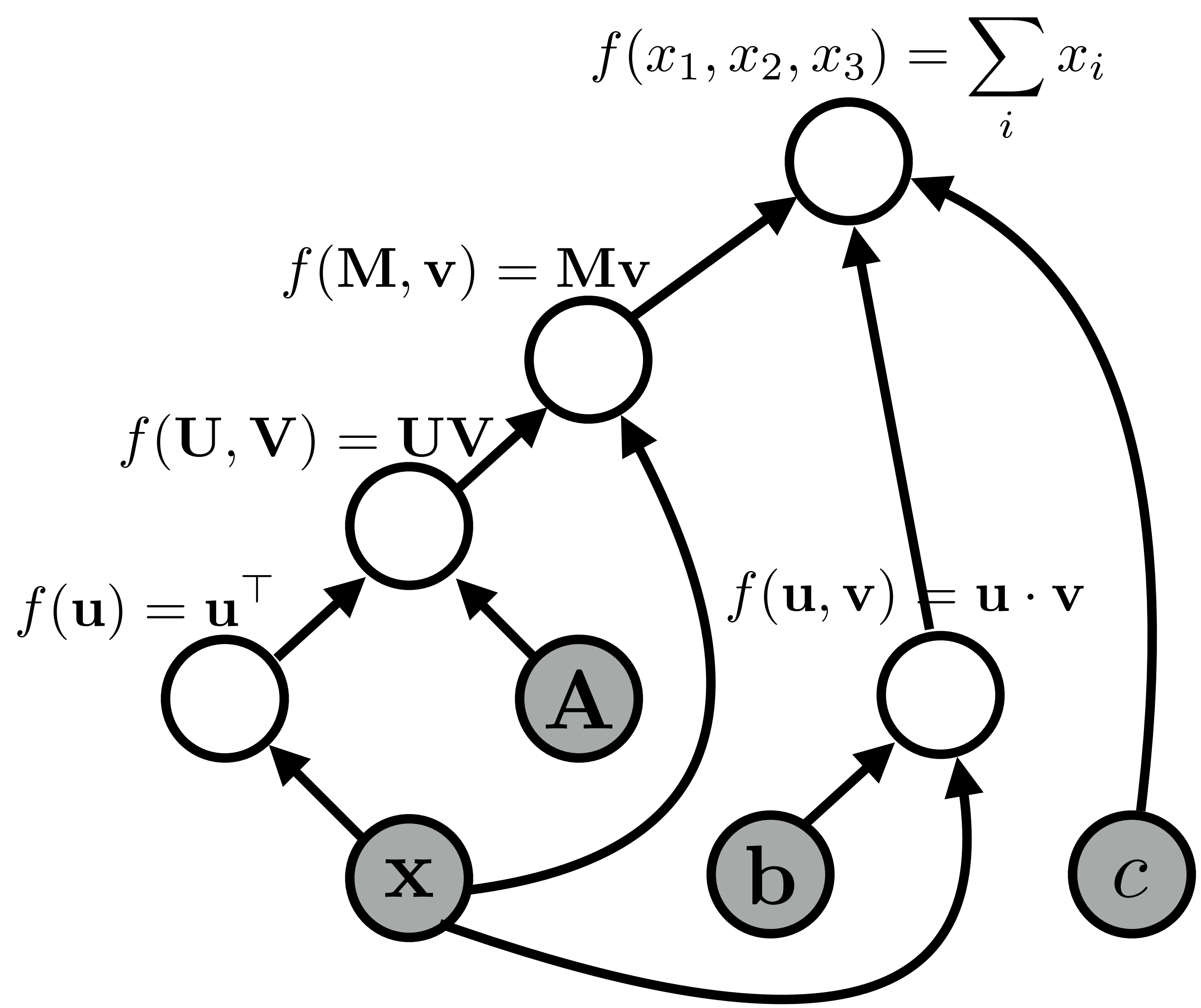
Forward Propagation

graph:



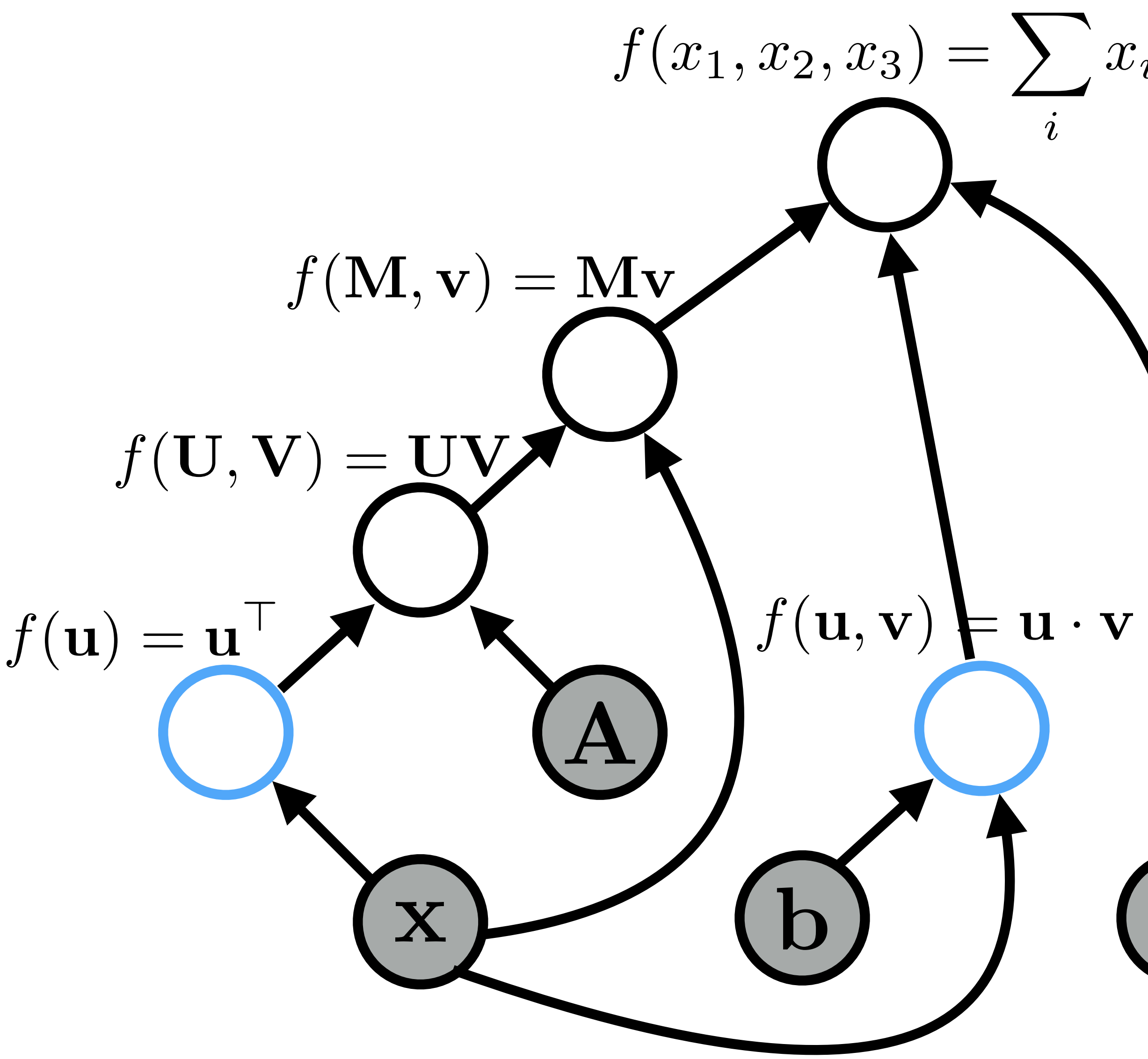
Forward Propagation

graph:



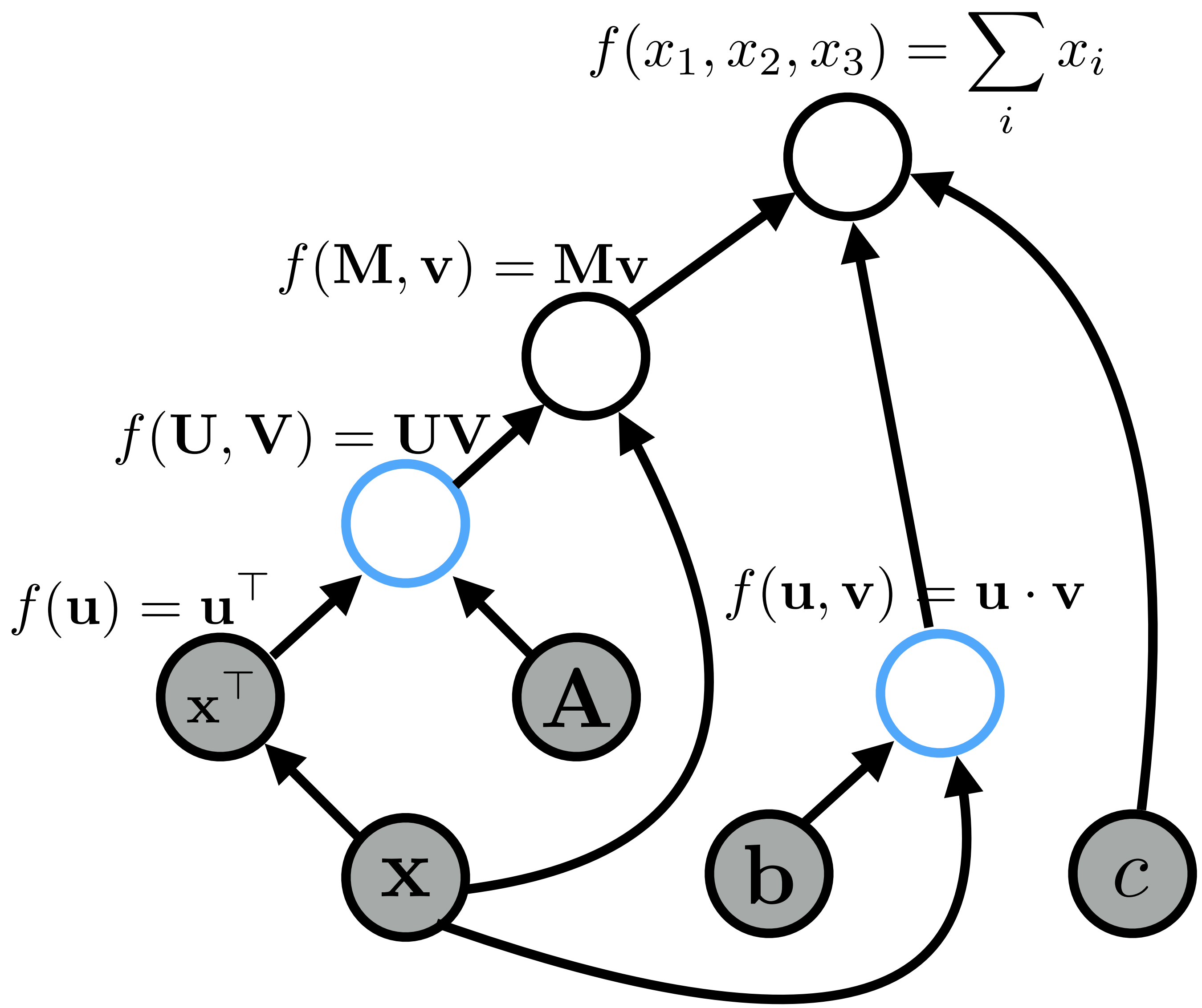
Forward Propagation

graph:



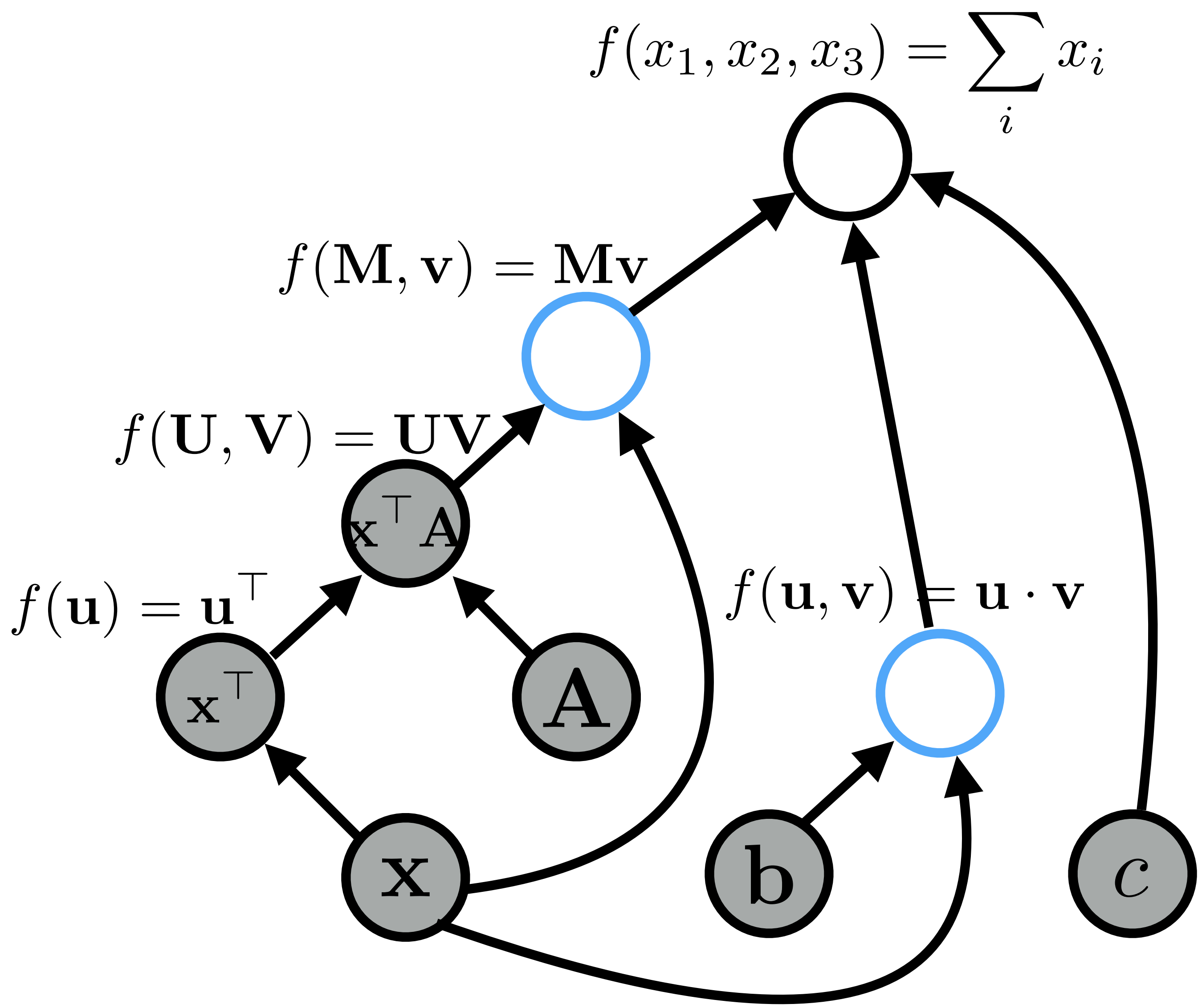
Forward Propagation

graph:



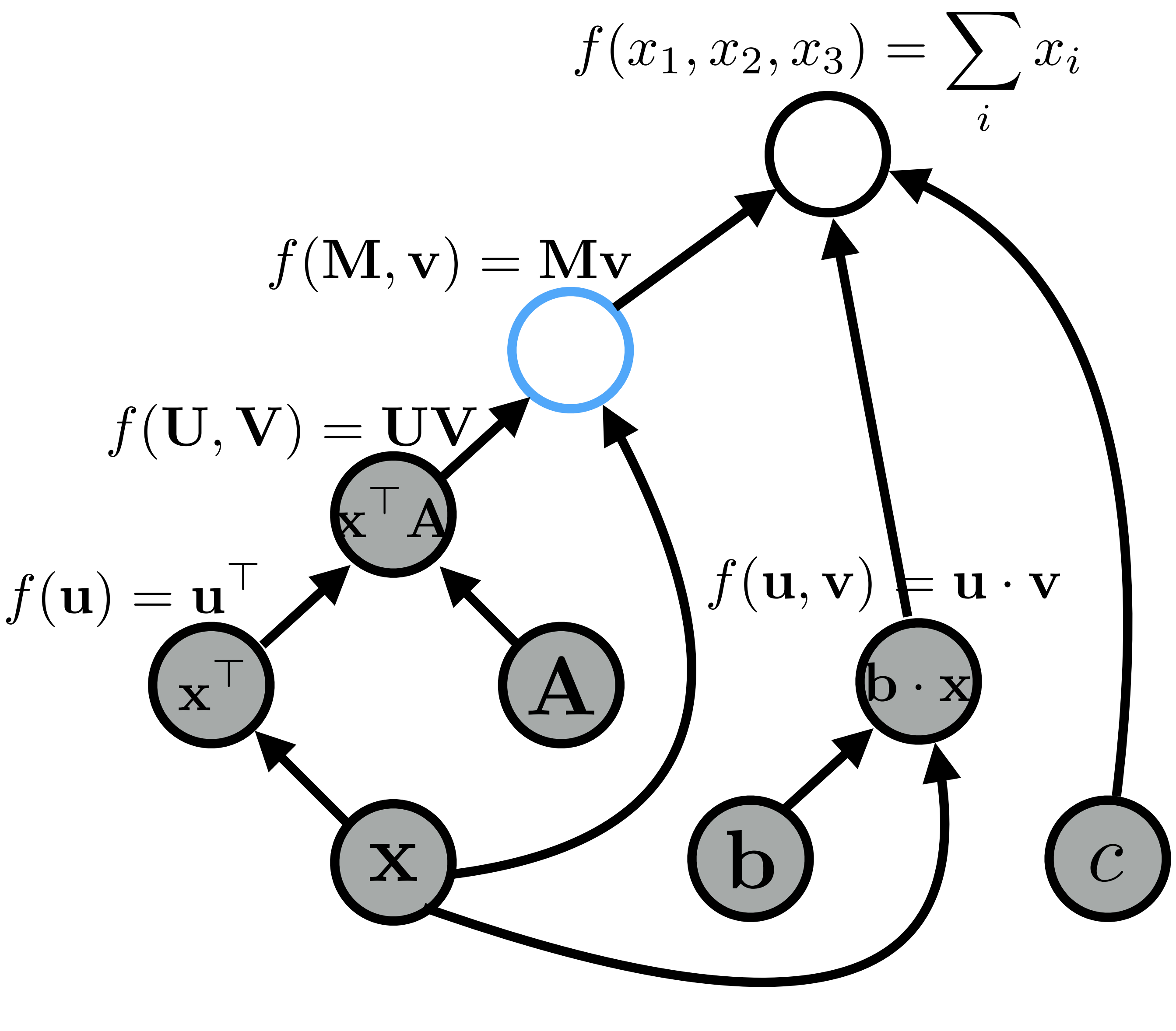
Forward Propagation

graph:



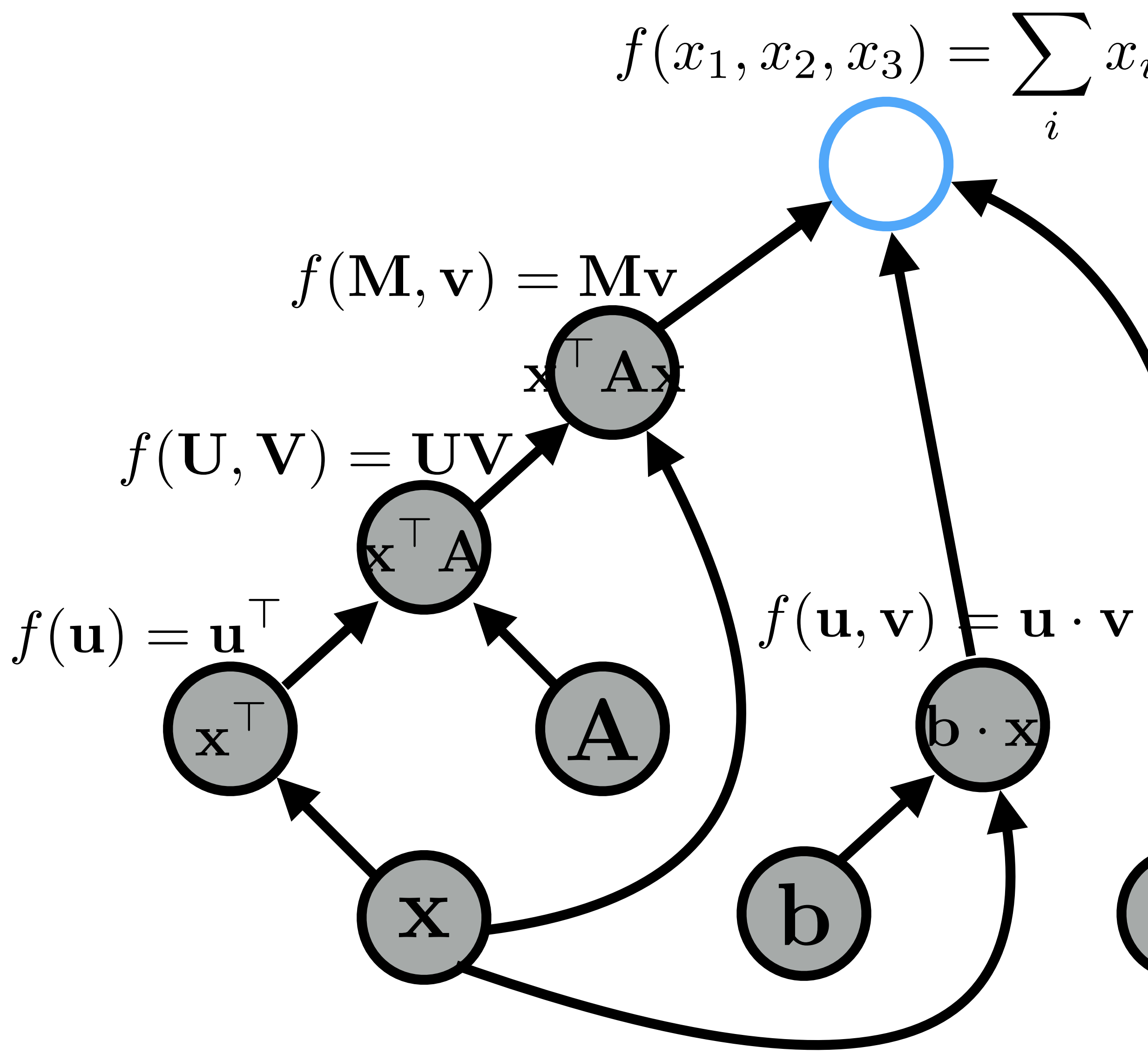
Forward Propagation

graph:



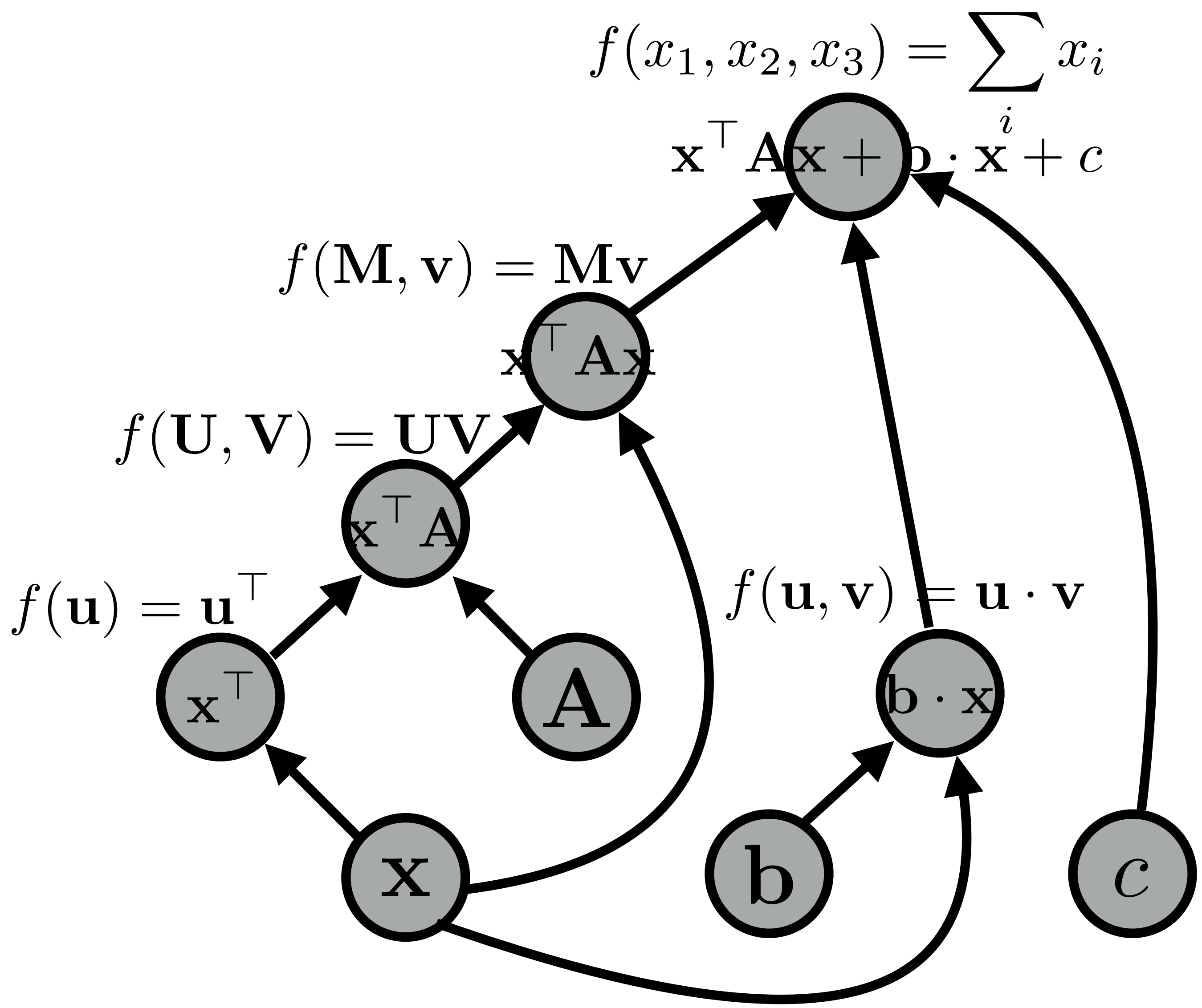
Forward Propagation

graph:



Forward Propagation

graph:

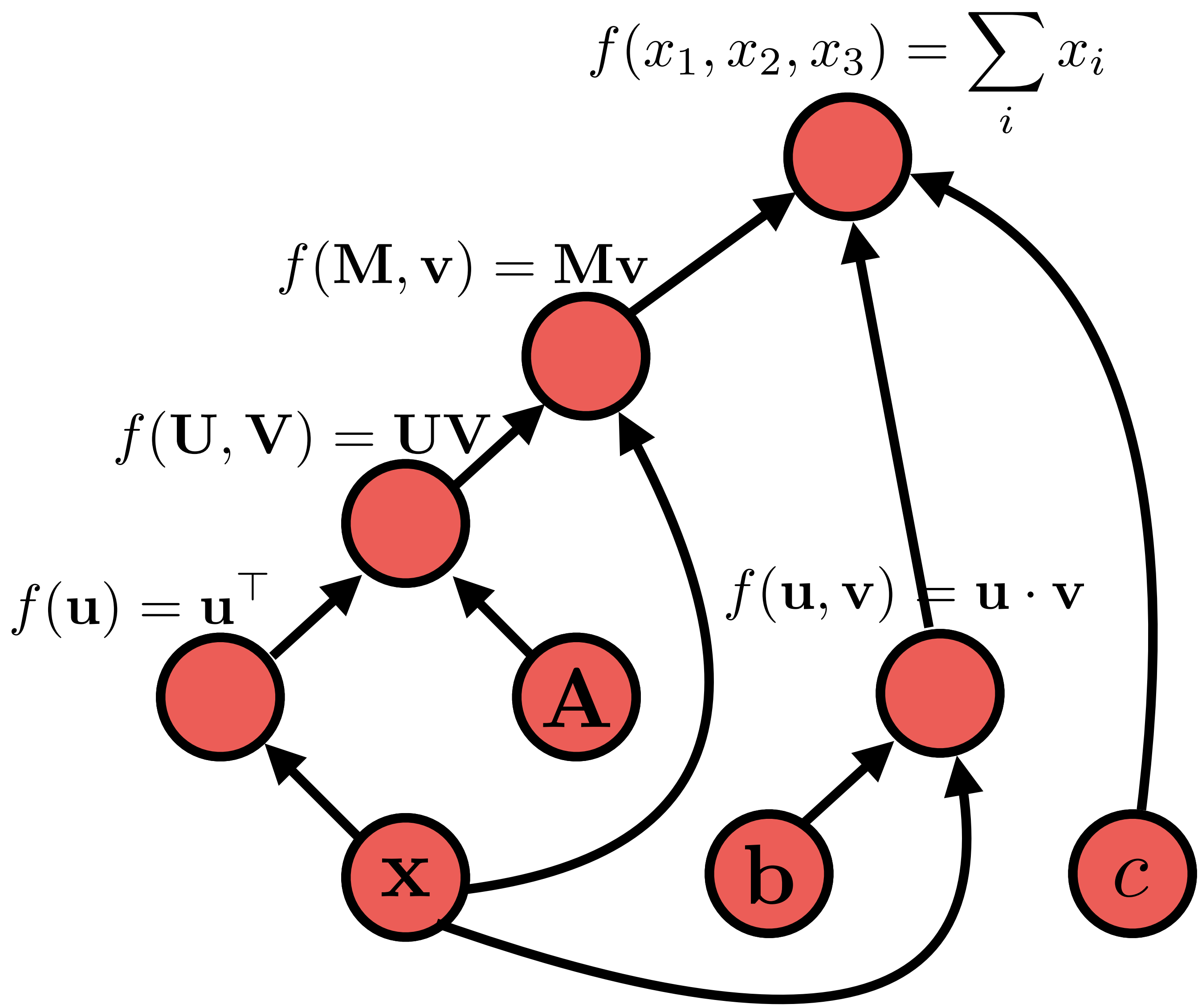


Algorithms (2)

- **Back-propagation:**
  - Process examples in reverse topological order
  - Calculate the derivatives of the parameters with respect to the final value
- **Parameter update:**
  - Move the parameters in the direction of this derivative
    W -= α * dl/dW

Back Propagation

graph:

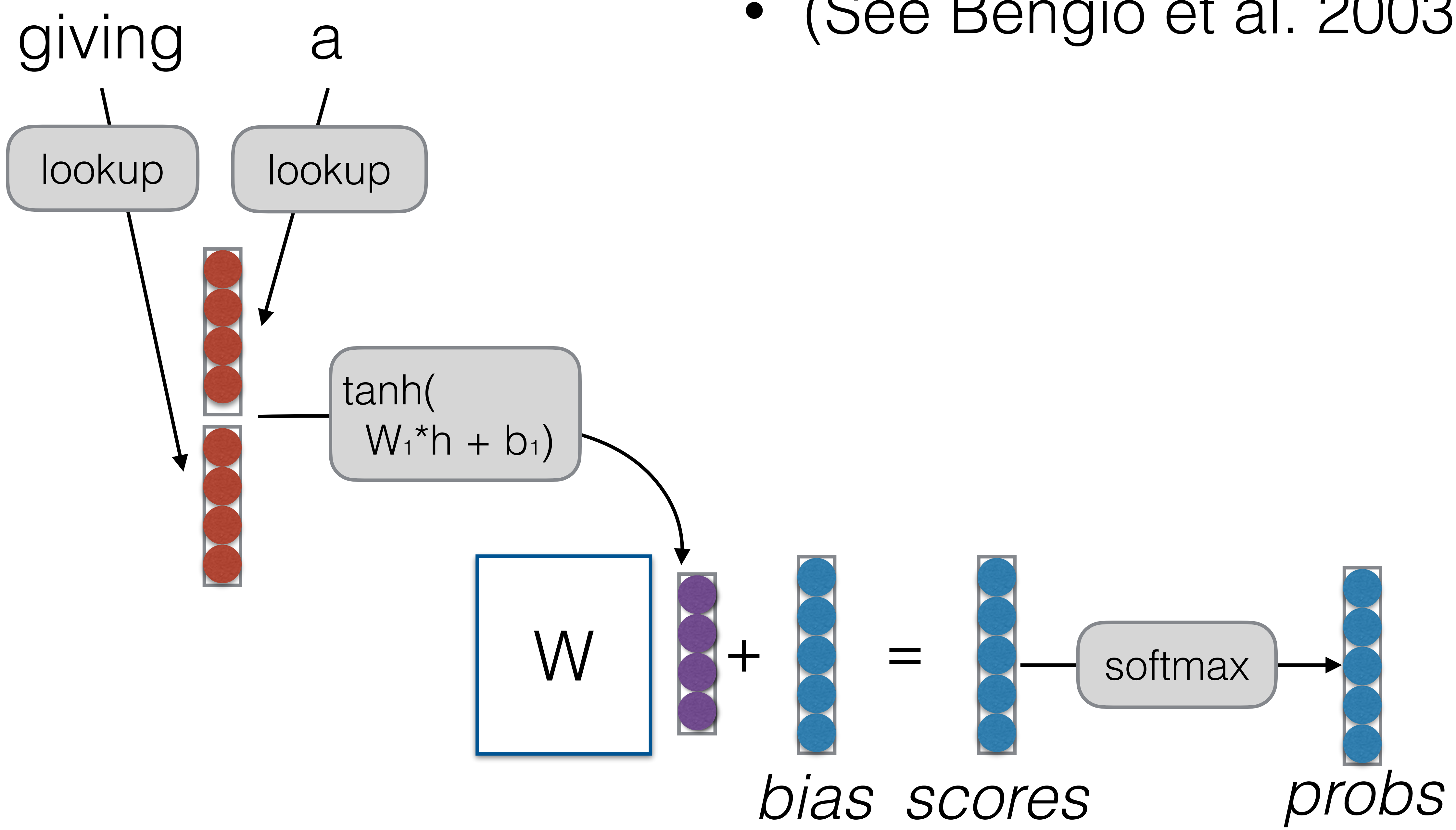


Much more detail next class!

Back to Language Modeling

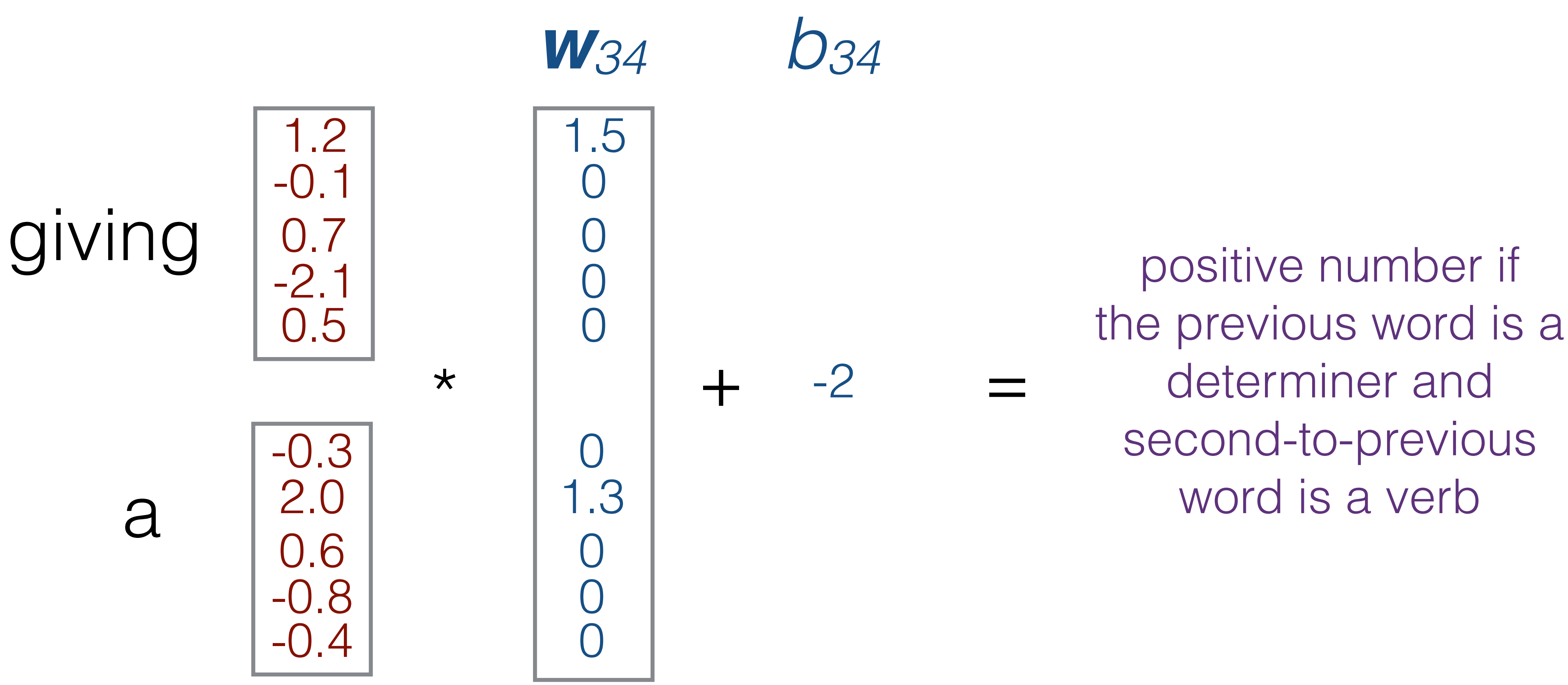
Feed-forward Neural Language Models

- (See Bengio et al. 2003)

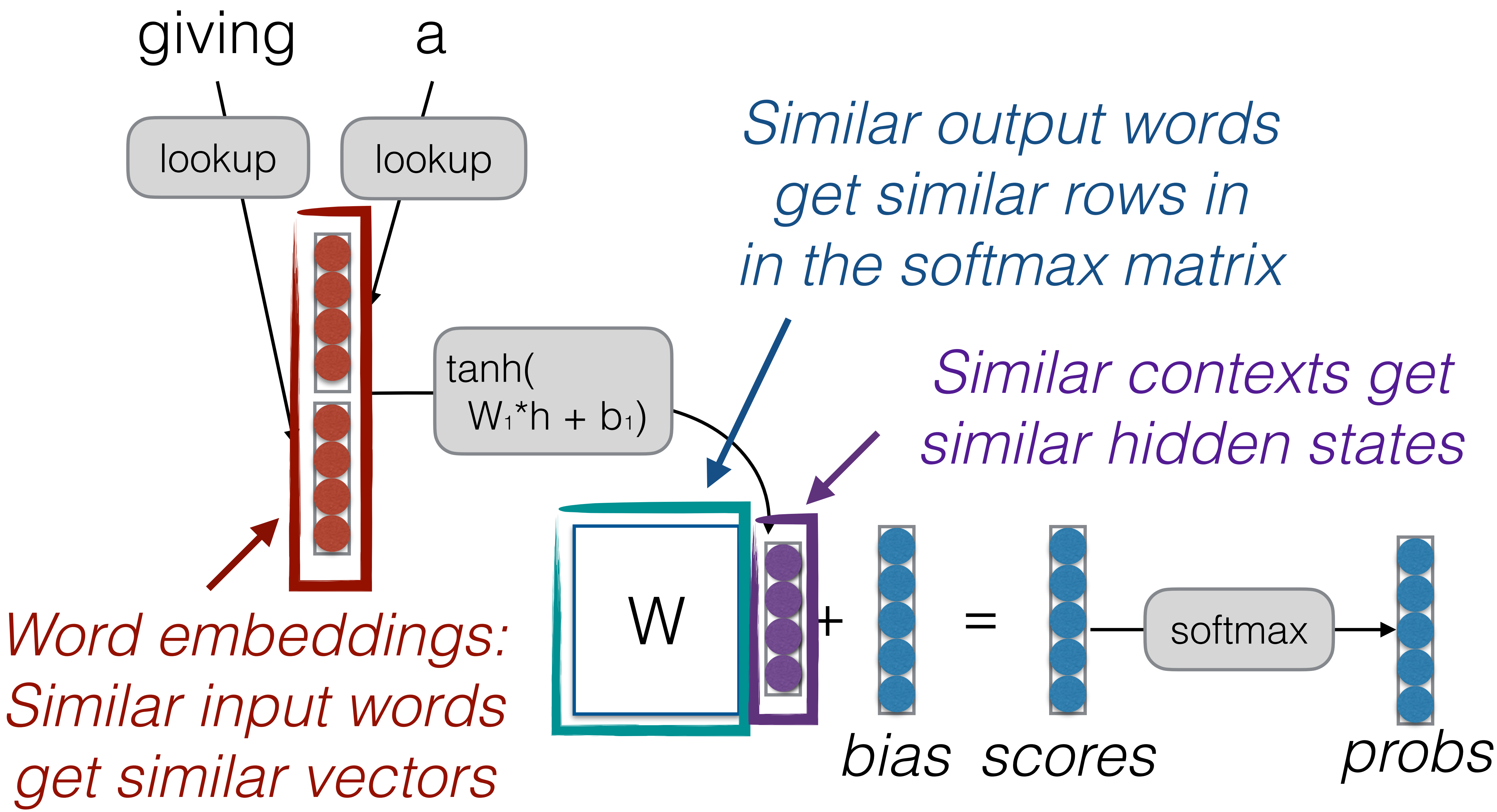


Example of Combination Features

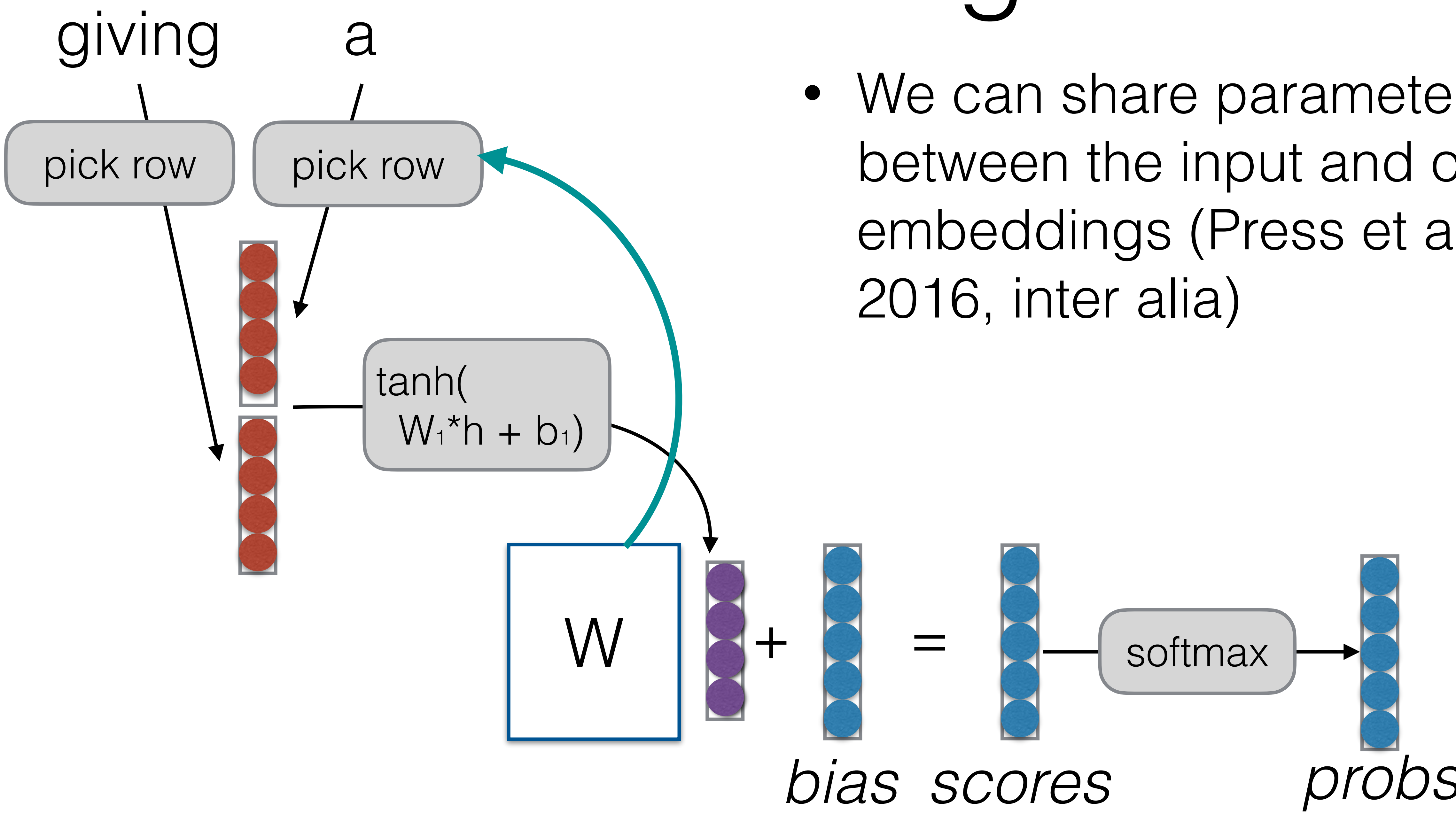
- Word embeddings capture features of words
  - e.g. feature 1 indicates verbs, feature 2 indicates determiners
- A row in the weight matrix (together with the bias) can capture particular *combinations* of these features
  - e.g. the 34th row in the weight matrix looks at feature 1 in the second-to-previous word, and feature 2 in the previous word



Where is Strength Shared?



Tying Input/Output Embeddings



- We can share parameters between the input and output embeddings (Press et al. 2016, inter alia)

Want to try? Delete the input embeddings, and instead pick a row from the softmax matrix.

What Problems are Handled?

- Cannot share strength among **similar words**

she bought a car she purchased a car	she bought a bicycle she purchased a bicycle

→ solved, and similar contexts as well! 😀

- Cannot condition on context with **intervening words**

Dr. Jane Smith	Dr. Gertrude Smith

→ solved! 😀

- Cannot handle **long-distance dependencies**

for tennis class he wanted to buy his own racquet

for programming class he wanted to buy his own computer

→ not solved yet 😞

Many Other Potential Designs!

- Neural networks allow design of arbitrarily complex functions!
- In future classes:
  - **Recurrent neural network LMs**
  - **Transformer LMs**

LM Problem Definition

Count-based LMs

Evaluating LMs

Log-linear LMs

Neural Net Basics

Feed-forward NN LMs

Questions?