CS11-747 Neural Networks for NLP

Intro/
Why Neural Nets for NLP?

Graham Neubig

Carnegie Mellon University
Language Technologies Institute

Site
https://phontron.com/class/nn4nlp2017/

Language is Hard!

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

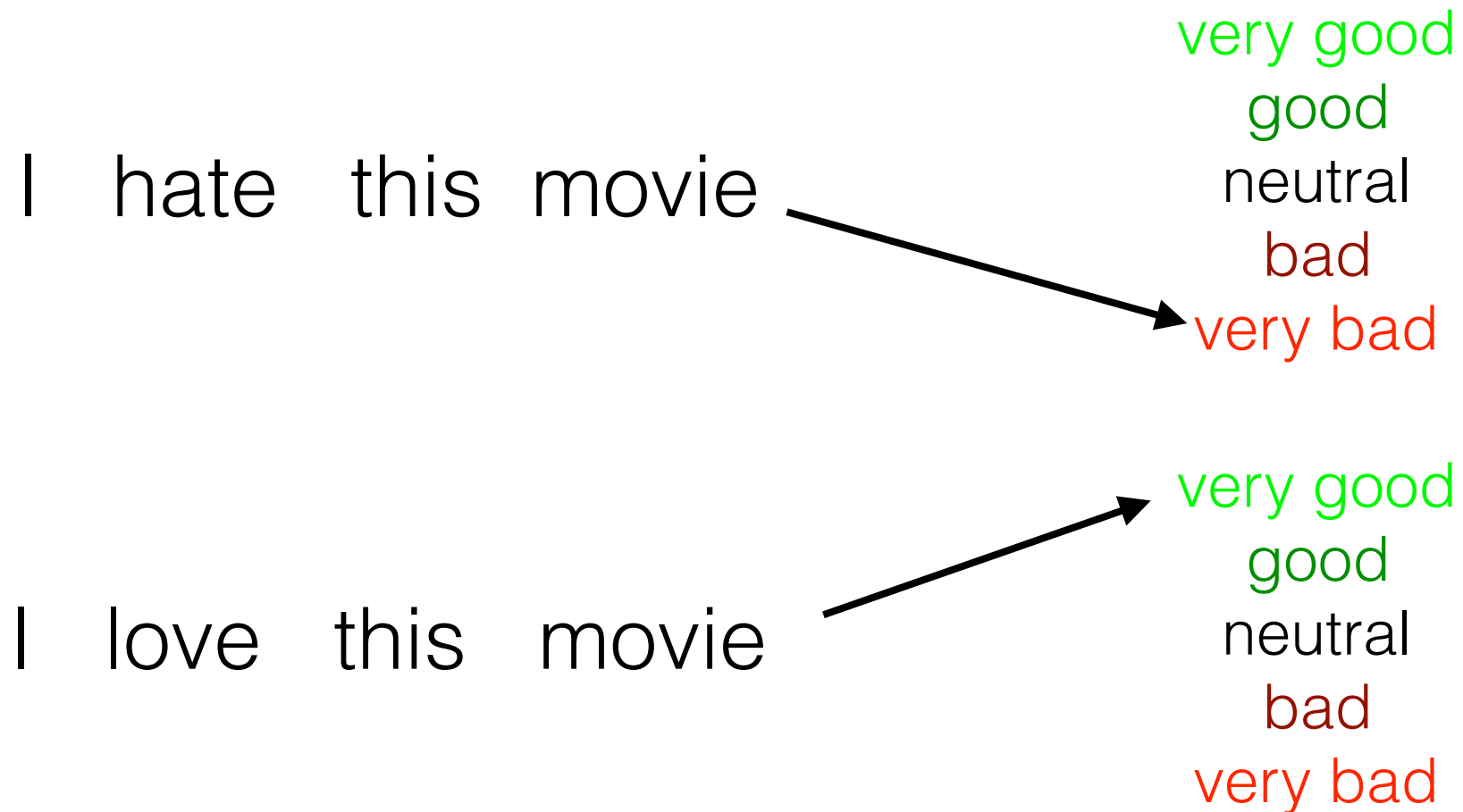# Are These Sentences OK?

- ジェインは店へ行った。

- は店行ったジェインは。

- ジェインは店へ行た。

- 店はジェインへ行った。
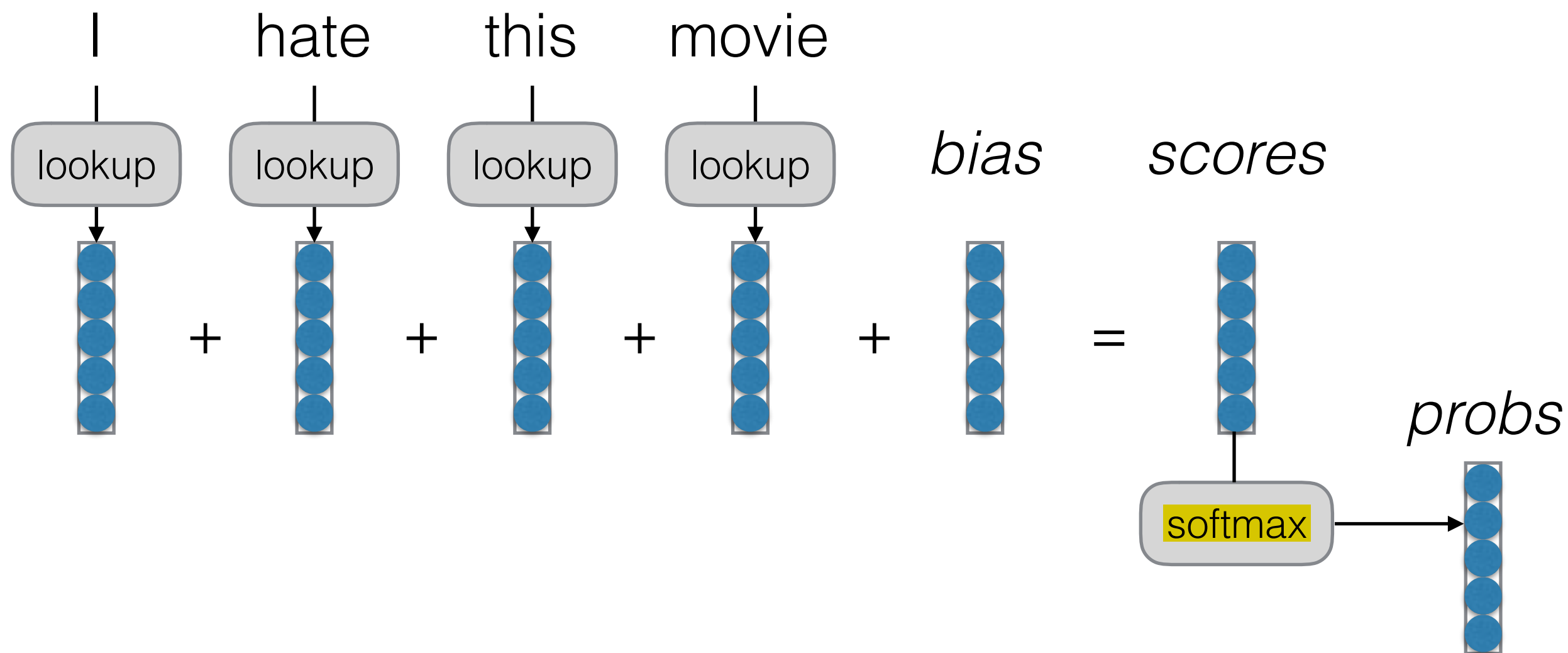
- 屋台はジェインのところへ行った。

Phenomena to Handle

- Morphology

- Syntax

- Semantics/World Knowledge

- Discourse

- Pragmatics

- Multilinguality

Neural Networks:
A Tool for Doing Hard Things

An Example Prediction Problem: Sentence Classification

I hate this movie ⟶

very good
good
neutral
bad
very bad

I love this movie ⟶

very good
good
neutral
bad
very bad

A First Try:
Bag of Words (BOW)

I   hate   this   movie

lookup   lookup   lookup   lookup   *bias*   *scores*

$+$ $+$ $+$ $+$ $=$

probs

softmax

Build It, Break It

I don't love this movie →
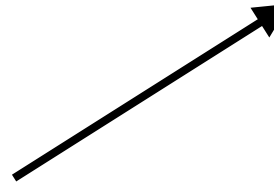
very good
good
neutral
bad
very bad

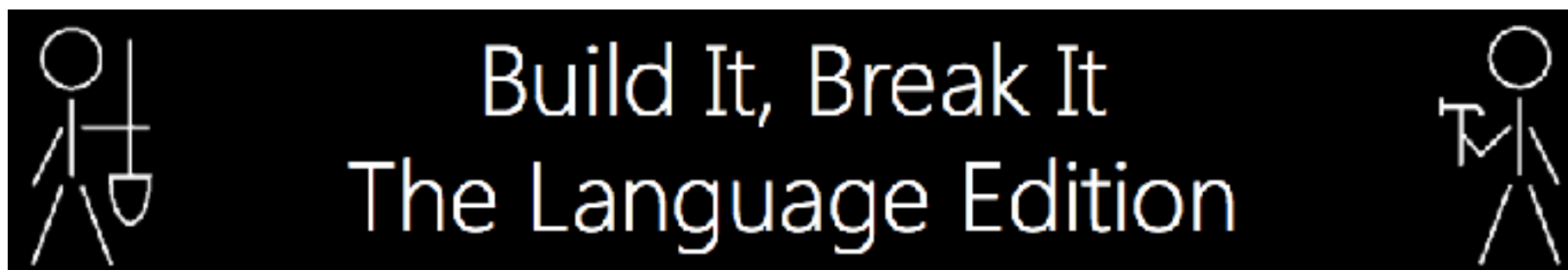There's nothing I don't love about this movie →

very good
good
neutral
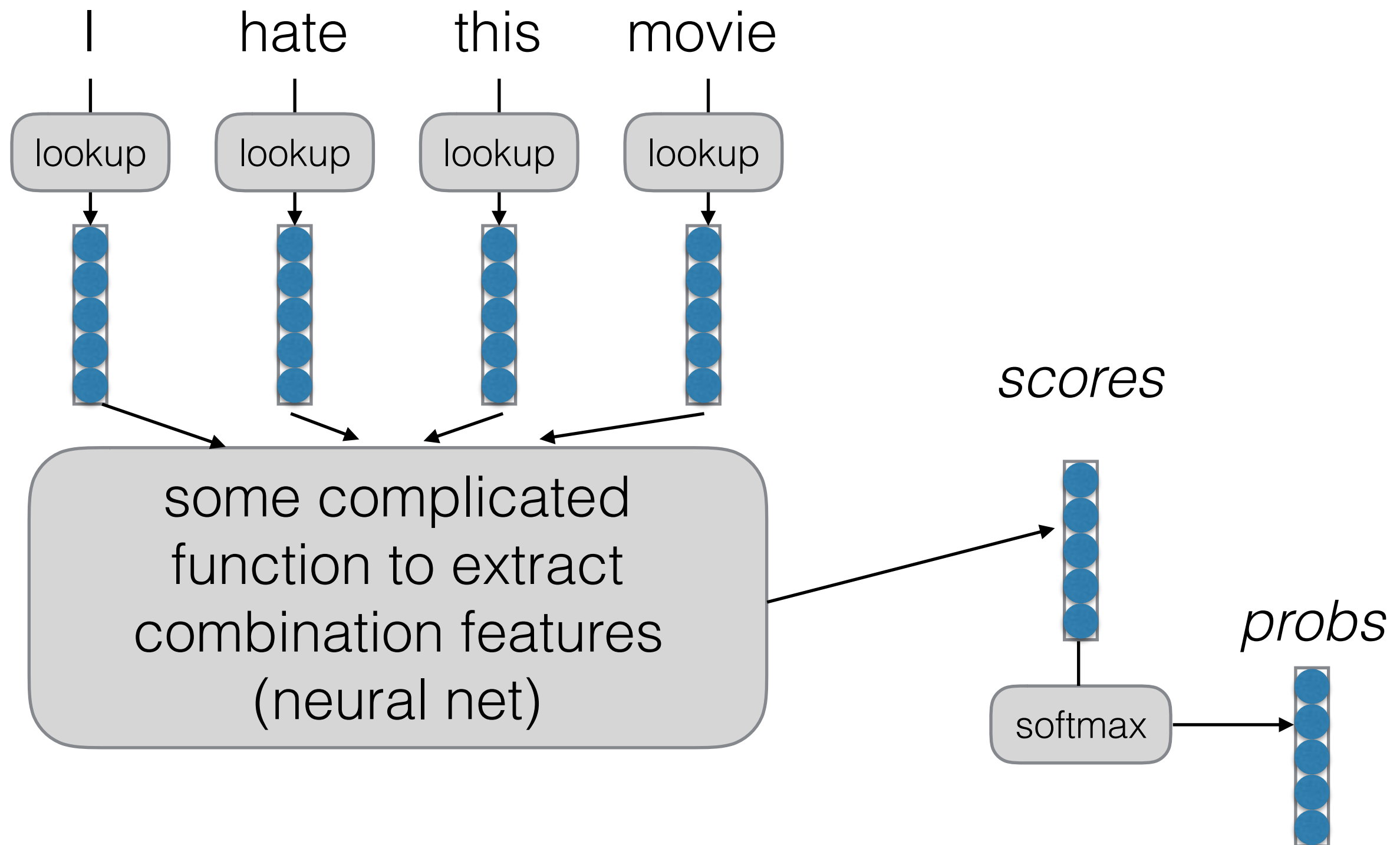bad
very bad

Build It, Break It
The Language Edition

https://bibinlp.umiacs.umd.edu

Combination Features

- Does it contain "don't" and "love"?

- Does it contain "don't", "i", "love", and "nothing"?

Basic Idea of Neural Networks
(for NLP Prediction Tasks)

Computation Graphs

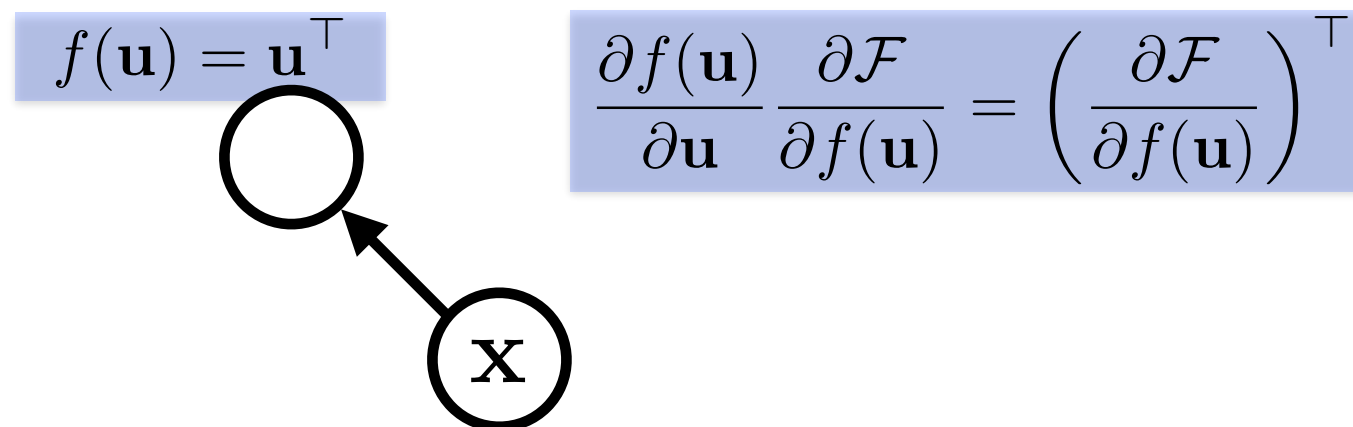## The Lingua Franca of Neural Nets

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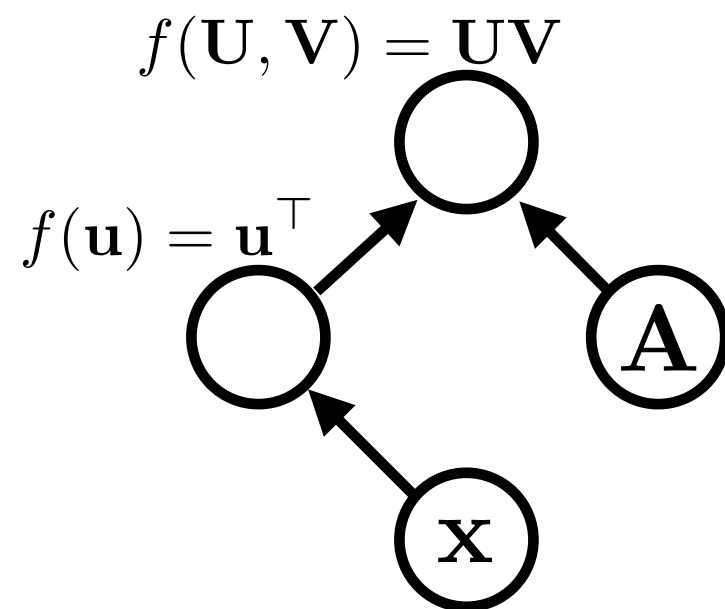
$$f(\mathbf{u}) = \mathbf{u}^\top$$

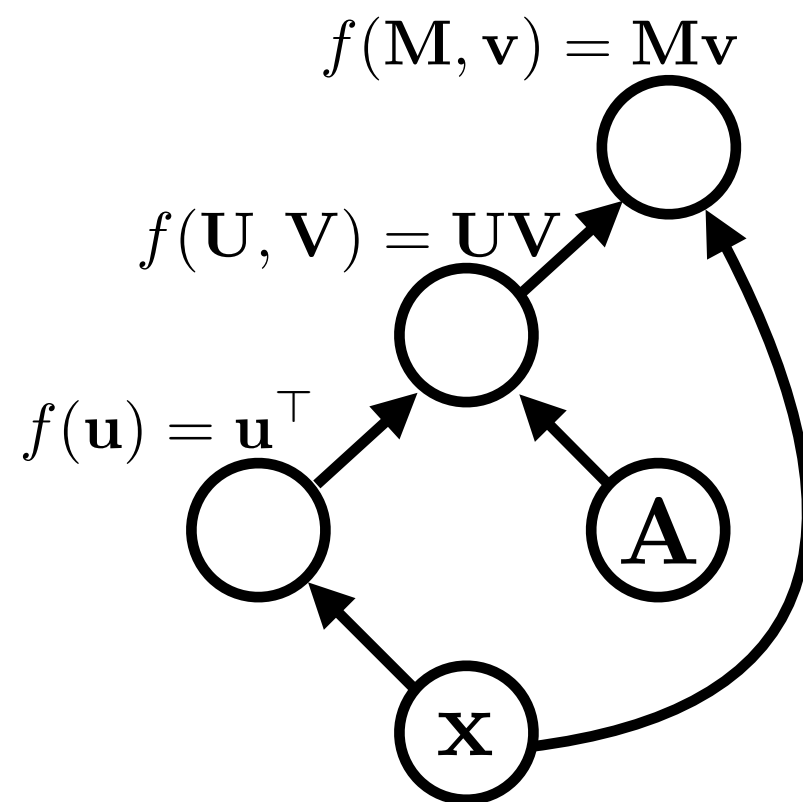$$\frac{\partial f(\mathbf{u})}{\partial \mathbf{u}} \frac{\partial \mathcal{F}}{\partial f(\mathbf{u})} = \left(\frac{\partial \mathcal{F}}{\partial f(\mathbf{u})} \right)^\top$$

expression:

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

graph:

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

$f(\mathbf{U}, \mathbf{V}) = \mathbf{UV}$

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

expression:

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

graph:
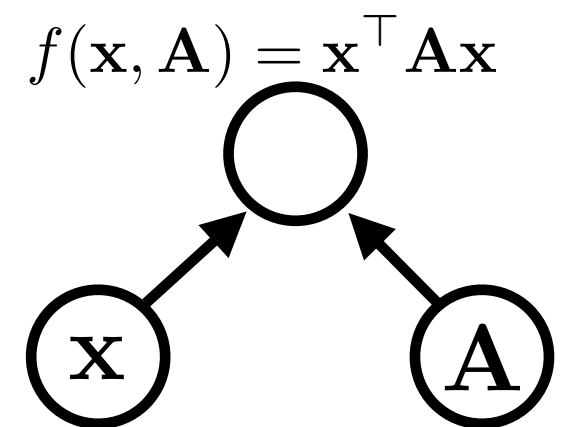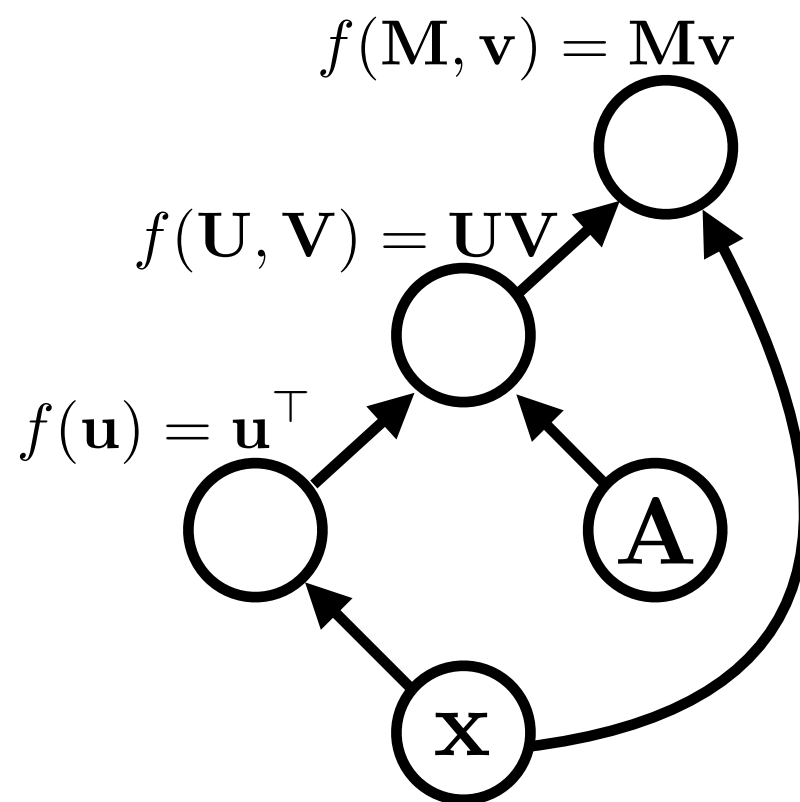

Computation graphs are directed and acyclic (in DyNet)

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

graph:

$f(\mathbf{M}, \mathbf{v}) = \mathbf{M}\mathbf{v}$

$f(\mathbf{U}, \mathbf{V}) = \mathbf{U}\mathbf{V}$

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

$\mathbf{A}$

$\mathbf{x}$

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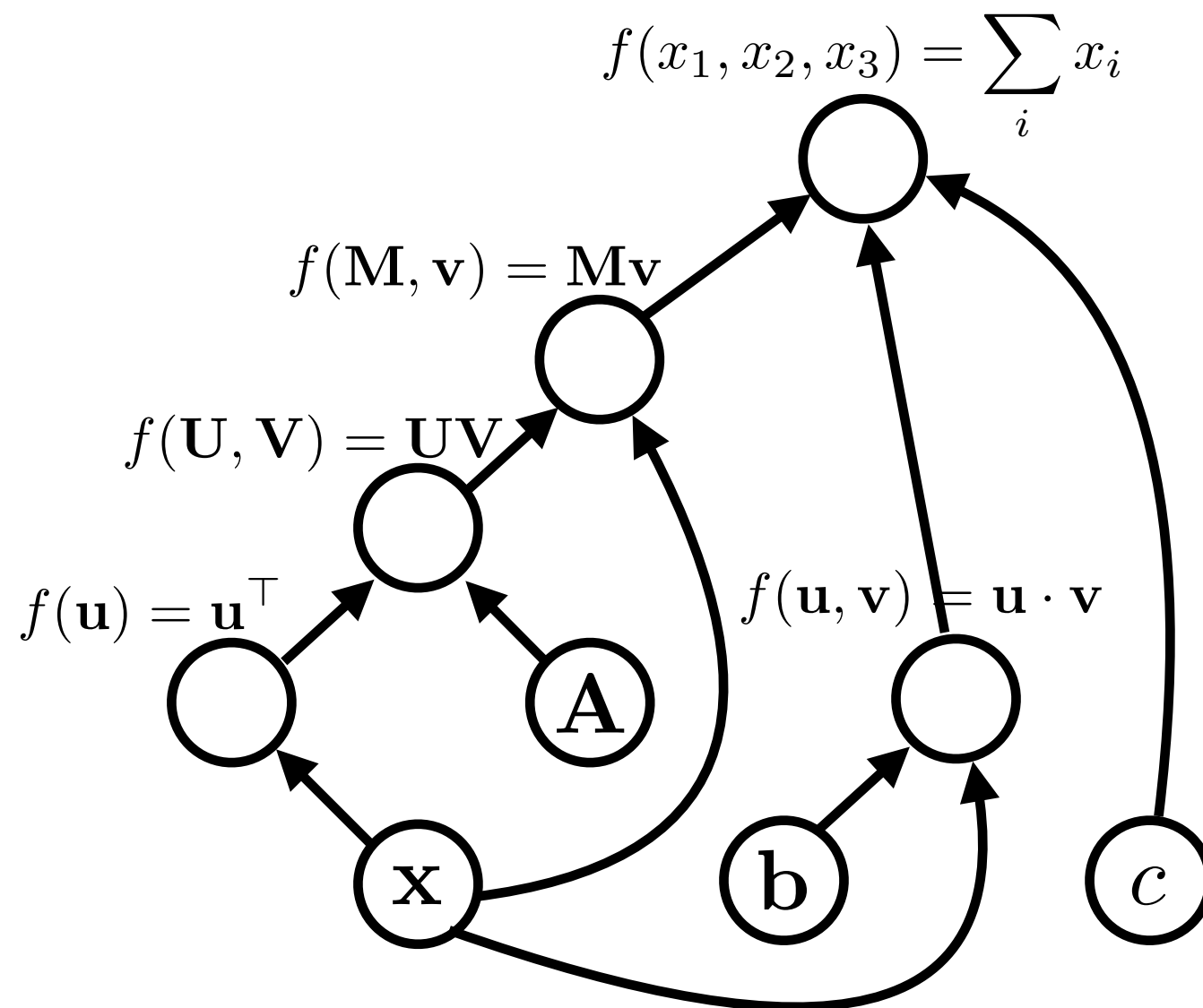
$\mathbf{x}$

$\mathbf{A}$

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

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

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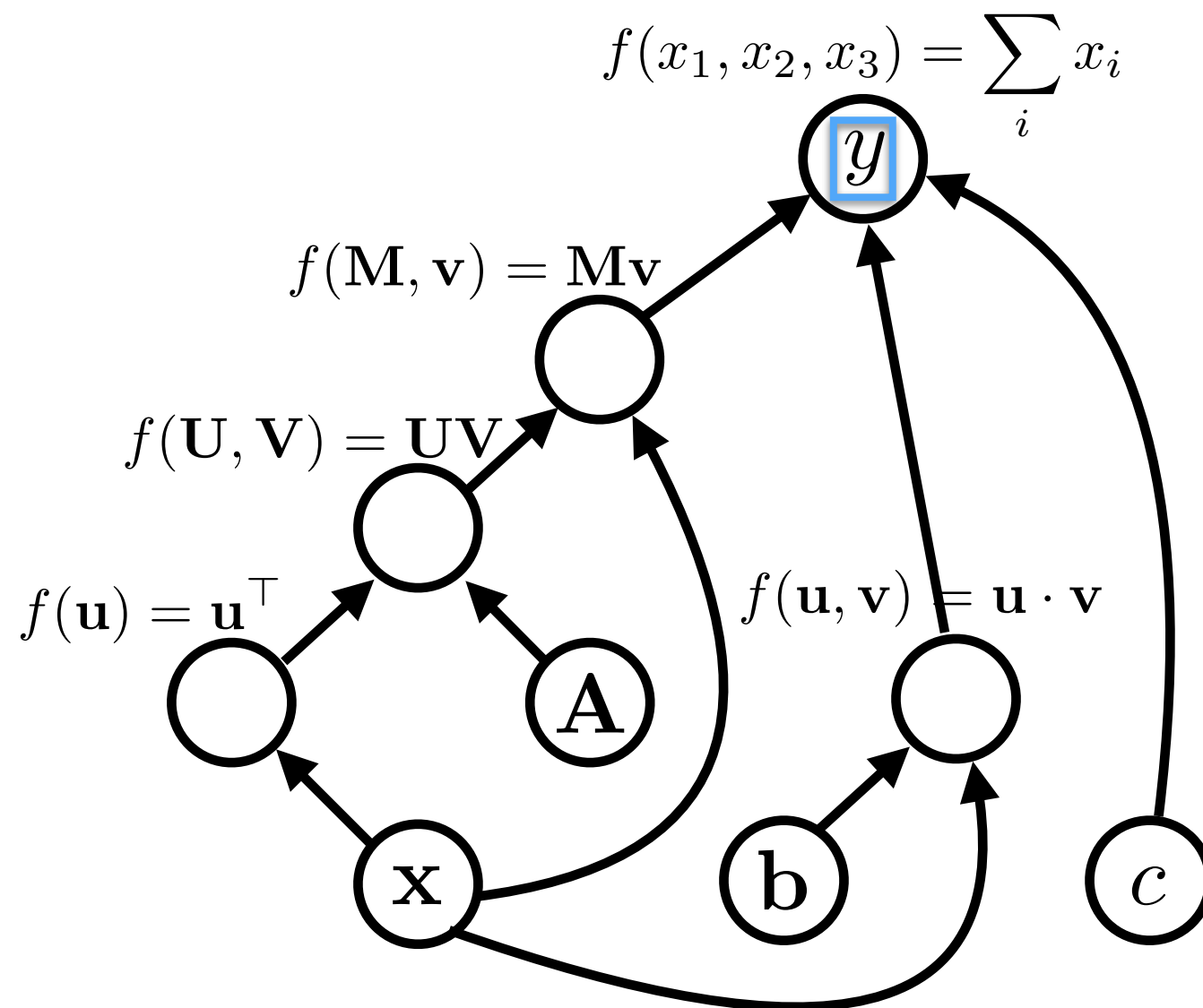$$\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c$$

graph:

expression:

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

graph:

$f(x_1, x_2, x_3) = \sum_i x_i$

$f(\mathbf{M}, \mathbf{v}) = \mathbf{M}\mathbf{v}$

$f(\mathbf{U}, \mathbf{V}) = \mathbf{U}\mathbf{V}$

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

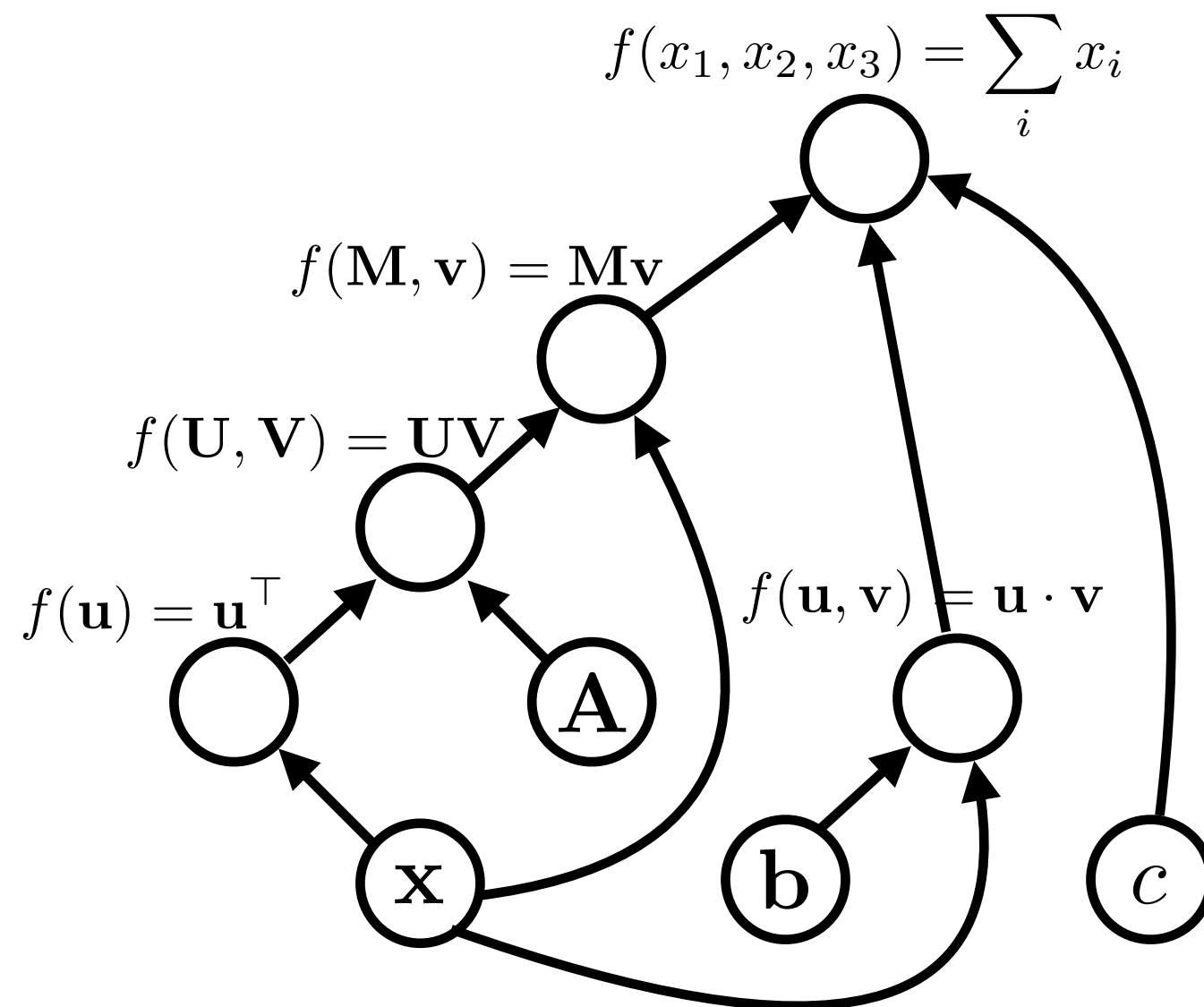$f(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$

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
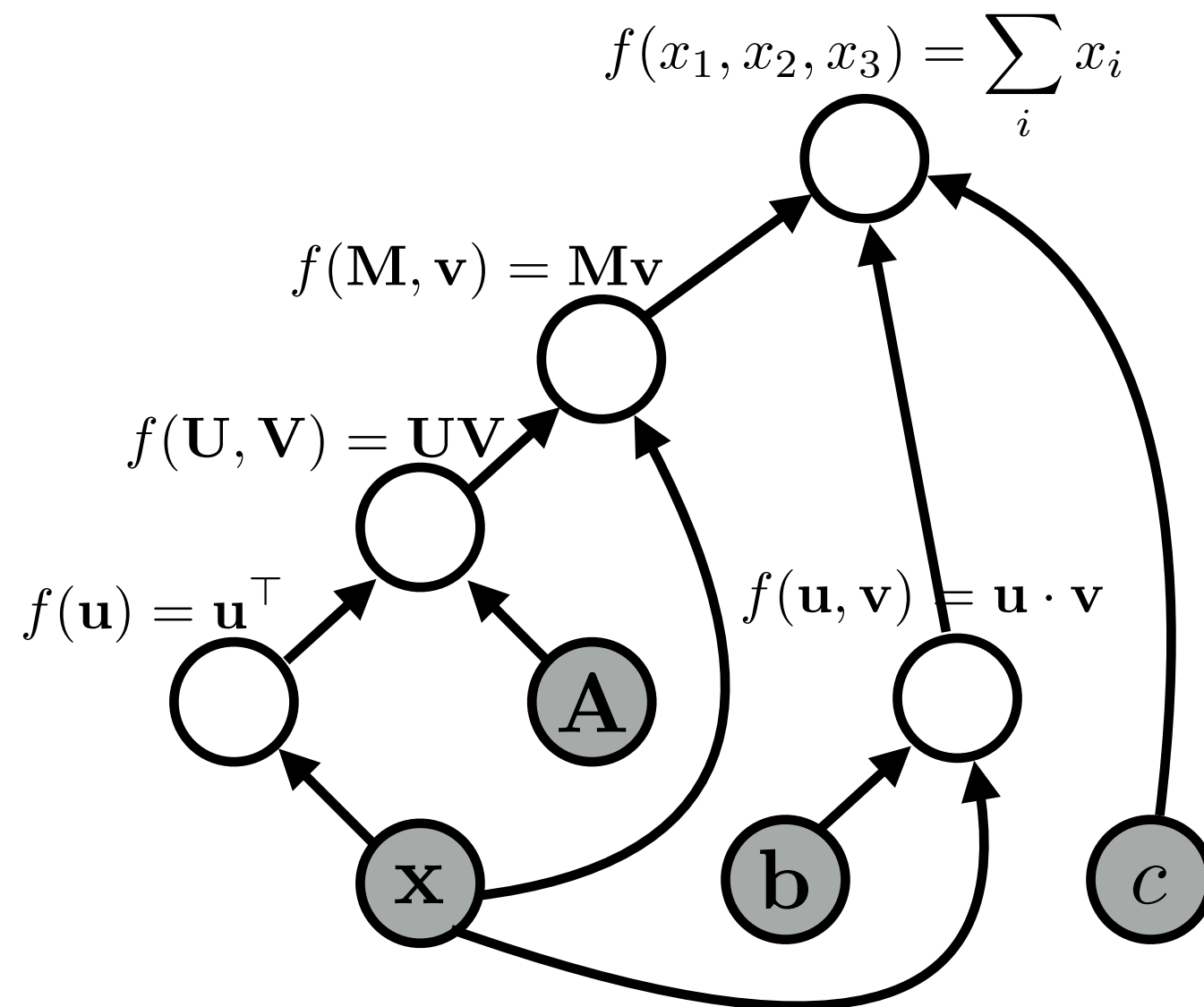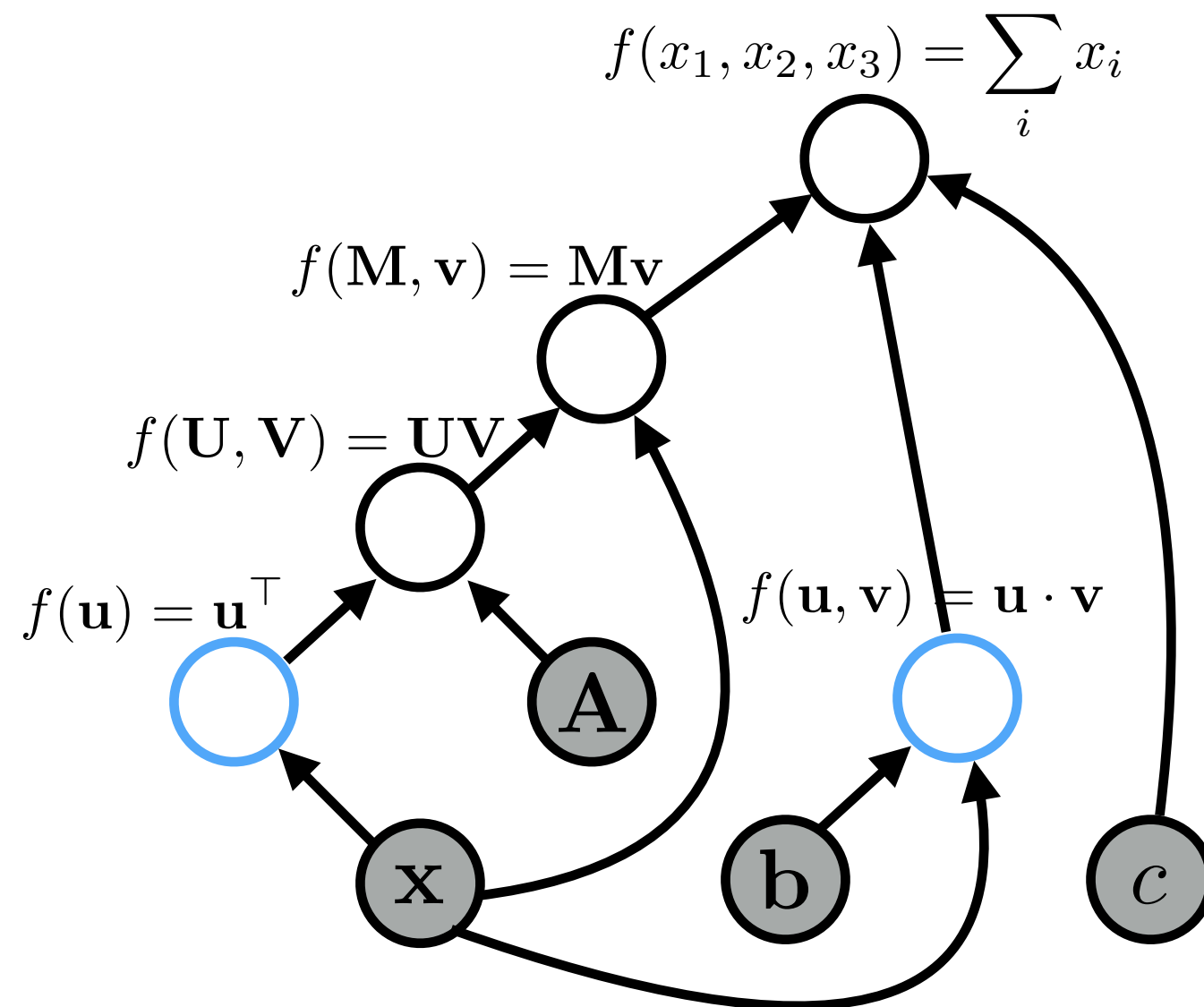
Forward Propagation
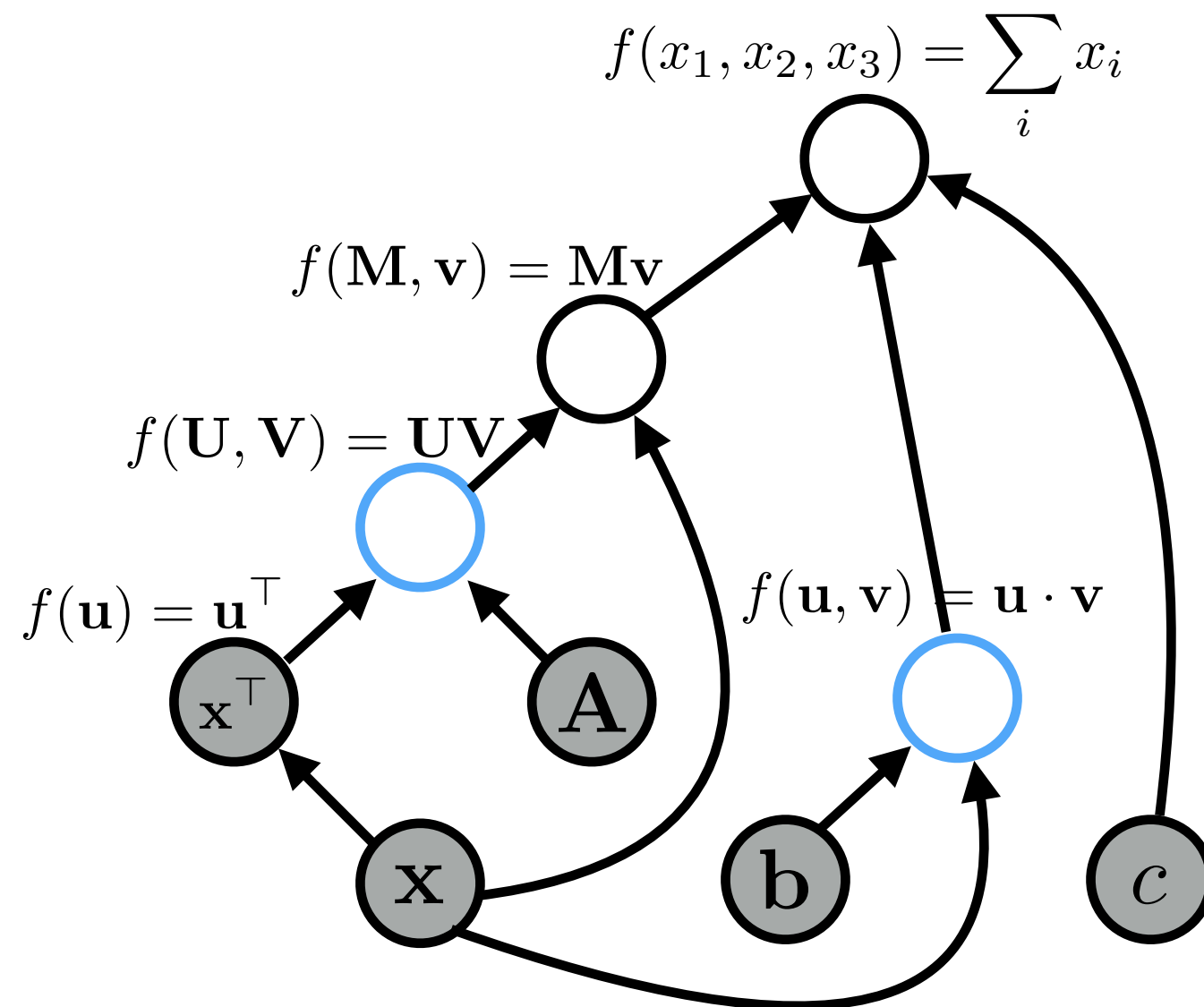
graph:

Forward Propagation

graph:

$$f(x_1, x_2, x_3) = \sum_i x_i$$

$$f(\mathbf{M}, \mathbf{v}) = \mathbf{M}\mathbf{v}$$

$$f(\mathbf{U}, \mathbf{V}) = \mathbf{U}\mathbf{V}$$

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

$$f(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$$

A

x

b

c

Forward Propagation

graph:

Forward Propagation

graph:

$$f(x_1, x_2, x_3) = \sum_i x_i$$

$$f(\mathbf{M}, \mathbf{v}) = \mathbf{Mv}$$

$$f(\mathbf{U}, \mathbf{V}) = \mathbf{UV}$$

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

$$f(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$$

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

Forward Propagation

graph:

Forward Propagation

graph:

$f(x_1, x_2, x_3) = \sum_i x_i$

$f(\mathbf{M}, \mathbf{v}) = \mathbf{M}\mathbf{v}$

$f(\mathbf{U}, \mathbf{V}) = \mathbf{U}\mathbf{V}$

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

$f(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$

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

$\mathbf{x}^\top$

$\mathbf{A}$

$\mathbf{b} \cdot \mathbf{x}$
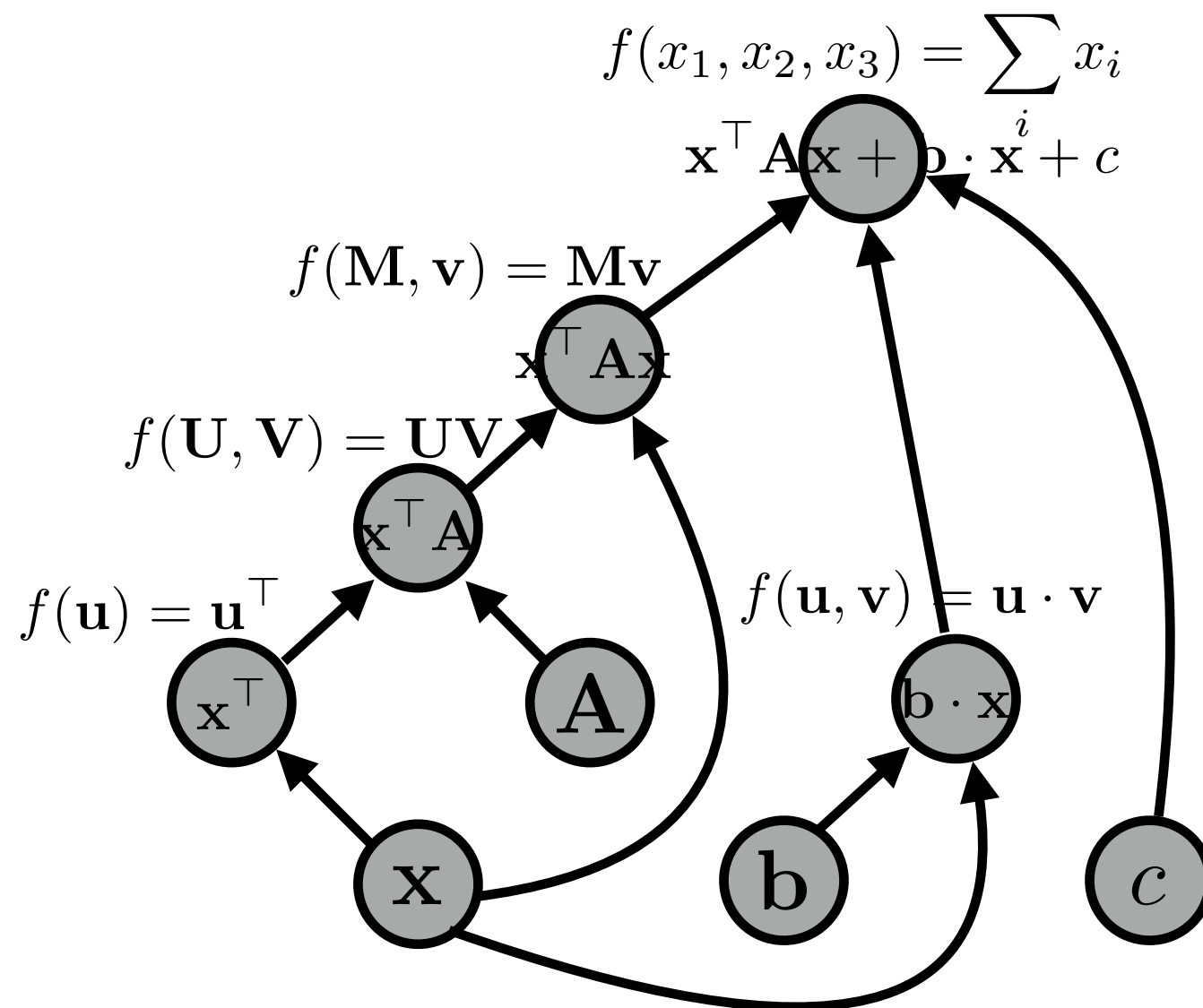
$\mathbf{x}$

$\mathbf{b}$

c
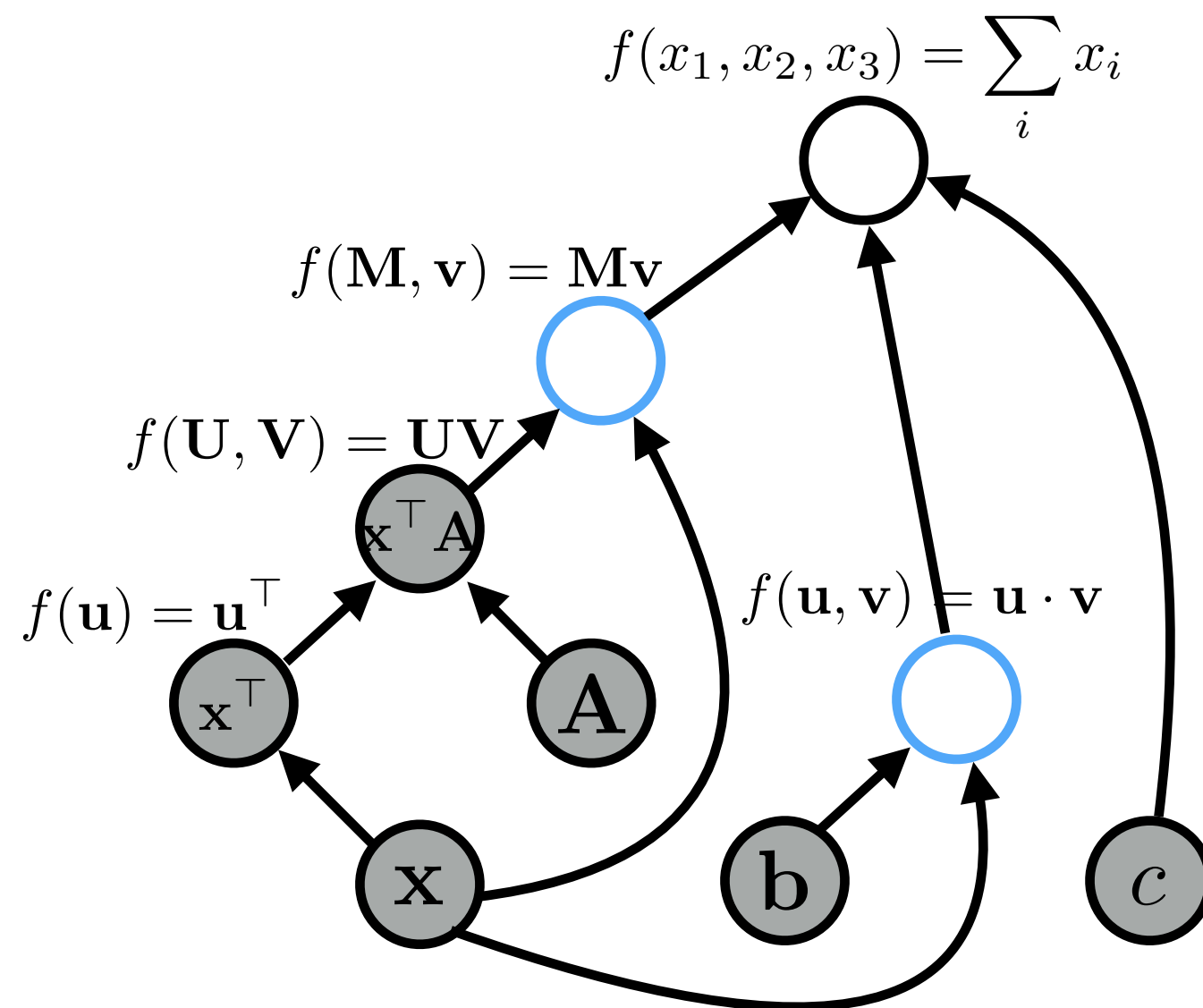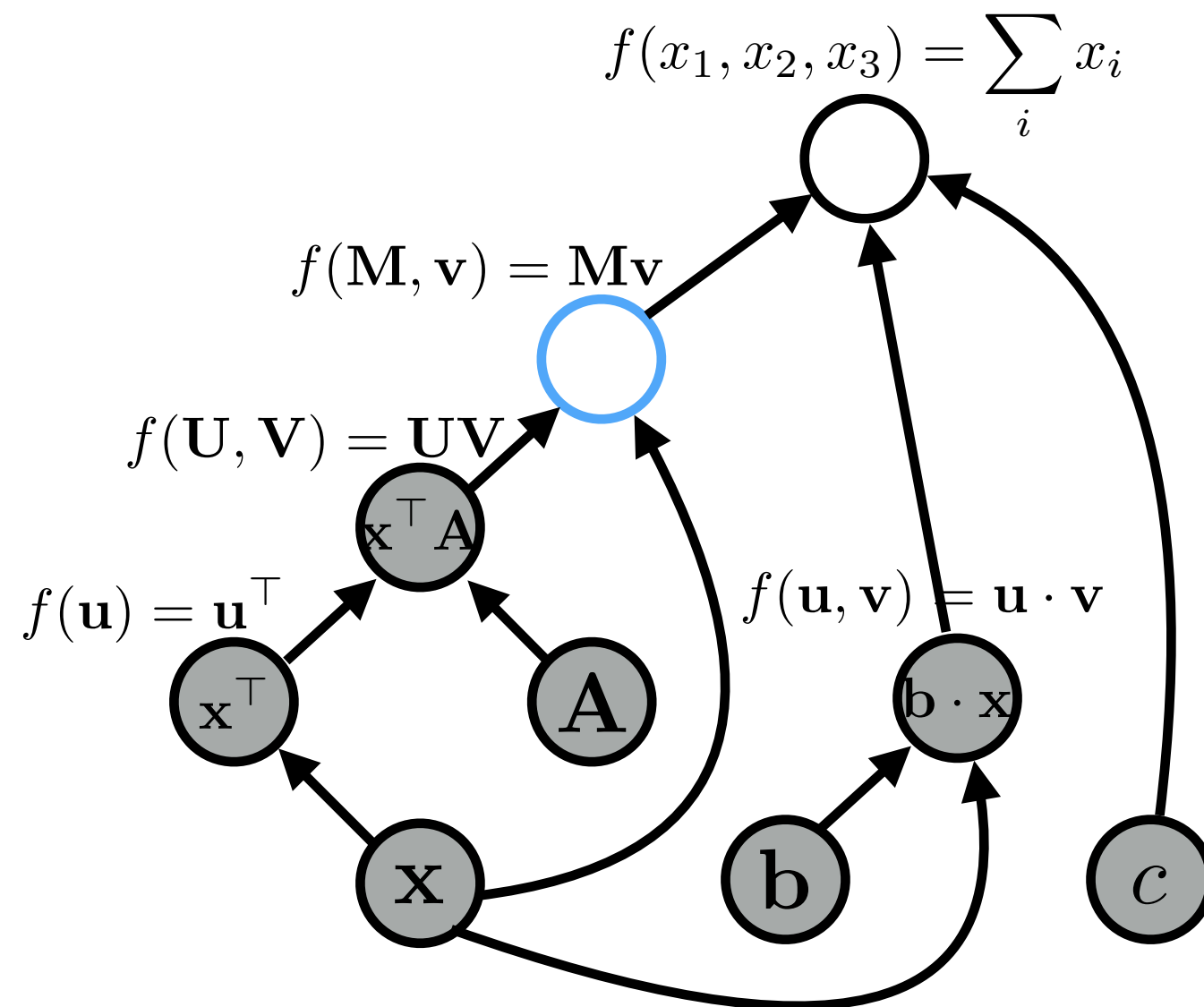
Forward Propagation
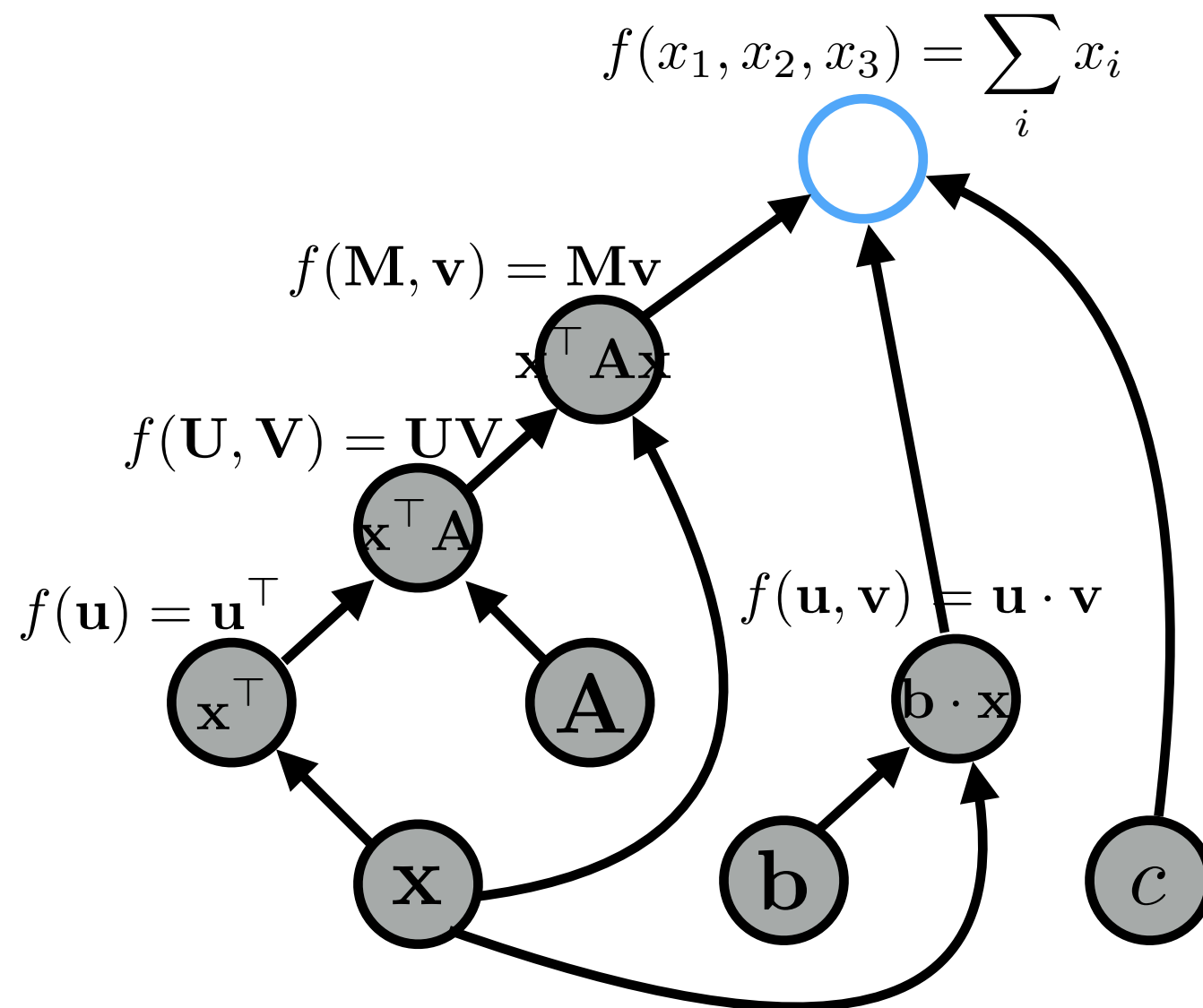
graph:

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
    (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

A Concrete Example

Neural Network Frameworks

Static Frameworks

Dynamic Frameworks
(Recommended!)

theano

Caffe

mxnet

TensorFlow

+Gluon

+Fold

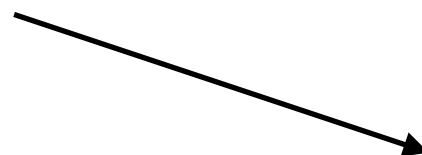∂y/net

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**

DyNet

- Examples in this class will be in DyNet:

  - **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)

Computation Graph and Expressions

```python
import dynet as dy

dy.renew_cg() # create a new computation graph

v1 = dy.inputVector([1,2,3,4])
v2 = dy.inputVector([5,6,7,8])
# v1 and v2 are expressions


v3 = v1 + v2
v4 = v3 * 2
v5 = v1 + 1


v6 = dy.concatenate([v1,v2,v3,v5])

print v6
print v6.npvalue()
```

Computation Graph and Expressions

```python
import dynet as dy

dy.renew_cg() # create a new computation graph

v1 = dy.inputVector([1,2,3,4])
v2 = dy.inputVector([5,6,7,8])
# v1 and v2 are expressions



v3 = v1 + v2
v4 = v3 * 2
v5 = v1 + 1


v6 = dy.concatenate([v1,v2,v3,v5])

print v6    expression 5/1
print v6.npvalue()
```

Computation Graph and Expressions

```python
import dynet as dy

dy.renew_cg() # create a new computation graph

v1 = dy.inputVector([1,2,3,4])
v2 = dy.inputVector([5,6,7,8])
# v1 and v2 are expressions


v3 = v1 + v2
v4 = v3 * 2
v5 = v1 + 1


v6 = dy.concatenate([v1,v2,v3,v5])

print v6
print v6.npvalue()
```

```
array([  1.,   2.,   3.,   4.,   2.,   4.,   6.,   8.,   4.,   8.,  12.,  16.])
```

Computation Graph and Expressions

- Create basic expressions.

- Combine them using *operations*.

- Expressions represent *symbolic computations*.

- Use:
  ```
  .value()
  .npvalue()
  .scalar_value()
  .vec_value()
  .forward()
  ```
  to perform actual computation.

Model and Parameters

- **Parameters** are the things that we optimize over (vectors, matrices).

- **Model** is a collection of parameters.

- Parameters **out-live** the computation graph.

Model and Parameters

```python
model = dy.Model()

pW = model.add_parameters((20,4))
pb = model.add_parameters(20)


dy.renew_cg()
x = dy.inputVector([1,2,3,4])
W = dy.parameter(pW) # convert params to expression
b = dy.parameter(pb) # and add to the graph

y = W * x + b
```

Parameter Initialization

```python
model = dy.Model()

pW = model.add_parameters((4,4))

pW2 = model.add_parameters((4,4), init=dy.GlorotInitializer())

pW3 = model.add_parameters((4,4), init=dy.NormalInitializer(0,1))

pW4 = model.parameters_from_numpu(np.eye(4))
```

Trainers and Backdrop

- Initialize a **Trainer** with a given model.

- Compute gradients by calling `expr.backward()` from a scalar node.

- Call `trainer.update()` to update the model parameters using the gradients.

Trainers and Backdrop

```python
model = dy.Model()

trainer = dy.SimpleSGDTrainer(model)

p_v = model.add_parameters(10)

for i in xrange(10):
    dy.renew_cg()

    v = dy.parameter(p_v)
    v2 = dy.dot_product(v,v)
    v2.forward()

    v2.backward()  # compute gradients

    trainer.update()
```

Trainers and Backdrop

```
model = dy.Model()

trainer =        dy.SimpleSGDTrainer(model,...)

p_v = mode       dy.MomentumSGDTrainer(model,...)

for i in        dy.AdagradTrainer(model,...)
    dy.rer
                 dy.AdadeltaTrainer(model,...)
    v = dy
    v2 = d       dy.AdamTrainer(model,...)
    v2.for

    v2.backward()   # compute gradients

    trainer.update()
```
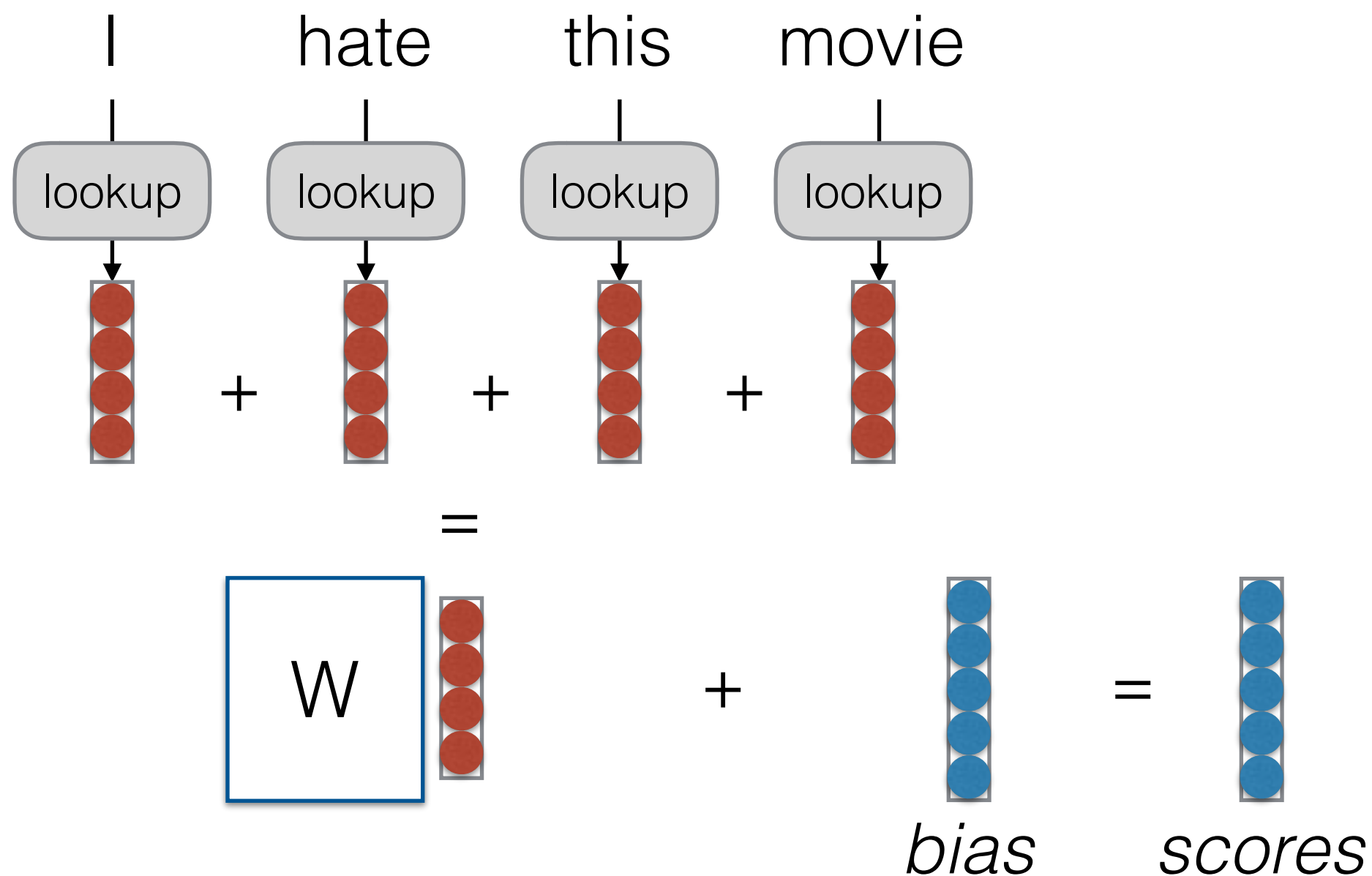
Training with DyNet

- Create model, add parameters, create trainer.

- For each training example:

  - create computation graph for the loss

  - run forward (compute the loss)

  - run backward (compute the gradients)

  - update parameters
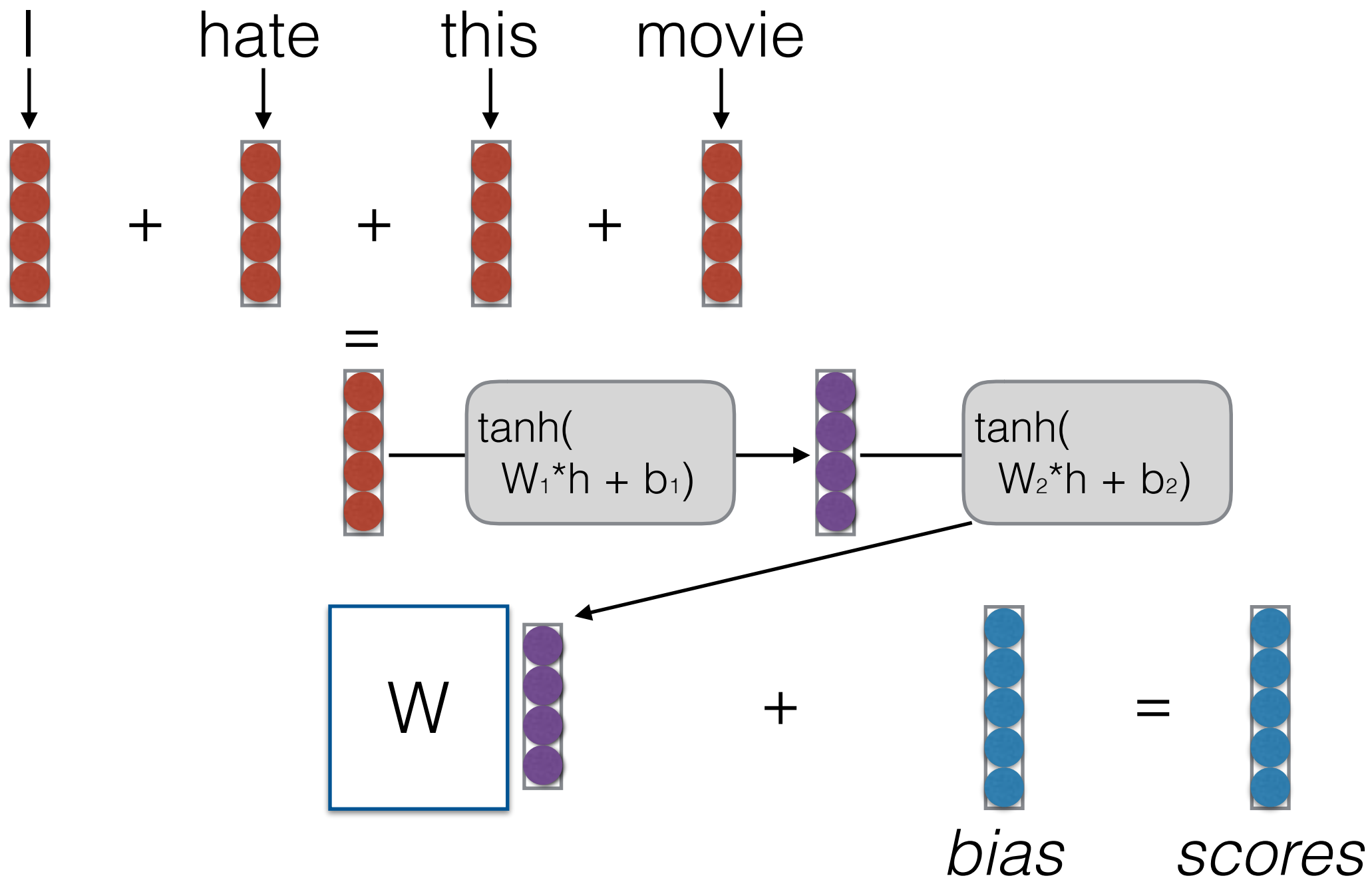
Example Implementation
(in DyNet)

Bag of Words (BOW)

I   hate   this   movie

bias   *scores*

lookup + lookup + lookup + lookup + =

probs

softmax →
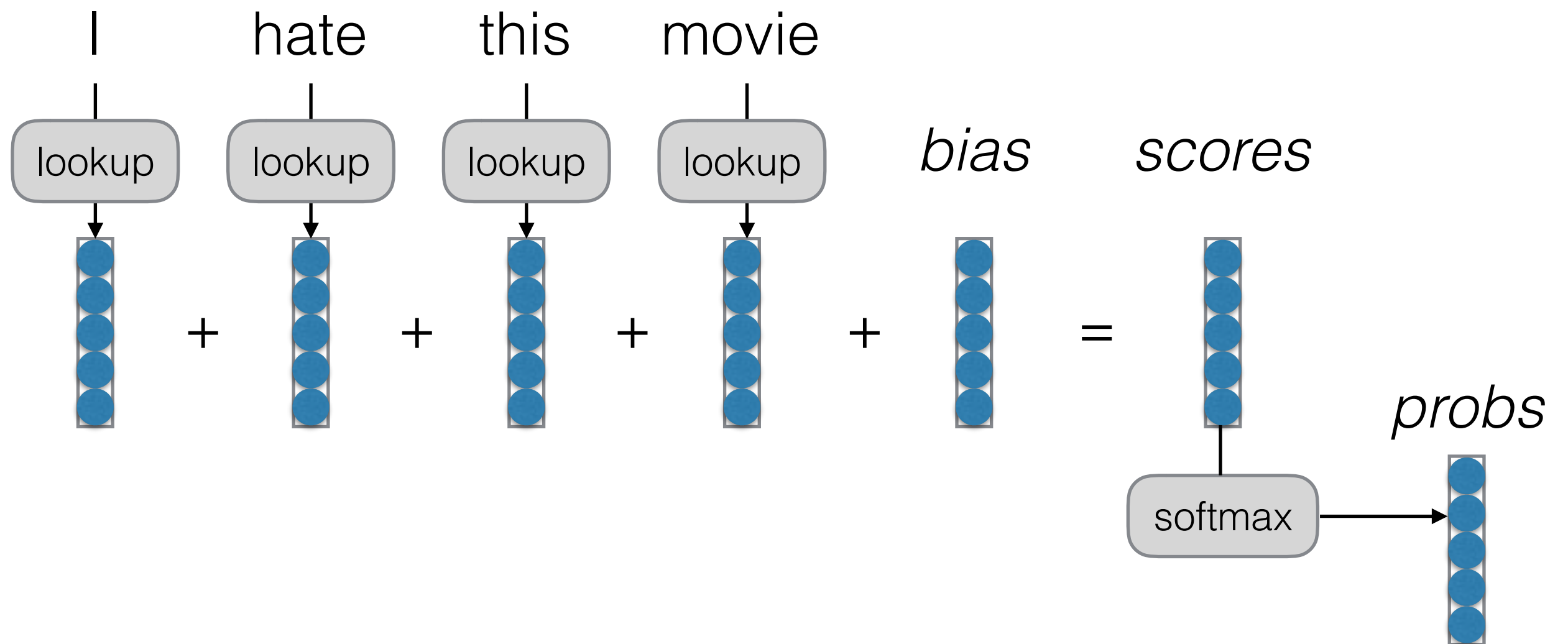
Continuous Bag of Words (CBOW)

Deep CBOW

I + hate + this + movie

=

$\tanh(W_1 * h + b_1)$

$\tanh(W_2 * h + b_2)$

W

+ *bias* = *scores*

Class Format/Structure

Class Format

- **Reading:** Before the class

- **Quiz:** Simple questions about the required reading (should be easy)

- **Summary/Elaboration/Questions:** Instructor or TAs will summarize the material, elaborate on details, and field questions

- **Code Walk:** The TAs (or instructor) will walk through some demonstration code

Assignments

- Course is group (2-3) assignment/project based

- **Assignment 1:** Survey the field and implement a baseline model

- **Assignment 2:** Re-implement and reproduce results from a state-of-the-art model

- **Project:** Perform a unique research project that either (1) improves on state-of-the-art, or (2) applies neural net models to a unique task

Instructors/Office Hours

- **Instructor:** Graham Neubig
  (Mon., 4:00-5:00PM GHC5409)

- **TAs:**

  - Zhengzhong (Hector) Liu (Mon. 1:00-2:00PM, GHC5517)

  - Xuezhe (Max) Ma (Tue. 12:00-1:00PM, GHC5517)

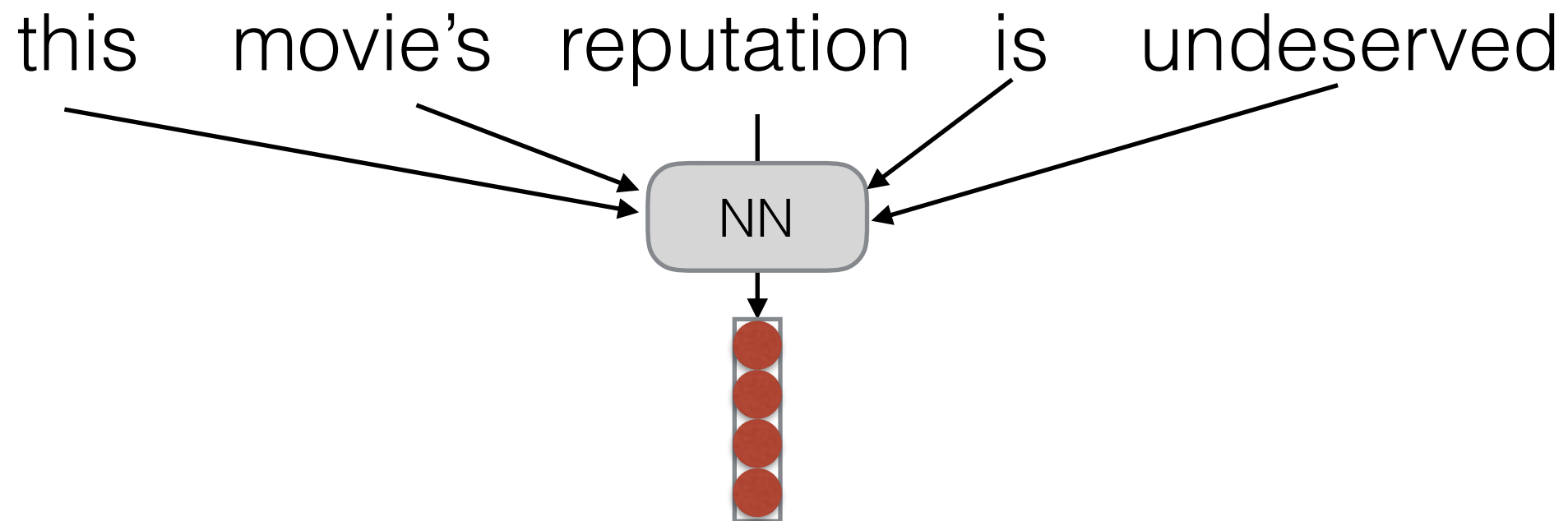 - Daniel Clothiaux (Fri. 9:00-10:00AM, GHC5505)

- **Piazza:** http://piazza.com/cmu/fall2017/cs11747/home

Class Plan

Section 1:
Models of Words

undeserved
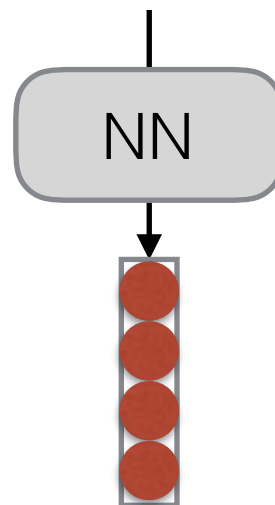
- Word representations using context
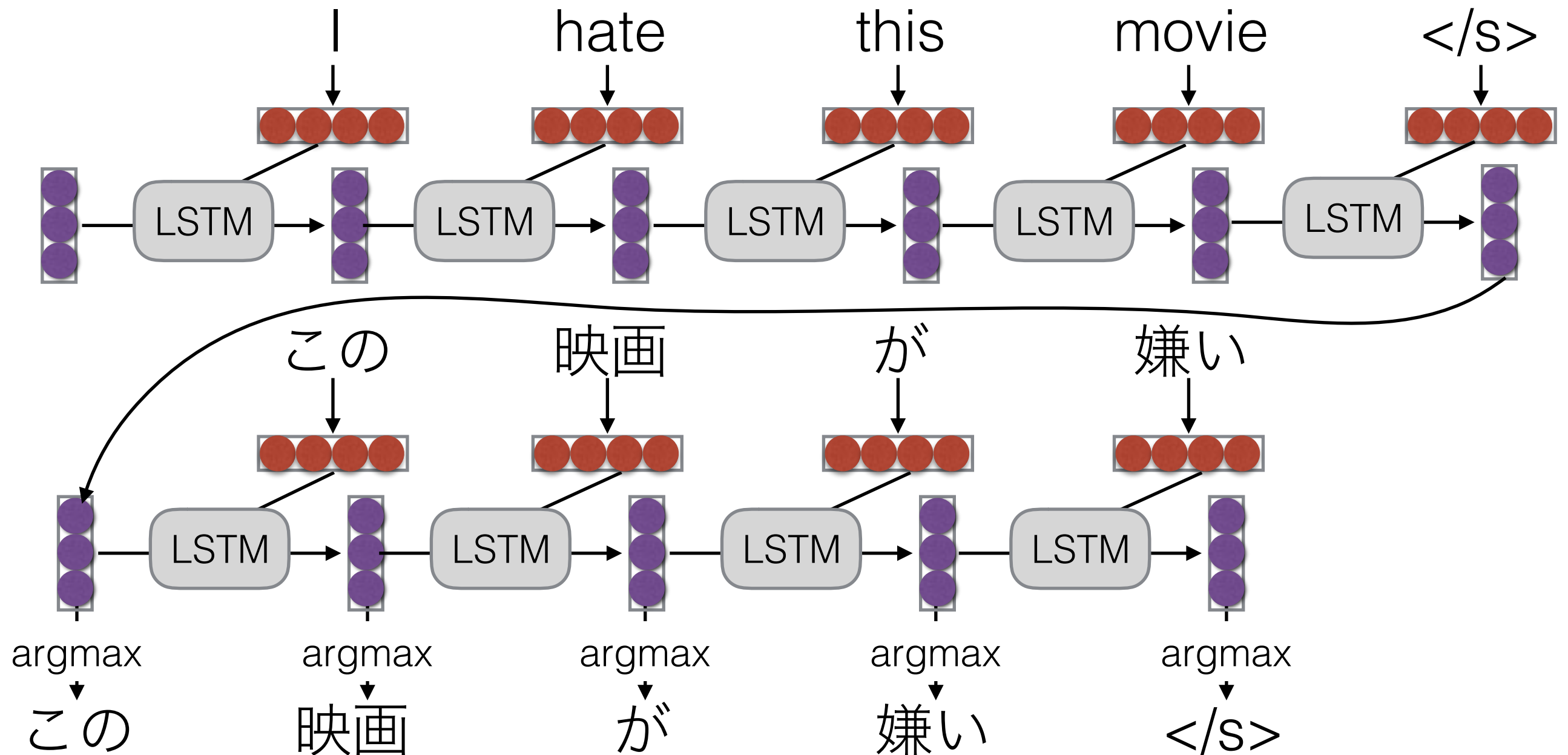
- Word representations using word form

- Speed tricks for neural networks

Section 2:
Models of Sentences

- Bag of words, bag of n-grams, convolutional nets

- Recurrent neural networks and variations
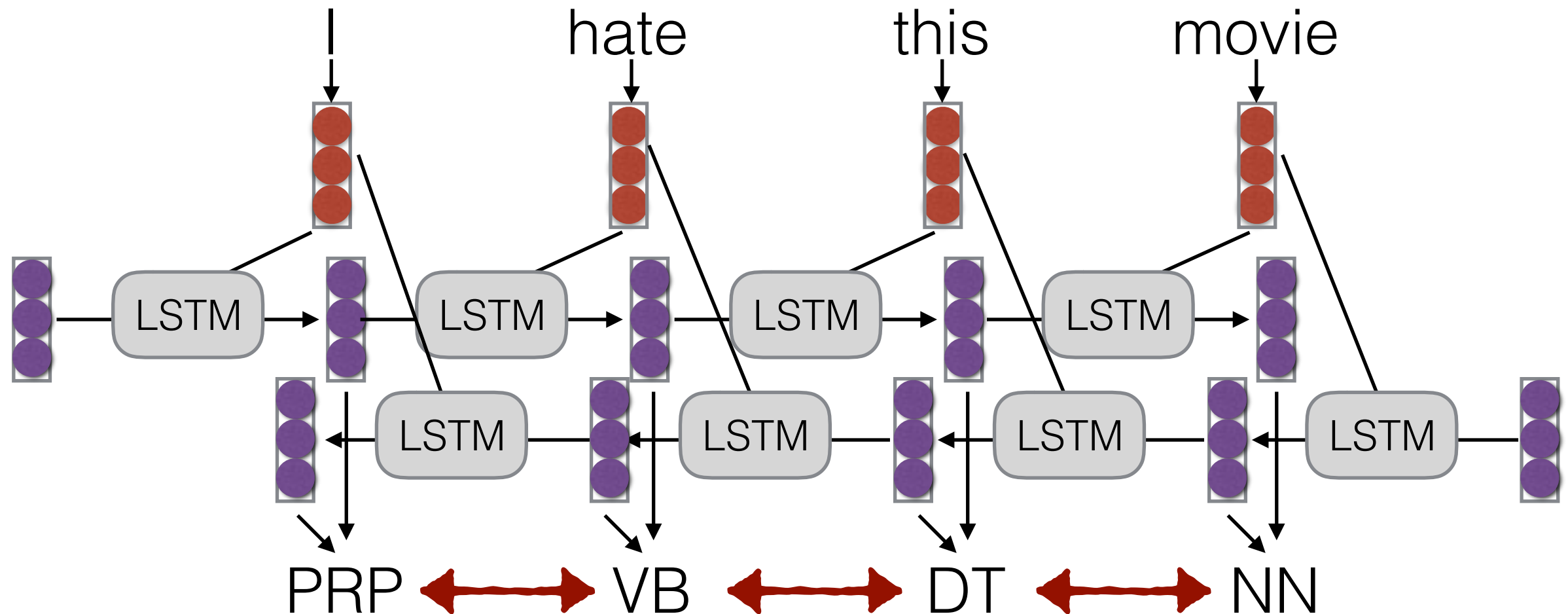
- Applications of sentence modeling

Sec.3: Sequence-to-sequence Models

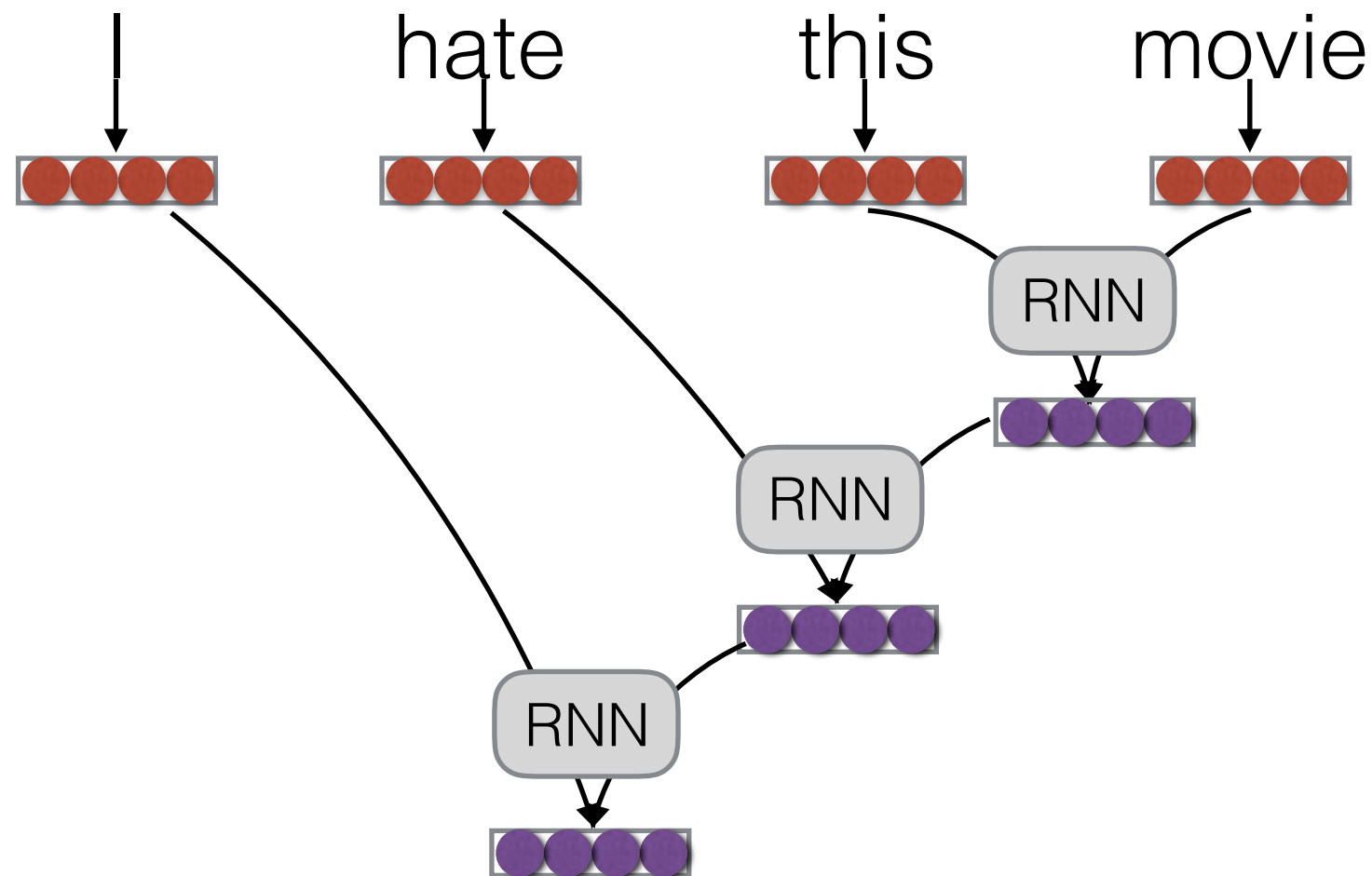- Encoder decoder models

- Attentional models

Section 4: Structured Prediction Models

- Structured perceptron, structured max margin
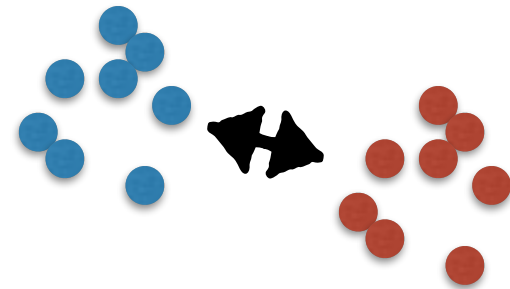
- Conditional random fields

Section 5:
Models of Tree Structure

- Shift reduce, minimum spanning tree parsing

- Tree structured compositions

- Models of graph structures
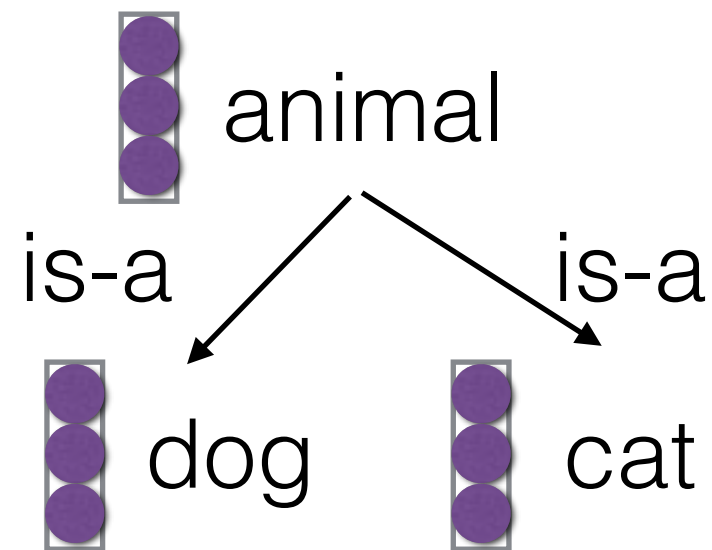
Section 6:
Advanced Learning Techniques

- Variational Auto-encoders

- Adversarial Networks

- Marginal Likelihood, Reinforcement Learning

- Semi-supervised and Unsupervised Learning

Section 7:
Neural Networks and Knowledge
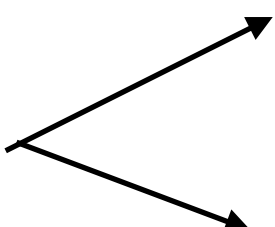
animal

is-a   is-a

dog   cat

- Learning from/for Relational Databases

- Interfacing with Relational Databases

- Machine Reading Models

- Reasoning with Neural Nets
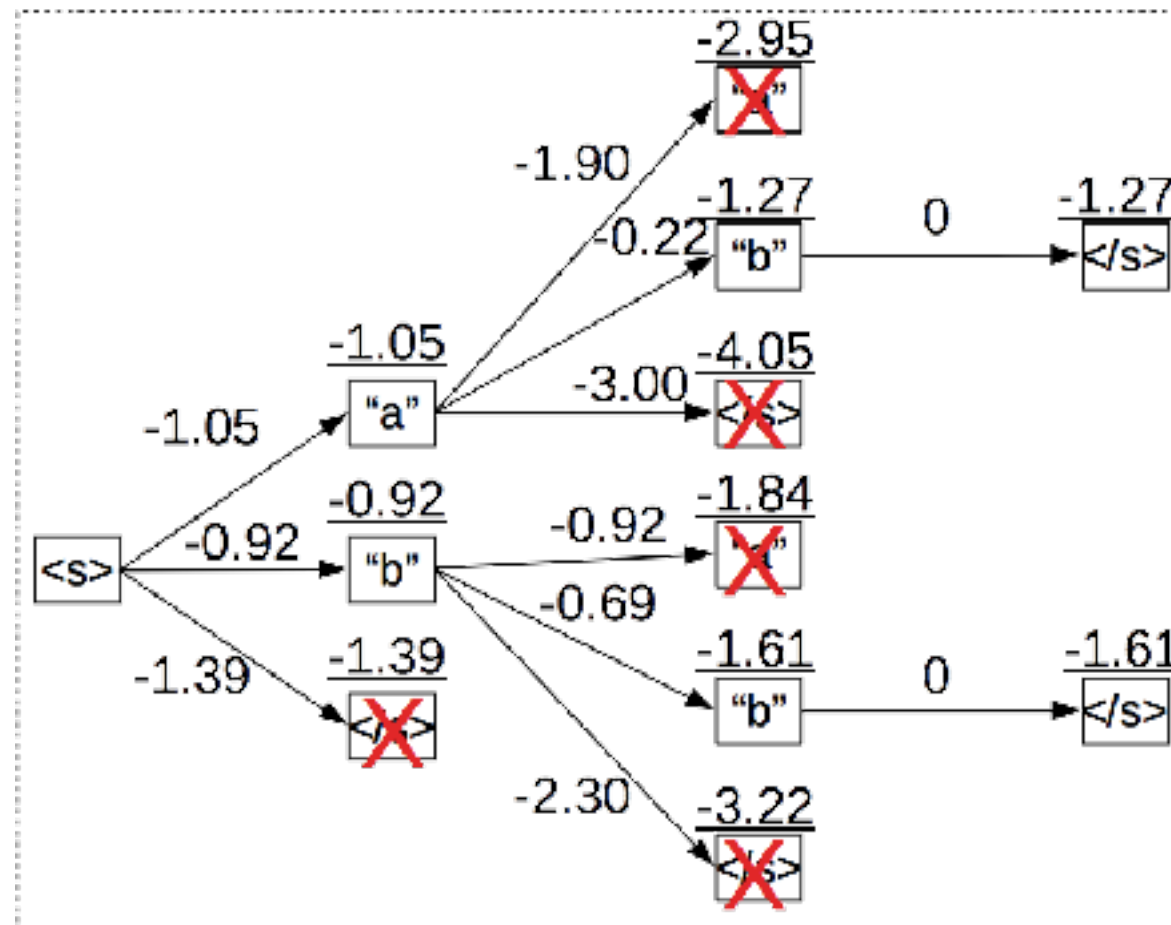
Section 8:
Multi-task and Multilingual Learning

I hate this movie → この 映画 が 嫌い

PRP VB DT NN

- Multi-task Learning Models

- Multilingual Learning of Representations

- Universal Analysis Models

Section 9:
Advanced Search Techniques

- Beam search and its variants

- A* search

Any Questions?