CS11-711 Advanced NLP

Language and Sequence Modeling

Graham Neubig

Carnegie Mellon University
Language Technologies Institute

https://phontron.com/class/anlp-fall2024/

Types of Sequential Prediction Problems

Types of Prediction: Binary, Multi-class, Structured

- Two classes (**binary classification**)

I hate this movie → positive / negative

- Multiple classes (**multi-class classification**)

I hate this movie → very good / good / neutral / bad / very bad

- Exponential/infinite labels (**structured prediction**)

I hate this movie → PRP VBP DT NN

I hate this movie → *kono eiga ga kirai*

Types of Prediction: Unconditioned vs. Conditioned

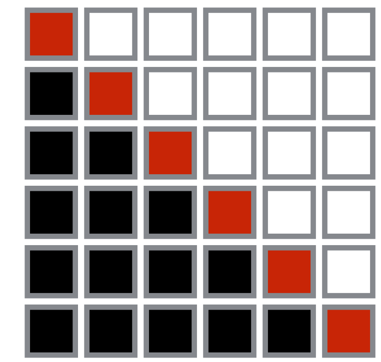
- **Unconditioned Prediction:** Predict the probability of a single variable $P(X)$
- **Conditioned Prediction:** Predict the probability of an output variable given an input $P(Y|X)$

Types of Unconditioned Prediction

Left-to-right Autoregressive Prediction

$$P(X) = \prod_{i=1}^{|X|} P(x_i | x_1, \ldots, x_{i-1})$$

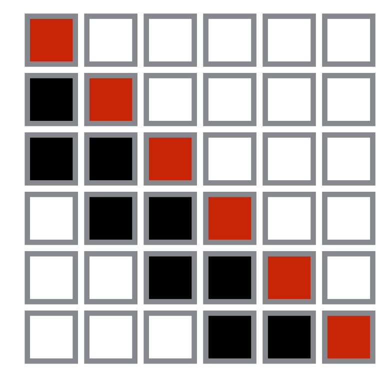
(e.g. RNN or Transformer LM)



Left-to-right Markov Chain (order n-1*)*

$$P(X) = \prod_{i=1}^{|X|} P(x_i | x_{i-n+1}, \ldots, x_{i-1})$$

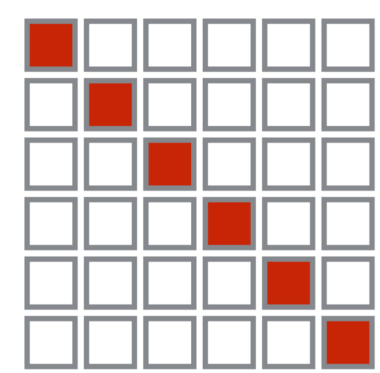
(e.g. n-gram LM, feed-forward LM)



Independent Prediction

$$P(X) = \prod_{i=1}^{|X|} P(x_i)$$

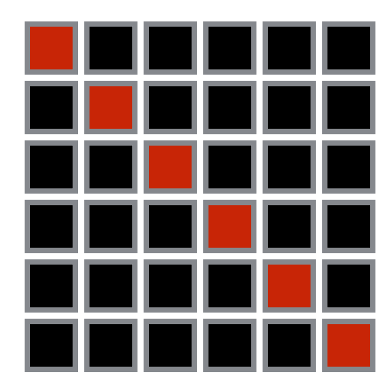
(e.g. unigram model)



Bidirectional Prediction

$$P(X) \neq \prod_{i=1}^{|X|} P(x_i | x_{\neq i})$$

(e.g. masked language model)

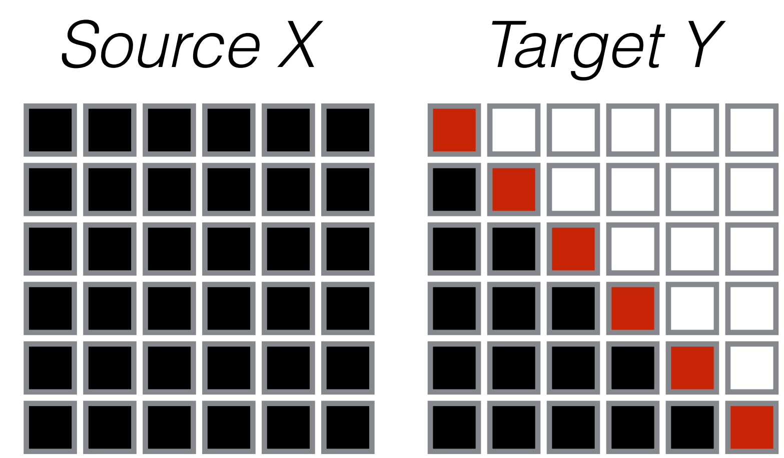


Types of Conditioned Prediction

Autoregressive Conditioned Prediction

$$P(Y|X) = \prod_{i=1}^{|Y|} P(y_i|X, y_1, \ldots, y_{i-1})$$

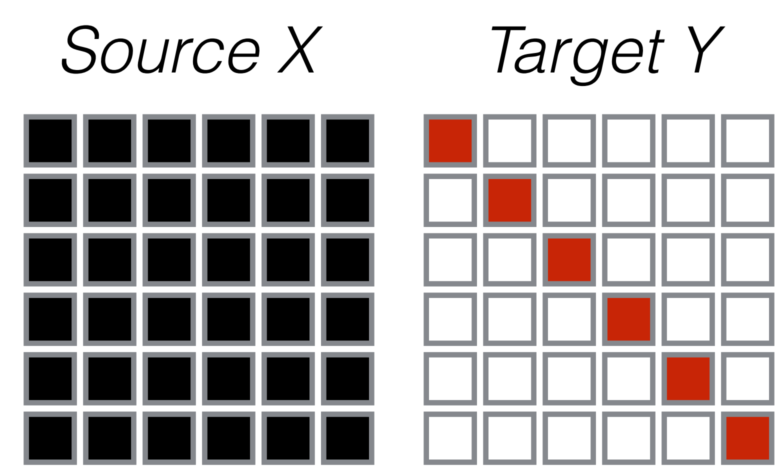
(e.g. seq2seq model)



Non-autoregressive Conditioned Prediction

$$P(Y|X) = \prod_{i=1}^{|Y|} P(y_i|X)$$

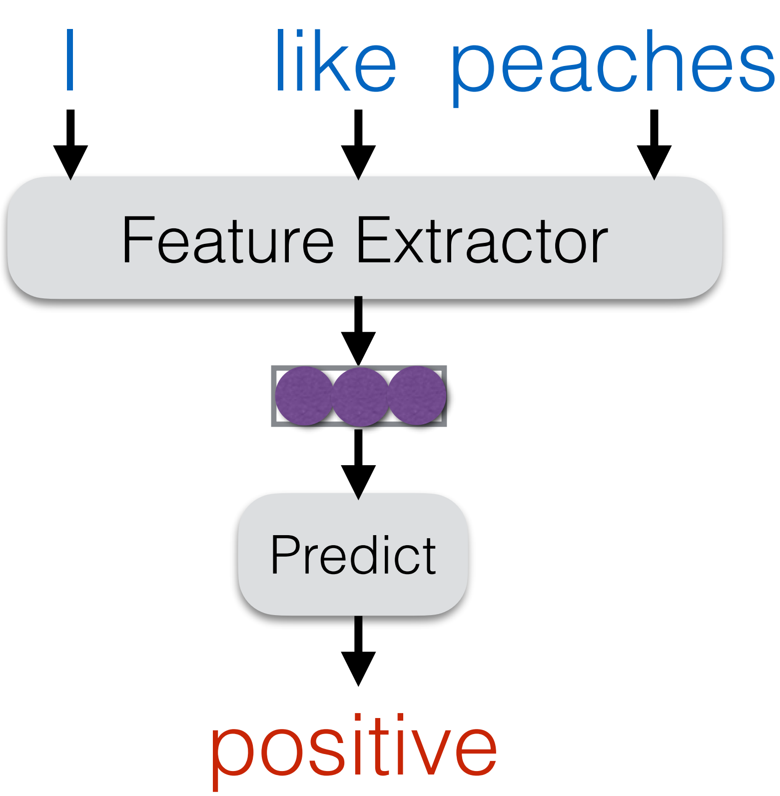
(e.g. sequence labeling, non-autoregressive MT)



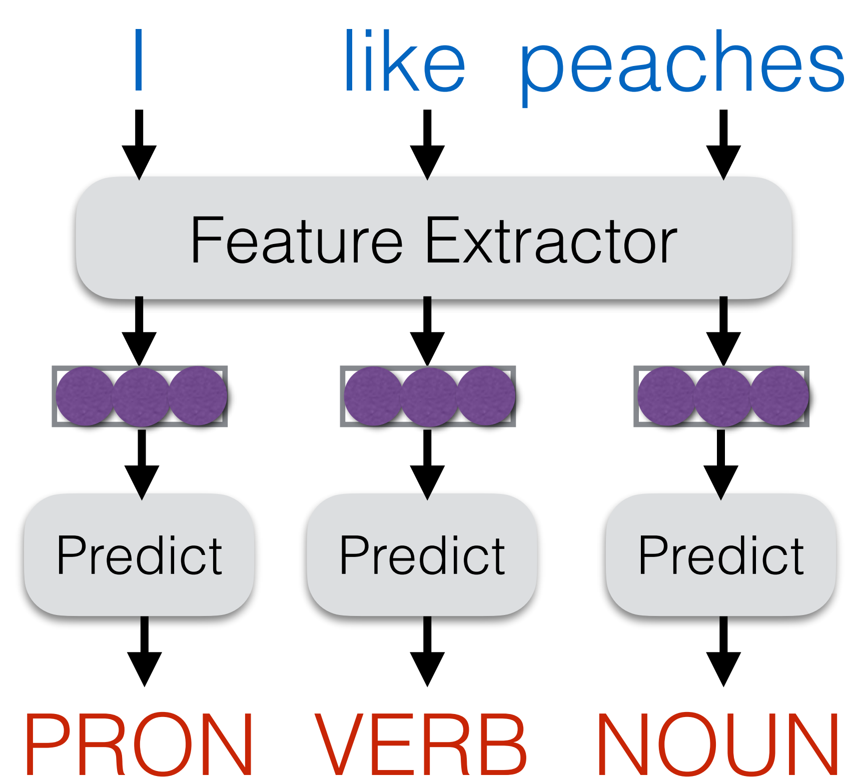
Basic Modeling Paradigm: Extract Features -> Predict

- Given an input text *X*
- Extract features *H*
- Predict labels *Y*

Text Classification



Sequence Labeling



Language Modeling

Generative vs. Discriminative Models

- **Discriminative model:** a model that calculates the probability of a latent trait given the data

$$P(Y \mid X)$$

conditional

- **Generative model:** a model that calculates the probability of the input data itself

$$P(X) \qquad P(X, Y)$$

stand-alone *joint*

Probabilistic Language Models

$$P(X)$$

Sentence/Document

A generative model that calculates the probability of language

What Can we Do w/ LMs?

- **Score** sentences:

P(Jane went to the store .) → high
P(store to Jane went the .) → low

(same as calculating loss for training)

- **Generate** sentences:

$$\tilde{x} \sim P(X)$$

How Can we Apply These?

- **Answer questions**
  - *Score* possible multiple choice answers
  - *Generate* a continuation of a question prompt
- **Classify text**
  - *Score* the text conditioned on a label
  - *Generate* a label given a classification prompt
- **Correct grammar**
  - *Score* each word and replace low-scoring ones
  - *Generate* a paraphrase of the output

Auto-regressive Language Models

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

The big problem: How do we predict

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

?!?!

Aside: there are also *masked* and *energy-based* language models, but we'll not cover them today.

Unigram Language Models

The Simplest Language Model: Count-based Unigram Models

- Let's choose the simplest one for now!

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

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

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

Handling Unknown Words

- If a token doesn't exist in training data becomes zero! $\frac{c_{\text{train}}(x_i)}{\sum_{\tilde{x}} c_{\text{train}}(\tilde{x})}$

- Two options:
  - **Segment to characters/subwords:** Make sure that all tokens are in vocabulary.
  - **Unknown word model:** create a character/word based model for unknown words and interpolate.

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

Parameterizing in Log Space

- Multiplication of probabilities can be re-expressed as addition of log probabilities

$$P(X) = \prod_{i=1}^{|X|} P(x_i) \longrightarrow \log P(X) = \sum_{i=1}^{|X|} \log P(x_i)$$

- **Why?:** numerical stability, other conveniences

- We will define these parameters θ_{xi}

$$\theta_{x_i} := \log P(x_i)$$

Quiz: how many parameters does a unigram model have?

Higher-order Language Models

Higher-order *n*-gram Models

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

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

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

- 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

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

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 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

Likelihood

- **Log-likelihood:**

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

- **Per-word Log Likelihood:**

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

Papers often also report negative log likelihood (lower better), as that is used in loss.

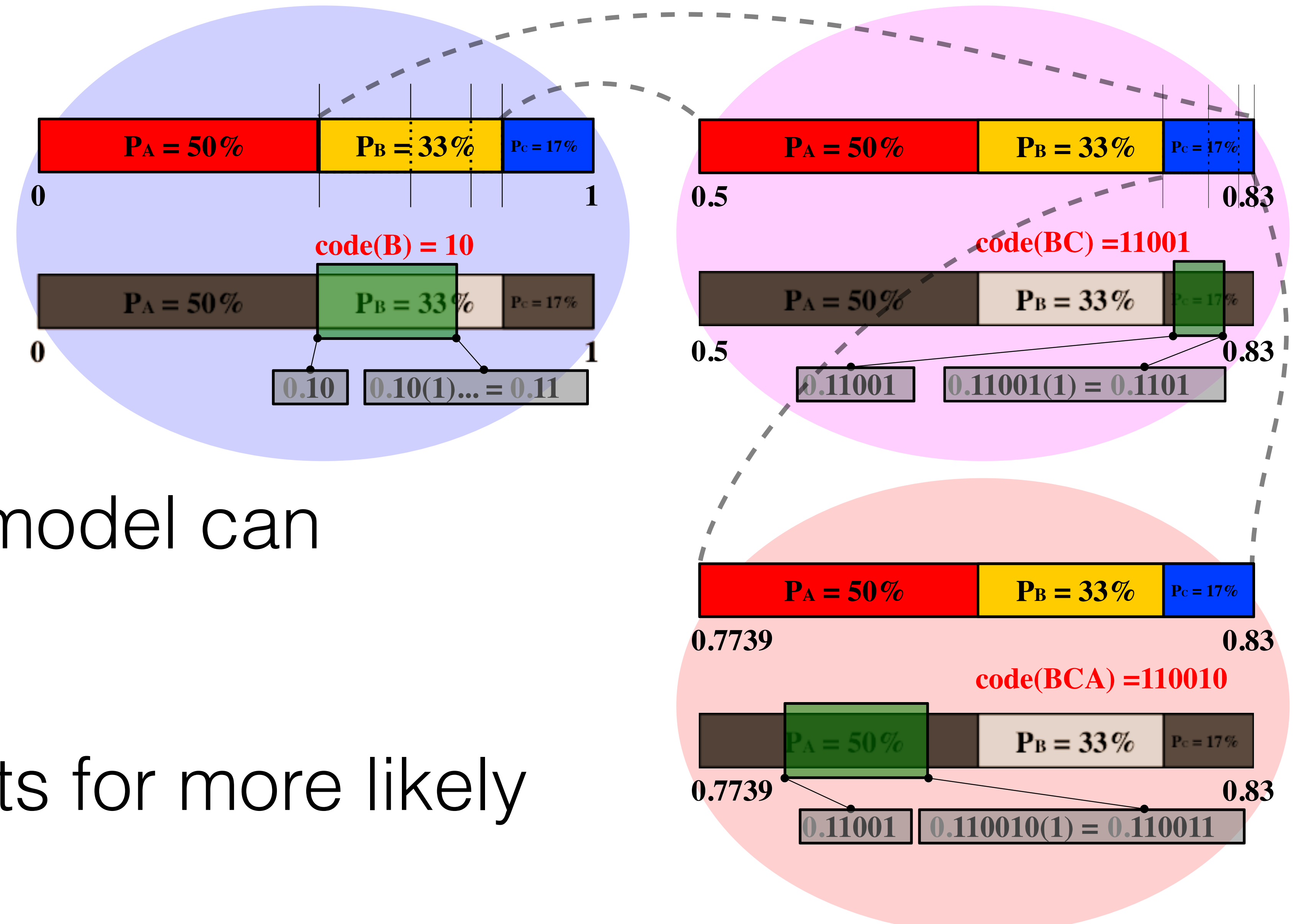
Entropy

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

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

Quiz: why log2?

An Aside: LMs and Compression



- Any probabilistic model can compress data
- Use shorter outputs for more likely inputs
- Method: arithmetic coding

Image credit: Wikipedia

Perplexity

- **Perplexity:**

$$PPL(\mathcal{X}_{\text{test}}) = 2^{H(\mathcal{X}_{\text{test}})} = e^{-WLL(\mathcal{X}_{\text{test}})}$$

When a dog sees a squirrel it will usually ___

Token: ' be' - Probability: 0.0352 → PPL= 28.4
Token: ' jump' - Probability: 0.0338 → PPL= 29.6
Token: ' start' - Probability: 0.0289 → PPL= 34.6
Token: ' run' - Probability: 0.0277 → PPL= 36.1
Token: ' try' - Probability: 0.0219 → PPL= 45.7

An Alternative:
Featurized Log-Linear Models

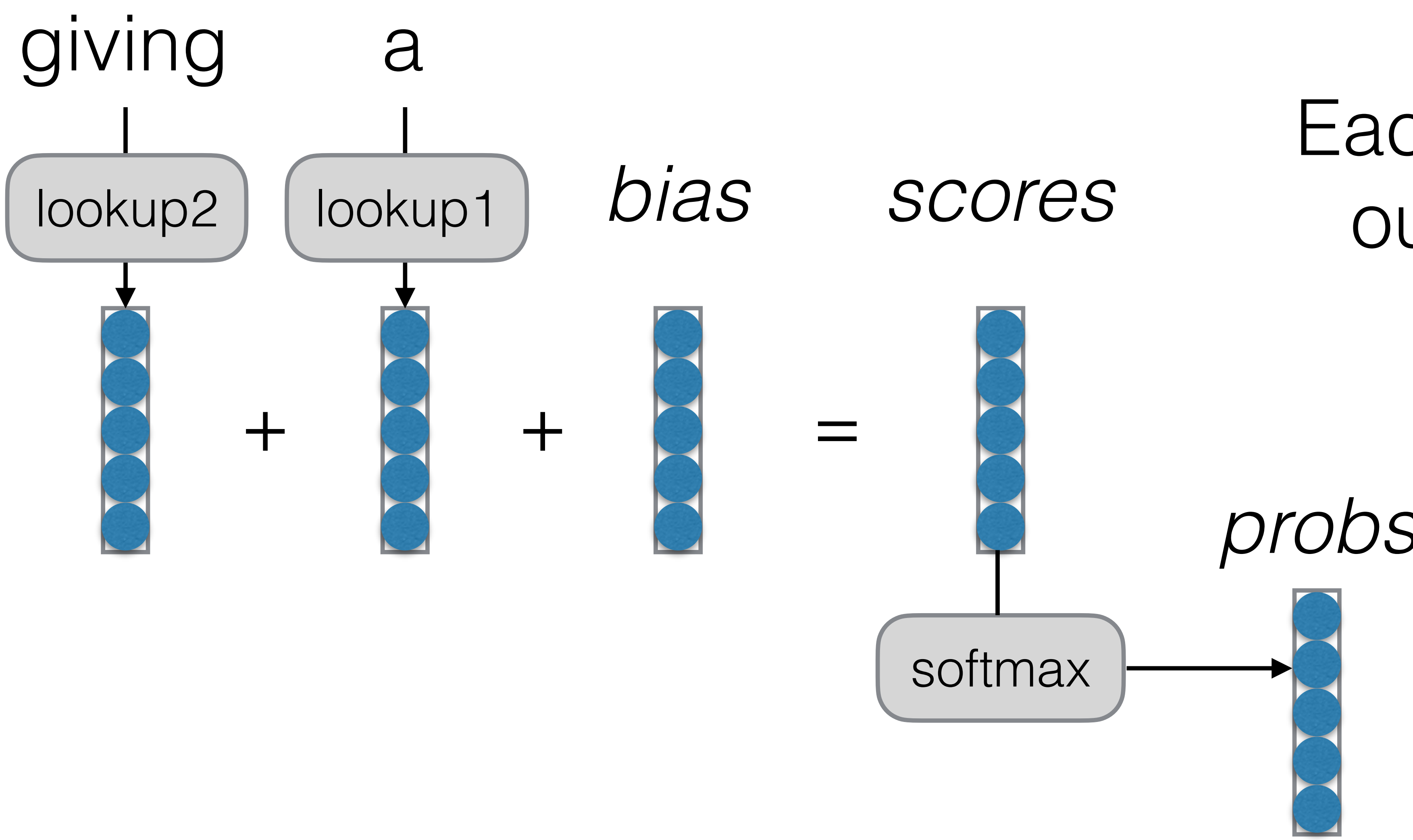
(Rosenfeld 1996)

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



Each vector is size of output vocabulary

- 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

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

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

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

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

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

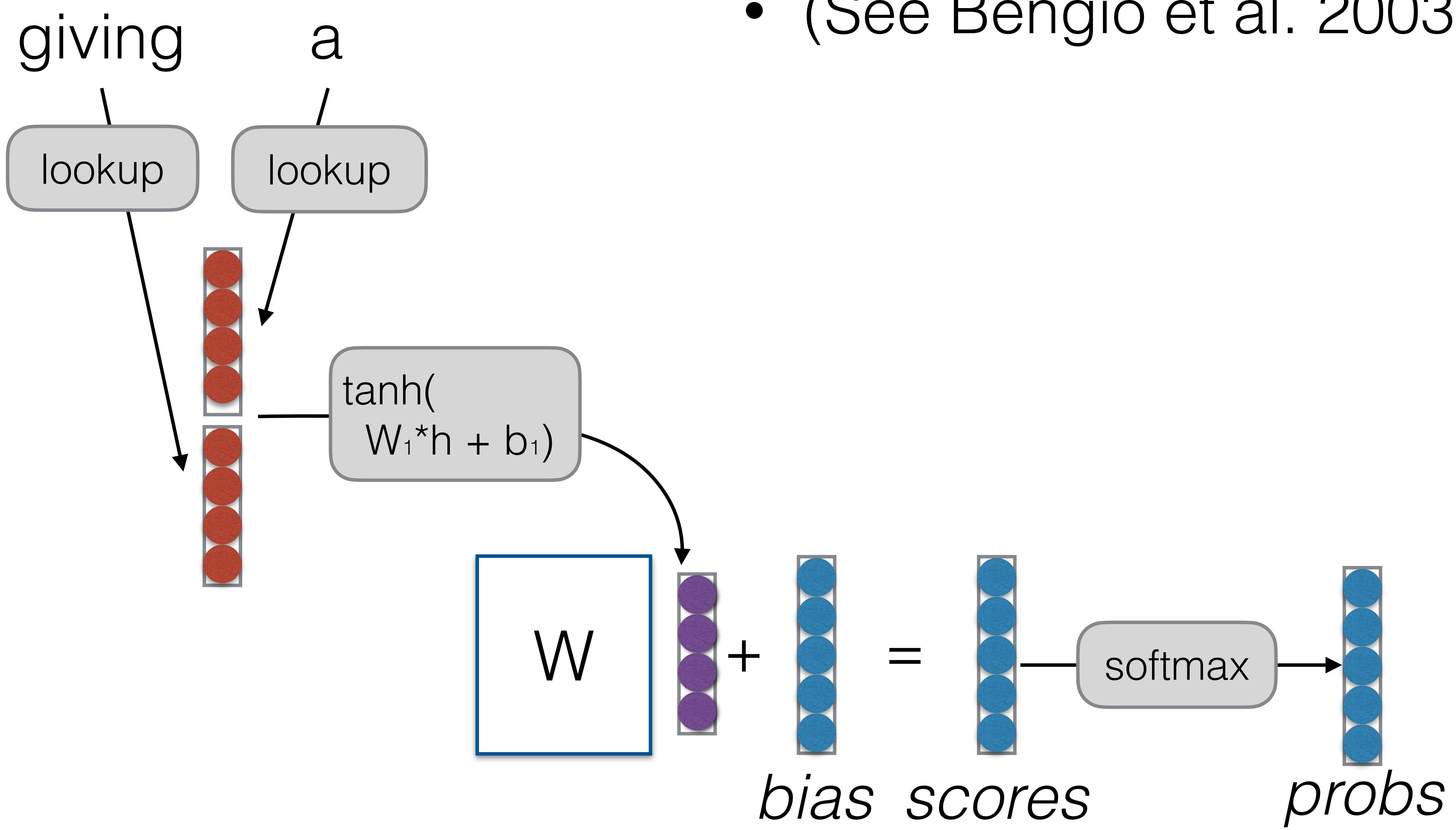
for programming class he wanted to buy his own computer

→ not solved yet 😞

Back to Language Modeling

Feed-forward Neural Language Models

- (See Bengio et al. 2003)



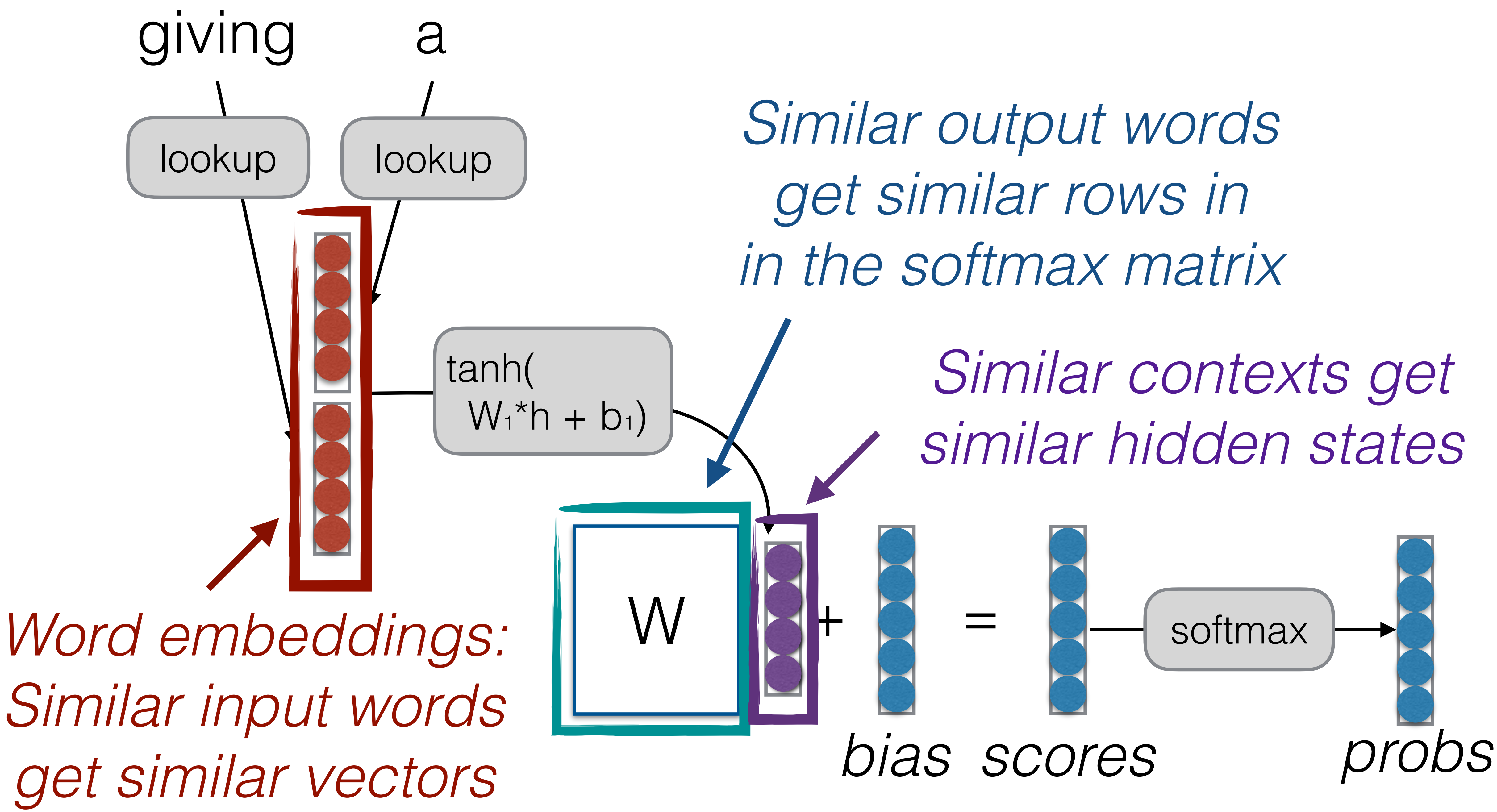
Example of Combination Features

- 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

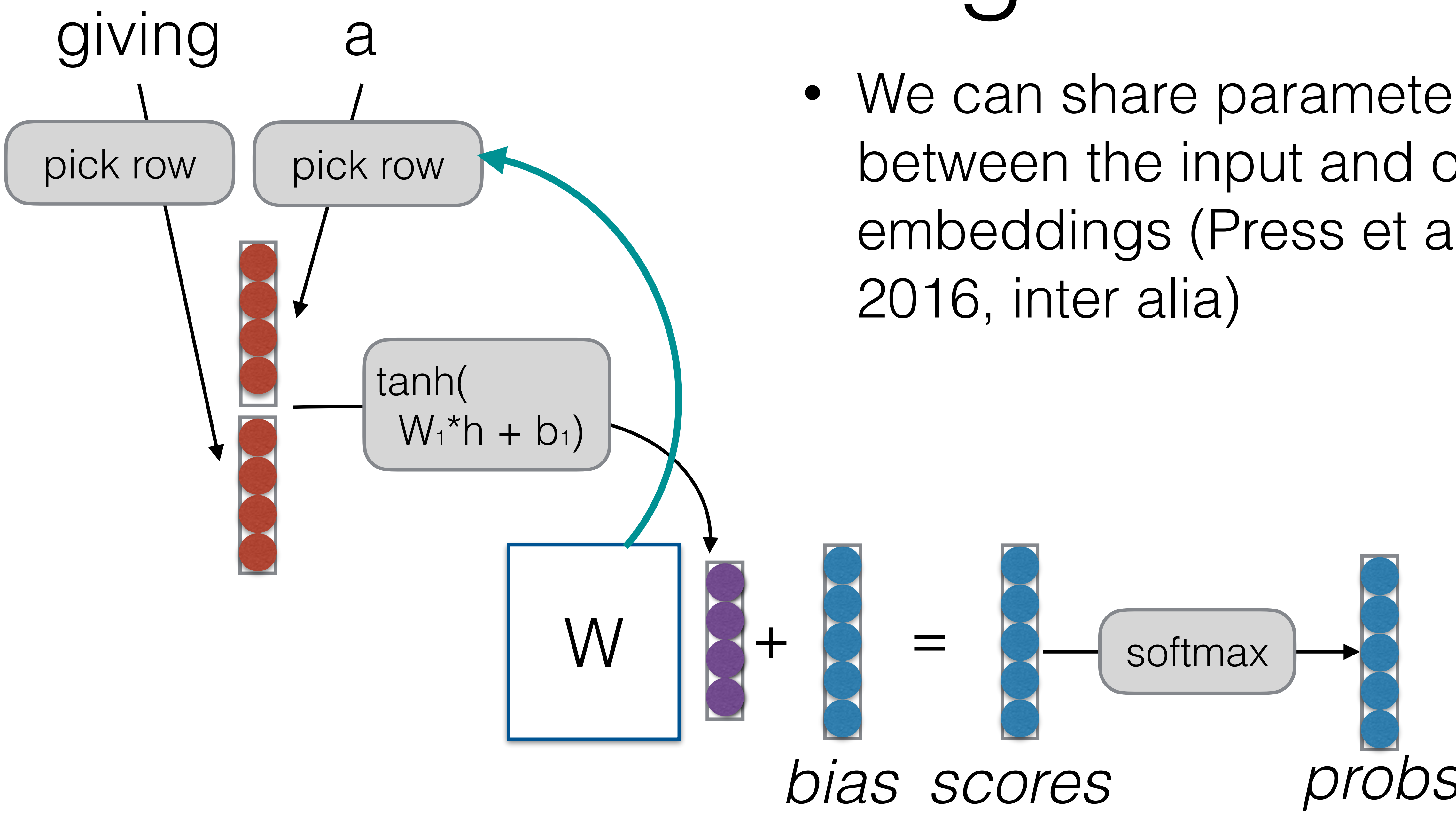
$$\begin{array}{cc} \text{giving} & \begin{bmatrix}1.2\\-0.1\\0.7\\-2.1\\0.5\end{bmatrix} \\ \text{a} & \begin{bmatrix}-0.3\\2.0\\0.6\\-0.8\\-0.4\end{bmatrix}\end{array} * \overset{\boldsymbol{w}_{34}}{\begin{bmatrix}1.5\\0\\0\\0\\0\\0\\1.3\\0\\0\\0\end{bmatrix}} + \overset{b_{34}}{-2} =$$

positive number if the previous word is a determiner and second-to-previous word is a verb

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

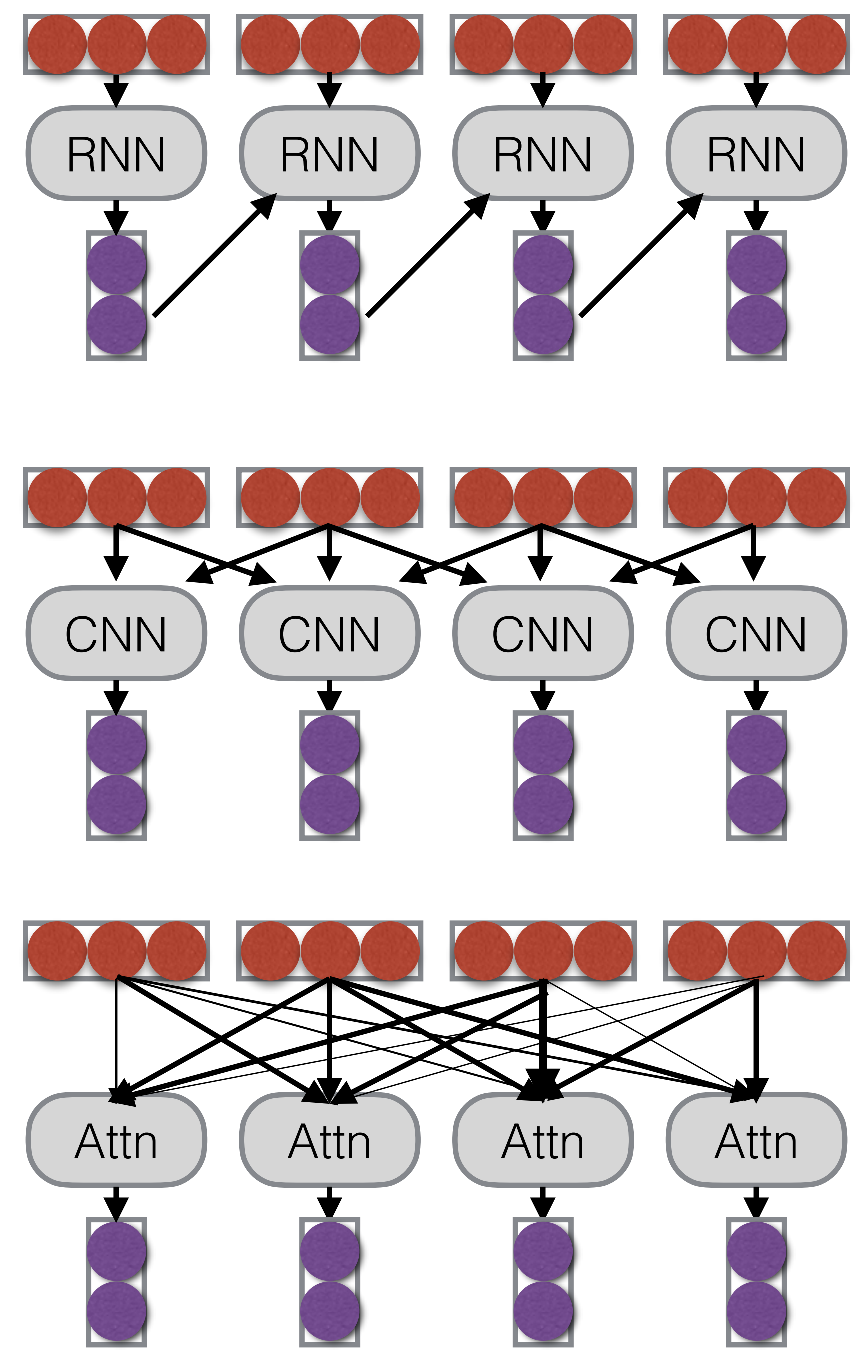
for programming class he wanted to buy his own computer

→ not solved yet 😞

Full Sequence Models

Three Major Types of Sequence Models

- **Recurrence:** Condition representations on an encoding of the history
- **Convolution:** Condition representations on local context
- **Attention:** Condition representations on a weighted average of all tokens



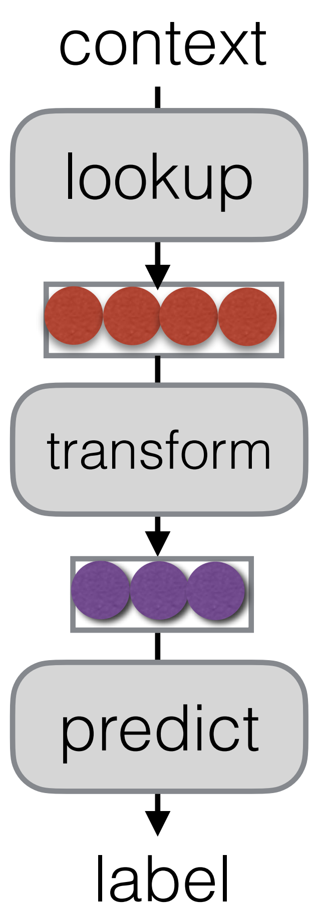
Recurrent Neural Networks

Recurrent Neural Networks

(Elman 1990)

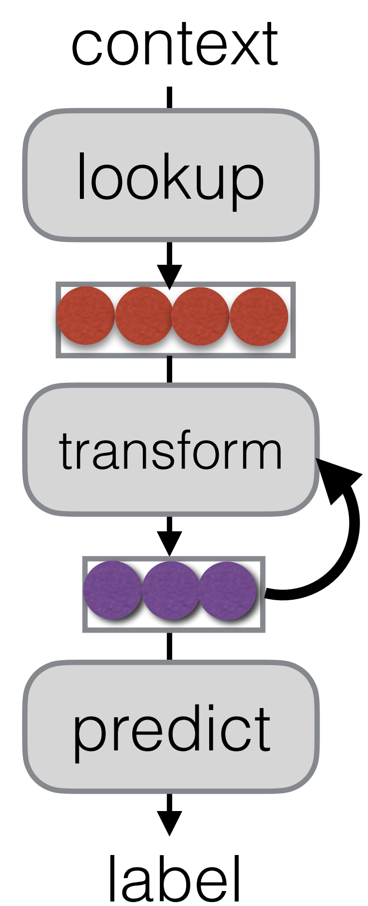
- Tools to “remember” information

Feed-forward NN



$$h_t = \mathrm{f}(W_x x_t + b)$$

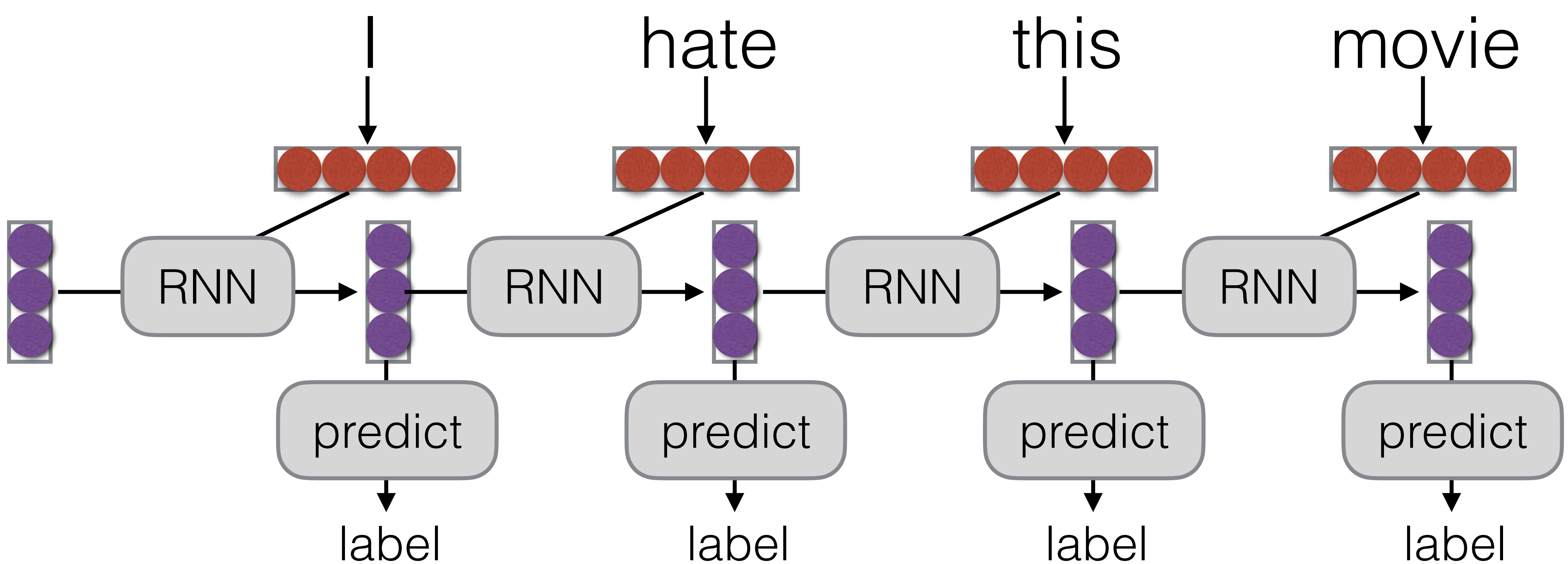
Recurrent NN



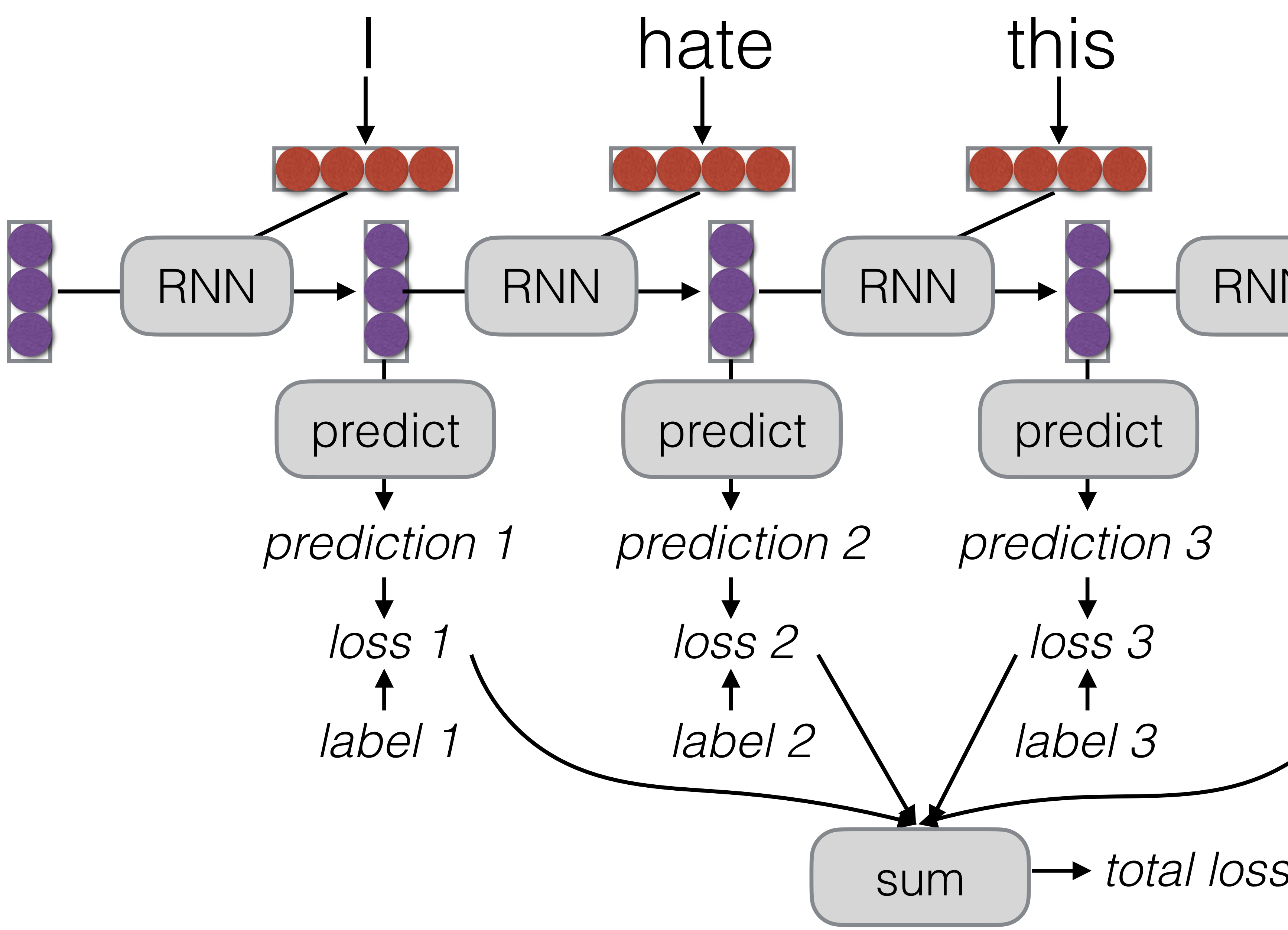
$$h_t = \mathrm{f}(W_h h_{t-1} + W_x x_t + b)$$

Unrolling in Time

- What does processing a sequence look like?

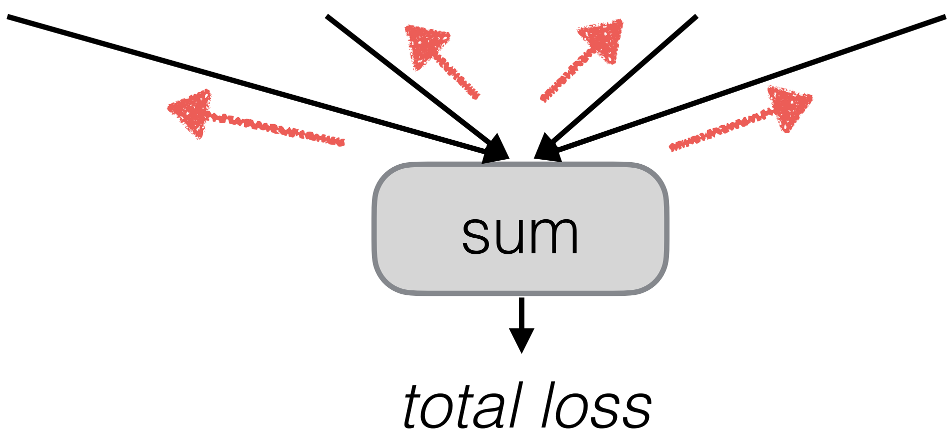


Training RNNs



RNN Training

- The unrolled graph is a well-formed (DAG) computation graph—we can run backprop

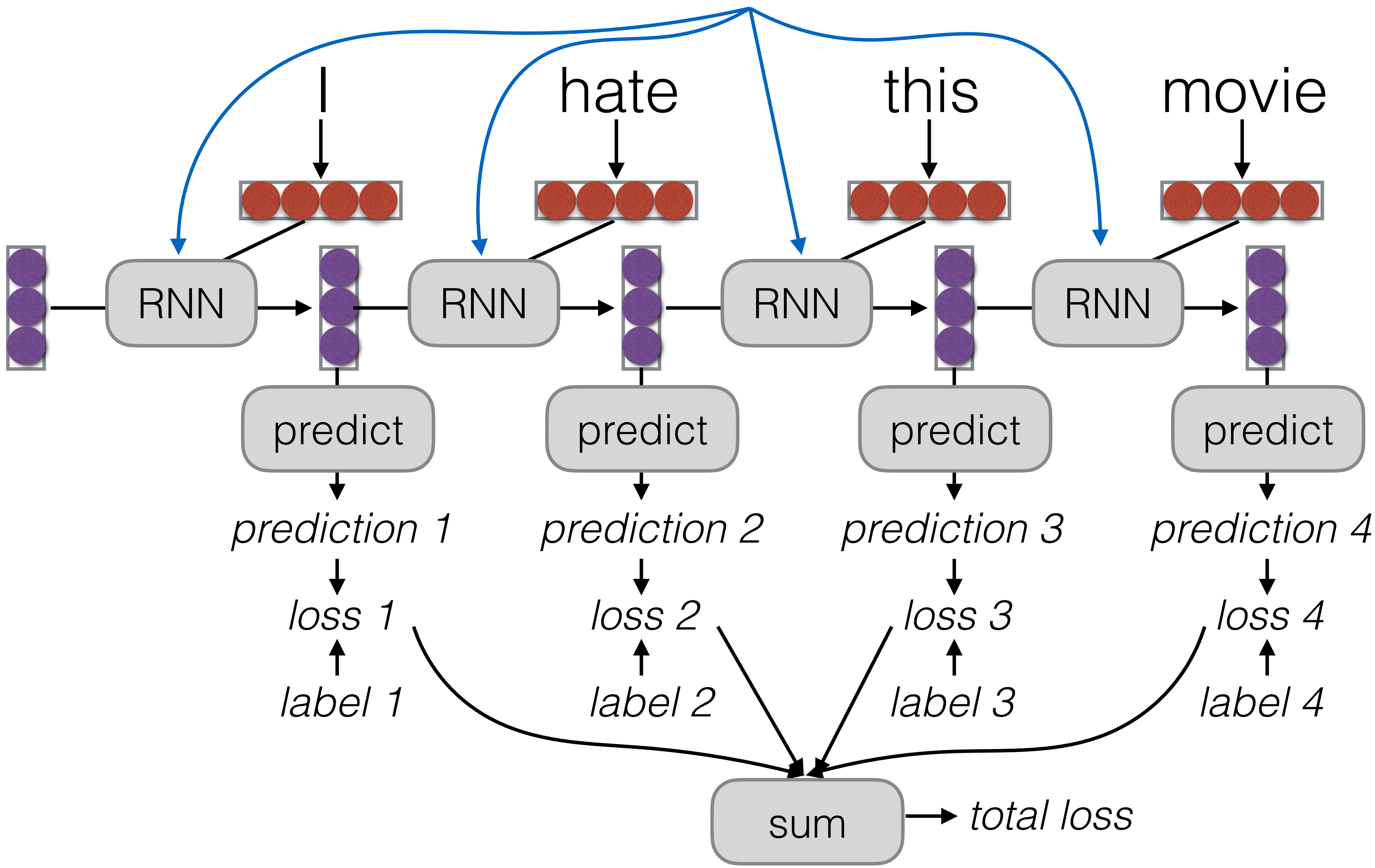


- Parameters are tied across time, derivatives are aggregated across all time steps

- This is historically called "backpropagation through time" (BPTT)

Parameter Tying

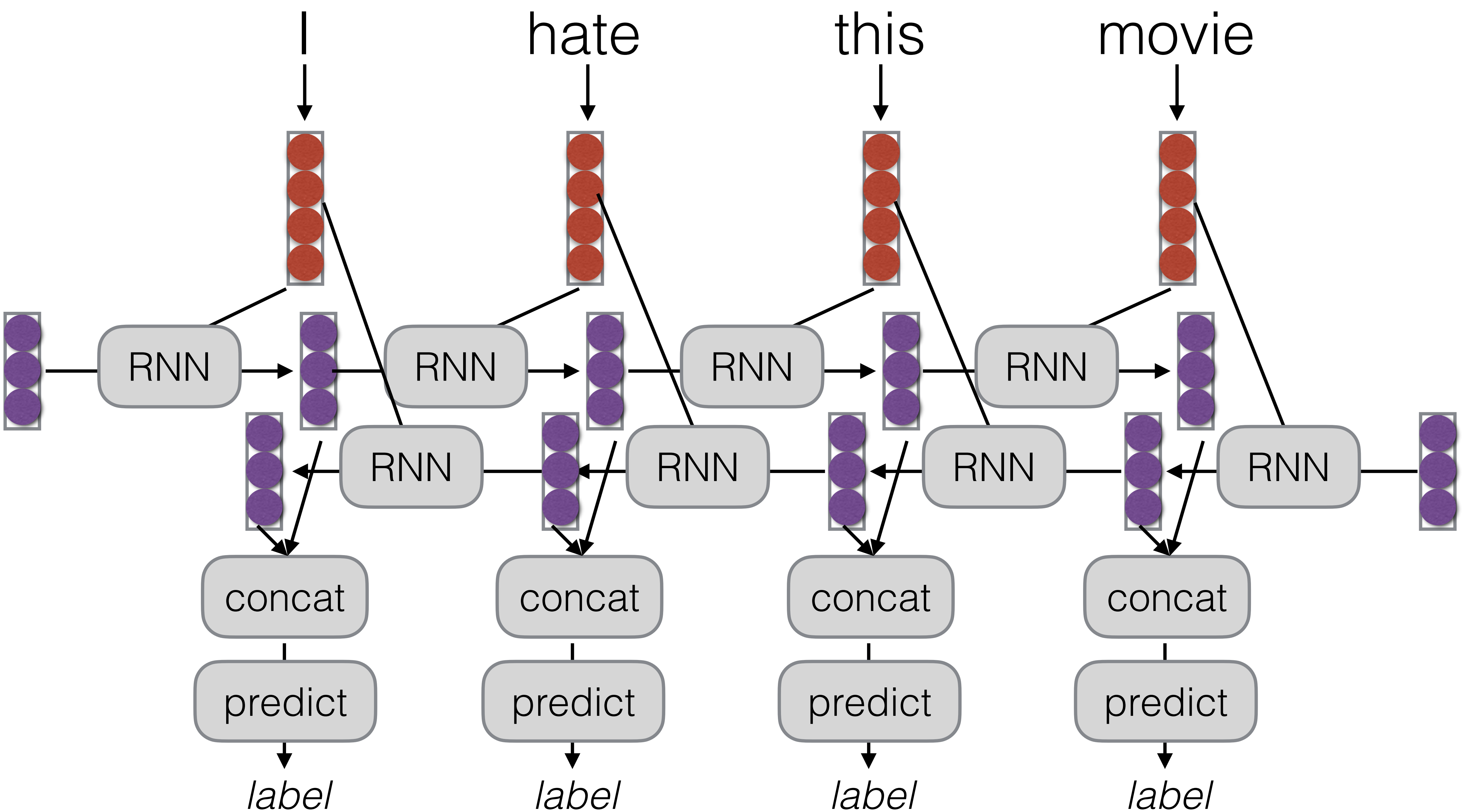
Parameters are shared! Derivatives are accumulated.



(Same for attention, convolutional networks)

Bi-RNNs

- A simple extension, run the RNN in both directions

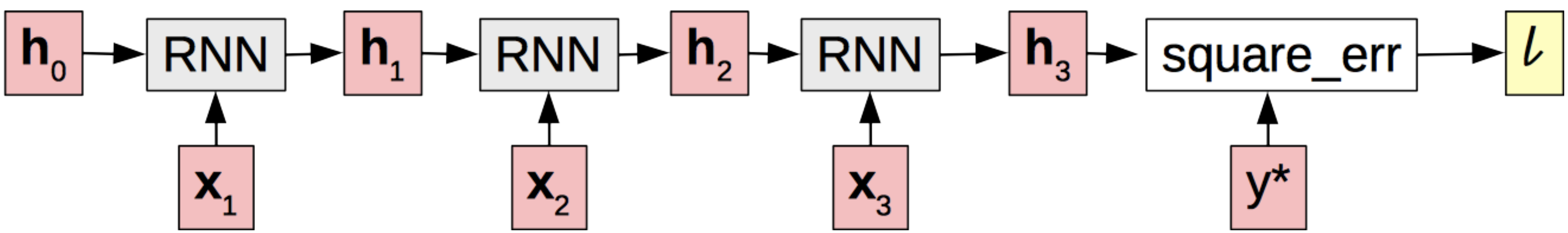


Vanishing Gradients

Vanishing Gradient

- Gradients decrease as they get pushed back

$$\frac{dl}{d_{h_0}} = \text{tiny} \qquad \frac{dl}{d_{h_1}} = \text{small} \qquad \frac{dl}{d_{h_2}} = \text{med.} \qquad \frac{dl}{d_{h_3}} = \text{large}$$



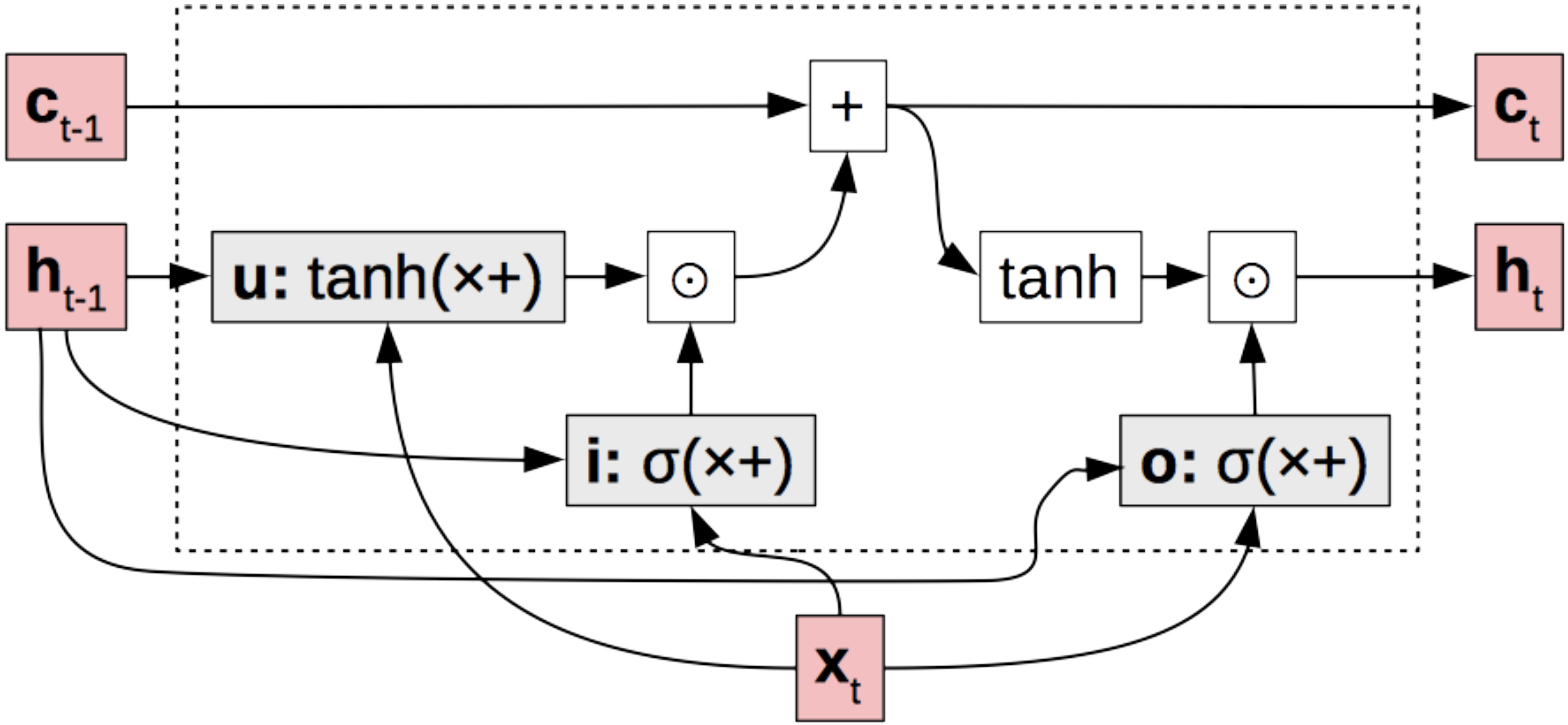
- Why? “Squashed” by non-linearities or small weights in matrices.

A Solution:
Long Short-term Memory

(Hochreiter and Schmidhuber 1997)

- **Basic idea:** make additive connections between time steps
- Addition does not modify the gradient, no vanishing
- Gates to control the information flow

LSTM Structure

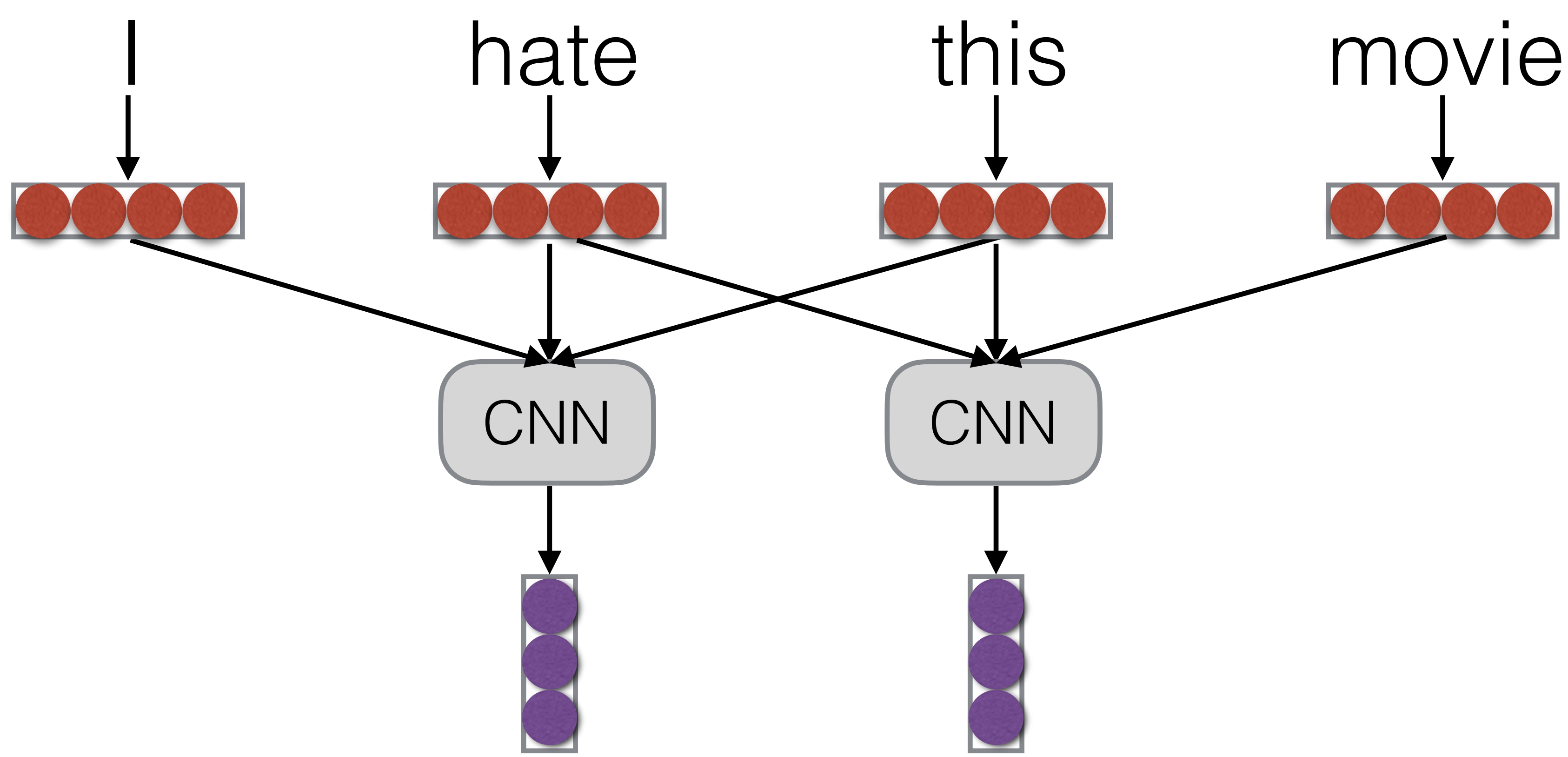


update **u**: what value do we try to add to the memory cell?
input **i**: how much of the update do we allow to go through?
output **o**: how much of the cell do we reflect in the next state?

Convolution

Convolution

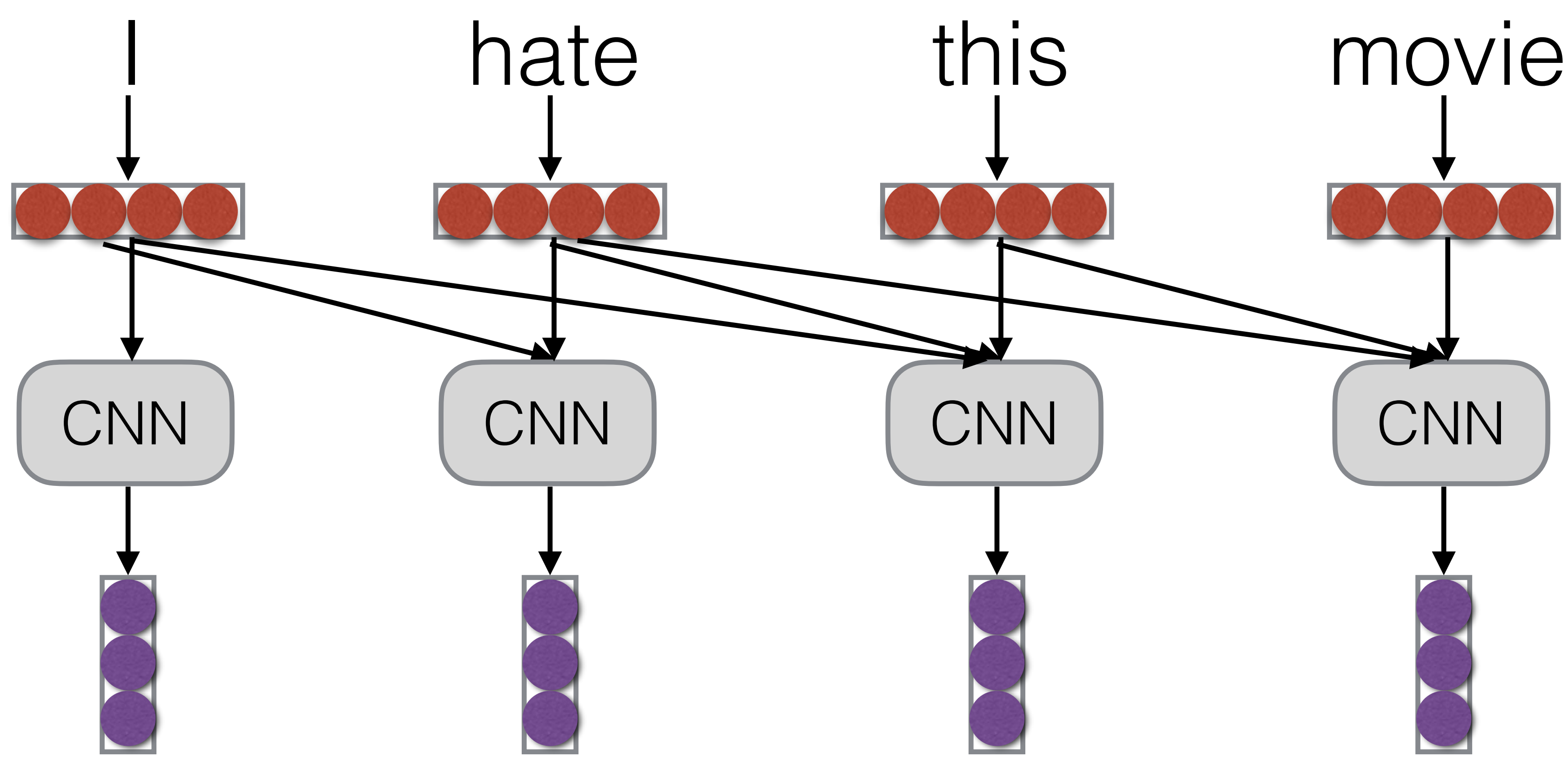
- Calculate based on local context



$$h_t = \mathrm{f}(W[x_{t-1}; x_t; x_{t+1}])$$

Convolution for Auto-regressive Models

- Functionally identical, just consider previous context



Other Desiderata of Sequence Models

Calibration (Guo+ 2017)

- The model "knows when it knows"
- More formally, the model probability of the answer matches the actual probability of getting it right
- Typically calculated by bucketing outputs and calculating "expected calibration error"

$$\mathrm{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} \left| \mathrm{acc}(B_m) - \mathrm{conf}(B_m) \right|$$

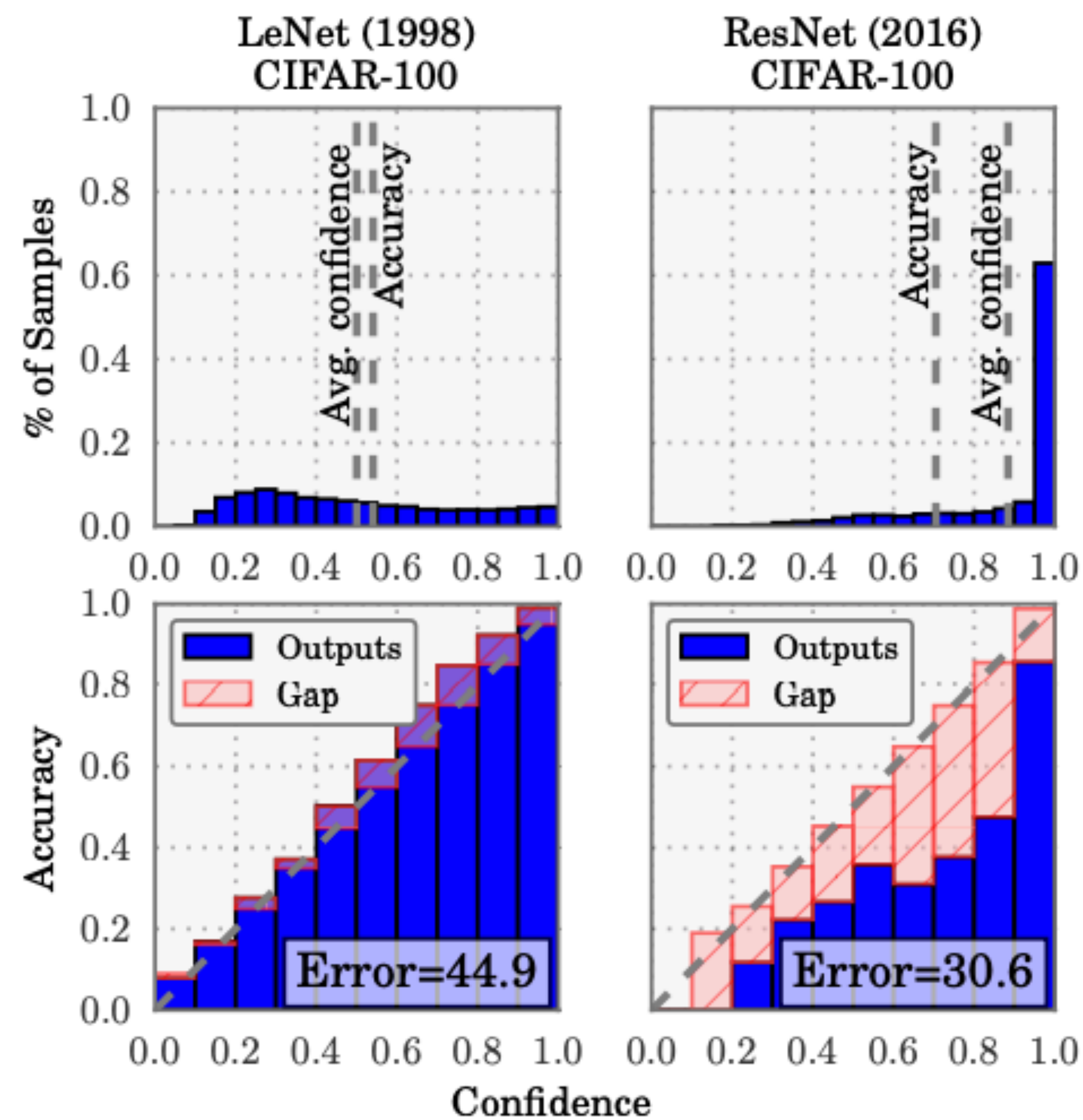


Figure 1. Confidence histograms (top) and reliability diagrams (bottom) for a 5-layer LeNet (left) and a 110-layer ResNet (right) on CIFAR-100. Refer to the text below for detailed illustration.

How to Calculate Answer Probability?

- Probability of the answer
- Probability of the answer + paraphrases (Jiang+ 2021)
- Sample multiple outputs, and count number of answers (Wang+ 2022)
- Ask the model what it thinks (Tian+ 2023)

Good comparison in Xiong+ (2023)

Efficiency

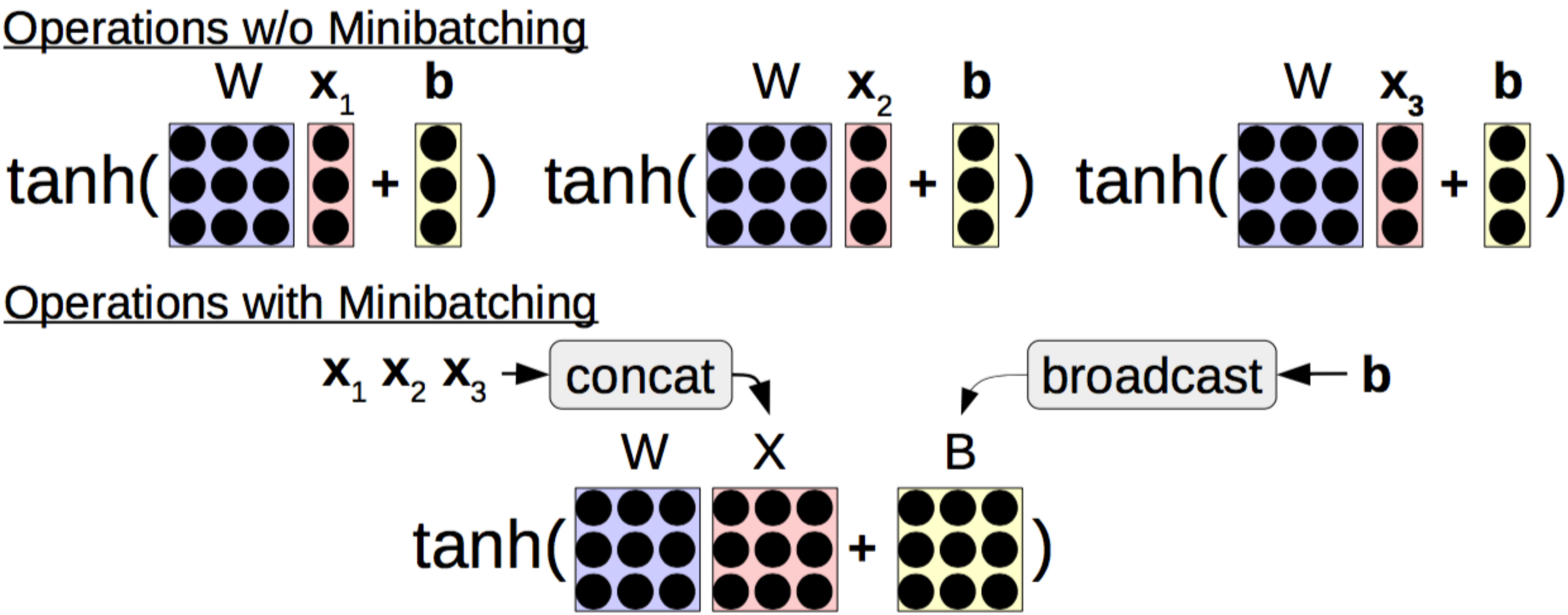
- The model is easy to run on limited hardware.
- **Metrics:**
    - Parameter count
    - Memory usage (model only, peak)
    - Latency (to first token, to last token)
    - Throughput
- See distillation/compression and generation algorithms classes

Efficiency Tricks

Efficiency Tricks: Mini-batching

- On modern hardware 10 operations of size 1 is **much slower than** 1 operation of size 10
- Minibatching combines together smaller operations into one big one

Minibatching



GPUs vs. CPUs

CPU, like a motorcycle

Quick to start, top speed not shabby

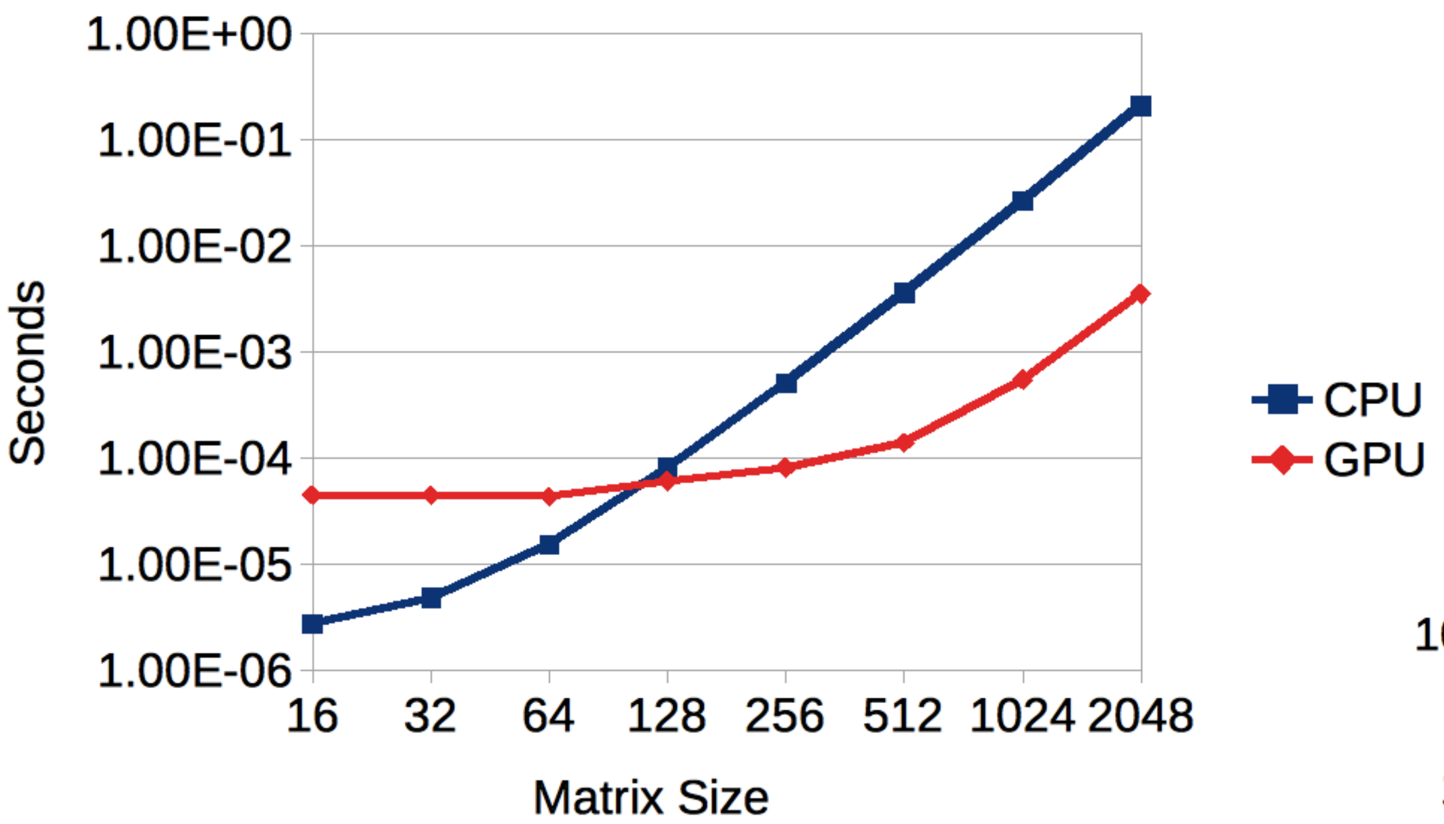
GPU, like an airplane

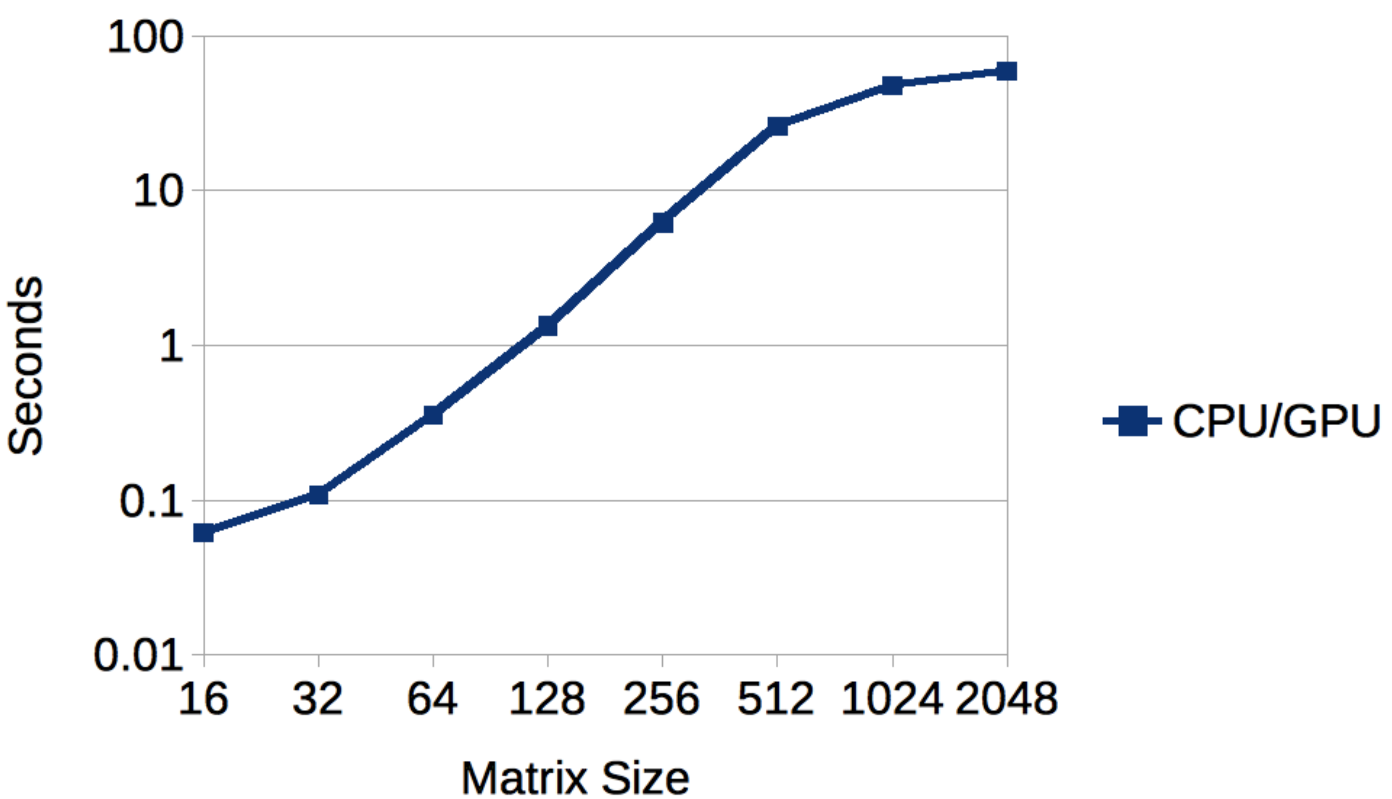
Takes forever to get off the ground, but super-fast once flying

Image Credit: Wikipedia

A Simple Example

- How long does a matrix-matrix multiply take?





Speed Trick 1: Don't Repeat Operations

- Something that you can do once at the beginning of the sentence, don't do it for every word!

Bad

```
for x in words_in_sentence:
  vals.append( W * c + x )
```

Good

```
W_c = W * c
for x in words_in_sentence:
  vals.append( W_c + x )
```

Speed Trick 2: Reduce # of Operations

- e.g. can you combine multiple matrix-vector multiplies into a single matrix-matrix multiply? Do so!

Bad

```
for x in words_in_sentence:
  vals.append( W * x )
val = dy.concatenate(vals)
```

Good

```
X = dy.concatenate_cols(words_in_sentence)
val = W * X
```

Speed Trick 3: Reduce CPU-GPU Data Movement

- Try to **avoid memory moves** between CPU and GPU.
- When you do move memory, try to do it as early as possible (GPU operations are asynchronous)

Bad

```
for x in words_in_sentence:
  # input data for x
  # do processing
```

Good

```
# input data for whole sentence
for x in words_in_sentence:
  # do processing
```

Questions?

(Attention in Next Class)