

CS11-747 Neural Networks for NLP

A Simple (?) Exercise: Predicting the Next Word

Graham Neubig



Carnegie Mellon University

Language Technologies Institute


Site

<https://phontron.com/class/nn4nlp2017/>

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.

Calculating the Probability of a Sentence

$$P(X) = \prod_{i=1}^I P(x_i \mid x_1, \dots, x_{i-1})$$


The diagram illustrates the components of the probability formula. A red horizontal line segment is positioned below the term x_i in the denominator, with a red arrow pointing from the text "Next Word" below it to this segment. A blue horizontal line segment is positioned below the terms x_1, \dots, x_{i-1} in the denominator, with a blue arrow pointing from the text "Context" below it to this segment.

The big problem: How do we predict

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

?!?!

Review: Count-based Language Models

Count-based Language Models

- Count up the frequency and divide:

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

- Add smoothing, to deal with zero counts:

$$P(x_i \mid x_{i-n+1}, \dots, x_{i-1}) = \lambda P_{ML}(x_i \mid x_{i-n+1}, \dots, x_{i-1}) \\ + (1 - \lambda) P(x_i \mid x_{1-n+2}, \dots, x_{i-1})$$

- Modified Kneser-Ney smoothing

A Refresher on Evaluation

- **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:**

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

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

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.

An Alternative: Featurized Log-Linear Models

An Alternative: Featurized Models

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

Example:

Previous words: “giving a”

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

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

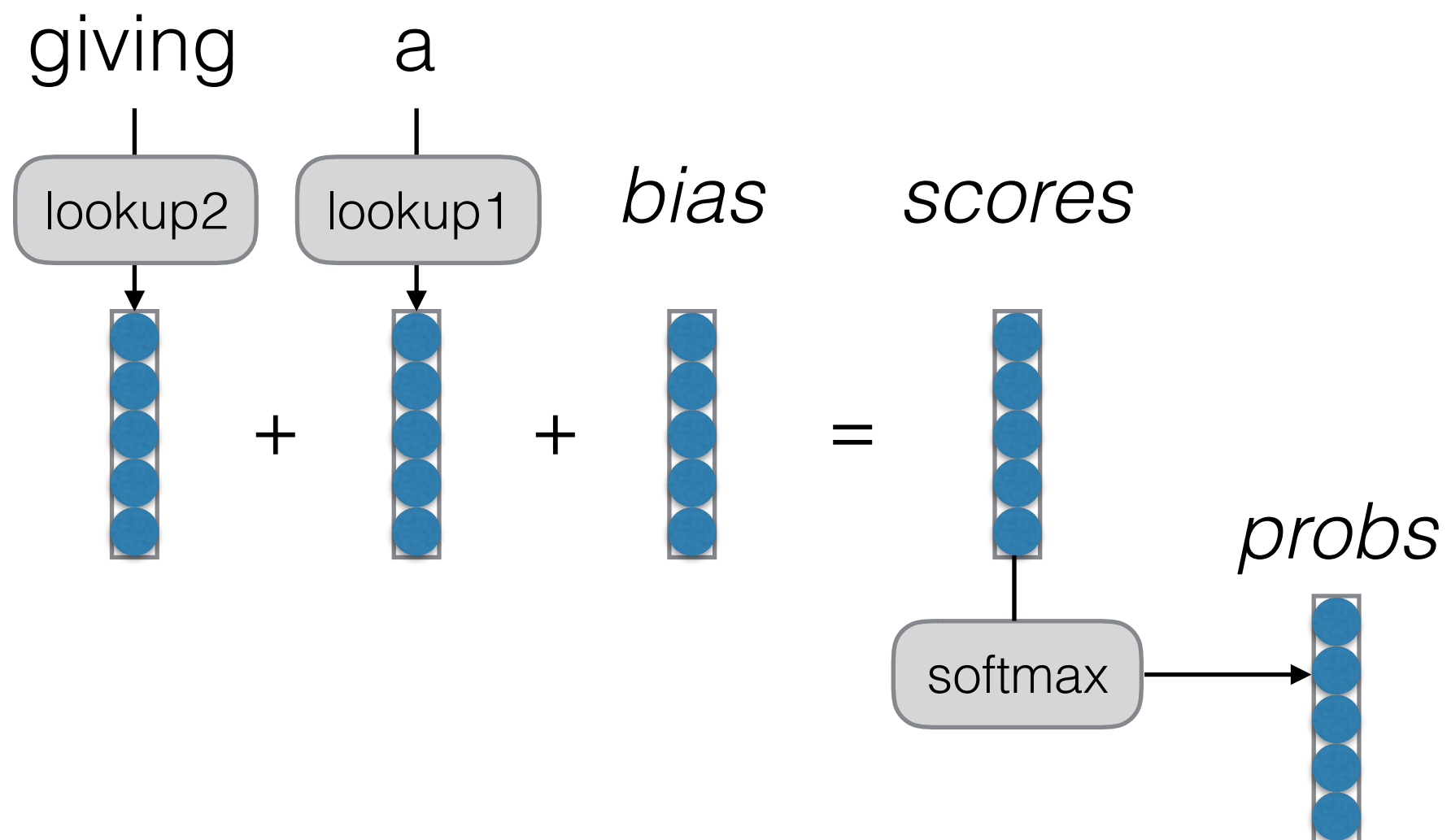
Softmax

- Convert scores into probabilities by taking the exponent and normalizing (softmax)

$$P(x_i \mid x_{i-n+1}^{i-1}) = \frac{e^{s(x_i \mid x_{i-n+1}^{i-1})}}{\sum_{\tilde{x}_i} e^{s(\tilde{x}_i \mid x_{i-n+1}^{i-1})}}$$

$$s = \begin{pmatrix} -3.2 \\ -2.9 \\ 1.0 \\ 2.2 \\ 0.6 \\ \dots \end{pmatrix} \longrightarrow p = \begin{pmatrix} 0.002 \\ 0.003 \\ 0.329 \\ 0.444 \\ 0.090 \\ \dots \end{pmatrix}$$

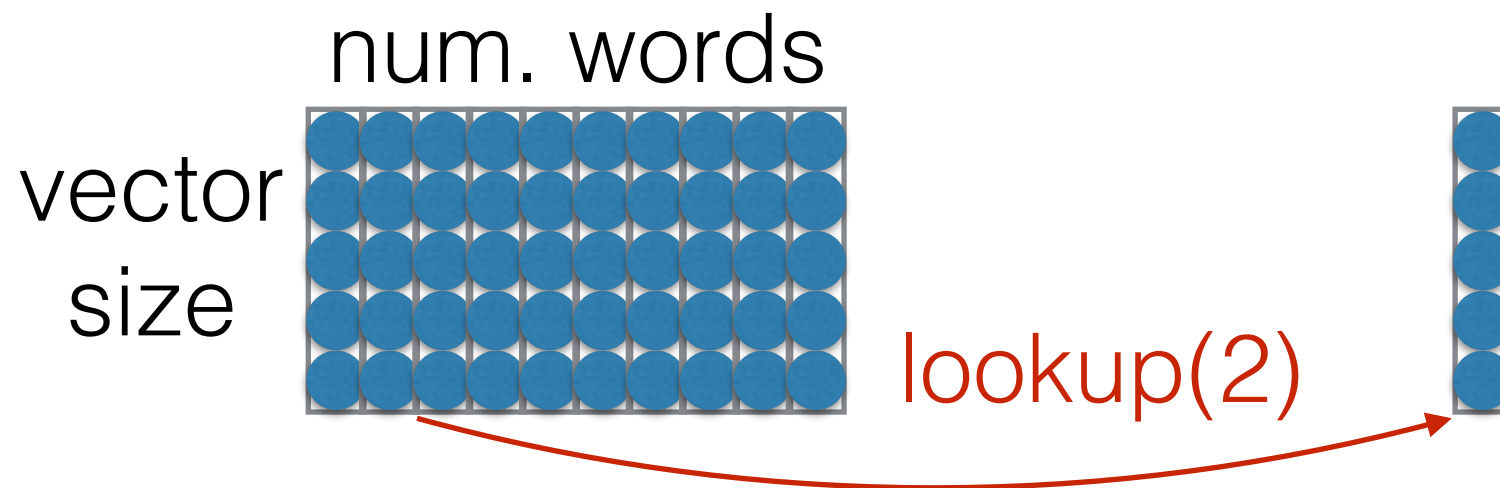
A Computation Graph View



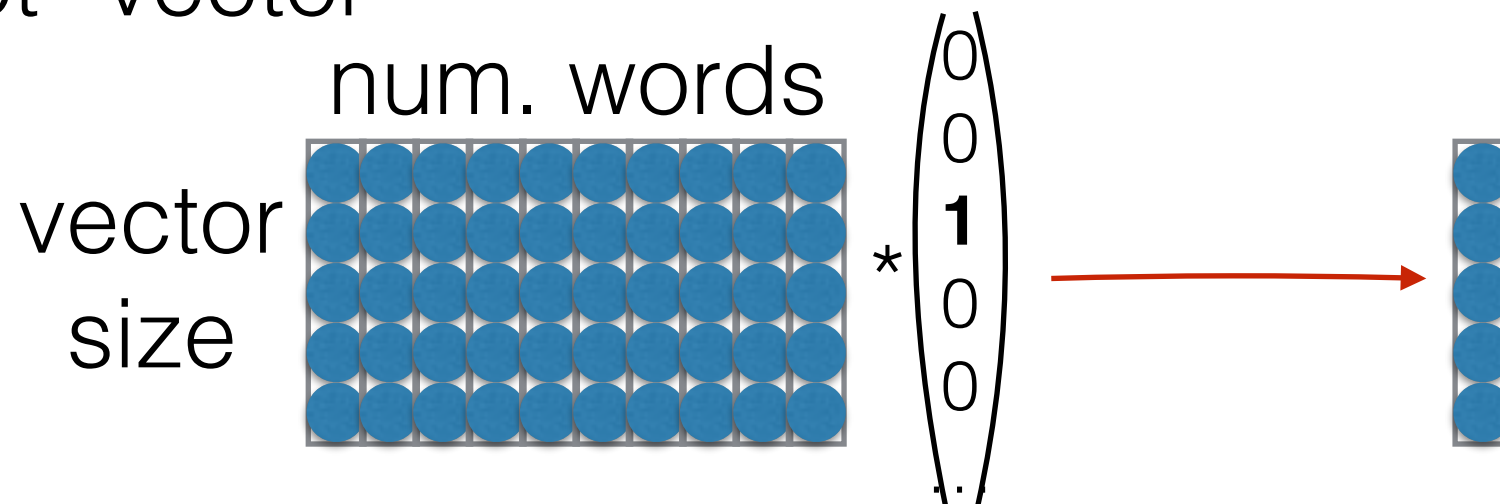
Each vector is size of output vocabulary

A Note: “Lookup”

- Lookup can be viewed as “grabbing” a single vector from a big matrix of word embeddings



- Similarly, can be viewed as multiplying by a “one-hot” vector



- Former tends to be faster

Training a Model

- **Reminder:** to train, we calculate a “loss function” (a measure of how bad our predictions are), and move the parameters to reduce the loss
- The most common loss function for probabilistic models is “negative log likelihood”

If element 3
(or zero-indexed, 2)
is the correct answer:

$$p = \begin{pmatrix} 0.002 \\ 0.003 \\ \boxed{0.329} \\ 0.444 \\ 0.090 \\ \dots \end{pmatrix} \xrightarrow{-\log} 1.112$$

Parameter Update

- Back propagation allows us to calculate the derivative of the loss with respect to the parameters

$$\frac{\partial \ell}{\partial \theta}$$

- Simple stochastic gradient descent optimizes parameters according to the following rule

$$\theta \leftarrow \theta - \alpha \frac{\partial \ell}{\partial \theta}$$

Choosing a Vocabulary

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:
 - Frequency threshold (usually $\text{UNK} \leq 1$)
 - Rank threshold

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

Let's try it out!
(loglin-lm.py)

What Problems are Handled?

- Cannot share strength among **similar words**

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

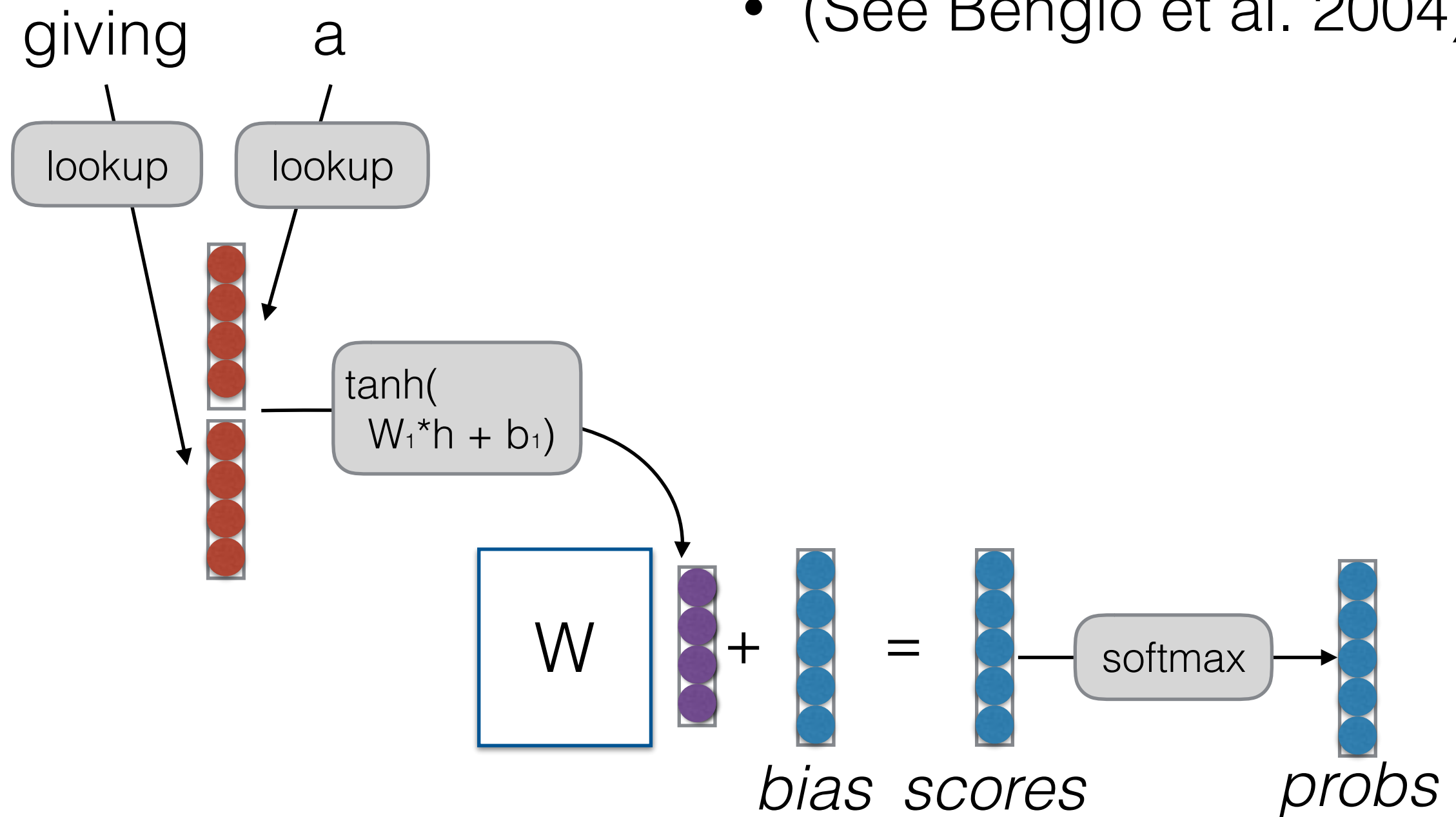
Linear Models can't Learn Feature Combinations

farmers eat steak → high	cows eat steak → low
farmers eat hay → low	cows eat hay → high

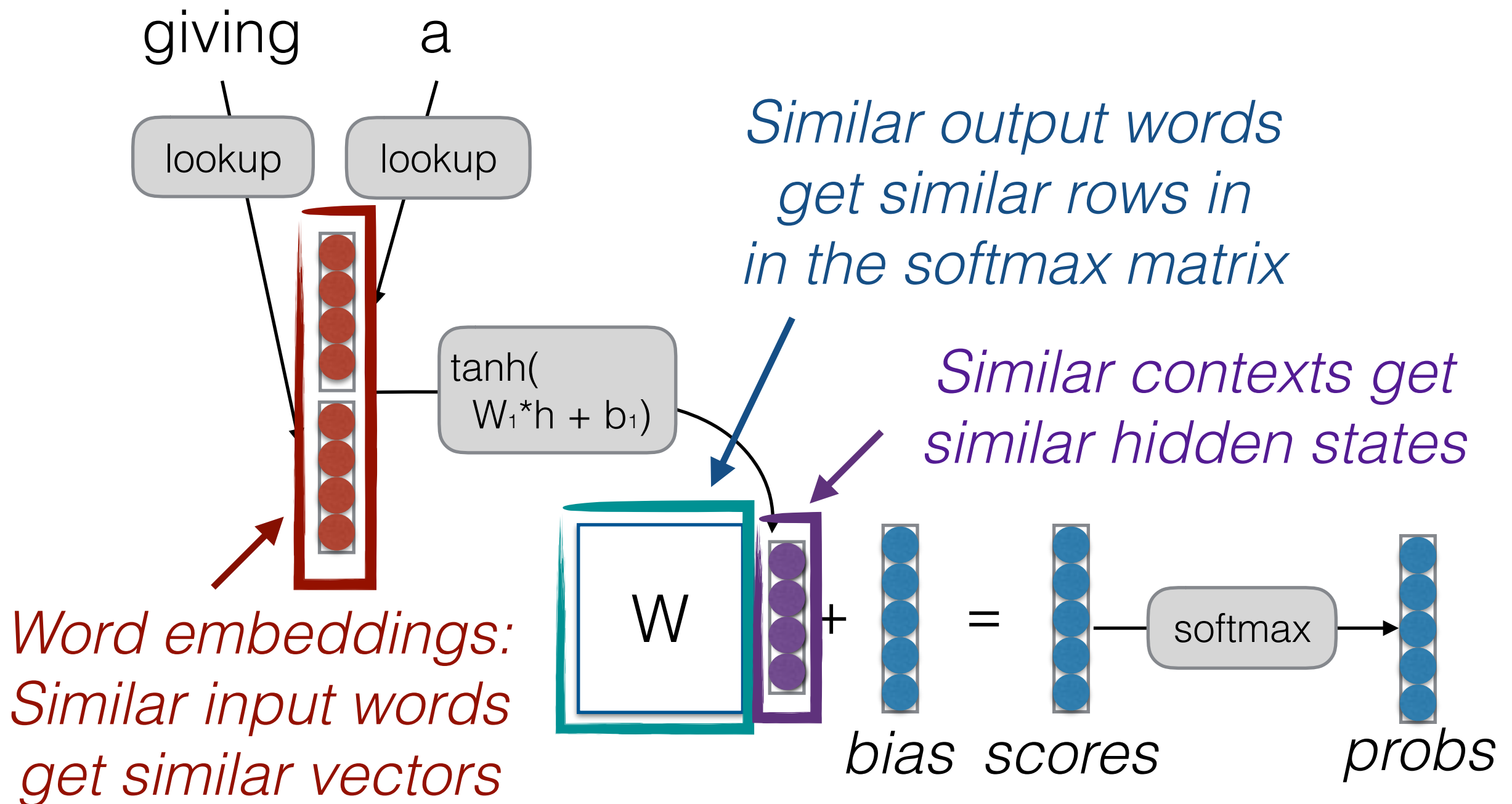
- These can't be expressed by linear features
- What can we do?
 - Remember combinations as features (individual scores for “farmers eat”, “cows eat”) → Feature space explosion!
 - Neural nets

Neural Language Models

- (See Bengio et al. 2004)



Where is Strength Shared?



What Problems are Handled?

- Cannot share strength among **similar words**

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

→ solved, and similar contexts as well! 😊

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

Dr. Jane Smith Dr. Gertrude Smith

→ solved! 😊

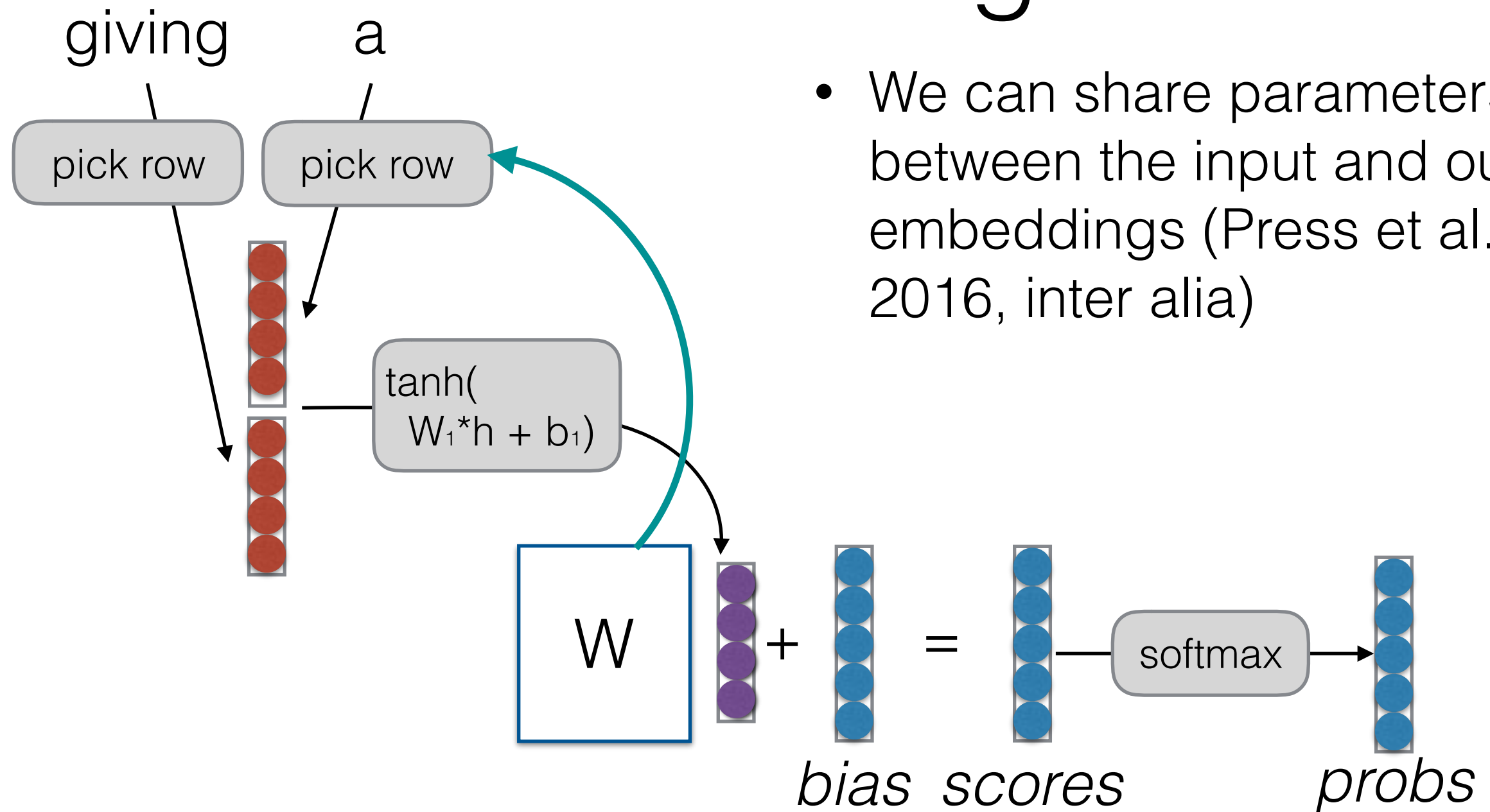
- Cannot handle **long-distance dependencies**

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

→ not solved yet 😞

Let's Try it Out!
(nn-lm.py)

Tying Input/Output Embeddings



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

Training Tricks

Shuffling the Training Data

- Stochastic gradient methods update the parameters a little bit at a time
 - What if we have the sentence “I love this sentence so much!” at the end of the training data 50 times?
- To train correctly, we should randomly shuffle the order at each time step

Other Optimization Options

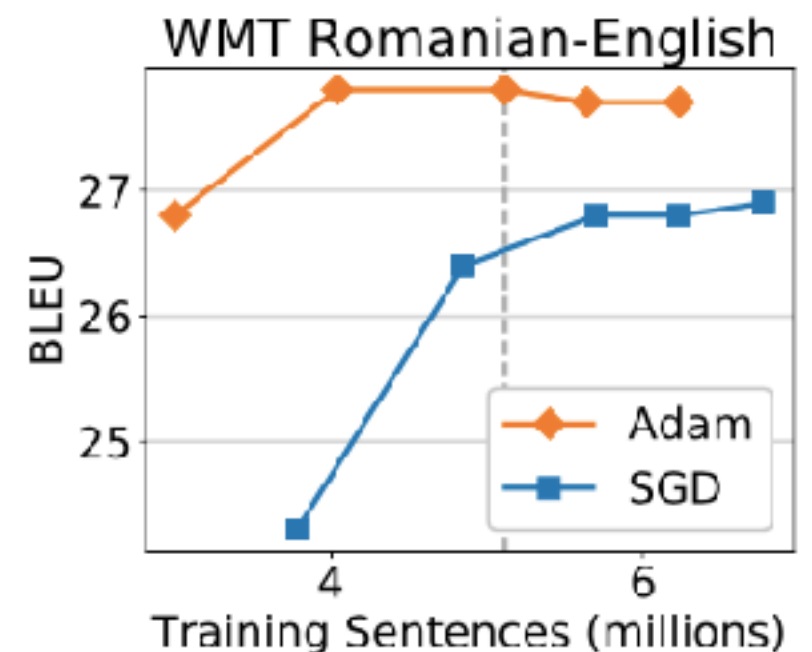
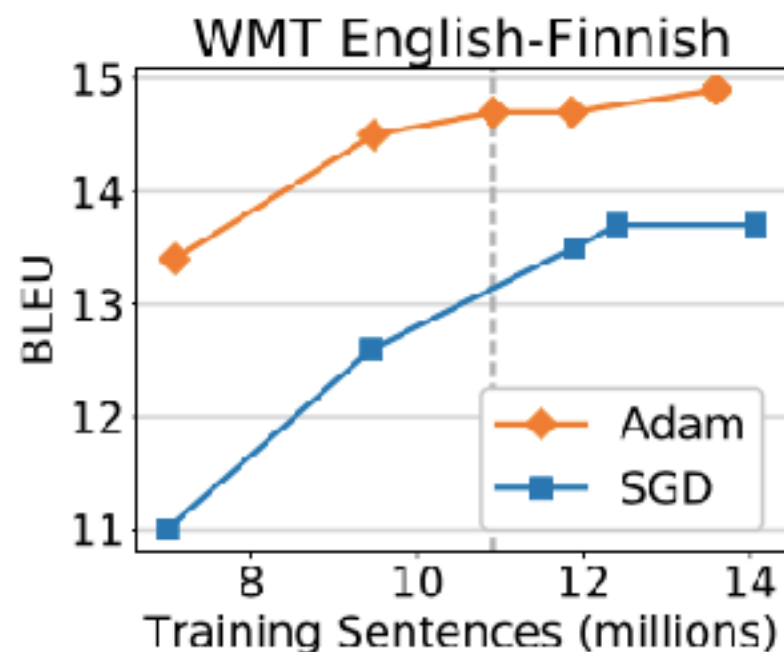
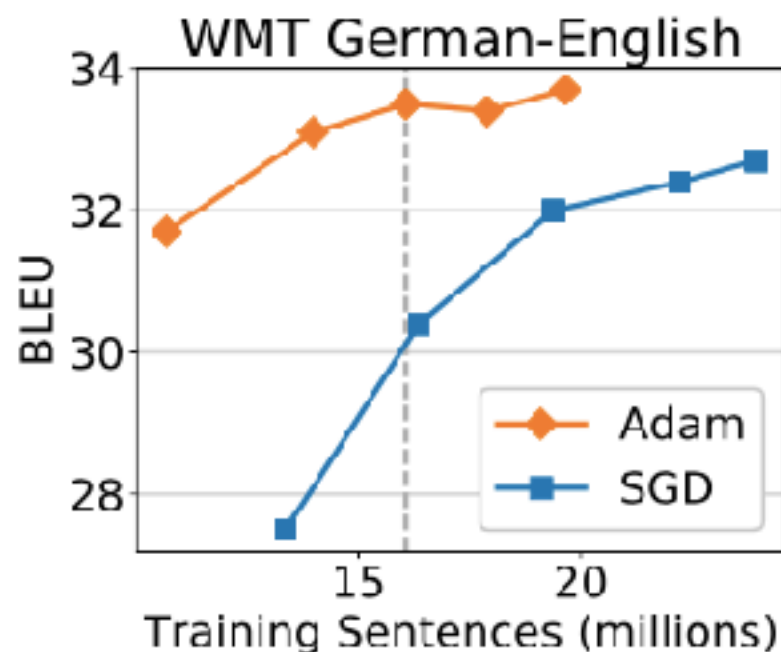
- **SGD with Momentum:** Remember gradients from past time steps to prevent sudden changes
- **Adagrad:** Adapt the learning rate to reduce learning rate for frequently updated parameters (as measured by the variance of the gradient)
- **Adam:** Like Adagrad, but keeps a running average of momentum and gradient variance
- **Many others:** RMSProp, Adadelata, etc.
(See Ruder 2016 reference for more details)

Early Stopping, Learning Rate Decay

- Neural nets have tons of parameters: we want to prevent them from over-fitting
- We can do this by monitoring our performance on held-out development data and stopping training when it starts to get worse
- It also sometimes helps to reduce the learning rate and continue training

Which One to Use?

- Adam is usually fast to converge and stable
- But simple SGD tends to do very well in terms of generalization
- You should use learning rate decay, (e.g. on Machine translation results by Denkowski & Neubig 2017)



Dropout

- Neural nets have lots of parameters, and are prone to overfitting
- Dropout: randomly zero-out nodes in the hidden layer with probability p at **training time only**



- Because the number of nodes at training/test is different, scaling is necessary:
 - Standard dropout: scale by p at test time
 - Inverted dropout: scale by $1/(1-p)$ at training time

Let's Try it Out!
(nn-lm-optim.py)

Efficiency Tricks: Operation Batching

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

Operations w/o Minibatching

$$\tanh\left(\begin{matrix} W \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{matrix} \begin{matrix} x_1 \\ \bullet \\ \bullet \\ \bullet \end{matrix} + \begin{matrix} b \\ \bullet \\ \bullet \\ \bullet \end{matrix}\right) \quad \tanh\left(\begin{matrix} W \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{matrix} \begin{matrix} x_2 \\ \bullet \\ \bullet \\ \bullet \end{matrix} + \begin{matrix} b \\ \bullet \\ \bullet \\ \bullet \end{matrix}\right) \quad \tanh\left(\begin{matrix} W \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{matrix} \begin{matrix} x_3 \\ \bullet \\ \bullet \\ \bullet \end{matrix} + \begin{matrix} b \\ \bullet \\ \bullet \\ \bullet \end{matrix}\right)$$

Operations with Minibatching

$$\begin{matrix} x_1 & x_2 & x_3 \end{matrix} \rightarrow \text{concat} \rightarrow \begin{matrix} W & X \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \end{matrix} \quad \begin{matrix} \text{broadcast} \leftarrow b \\ \downarrow \\ B \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{matrix}$$
$$\tanh\left(\begin{matrix} W & X \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \end{matrix} + \begin{matrix} B \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{matrix}\right)$$

Manual Mini-batching

- Group together similar operations (e.g. loss calculations for a single word) and execute them all together
- In the case of a feed-forward language model, each word prediction in a sentence can be batched
- For recurrent neural nets, etc., more complicated
- DyNet has special minibatch operations for lookup and loss functions, everything else automatic

Mini-batched Code Example

```
1 # in_words is a tuple (word_1, word_2)
2 # out_label is an output label
3 word_1 = E[in_words[0]]
4 word_2 = E[in_words[1]]
5 scores_sym = W*dy.concatenate([word_1, word_2])+b
6 loss_sym = dy.pickneglogsoftmax(scores_sym, out_label)
```

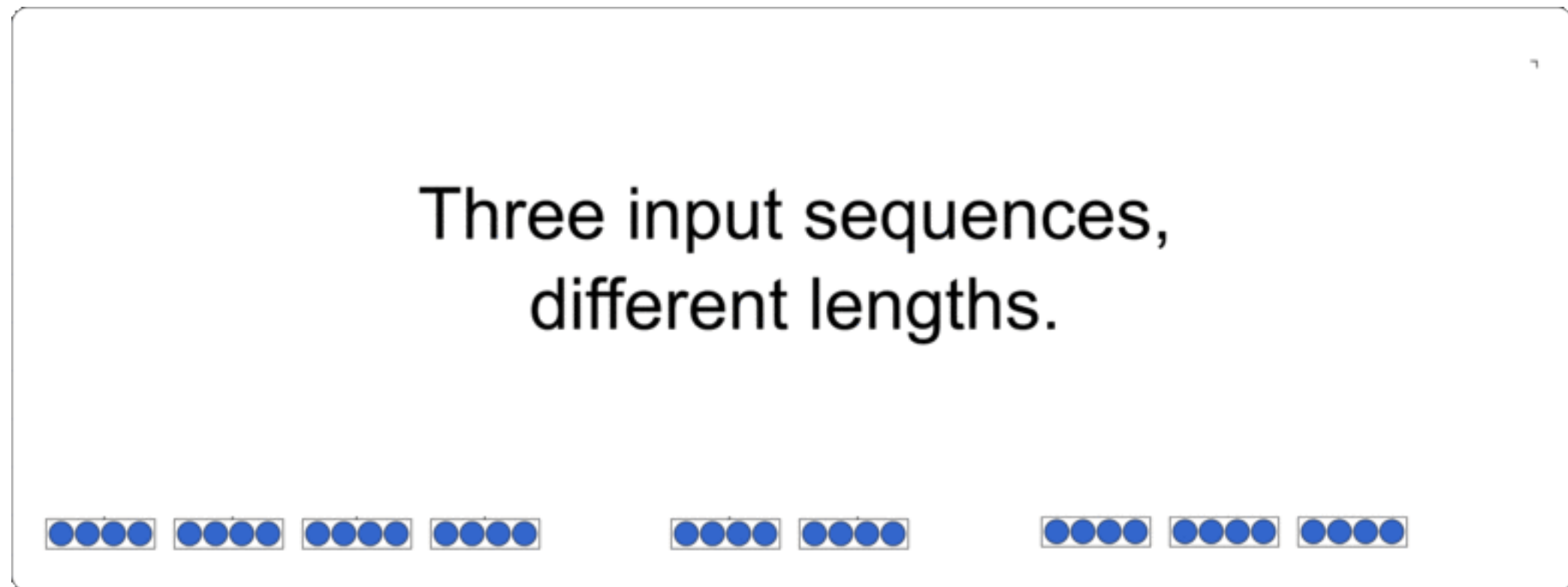
(a) Non-minibatched classification.

```
1 # in_words is a list [(word_{1,1}, word_{1,2}), (word_{2,1}, word_{2,2}), ...]
2 # out_labels is a list of output labels [label_1, label_2, ...]
3 word_1_batch = dy.lookup_batch(E, [x[0] for x in in_words])
4 word_2_batch = dy.lookup_batch(E, [x[1] for x in in_words])
5 scores_sym = W*dy.concatenate([word_1_batch, word_2_batch])+b
6 loss_sym = dy.sum_batches( dy.pickneglogsoftmax_batch(scores_sym, out_labels) )
```

(b) Minibatched classification.

Let's Try it Out!
(nn-lm-batch.py)

Automatic Mini-batching!



- (see Neubig et al. 2017)
- Try it with the `-dynet-autobatch` command line option

Autobatching Usage

- for each minibatch:
 - for each data point in mini-batch:
 - **define/add data**
 - **sum losses**
 - **forward** (autobatch engine does magic!)
 - **backward**
 - **update**

Speed Improvements

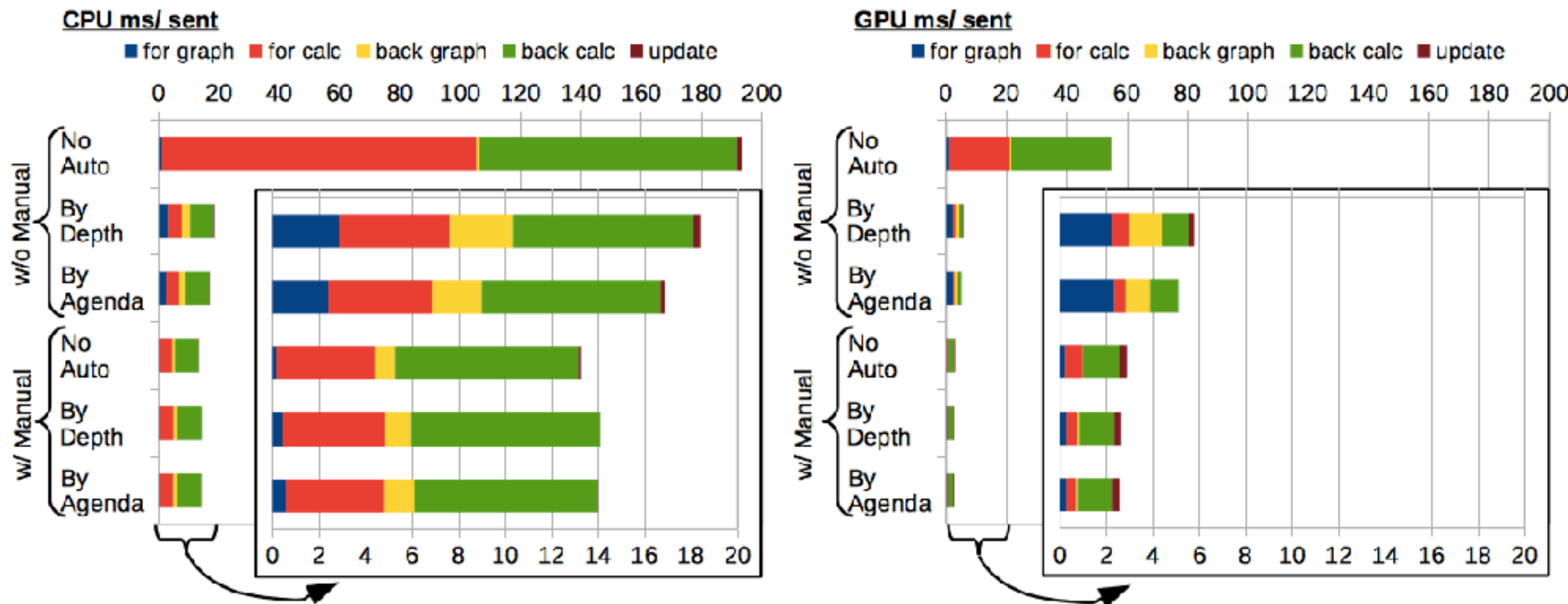


Table 1: Sentences/second on various training tasks for increasingly challenging batching scenarios.

Task	CPU			GPU		
	NOAUTO	BYDEPTH	BYAGENDA	NOAUTO	BYDEPTH	BYAGENDA
BiLSTM	16.8	139	156	56.2	337	367
BiLSTM w/ char	15.7	93.8	132	43.2	183	275
TreeLSTM	50.2	348	357	76.5	672	661
Transition-Parsing	16.8	61.0	61.2	33.0	89.5	90.1

Questions?