

CS11-747 Neural Networks for NLP

Language Modeling, Efficiency/Training Tricks

Graham Neubig



Carnegie Mellon University

Language Technologies Institute

Site

<https://phontron.com/class/nn4nlp2021/>

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.

Language Modeling: Calculating the Probability of a Sentence

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

Next Word Context

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
the
talk
gift
hat
...

$$b = \begin{pmatrix} 3.0 \\ 2.5 \\ -0.2 \\ 0.1 \\ 1.2 \\ \dots \end{pmatrix}$$

$$w_{1,a} = \begin{pmatrix} -6.0 \\ -5.1 \\ 0.2 \\ 0.1 \\ 0.5 \\ \dots \end{pmatrix}$$

$$w_{2,giving} = \begin{pmatrix} -0.2 \\ -0.3 \\ 1.0 \\ 2.0 \\ -1.2 \\ \dots \end{pmatrix}$$

$$s = \begin{pmatrix} -3.2 \\ -2.9 \\ 1.0 \\ 2.2 \\ 0.6 \\ \dots \end{pmatrix}$$

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

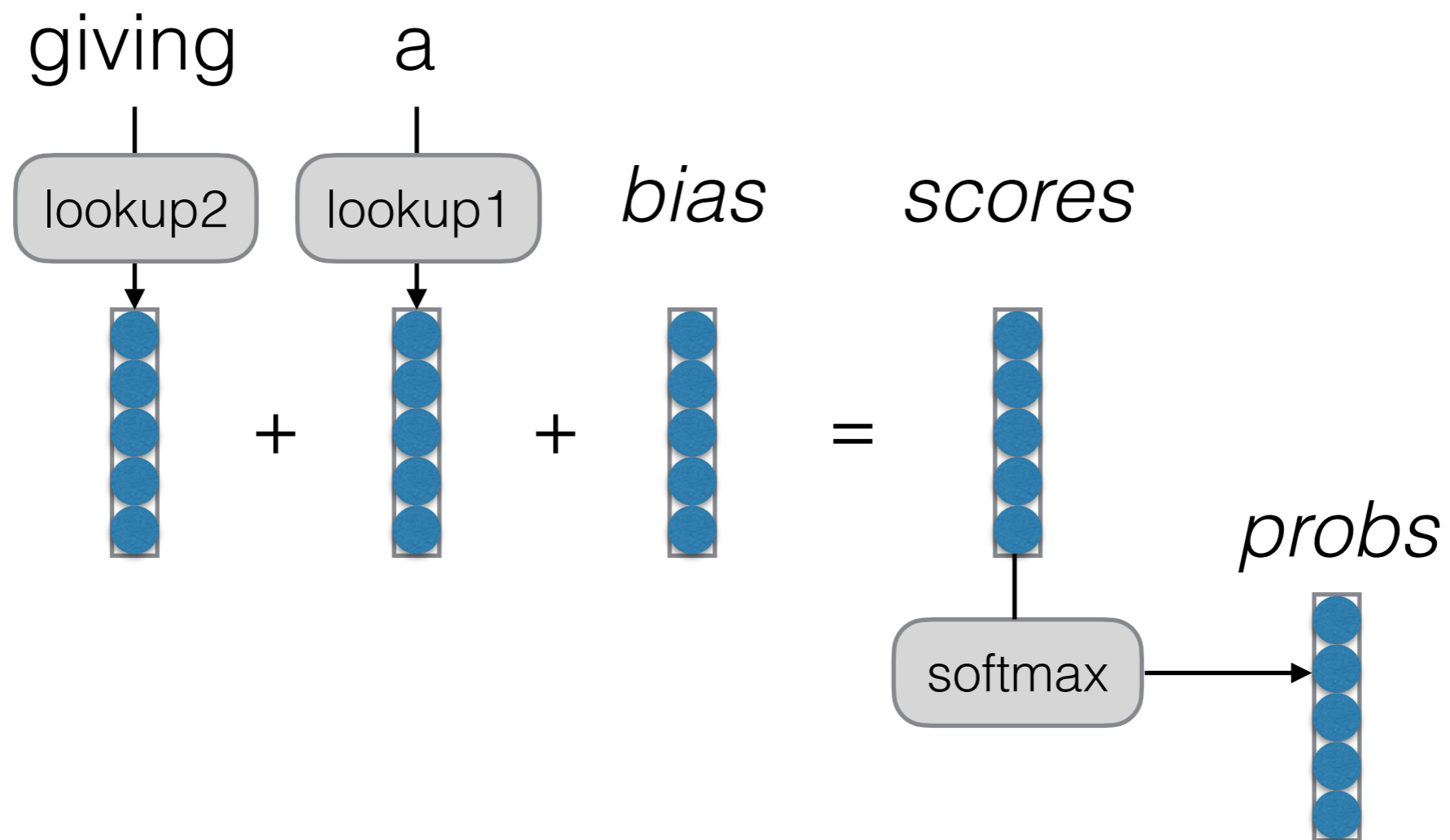
Softmax

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

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

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

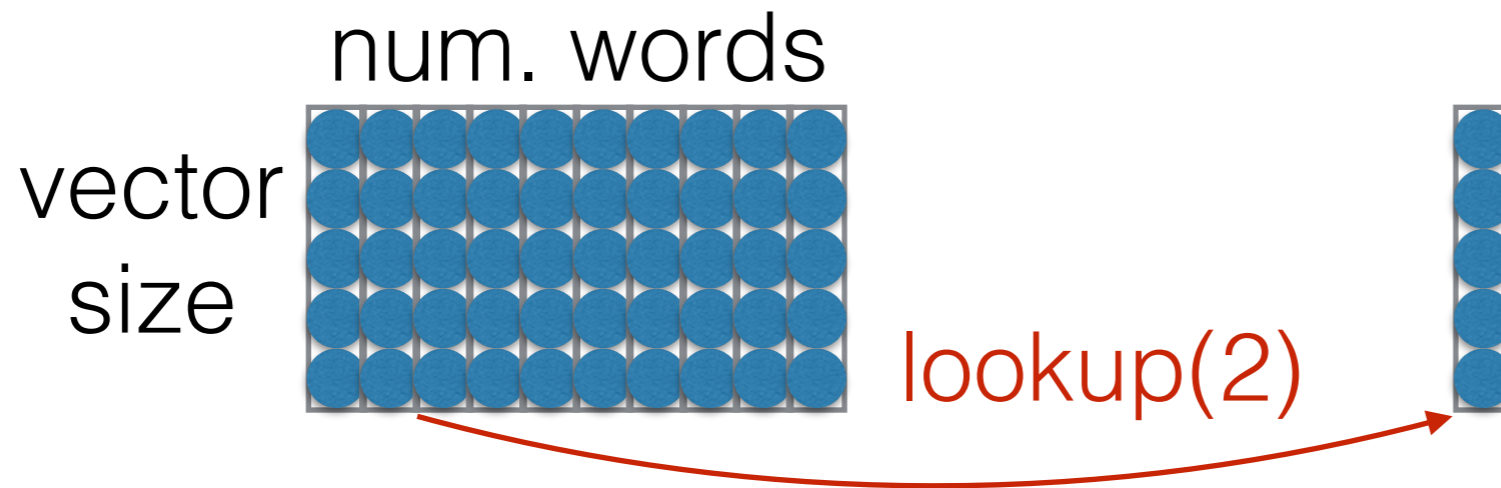
A Computation Graph View



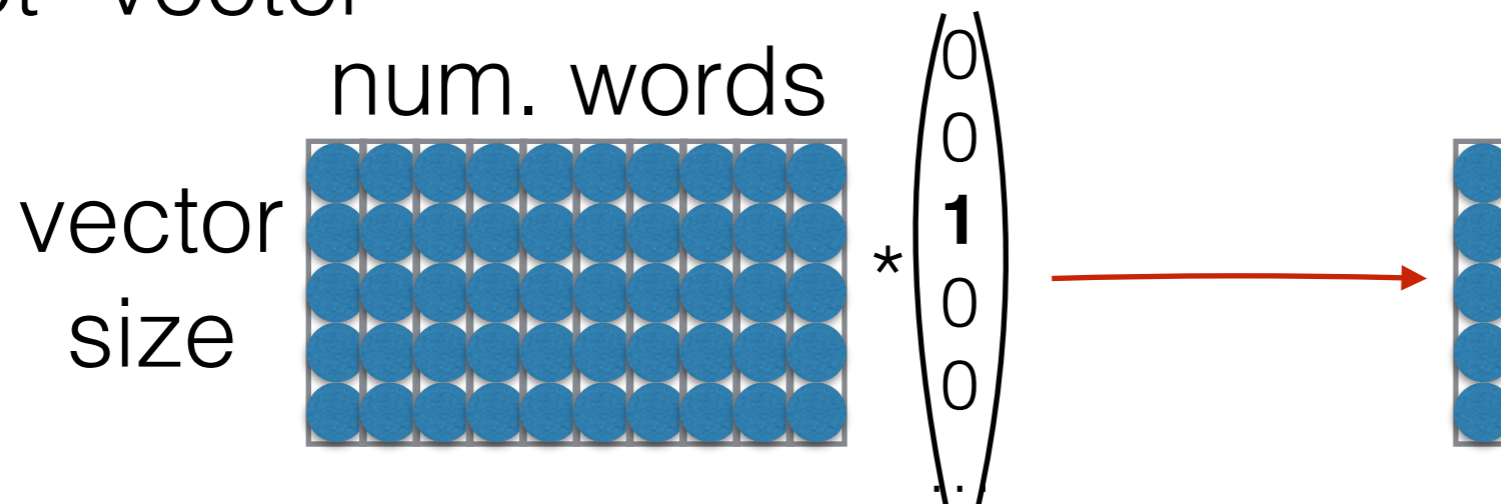
Each vector is size of output vocabulary

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

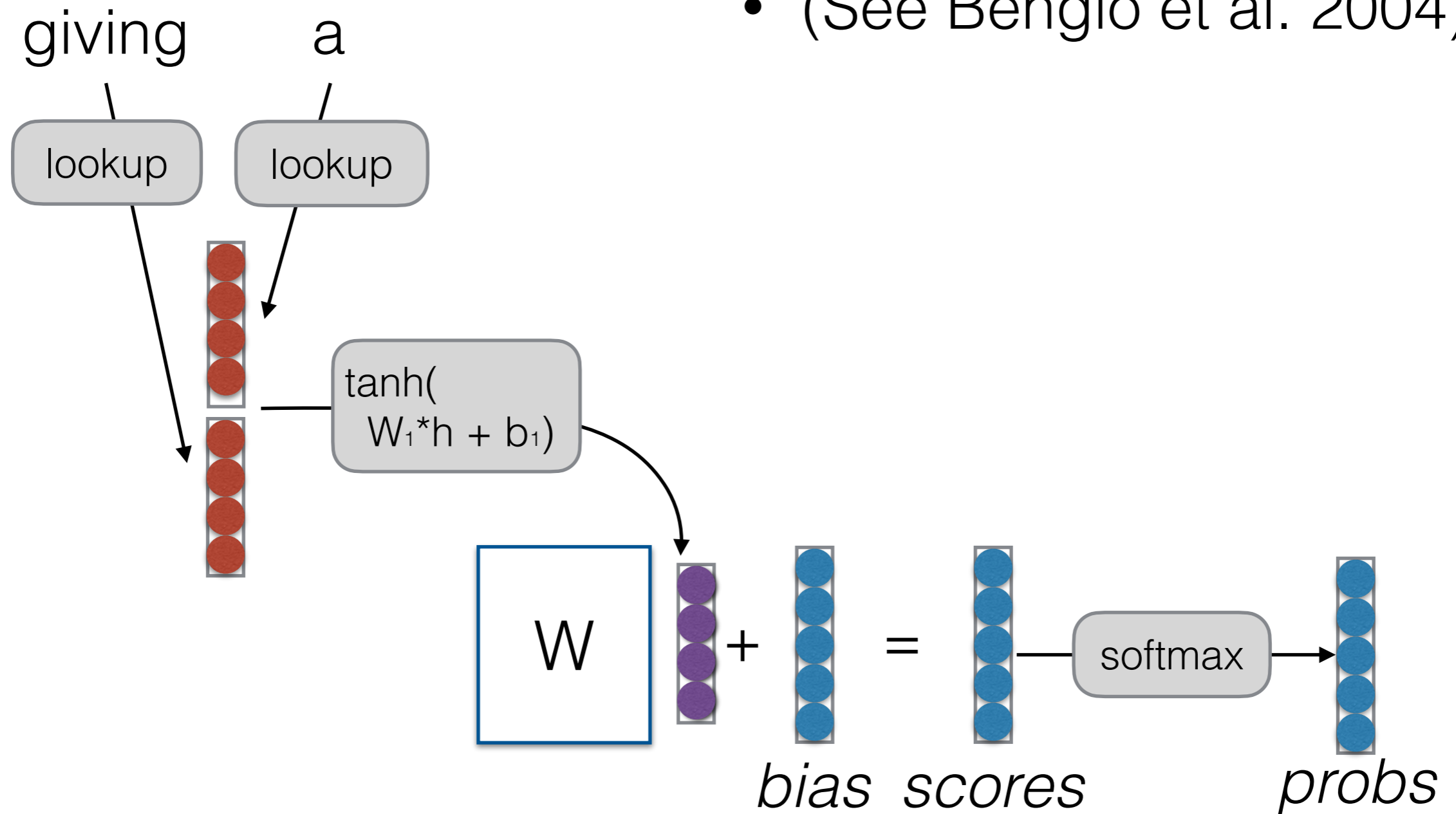
Linear Models can't Learn Feature Combinations

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

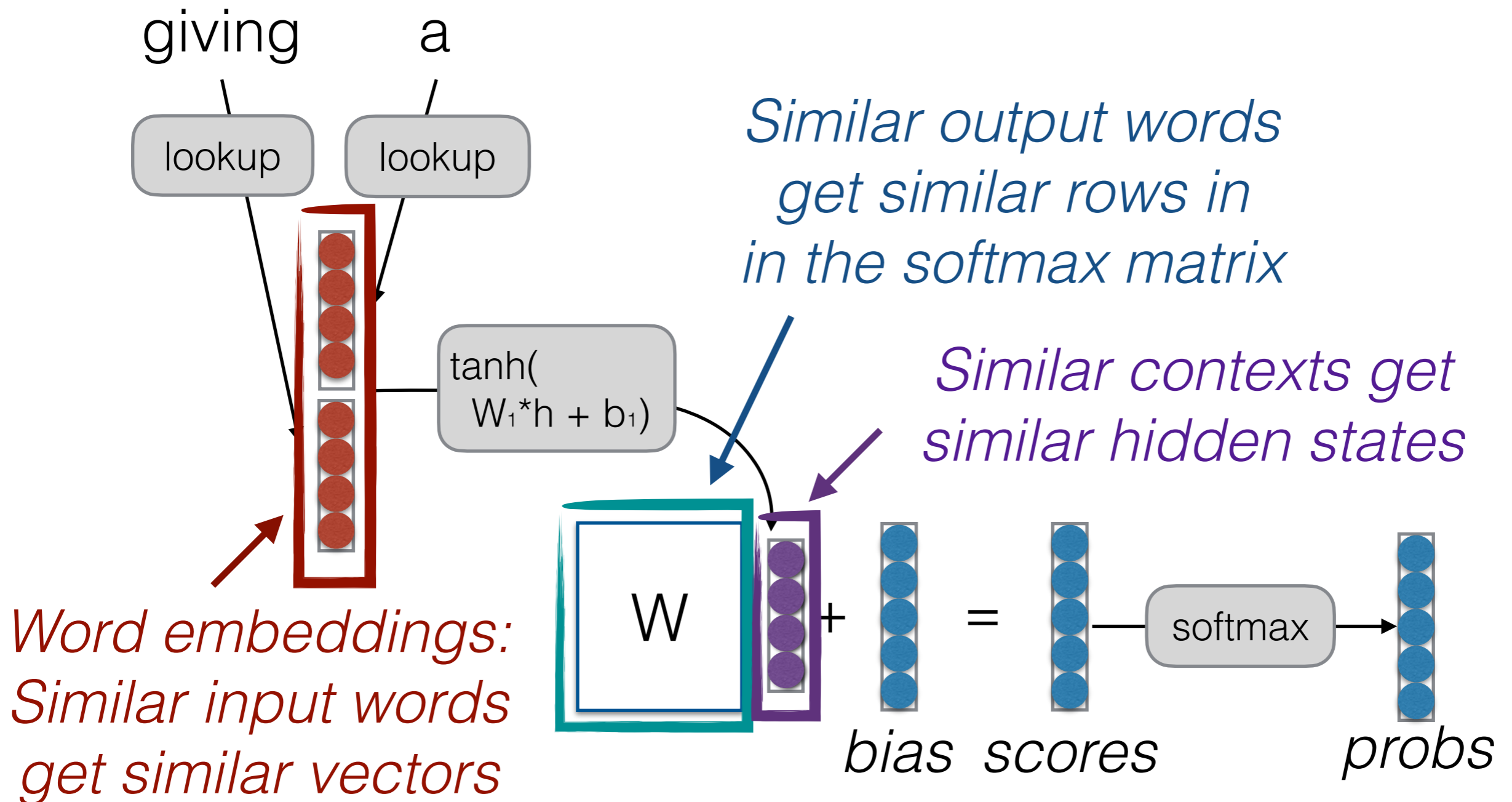
- These can't be expressed by linear features
- What can we do?
 - Remember combinations as features (individual scores for “students take”, “teachers write”) → Feature space explosion!
 - Neural nets

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! 😊

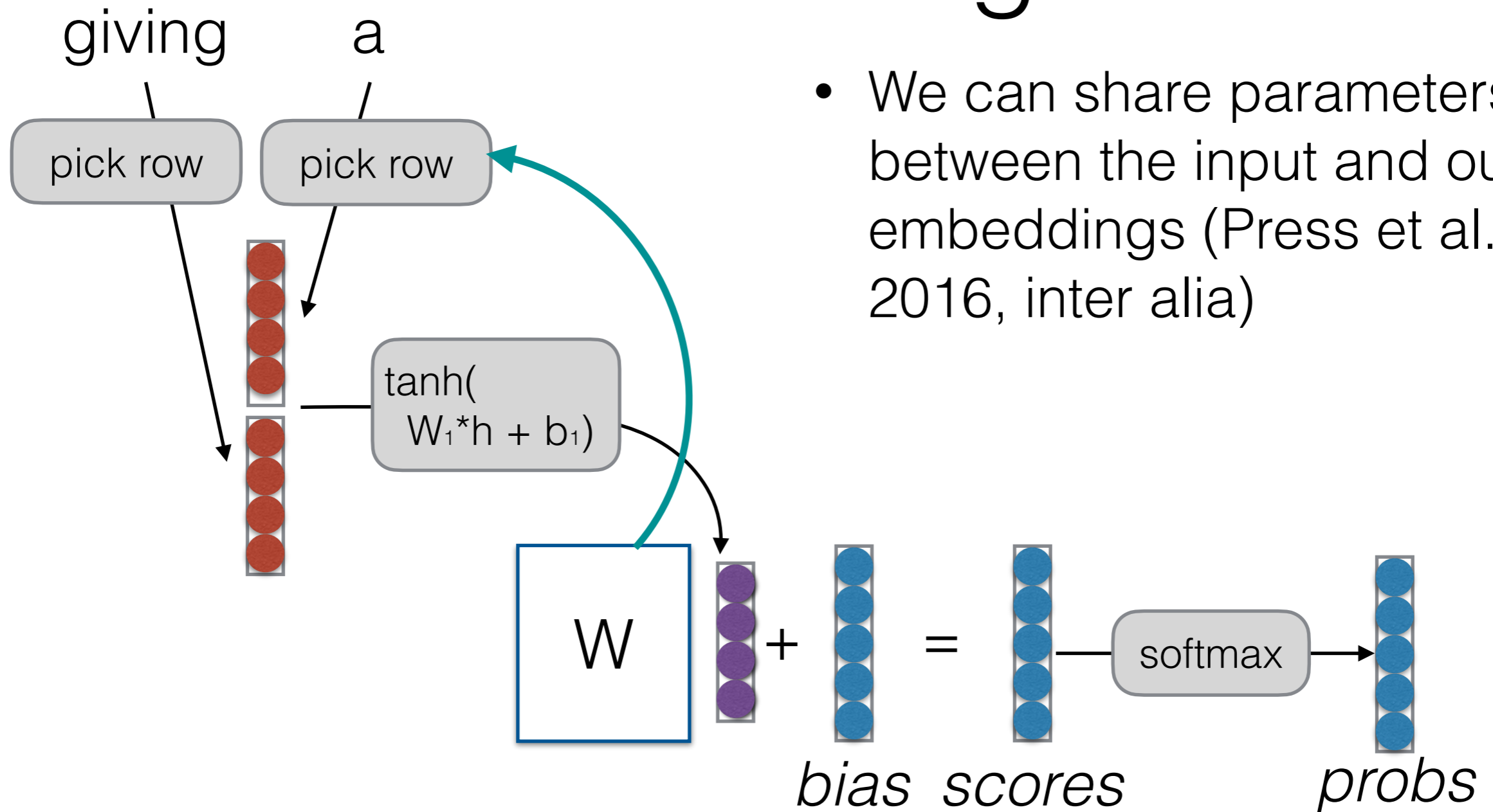
- Cannot handle **long-distance dependencies**

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



- 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.

Optimizers

Standard SGD

- **Reminder:** Standard stochastic gradient descent does

$$g_t = \nabla_{\theta_{t-1}} \ell(\theta_{t-1})$$

Gradient of Loss

$$\theta_t = \theta_{t-1} - \eta g_t$$

Learning Rate

- There are many other optimization options! (see Ruder 2016 in references)

SGD With Momentum

- Remember gradients from past time steps

$$v_t = \gamma v_{t-1} + \eta g_t$$

Momentum

Momentum
Conservation
Parameter

Previous Momentum

$$\theta_t = \theta_{t-1} - v_t$$

- Intuition:** Prevent instability resulting from sudden changes

Adagrad

- Adaptively reduce learning rate based on accumulated variance of the gradients

$$G_t = G_{t-1} + \underline{g_t \odot g_t}$$

Squared Current Gradient

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

— Small Constant

- **Intuition:** frequently updated parameters (e.g. common word embeddings) should be updated less
- **Problem:** learning rate continuously decreases, and training can stall -- fixed by using rolling average in *AdaDelta* and *RMSProp*

Adam

- Most standard optimization option in NLP and beyond
- Considers rolling average of gradient, and momentum

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad \text{Momentum}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t \odot g_t \quad \text{Rolling Average of Gradient}$$

- Correction of bias early in training

$$\hat{m}_t = \frac{m_t}{1 - (\beta_1)^t} \quad \hat{v}_t = \frac{v_t}{1 - (\beta_2)^t}$$

- Final update

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Training Tricks

Shuffling the Training Data

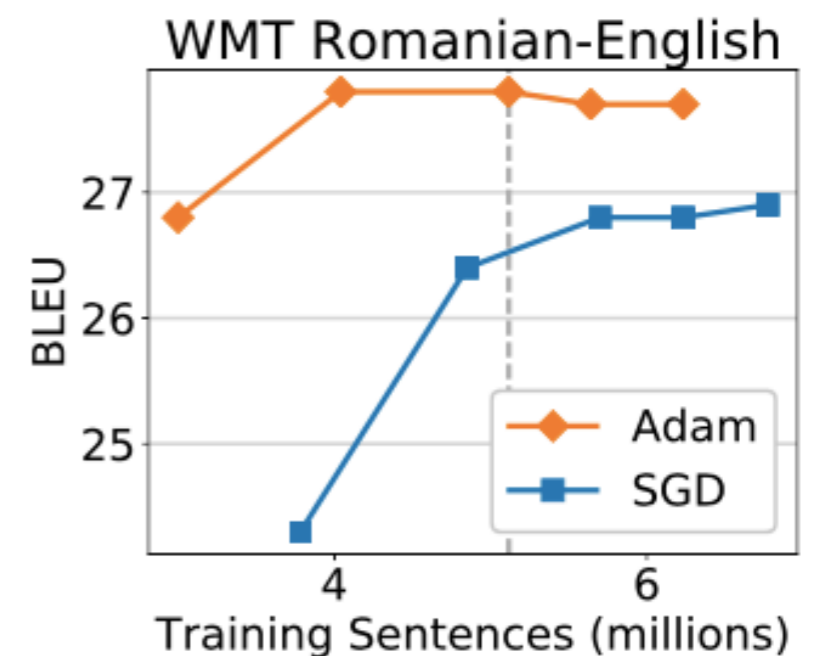
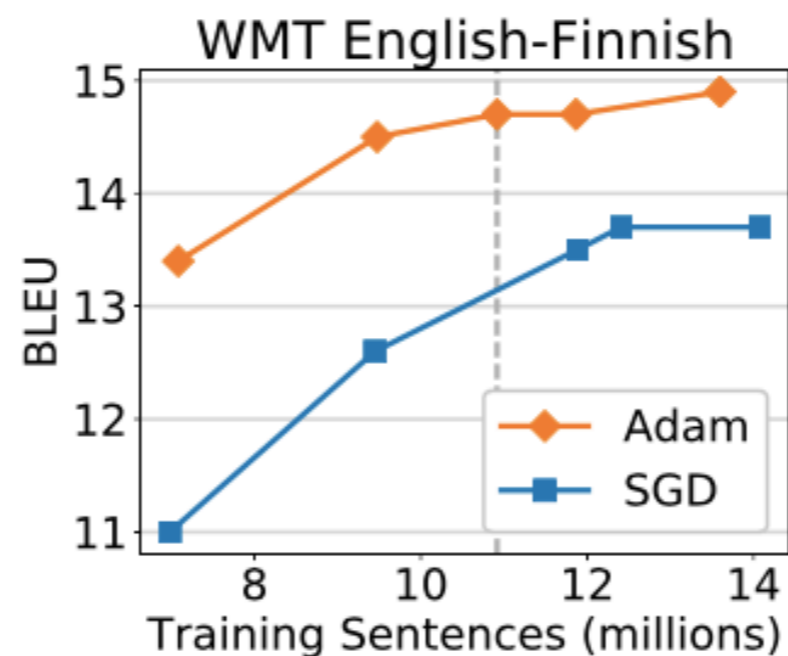
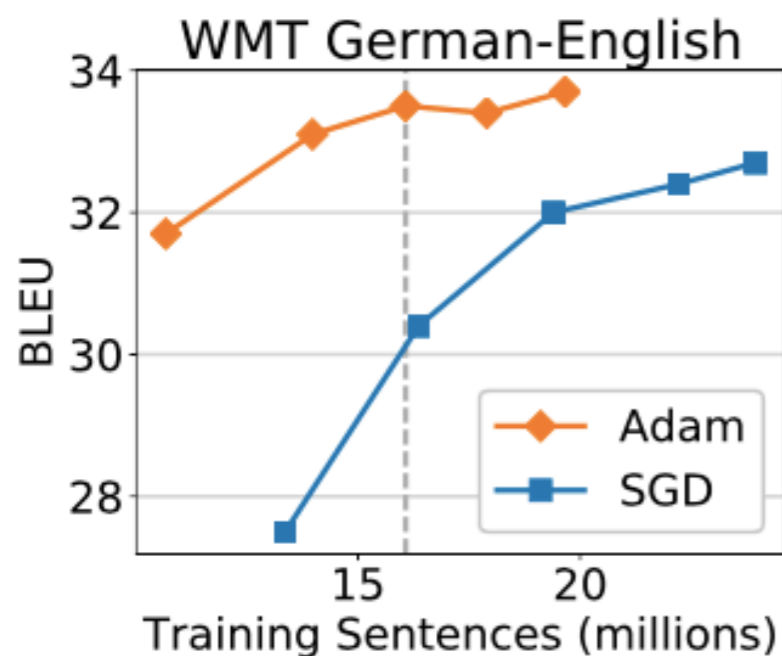
- Stochastic gradient methods update the parameters a little bit at a time
 - What if we have the sentence “I love this sentence so much!” at the end of the training data 50 times?
- To train correctly, we should randomly shuffle the order at each time step

Simple Methods to Prevent Over-fitting

- Neural nets have tons of parameters: we want to prevent them from over-fitting
- **Early stopping:**
 - monitor performance on held-out development data and stop training when it starts to get worse
- **Learning rate decay:**
 - gradually reduce learning rate as training continues, or
 - reduce learning rate when dev performance plateaus
- **Patience:**
 - learning can be unstable, so sometimes avoid stopping or decay until the dev performance gets worse n times

Which One to Use?

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



Dropout

(Srivastava+ 14)

- 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
- An alternative: **DropConnect** (Wan+ 2013) instead zeros out weights in the NN

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

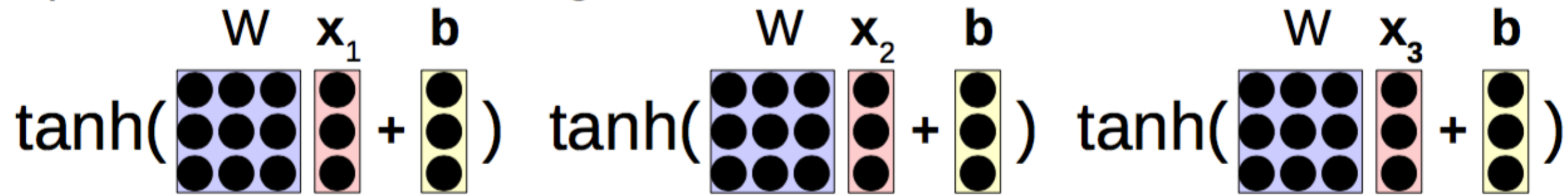
Efficiency Tricks: Operation Batching

Efficiency Tricks: Mini-batching

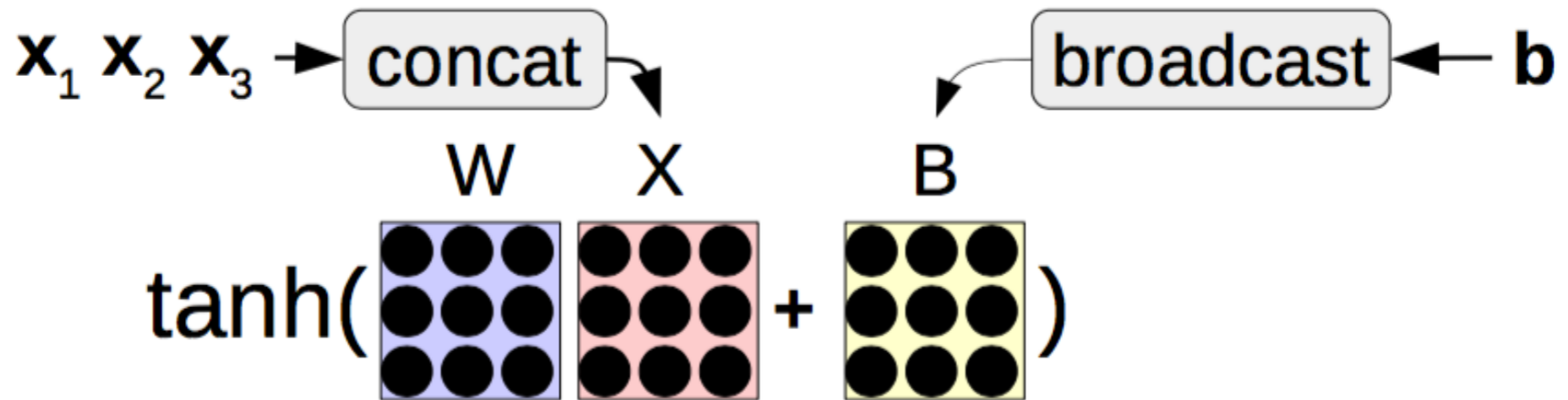
- On modern hardware 10 operations of size 1 is **much slower than** 1 operation of size 10
- Minibatching combines together smaller operations into one big one

Minibatching

Operations w/o Minibatching



Operations with Minibatching



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
- How this works depends on toolkit
 - Most toolkits have require you to add an extra dimension representing the batch size
 - 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.

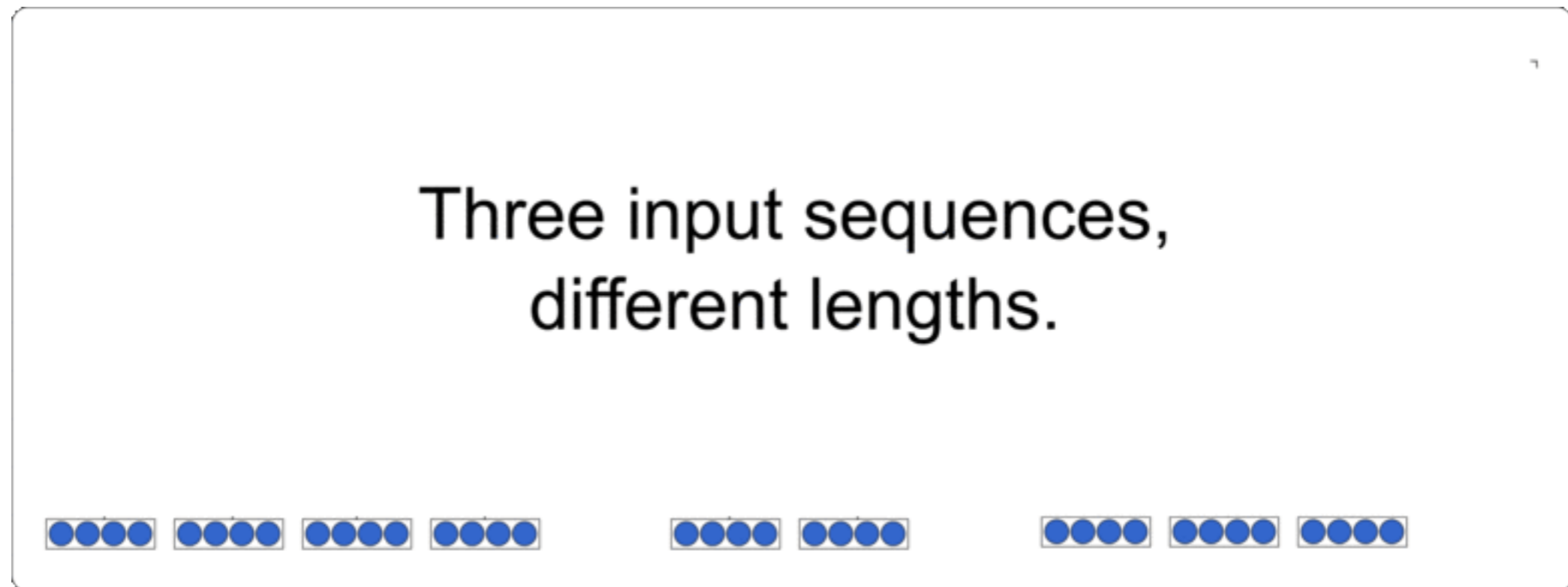
```
1 # in_words is a list [(word_{1,1}, word_{1,2}), (word_{2,1}, word_{2,2}), ...]
2 # out_labels is a list of output labels [label_1, label_2, ...]
3 word_1_batch = dy.lookup_batch(E, [x[0] for x in in_words])
4 word_2_batch = dy.lookup_batch(E, [x[1] for x in in_words])
5 scores_sym = W*dy.concatenate([word_1_batch,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

Automatic Mini-batching!



- TensorFlow Fold, DyNet Autobatching (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

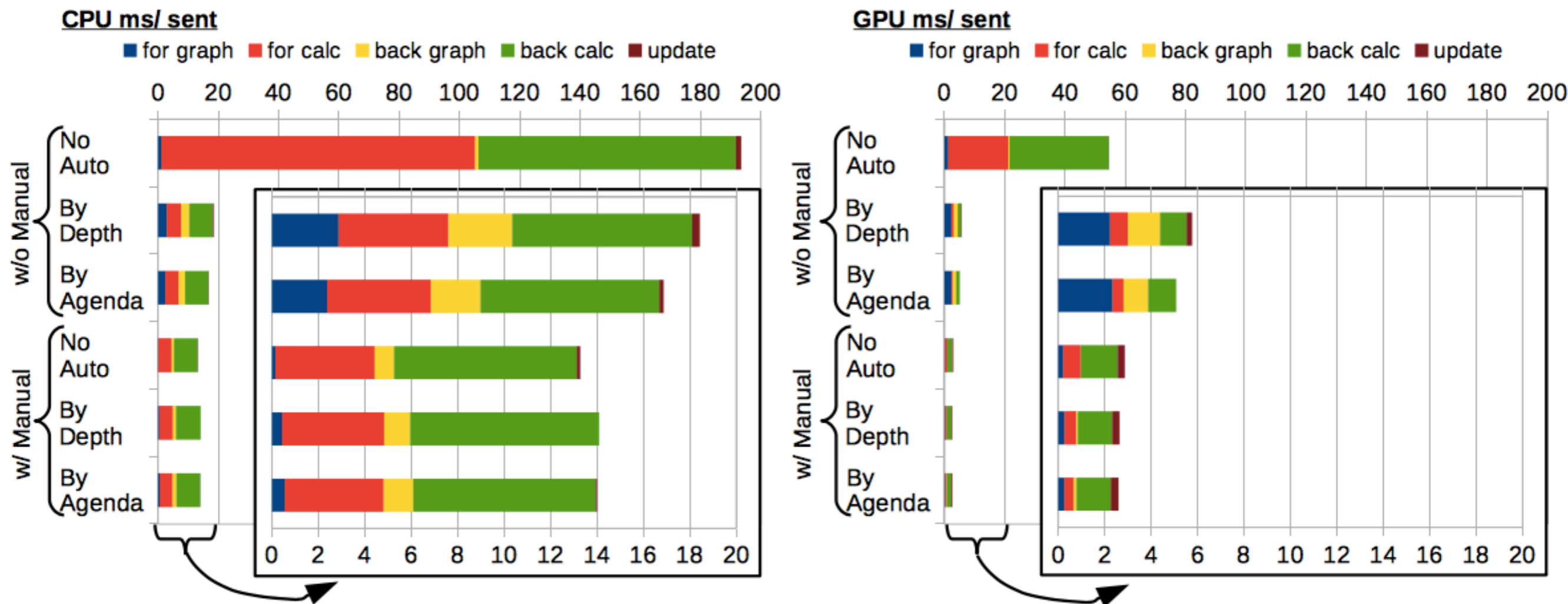


Table 1: Sentences/second on various training tasks for increasingly challenging batching scenarios.

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

Code-level Optimization

- e.g. TorchScript provides a restricted representation of a PyTorch module that can be run efficiently in C++

```
class MyCell(torch.nn.Module):
    def __init__(self):
        super(MyCell, self).__init__()
        self.linear = torch.nn.Linear(4, 4)

    def forward(self, x, h):
        new_h = torch.tanh(self.linear(x) + h)
        return new_h, new_h
```

```
my_cell = MyCell()
x, h = torch.rand(3, 4), torch.rand(3, 4)
traced_cell = torch.jit.trace(my_cell, (x, h))
print(traced_cell)
traced_cell(x, h)
```

```
import __torch__
import __torch__.torch.nn.modules.linear
def forward(self,
            input: Tensor,
            h: Tensor) -> Tuple[Tensor, Tensor]:
    _0 = self.linear
    weight = _0.weight
    bias = _0.bias
    _1 = torch.addmm(bias, input, torch.t(weight), beta=1, alpha=1)
    _2 = torch.tanh(torch.add(_1, h, alpha=1))
    return (_2, _2)
```

A Case Study:
Regularizing and Optimizing LSTM
Language Models (Merity et al. 2017)

Regularizing and Optimizing LSTM Language Models (Merity et al. 2017)

- Uses LSTMs as a backbone (discussed later)
- A number of tricks to improve stability and prevent overfitting:
 - DropConnect regularization
 - SGD w/ averaging triggered when model is close to convergence
 - Dropout on recurrent connections and embeddings
 - Weight tying
 - Independently tuned embedding and hidden layer sizes
 - Regularization of activations of the network
- Strong baseline for language modeling, SOTA at the time (without special model, just training methods)

In-class Discussion

- If you have experience with neural networks, what is *your* standard set of tricks that you use in most projects?
- Do you find that these tricks vary from genre to genre, some are more useful in vision and some are more useful in NLP?

Questions?