Simple and Efficient Learning with Automatic Operation Batching

Graham Neubig

Carnegie Mellon University
Language Technologies Institute

joint work w/ Yoav Goldberg and Chris Dyer

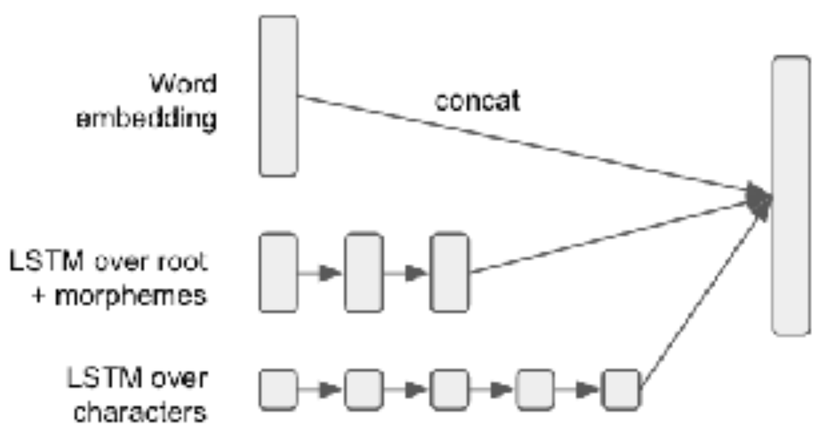
in dynet

http://dynet.io/autobatch/

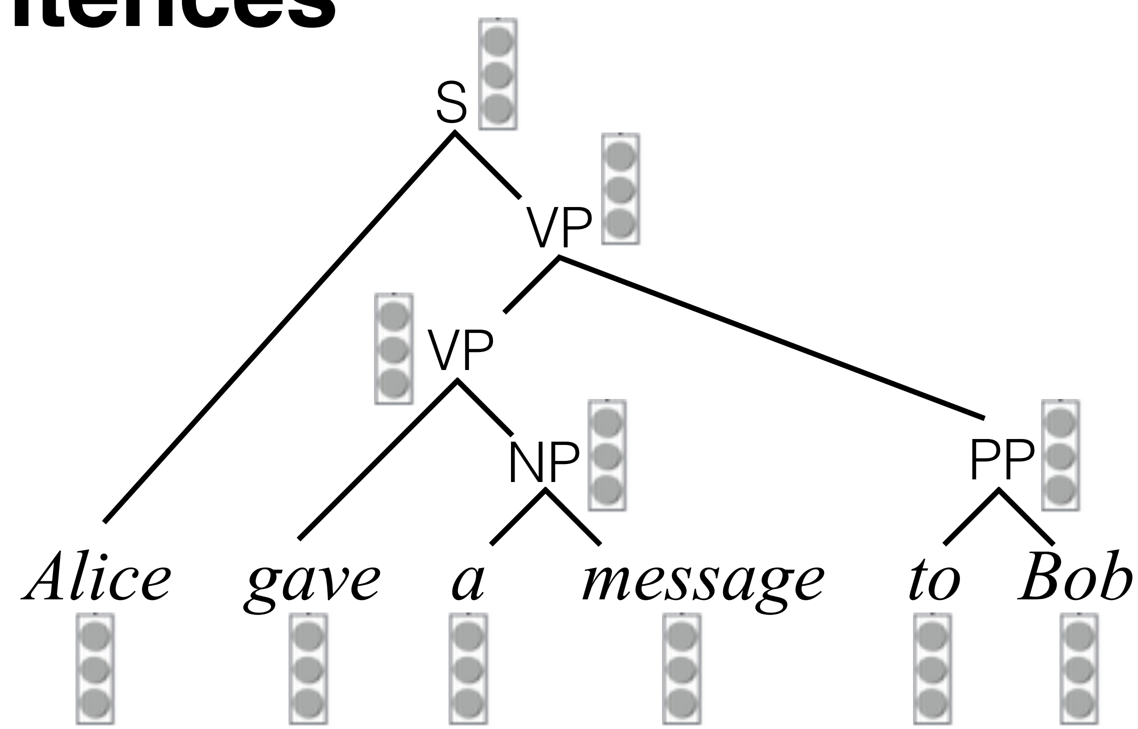
https://github.com/neubig/howtocode-2017

Neural Networks w/ Complicated Structures

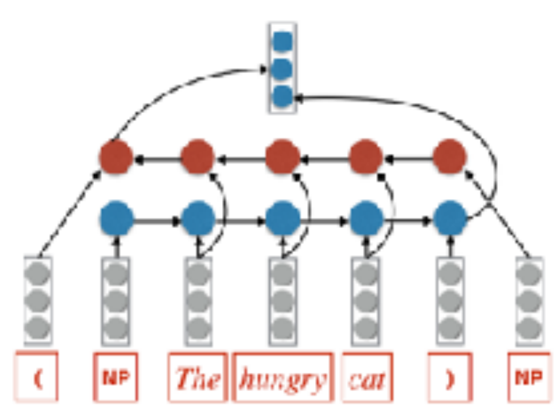
Words



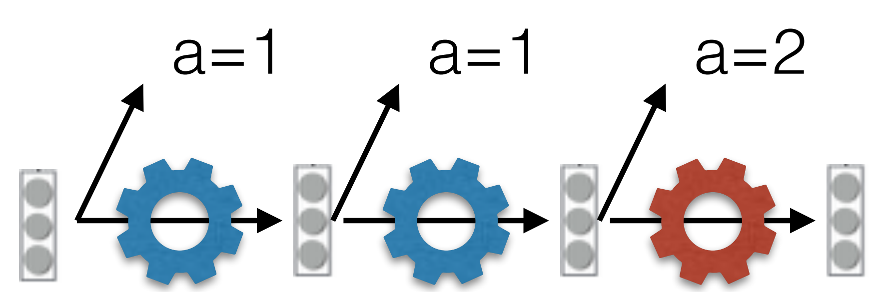
Sentences



Phrases



Dynamic Decisions



Neural Net Programming Paradigms

What is Necessary for Neural Network Training

- **define** computation
- **add** data
- calculate result (**forward**)
- calculate gradients (**backward**)
- **update** parameters

Paradigm 1: Static Graphs (Tensorflow, Theano)

- **define**
- for each data point:
    - **add data**
    - **forward**
    - **backward**
    - **update**

Advantages/Disadvantages of Static Graphs

- **Advantages:**
  - Can be optimized at definition time
  - Easy to feed data to GPUs, etc., via data iterators
- **Disadvantages:**
  - Difficult to implement nets with varying structure (trees, graphs, flow control)
  - Need to learn big API that implements flow control in the “graph” language

Paradigm 2: Dynamic+Eager Evaluation (PyTorch, Chainer)

- for each data point:
  - **define/add data/forward**
  - **backward**
  - **update**

Advantages/Disadvantages of Dynamic+Eager Evaluation

- **Advantages:**
  - Easy to implement nets with varying structure, API is closer to standard Python/C++
  - Easy to debug because errors occur immediately
- **Disadvantages:**
  - Cannot be optimized at definition time
  - Hard to serialize graphs w/o program logic, decide device placement, etc.

Paradigm 3: Dynamic+Lazy Evaluation (DyNet)

- for each data point:
  - **define/add data**
  - **forward**
  - **backward**
  - **update**

Advantages/Disadvantages of Dynamic+Lazy Evaluation

- **Advantages:**
  - Easy to implement nets with varying structure, API is closer to standard Python/C++
  - Can be optimized at definition time (this presentation!)
- **Disadvantages:**
  - Harder to debug because errors occur immediately
  - Still hard to serialize graphs w/o program logic, decide device placement, etc.

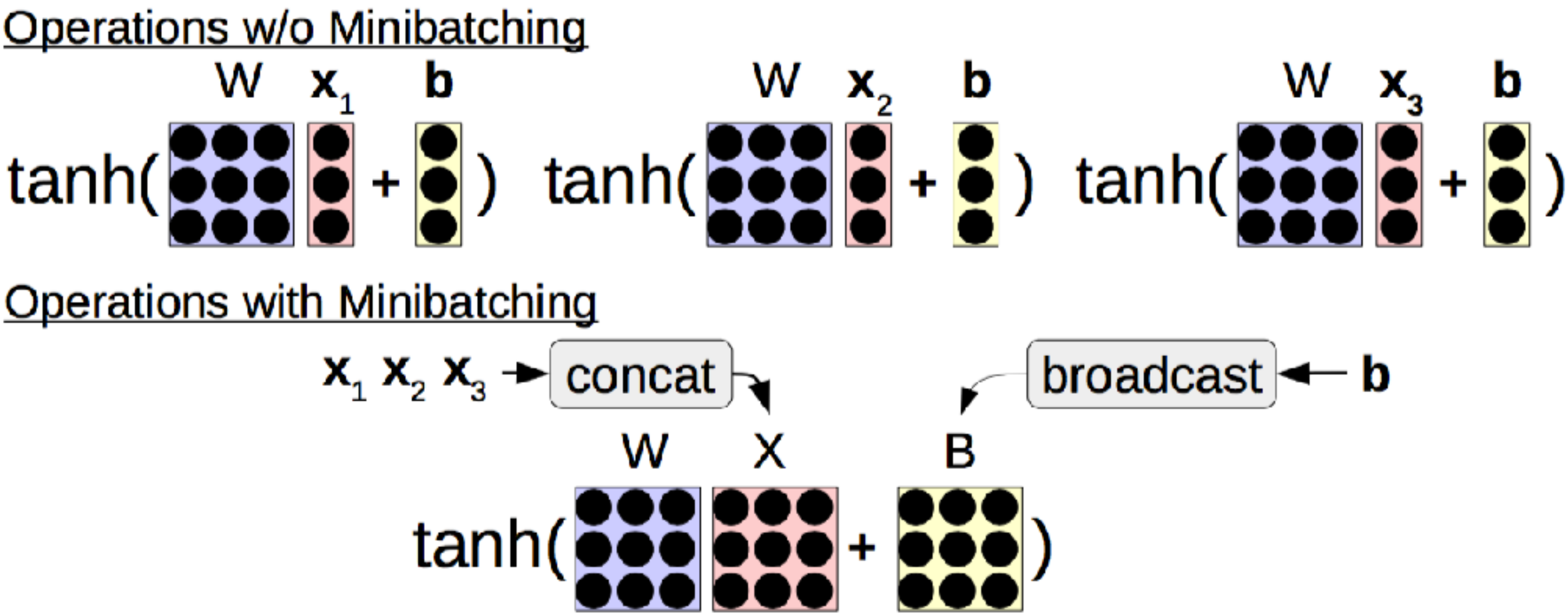
Efficiency Tricks: Operation Batching

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

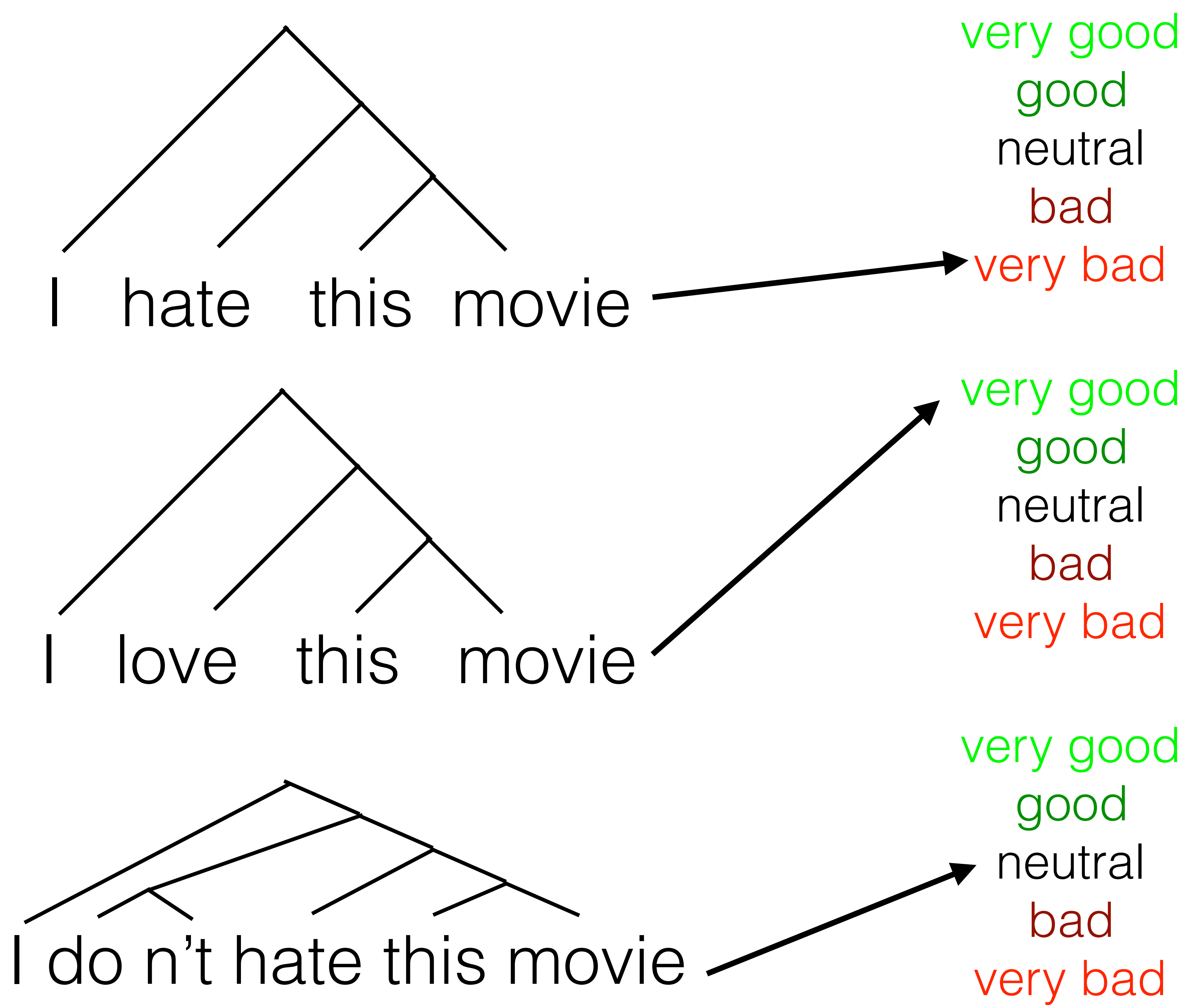
Minibatching



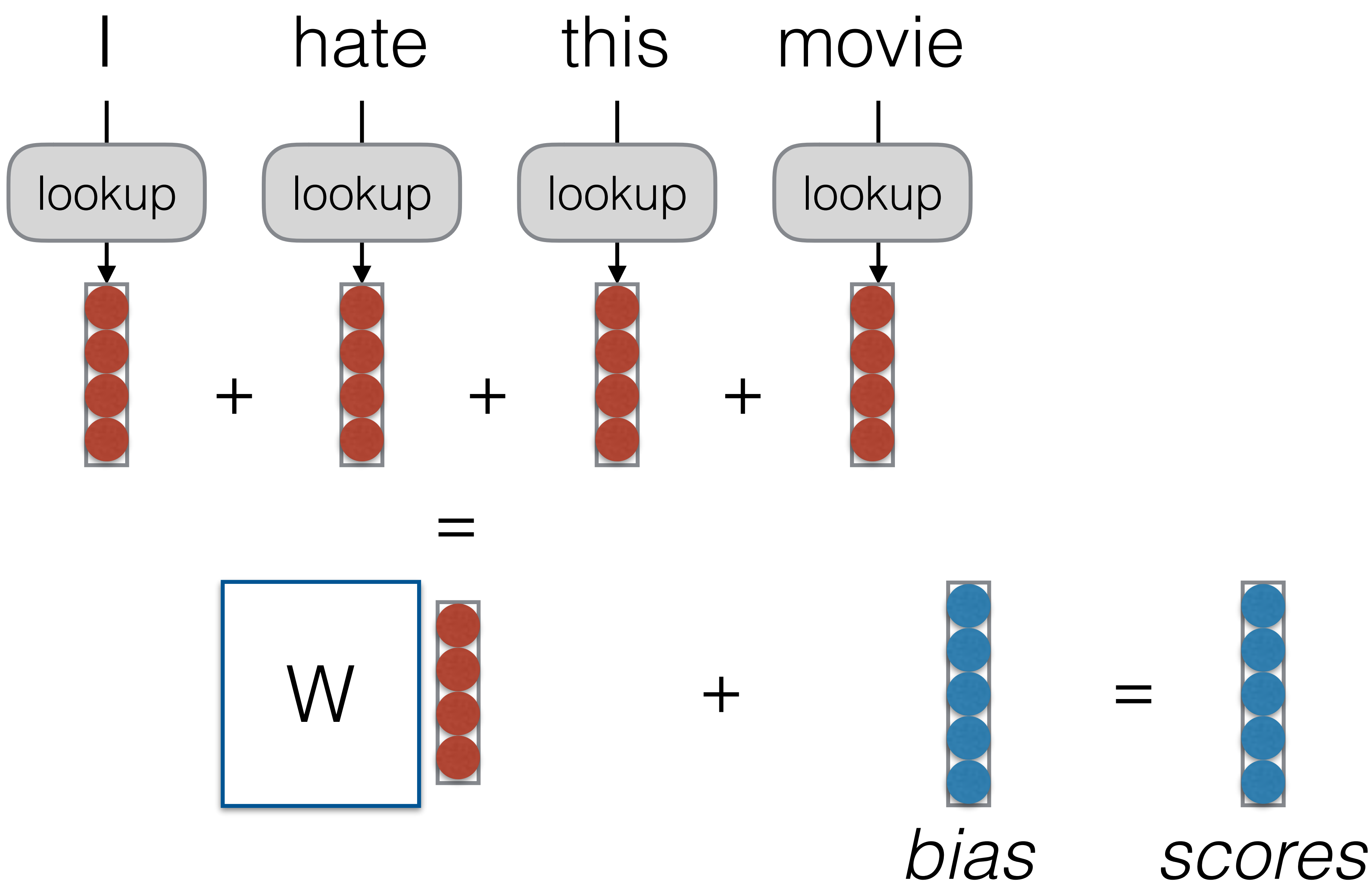
Manual Mini-batching

- DyNet has special minibatch operations for lookup and loss functions, everything else automatic
- You need to:
  - Group sentences into a mini batch (optionally, for efficiency group sentences by length)
  - Select the “t”th word in each sentence, and send them to the lookup and loss functions

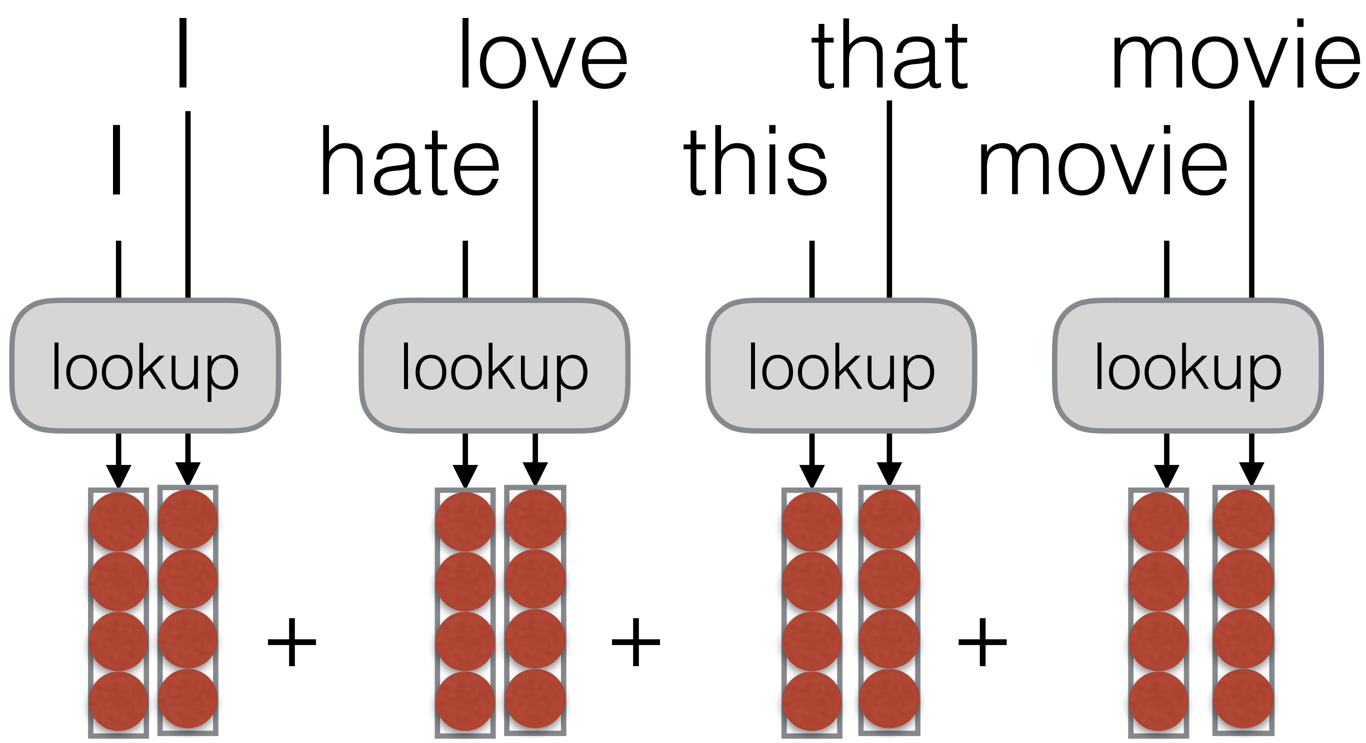
Example Task: Sentiment



Continuous Bag of Words (CBOW)



Batching CBOW



Mini-batched Code Example

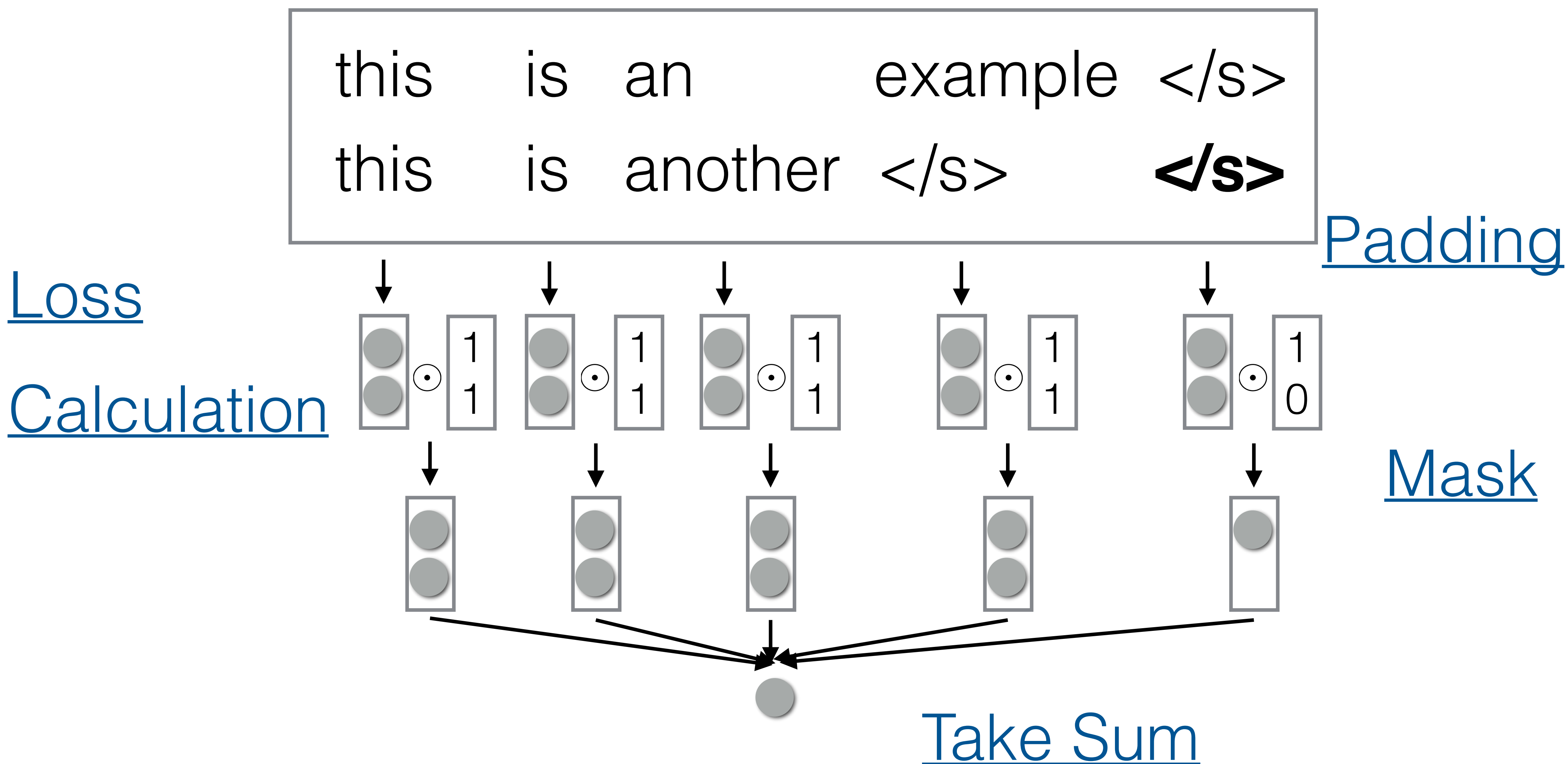
```
# in_words is a tuple (word_1, word_2)
# out_label is an output label
word_1 = E[in_words[0]]
word_2 = E[in_words[1]]
scores_sym = W*dy.concatenate([word_1, word_2])+b
loss_sym = dy.pickneglogsoftmax(scores_sym, out_label)
```

(a) Non-minibatched classification.

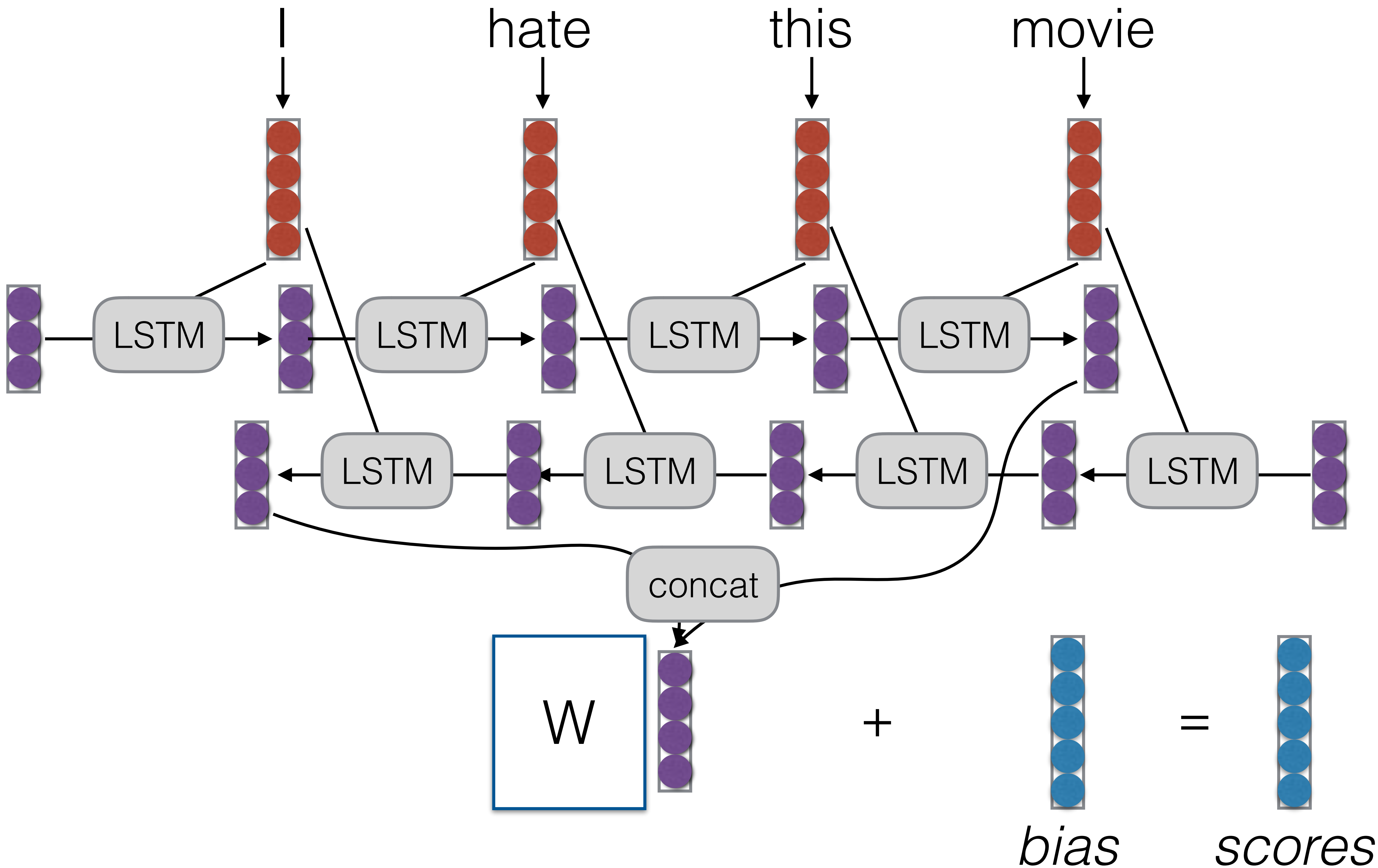
```
# in_words is a list [(word_{1,1}, word_{1,2}), (word_{2,1}, word_{2,2}), ...]
# out_labels is a list of output labels [label_1, label_2, ...]
word_1_batch = dy.lookup_batch(E, [x[0] for x in in_words])
word_2_batch = dy.lookup_batch(E, [x[1] for x in in_words])
scores_sym = W*dy.concatenate([word_1_batch,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

(b) Minibatched classification.

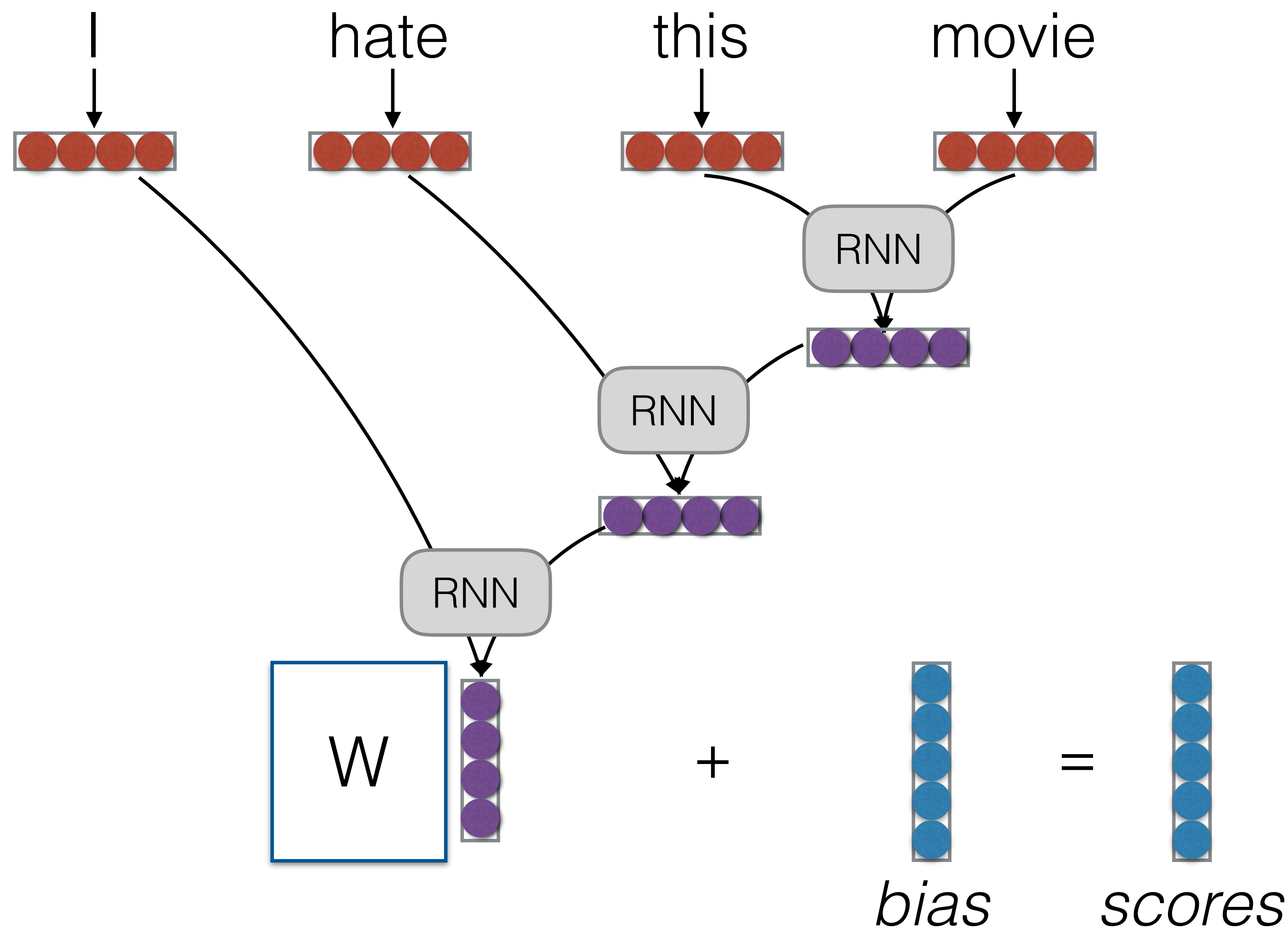
Mini-batching Sequences



Bi-directional LSTM

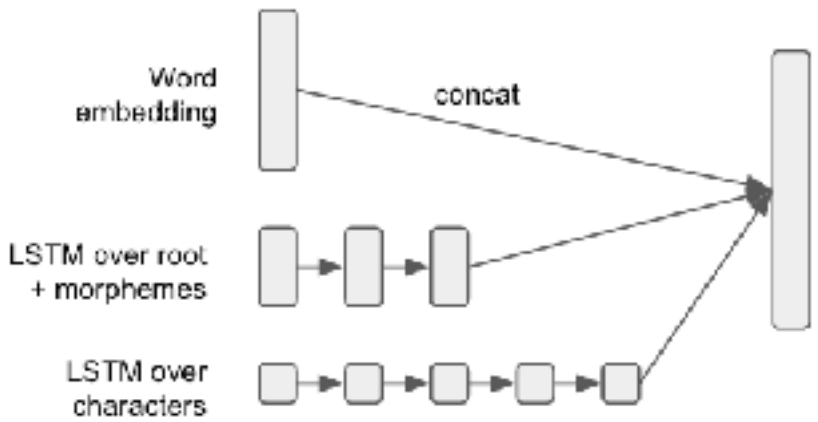


Tree-structured RNN/LSTM

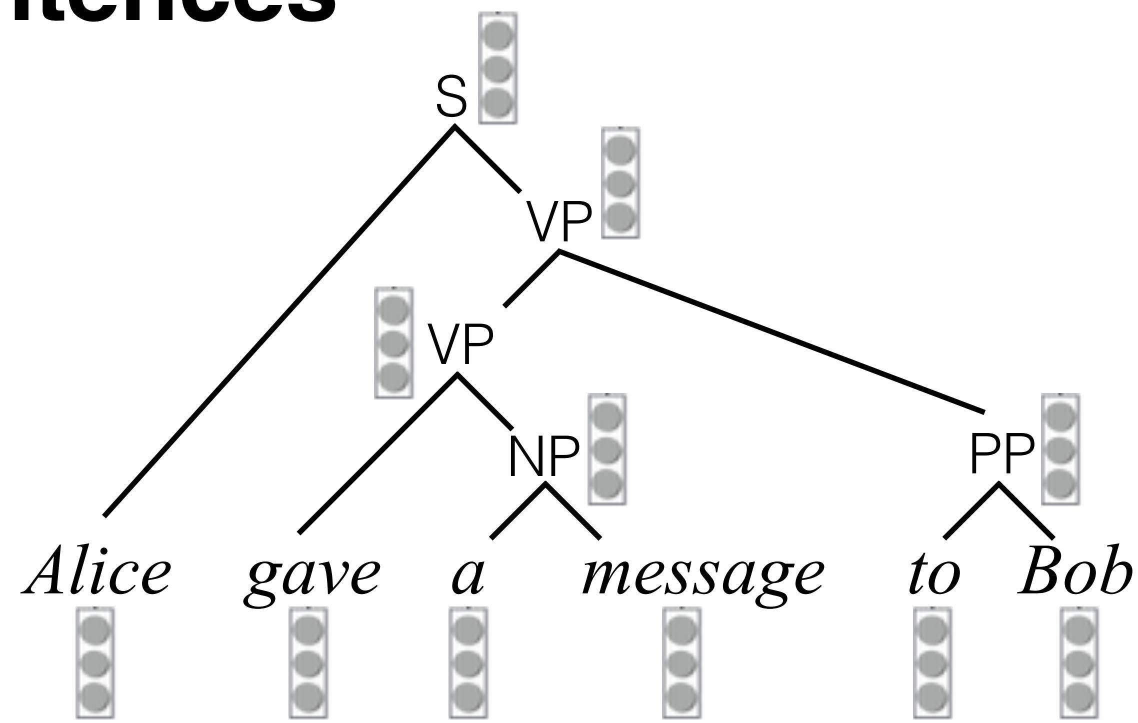


And What About These?

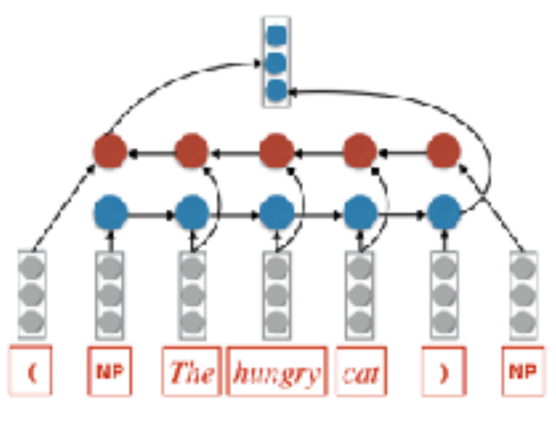
Words



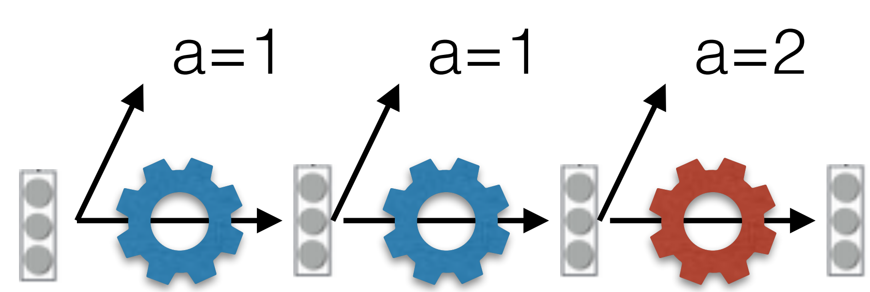
Sentences



Phrases

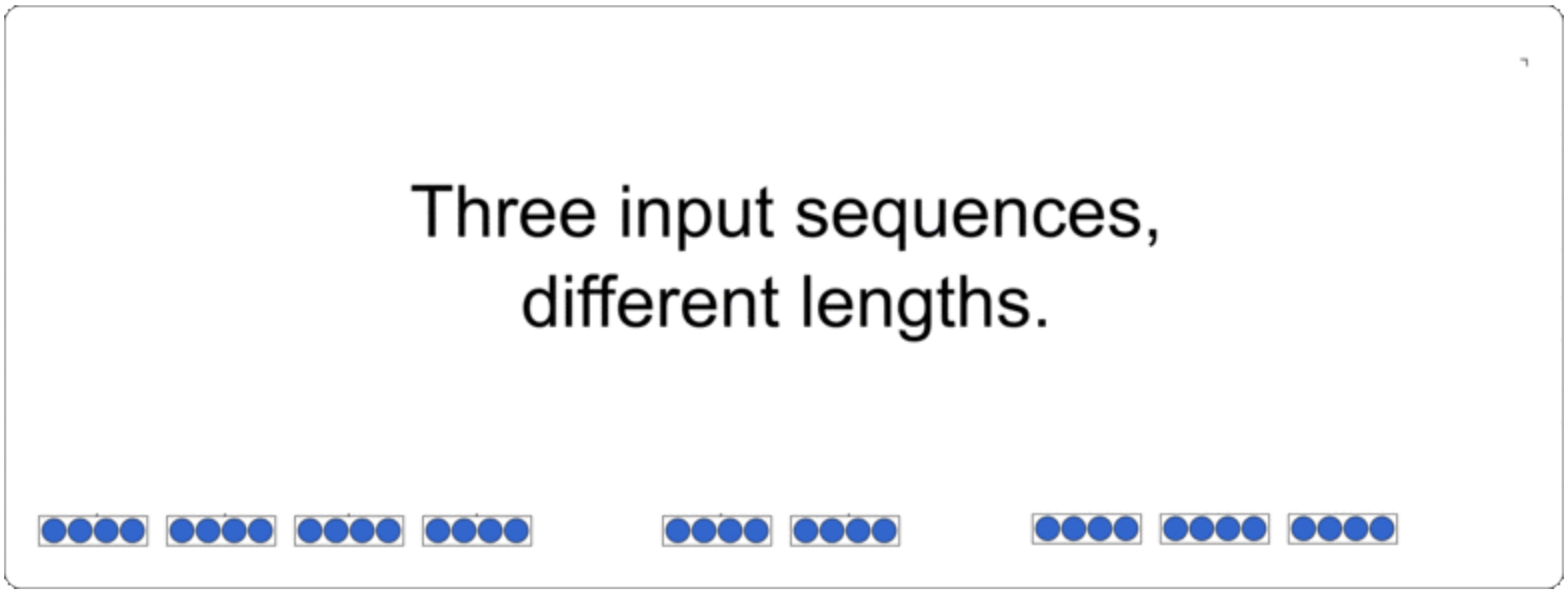


Dynamic Decisions



Automatic Operation Batching

Automatic Mini-batching!



- Innovatd by TensorFlow Fold (faster than unbatched, but implementation relatively complicated)
- DyNet Autobatch (basically effortless implementation)

Programming Paradigm

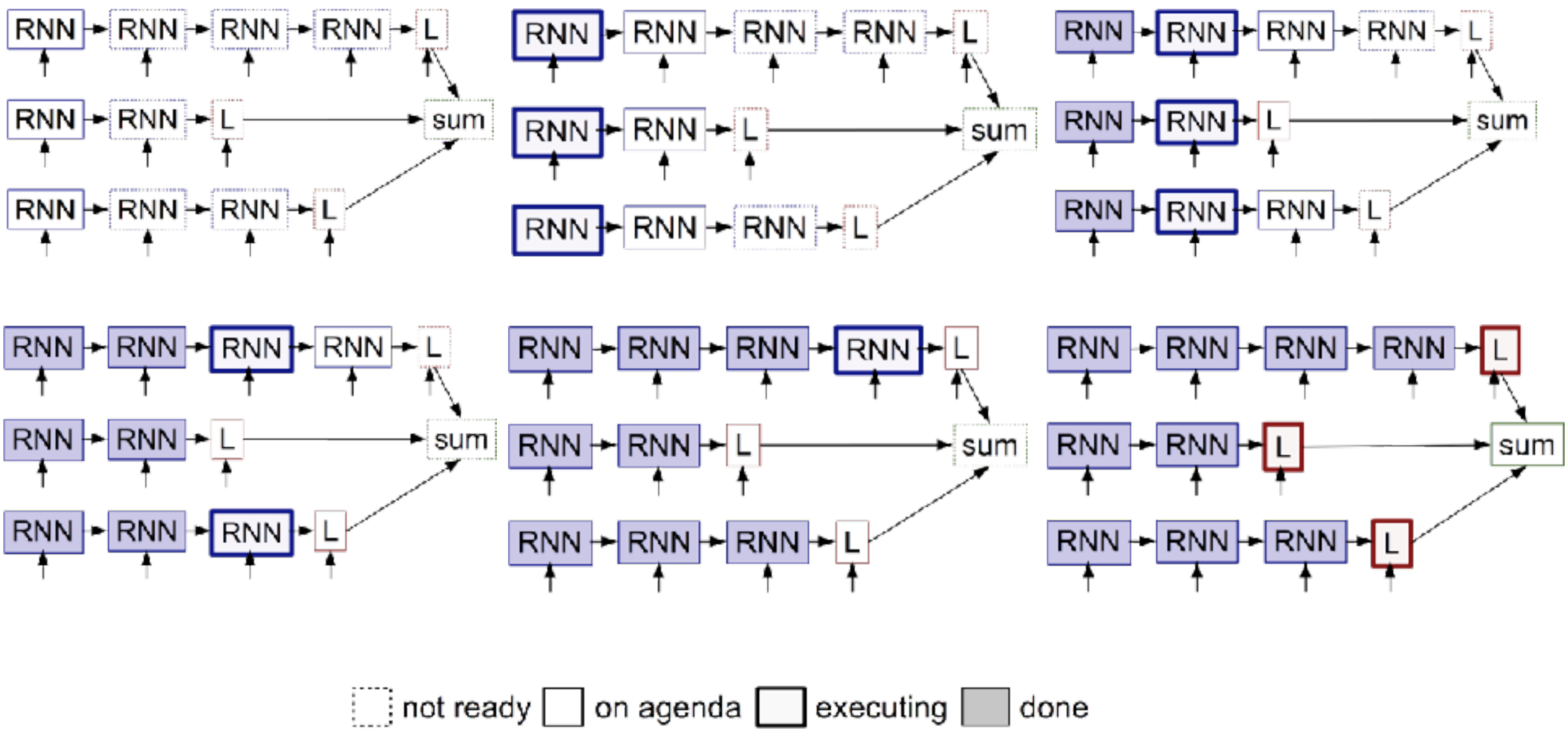
Just write a for loop!

```
for minibatch in training_data:
  loss_values = []
  for x, y in minibatch:
    loss_values.append(calculate_loss(x,y))
  loss_sum = sum(loss_values)
  loss_sum.forward()
  loss_sum.backward()
  trainer.update()
```

Batching occurs here

Under the Hood

- Each node has “profile”, same profile → batchable
- Batch and execute items with their dependencies satisfied

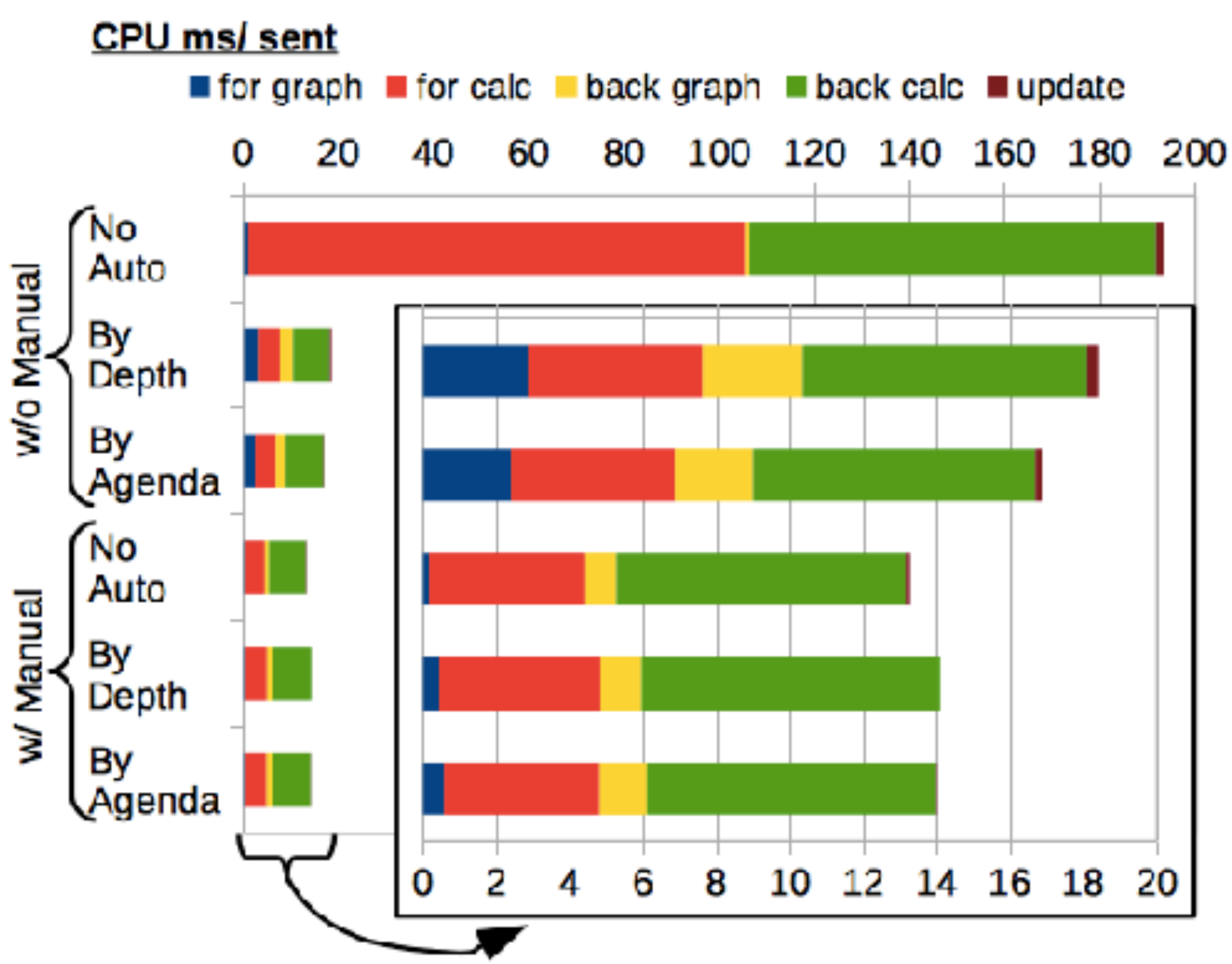


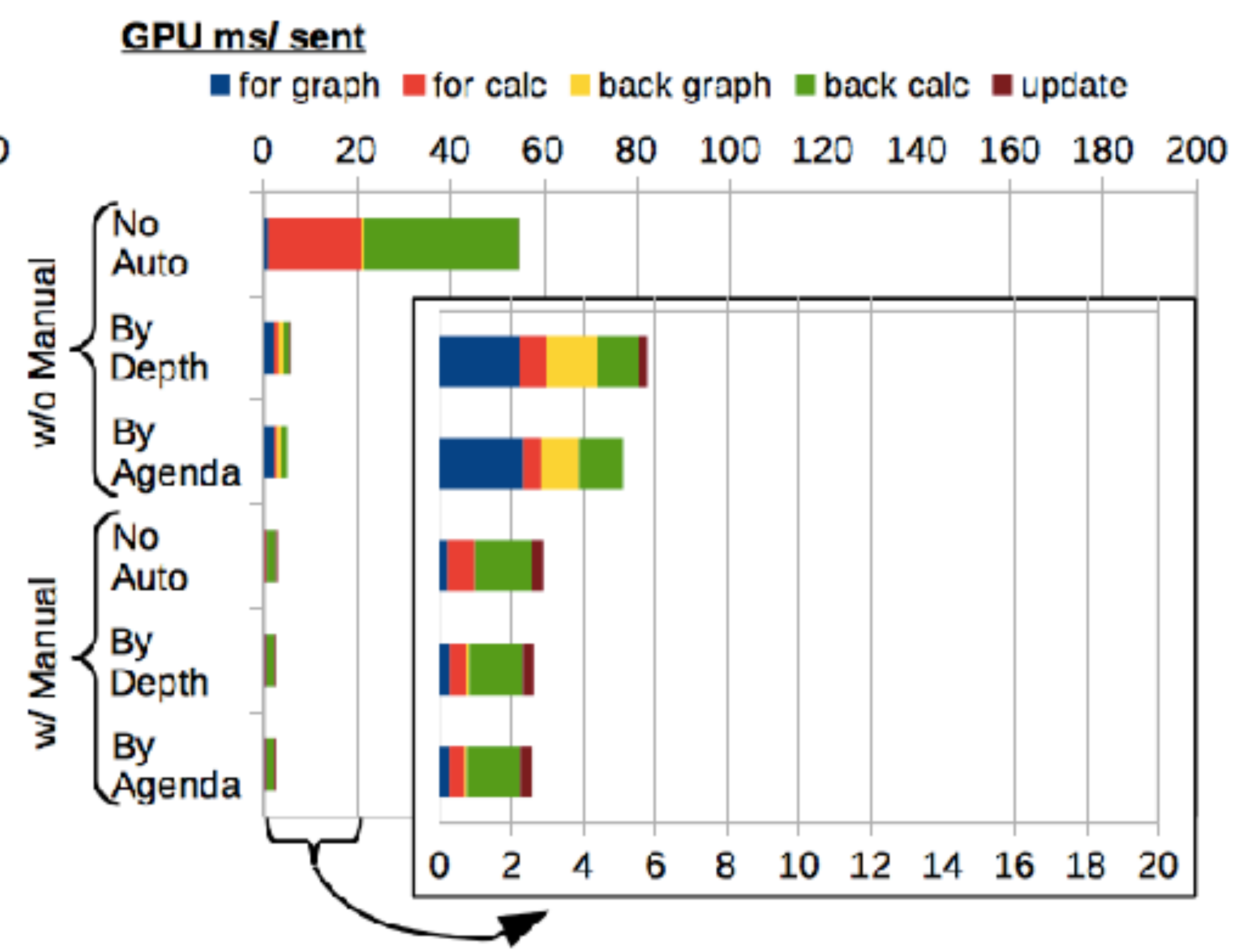
Challenges

- This goes in your training loop:
  must be blazing fast!
- DyNet’s C++ implementation is highly optimized
  - Profiles stored as hash functions
  - Minimize memory allocation overhead

Synthetic Experiments

- Fixed-length RNN → ideal case for manual batching
- How close can we get?





Real NLP Tasks

- Variably Lengthed RNN, RNN w/ character embeddings, tree LSTM, dependency parser

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**

Let's Try it Out!

http://dynet.io/autobatch/

https://github.com/neubig/howtocode-2017