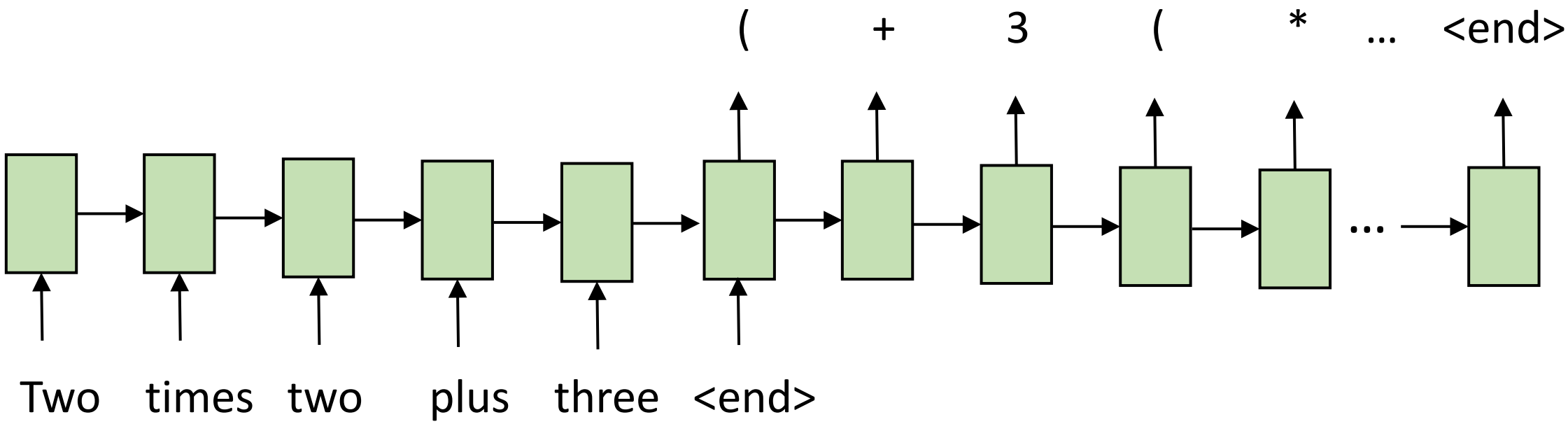
Neural Semantic Parsing

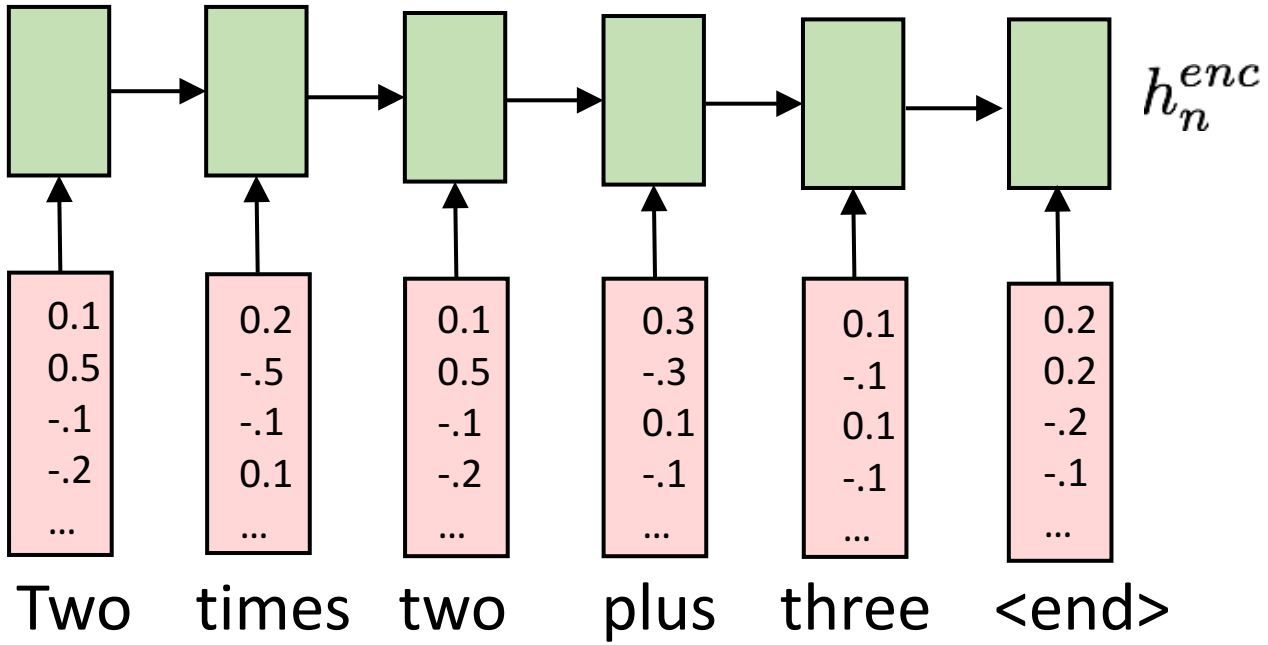
Bishan Yang and Igor Labutov

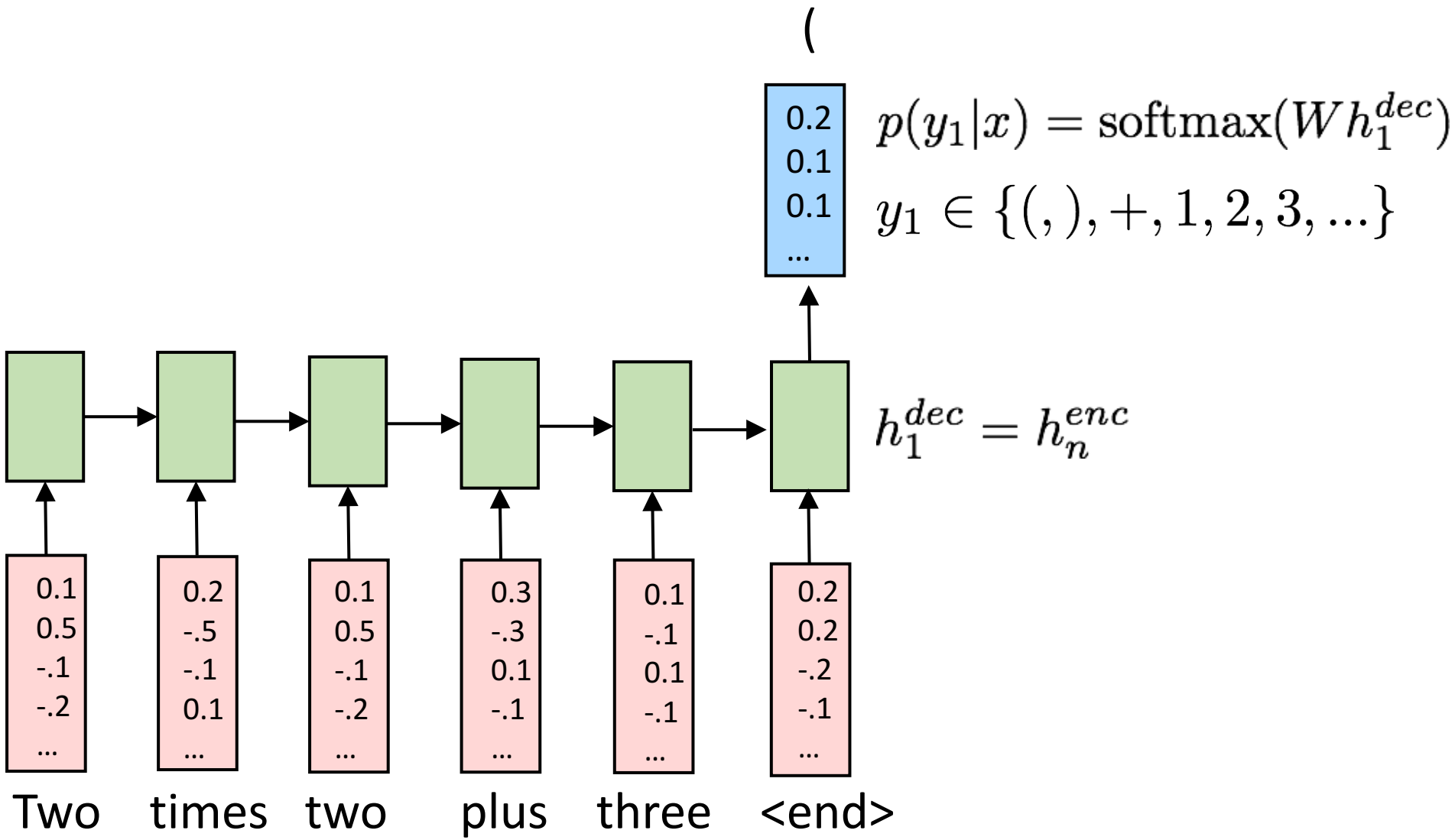
Recap

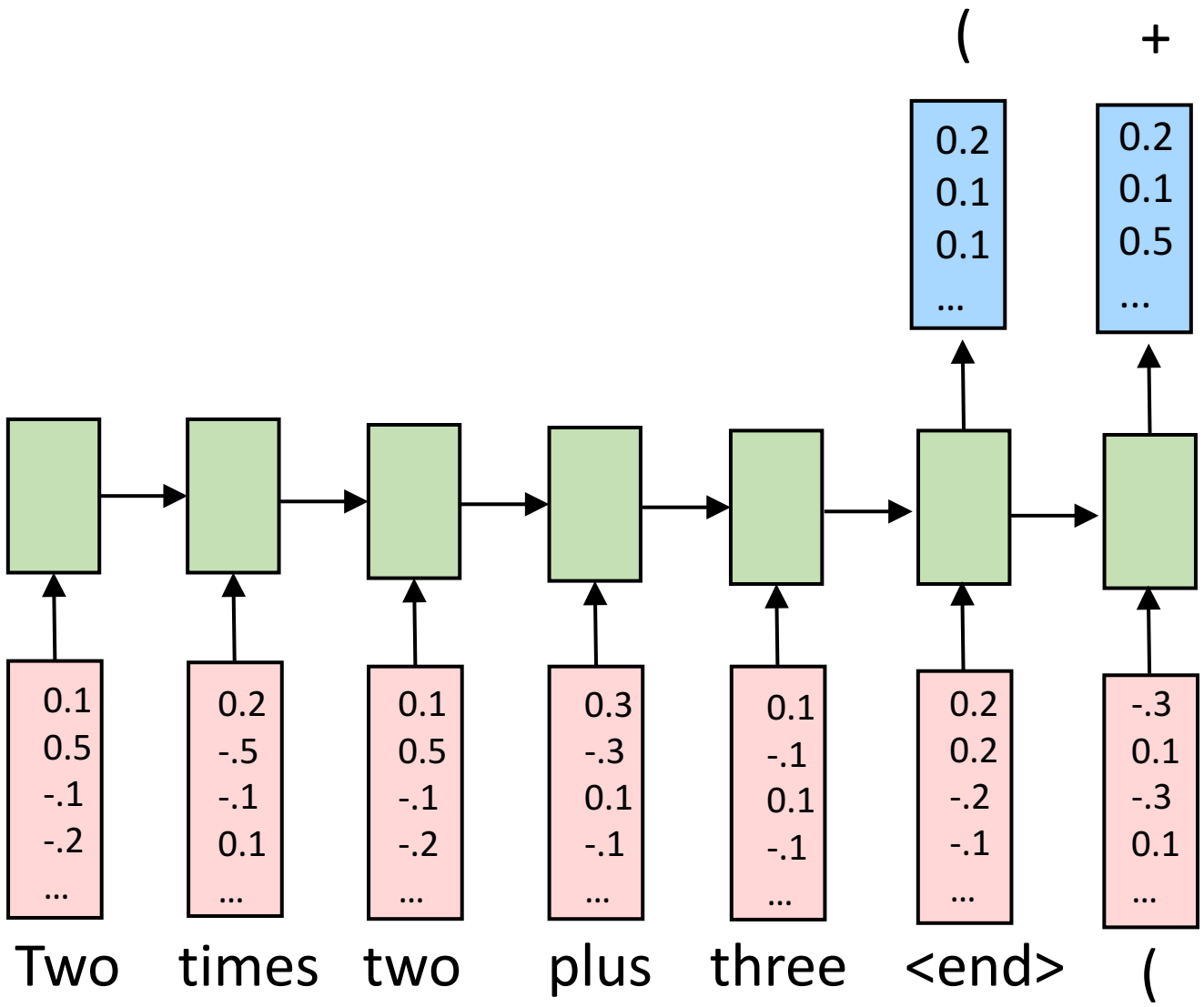
- What is Deep Learning?
- What is Neural Networks?
- What is Recurrent Neural Networks?
- Different formulations of RNNs
  - Sequence-to-sequence (Many-to-Many) is the most expressive
- Applications of RNNs: machine translation, summarization, image captioning, …

Semantic Parsing: Sequence to Sequence



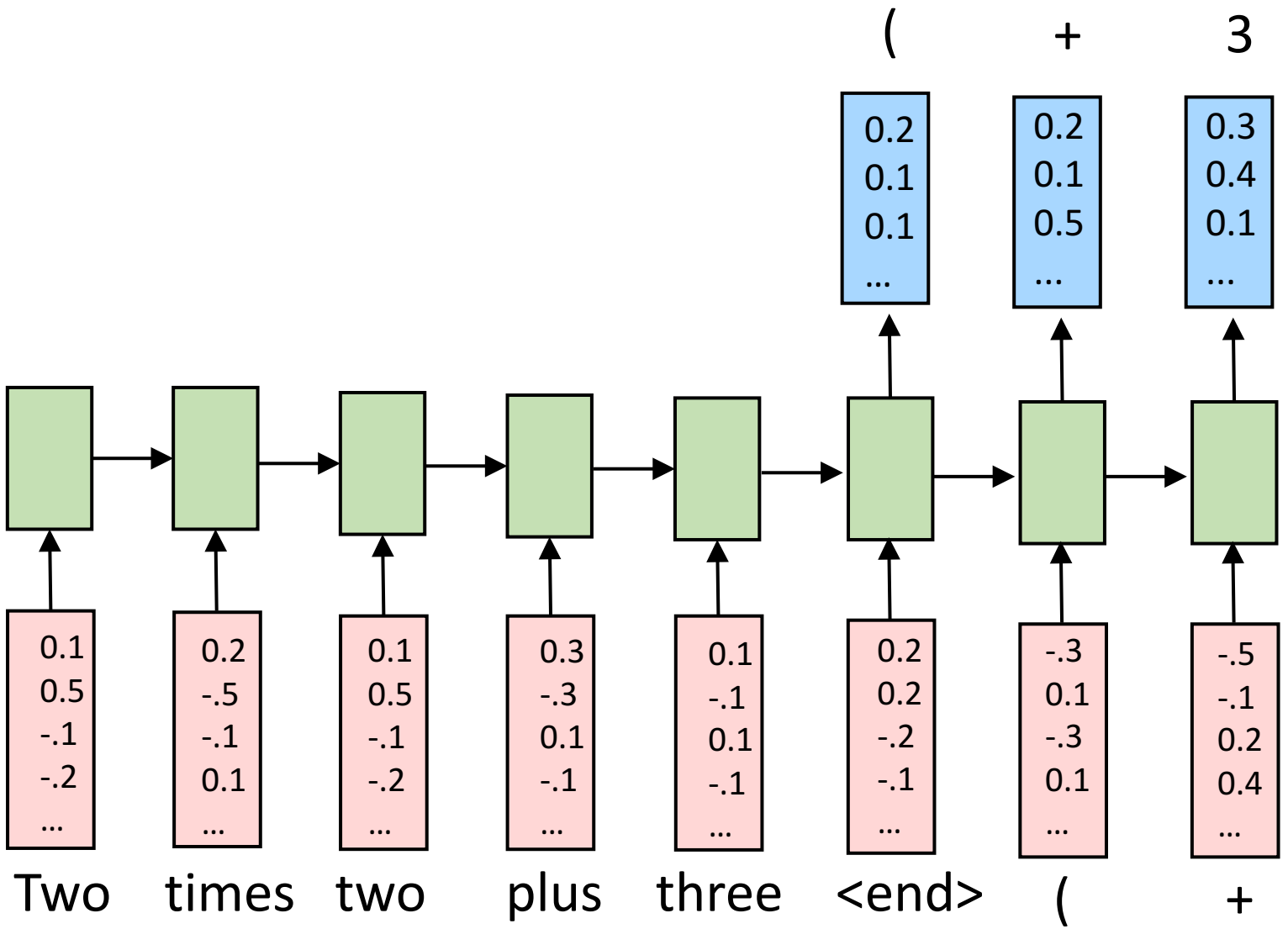

h_n^{enc}
0.1
0.5
-.1
-.2
...
0.2
-.5
-.1
0.1
...
0.1
0.5
-.1
-.2
...
0.3
-.3
0.1
-.1
...
0.1
-.1
0.1
-.1
...
0.2
0.2
-.2
-.1
...
Two
times
two
plus
three
<end>

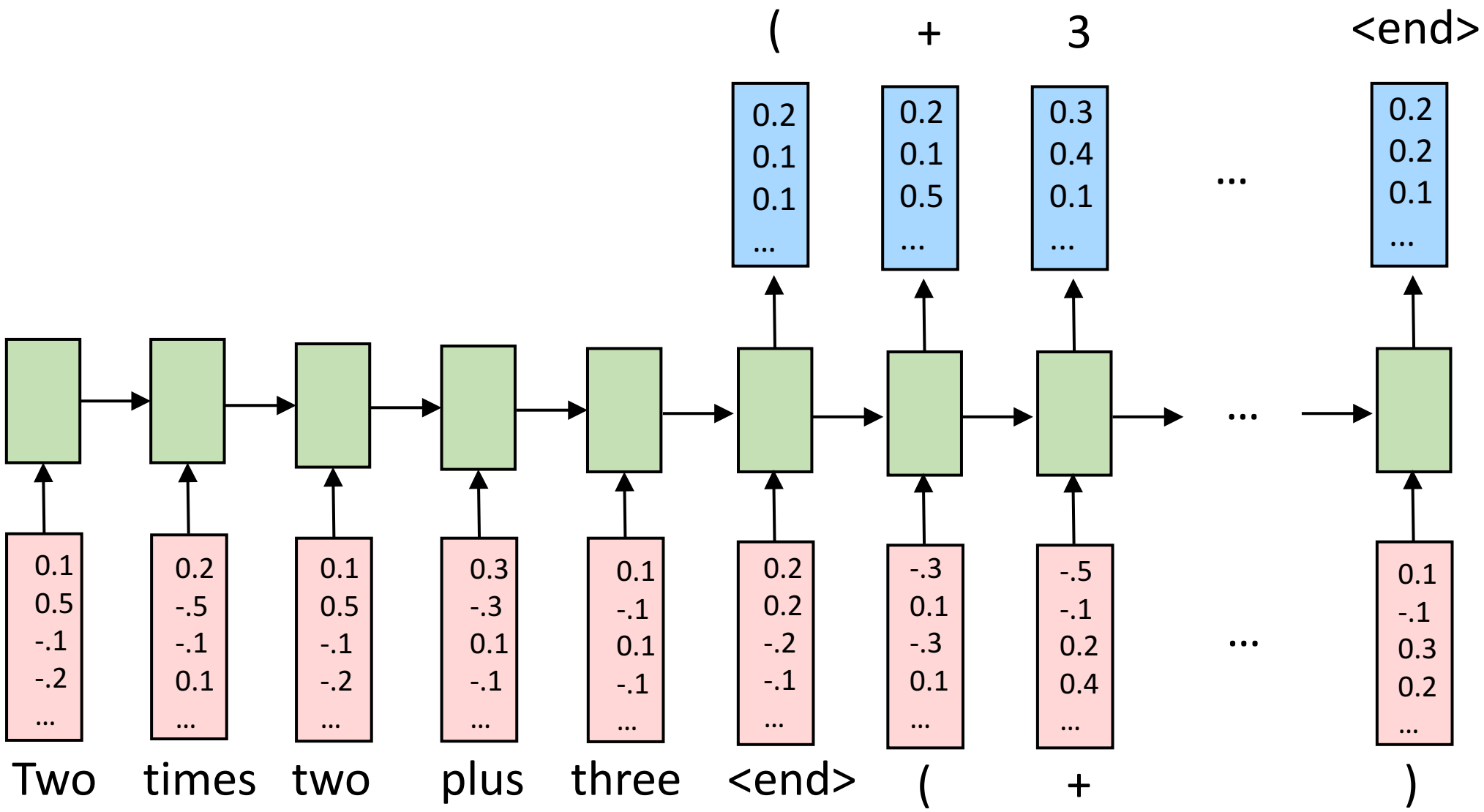

(
0.2
0.1
0.1
...
$p(y_1|x) = \text{softmax}(W h_1^{dec})$
$y_1 \in \{(,), +, 1, 2, 3, ...\}$
$h_1^{dec} = h_n^{enc}$
0.1
0.5
-.1
-.2
...
0.2
-.5
-.1
0.1
...
0.1
0.5
-.1
-.2
...
0.3
-.3
0.1
-.1
...
0.1
-.1
0.1
-.1
...
0.2
0.2
-.2
-.1
...
Two
times
two
plus
three
<end>



$$p(y_2|y_1, x) = \text{softmax}(W h_2^{dec})$$

$$y_2 \in \{(,), +, 1, 2, 3, ...\}$$


(
+
3
0.2
0.1
0.1
...
0.2
0.1
0.5
...
0.3
0.4
0.1
...
0.1
0.5
-.1
-.2
...
0.2
-.5
-.1
0.1
...
0.1
0.5
-.1
-.2
...
0.3
-.3
0.1
-.1
...
0.1
-.1
0.1
-.1
...
0.2
0.2
-.2
-.1
...
-.3
0.1
-.3
0.1
...
-.5
-.1
0.2
0.4
...
Two
times
two
plus
three
<end>
(
+


(
+
3
<end>
0.2 0.1 0.1 ...
0.2 0.1 0.5 ...
0.3 0.4 0.1 ...
...
0.2 0.2 0.1 ...
...
...
0.1 0.5 -.1 -.2 ...
0.2 -.5 -.1 0.1 ...
0.1 0.5 -.1 -.2 ...
0.3 -.3 0.1 -.1 ...
0.1 -.1 0.1 -.1 ...
0.2 0.2 -.2 -.1 ...
-.3 0.1 -.3 0.1 ...
-.5 -.1 0.2 0.4 ...
0.1 -.1 0.3 0.2 ...
Two
times
two
plus
three
<end>
(
+
)

Optimization

$$Loss(x, y) = -\log p(y_1, ..., y_T | x)$$

$$= -\sum_{t=1}^{T} \log p(y_t | y_{1:t-1}, x)$$

Optimization

$$Loss(x, y) = -\log p(y_1, ..., y_T|x)$$

$$= -\sum_{t=1}^{T} \log p(y_t|y_{1:t-1}, x)$$

$$p(y_t|y_{1:t-1}, x) = \text{softmax}(Wh_t)$$

Optimization

$$Loss(x, y) = -\log p(y_1, ..., y_T|x)$$

$$= -\sum_{t=1}^{T} \log p(y_t|y_{1:t-1}, x)$$

$$p(y_t|y_{1:t-1}, x) = \text{softmax}(Wh_t)$$

$$h_t = f(h_{t-1}, y_{t-1})$$

$$= \tanh(Vh_{t-1} + Ay_{t-1})$$

Computing the gradient

$$\frac{\partial Loss(x, y)}{\partial V}$$

Computing the gradient

$$\frac{\partial Loss(x, y)}{\partial V} \longrightarrow \frac{\partial p(y_t | y_{1:t-1}, x)}{\partial V}$$

Computing the gradient

$$\frac{\partial Loss(x,y)}{\partial V} \longrightarrow \frac{\partial p(y_t|y_{1:t-1},x)}{\partial V} \longrightarrow \frac{\partial h_t}{\partial V}$$

Computing the gradient

$$\frac{\partial Loss(x,y)}{\partial V} \longrightarrow \frac{\partial p(y_t|y_{1:t-1},x)}{\partial V} \longrightarrow \frac{\partial h_t}{\partial V} \longrightarrow$$

$$\frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial V}$$

Computing the gradient

$$\frac{\partial Loss(x,y)}{\partial V} \longrightarrow \frac{\partial p(y_t|y_{1:t-1},x)}{\partial V} \longrightarrow \frac{\partial h_t}{\partial V} \longrightarrow$$

$$\frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial V} \longrightarrow \prod_{i=2}^{t} \frac{\partial h_i}{\partial h_{i-1}} \frac{\partial h_1}{\partial V}$$

Computing the gradient

$$\frac{\partial Loss(x,y)}{\partial V} \longrightarrow \frac{\partial p(y_t|y_{1:t-1},x)}{\partial V} \longrightarrow \frac{\partial h_t}{\partial V} \longrightarrow$$

$$\frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial V} \longrightarrow \prod_{i=2}^{t} \frac{\partial h_i}{\partial h_{i-1}} \frac{\partial h_1}{\partial V}$$

$$\frac{\partial h_i}{\partial h_{i-1}} = V(1 - \tanh^2(Vh_{i-1} + Ay_{i-1}))$$

Computing the gradient

$$\frac{\partial Loss(x,y)}{\partial V} \longrightarrow \frac{\partial p(y_t|y_{1:t-1},x)}{\partial V} \longrightarrow \frac{\partial h_t}{\partial V} \longrightarrow$$

$$\frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial V} \longrightarrow \prod_{i=2}^{t} \frac{\partial h_i}{\partial h_{i-1}} \frac{\partial h_1}{\partial V}$$

$$\frac{\partial h_i}{\partial h_{i-1}} = V(1 - \tanh^2(Vh_{i-1} + Ay_{i-1}))$$

Gradient computation involves many multiplication of *V*!

Gradient computation problem

$$V^T$$

Largest singular value > 1: **exploding gradient**

Largest singular value < 1: **vanishing gradient**

Gradient computation problem

$$V^T$$

Largest singular value > 1: **exploding gradient**

Solution: gradient clipping. Rescale the gradient if its norm is too big.

Largest singular value < 1: **vanishing gradient**

Gradient computation problem

$$V^T$$

Largest singular value > 1: **exploding gradient**

Solution: gradient clipping. Rescale the gradient if its norm is too big.

Largest singular value < 1: **vanishing gradient**

Solution: LSTM. Long short term memory networks.

Long Short Term Memory (LSTM)

$$(h_t, c_t) = \mathrm{LSTM}(h_{t-1}, c_{t-1}, y_t)$$

Long Short Term Memory (LSTM)

$$(h_t, c_t) = \mathrm{LSTM}(h_{t-1}, c_{t-1}, y_t)$$

allow forgetting previous cell

Cell: $$c_t = i_t \odot \tanh(U_c h_{t-1} + W_c y_t) + f_t \odot c_{t-1}$$

Long Short Term Memory (LSTM)

$$(h_t, c_t) = \mathrm{LSTM}(h_{t-1}, c_{t-1}, y_t)$$

allow forgetting previous cell

Cell: $c_t = i_t \odot \tanh(U_c h_{t-1} + W_c y_t) + f_t \odot c_{t-1}$

Hidden state: $h_t = o_t \odot \tanh(c_t)$

Long Short Term Memory (LSTM)

$$(h_t, c_t) = \mathrm{LSTM}(h_{t-1}, c_{t-1}, y_t)$$

allow forgetting previous cell

Cell: $c_t = i_t \odot \tanh(U_c h_{t-1} + W_c y_t) + f_t \odot c_{t-1}$

Hidden state: $h_t = o_t \odot \tanh(c_t)$

Forget gate: $f_t = \sigma(U_f h_{t-1} + V_f c_{t-1} + W_f y_t)$

Input gate: $i_t = \sigma(U_i h_{t-1} + V_i c_{t-1} + W_i y_t)$

Output gate: $o_t = \sigma(U_o h_{t-1} + V_o c_t + W_o y_t)$

Training

Training Objective:

$$\arg\min_{\theta} \sum_{(x,y)\in\mathcal{D}_{train}} Loss(x, y, \theta)$$

Training

Training Objective:

$$\arg\min_{\theta} \sum_{(x,y)\in\mathcal{D}_{train}} Loss(x, y, \theta)$$

Algorithm: Stochastic Gradient Descent

for $i = 1$ to M **do**
  for $(x, y) \in \mathcal{D}_{train}$ **do**
    $\theta \leftarrow \theta - \eta_i \frac{\partial Loss(x,y,\theta)}{\partial \theta}$
  end for
end for

Training

Training Objective:

$$\arg\min_{\theta} \sum_{(x,y)\in\mathcal{D}_{train}} Loss(x, y, \theta)$$

Algorithm: Stochastic Gradient Descent

for $i = 1$ to M **do**
  for $(x, y) \in \mathcal{D}_{train}$ **do**
    $\theta \leftarrow \theta - \eta_i \frac{\partial Loss(x,y,\theta)}{\partial \theta}$
  end for
end for

Auto differentiate!

Prediction

Inference Objective: $\arg \max_{y_1,\ldots,y_T} p(y_1, \ldots, y_T | x)$

Prediction

Inference Objective: $\arg \max_{y_1,\ldots,y_T} p(y_1, \ldots, y_T|x)$

$$\arg \max_{y_1,\ldots,y_T} \sum_{t=1}^{T} \log p(y_t|y_{1:t-1}, x; \theta)$$

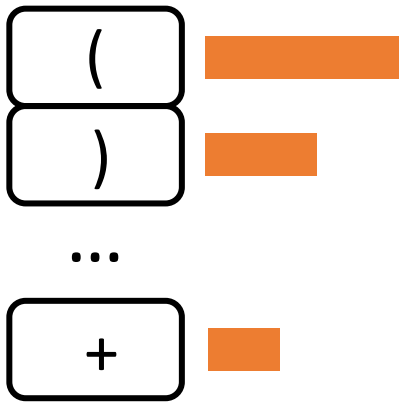
Prediction

Inference Objective: $\arg \max_{y_1,\ldots,y_T} p(y_1, \ldots, y_T|x)$

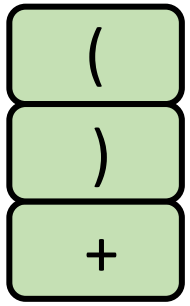
$$\arg \max_{y_1,\ldots,y_T} \sum_{t=1}^{T} \log p(y_t|y_{1:t-1}, x; \theta)$$

Approximate arg max with beam search

Beam Search (K=3)

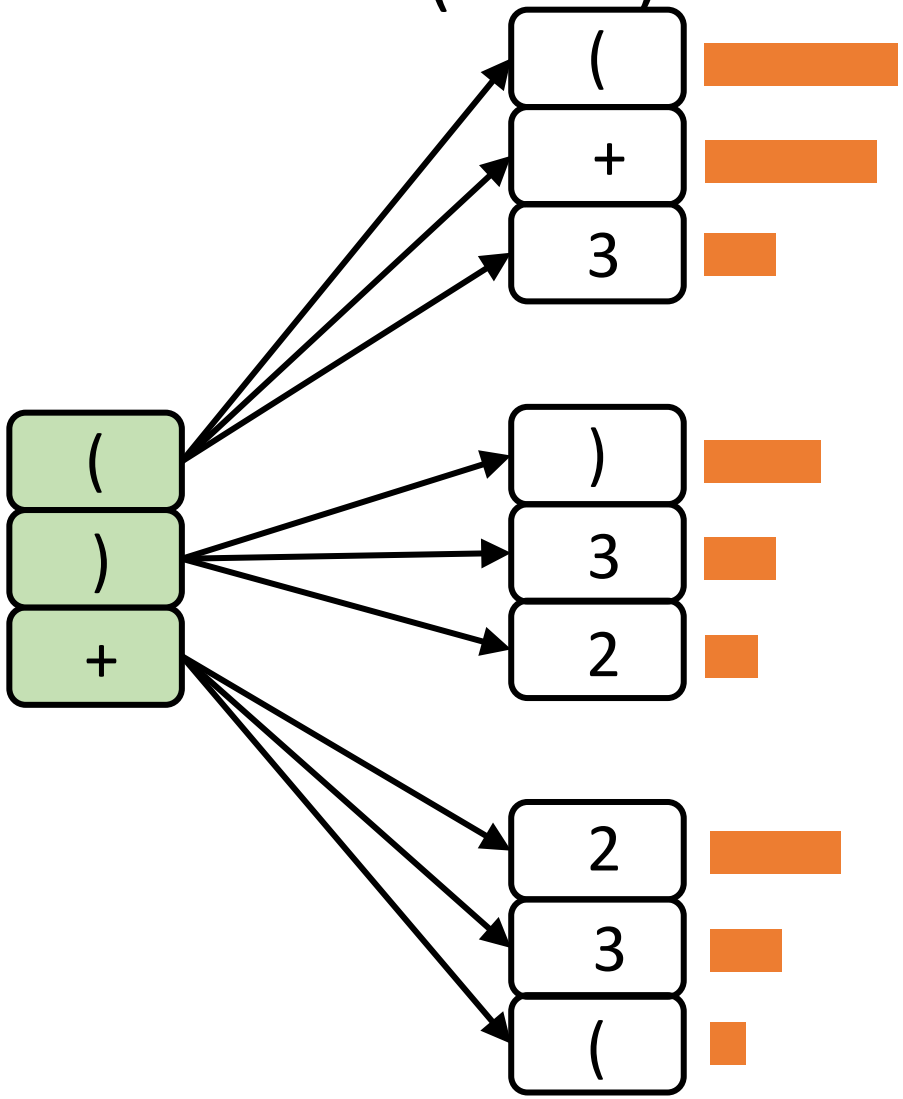


Beam Search (K=3)

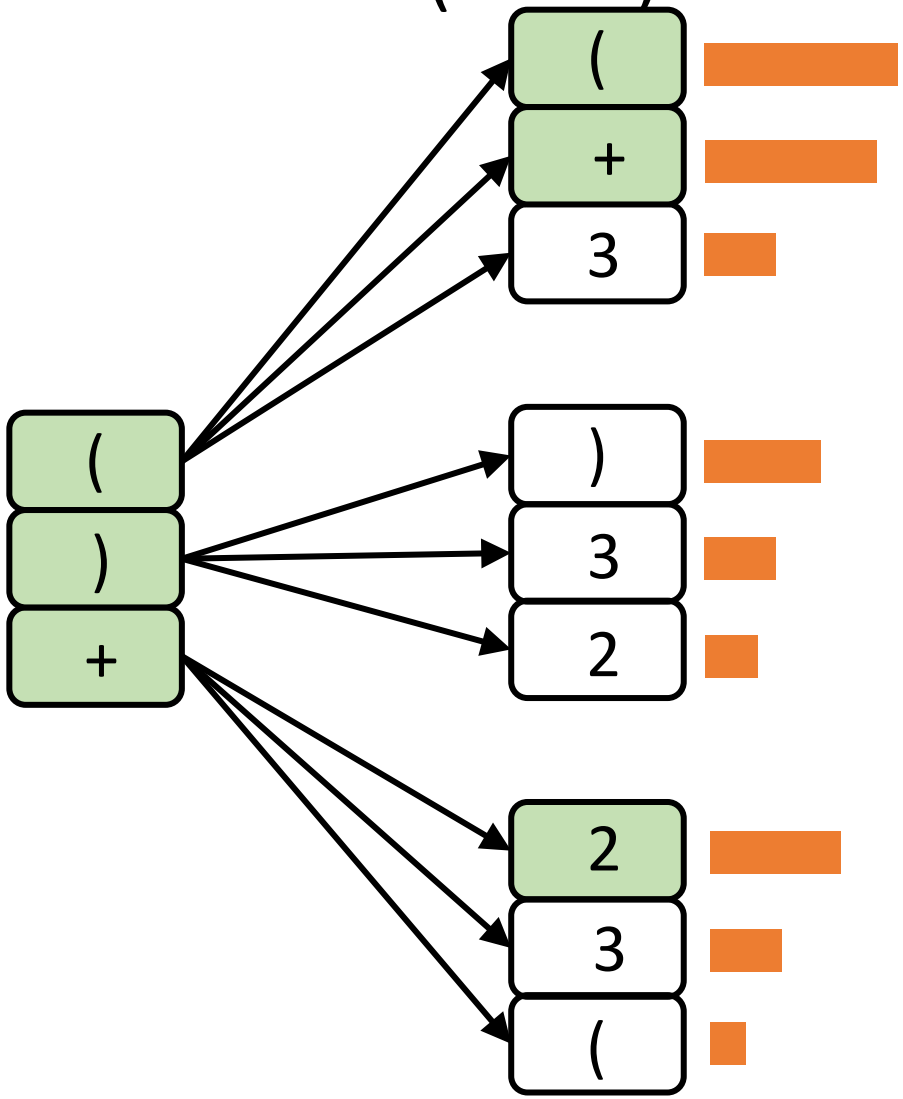


$$\{\hat{y}_1^1 = (, \hat{y}_1^2 =), \hat{y}_1^3 = +\}$$

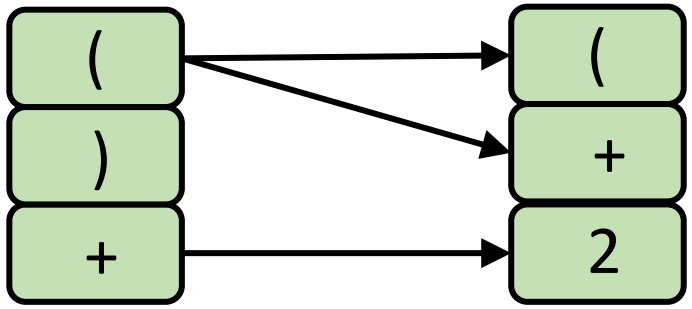
Beam Search (K=3)



Beam Search (K=3)

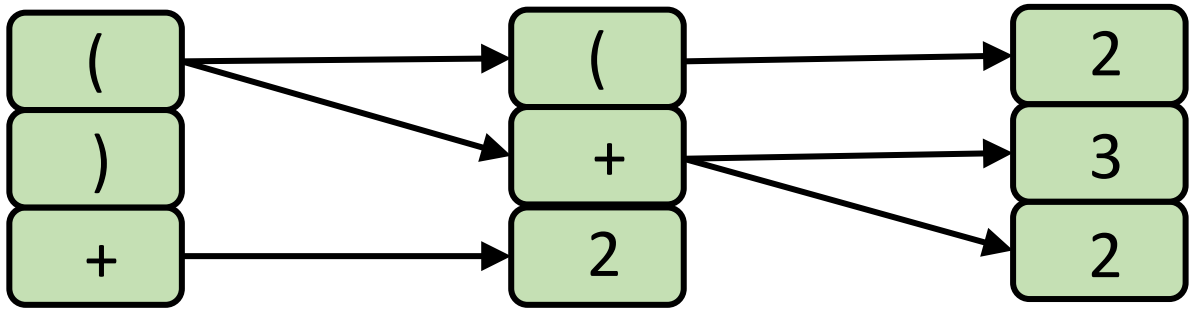


Beam Search (K=3)



$$\{\hat{y}^1_{1:2} = (\ (, \hat{y}^2_{1:2} = (\ +, \hat{y}^3_{1:2} = +\ 2\}$$

Beam Search (K=3)



$$\{\hat{y}^1_{1:3} = (\ (\ 2, \hat{y}^2_{1:3} = (\ +\ 3, \hat{y}^3_{1:3} = (\ +\ 2\}$$

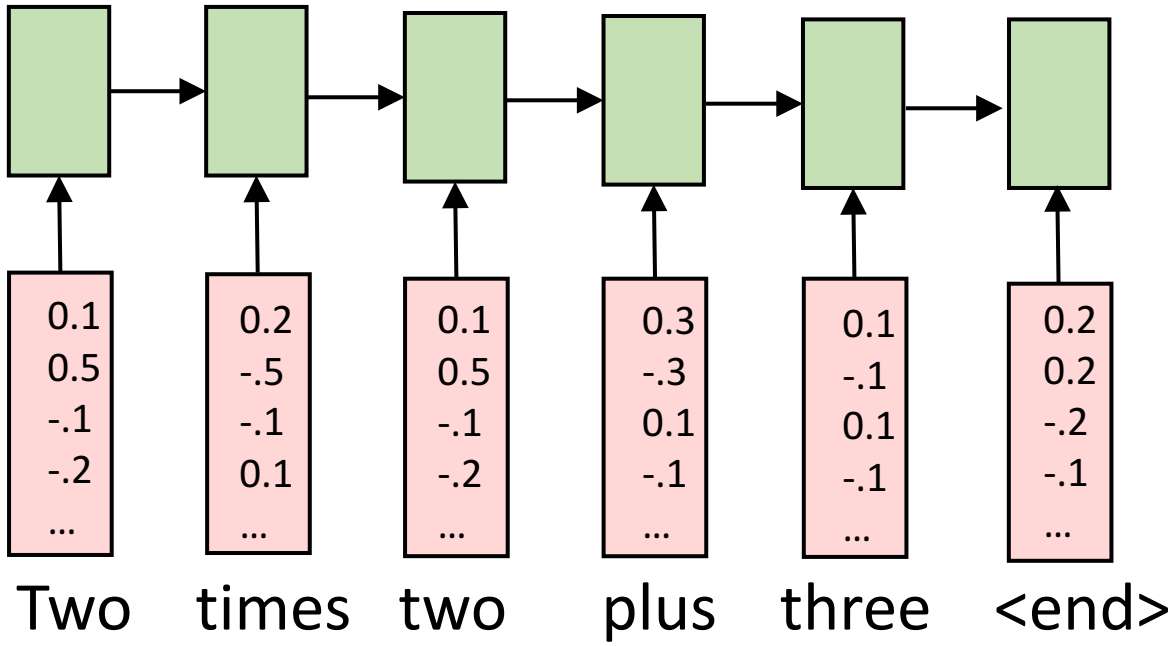
Beam Search

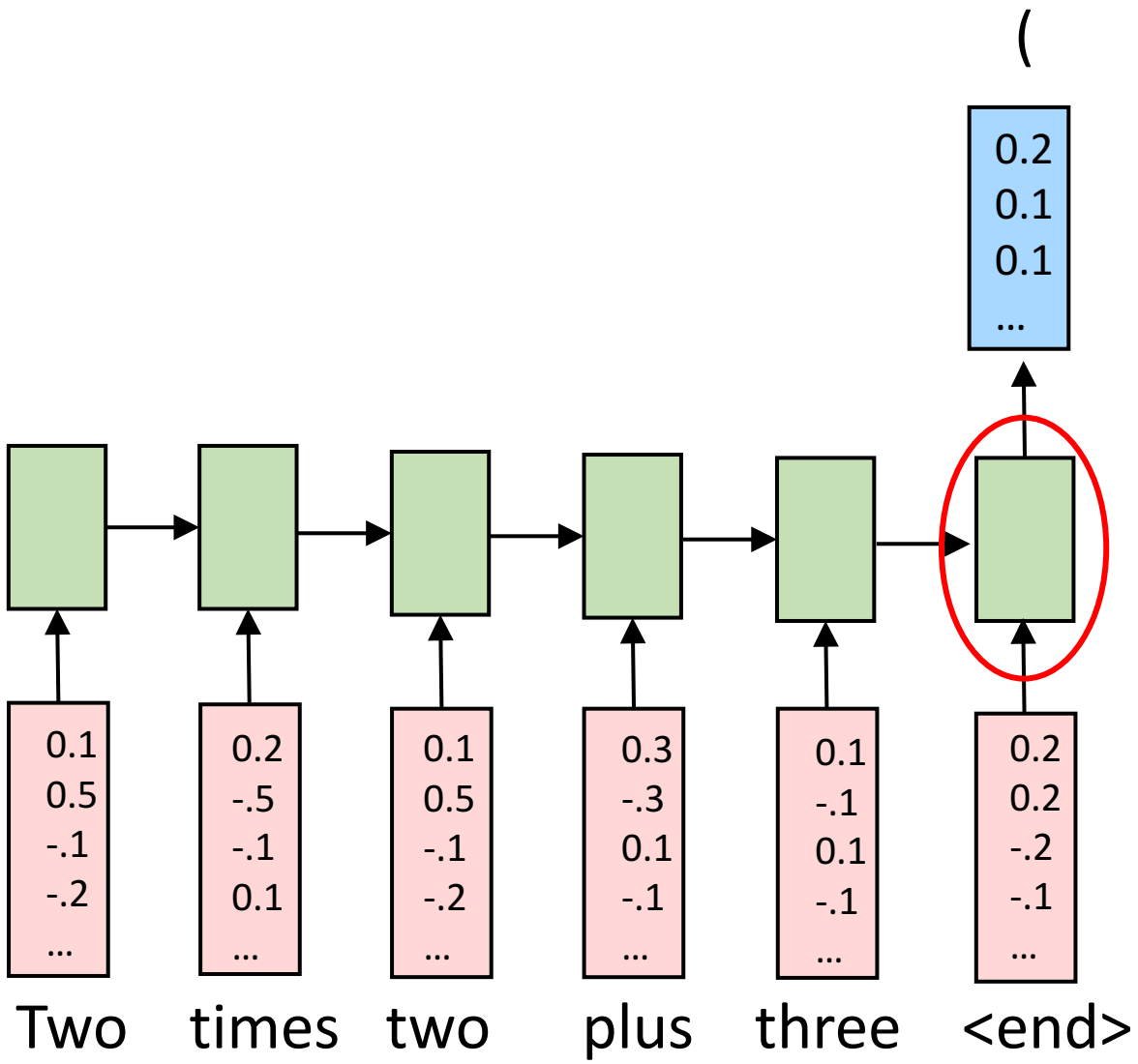
- Can be very slow in practice
- How to prune branches more efficiently?
- How to take into account global structure?

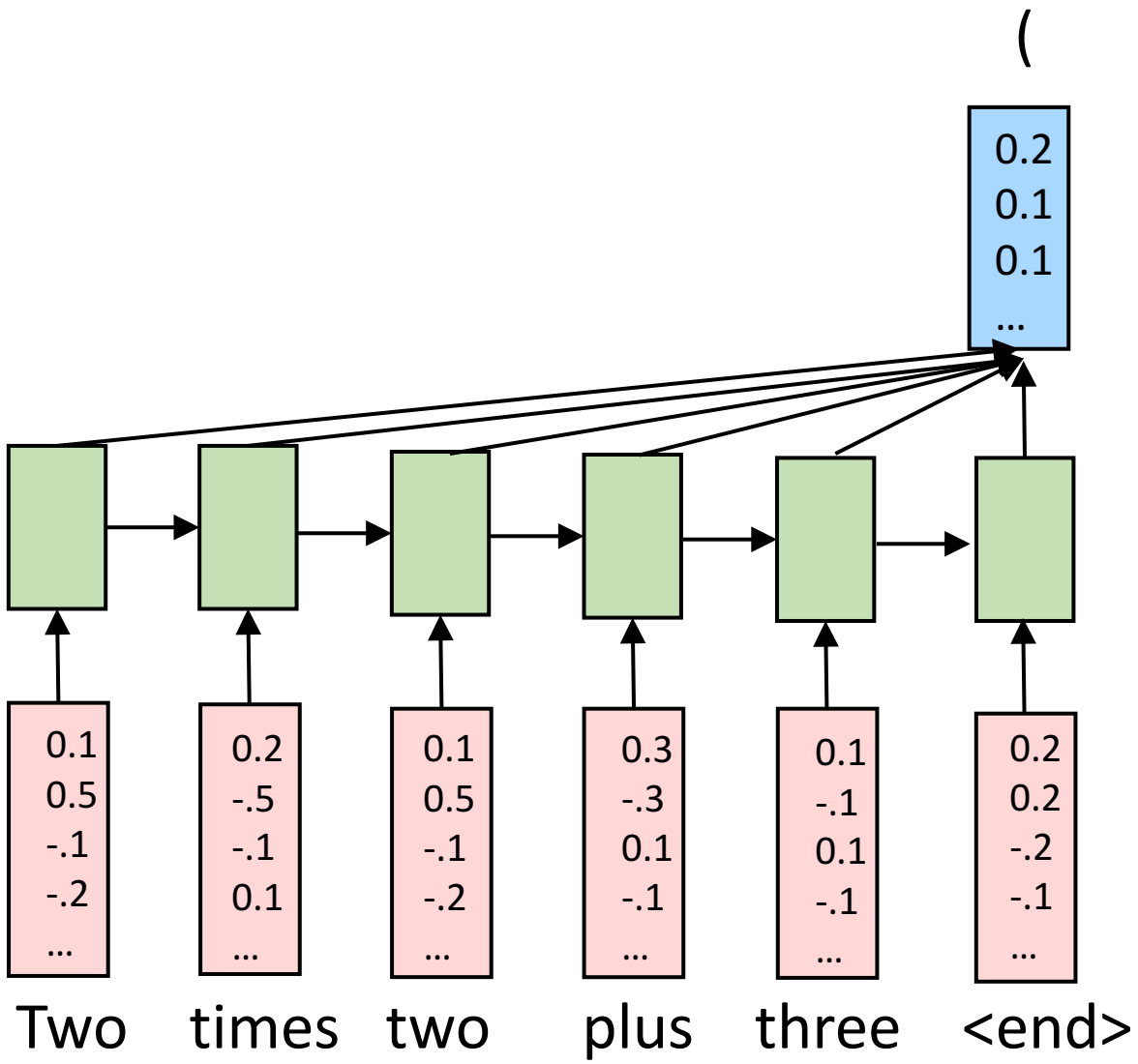
Improvements – Attention Mechanism

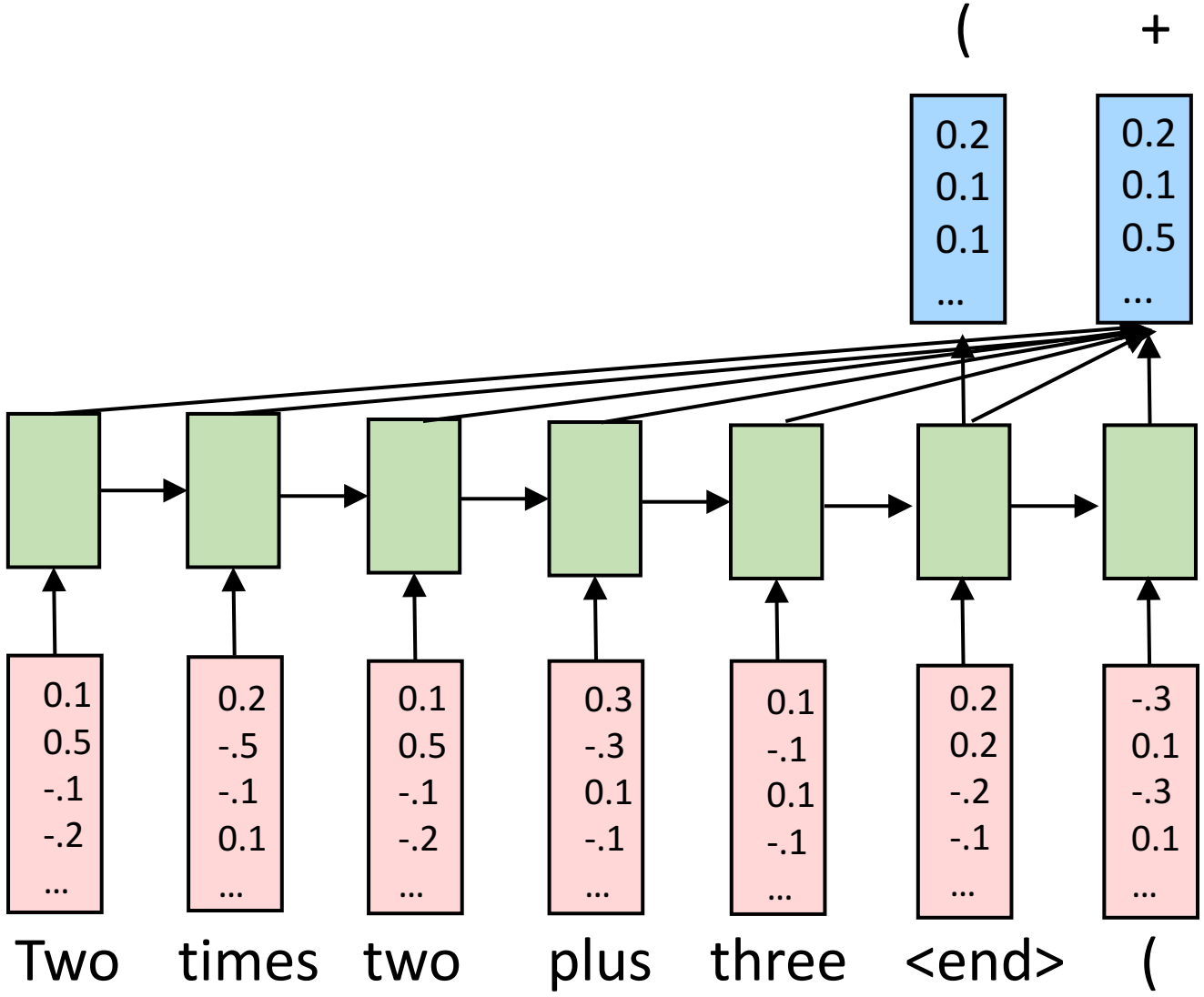
Issue: Compress a long sentence into a single vector?

Solution: Look back to the input at each time step


0.1
0.5
-.1
-.2
…
0.2
-.5
-.1
0.1
…
0.1
0.5
-.1
-.2
…
0.3
-.3
0.1
-.1
…
0.1
-.1
0.1
-.1
…
0.2
0.2
-.2
-.1
…
Two
times
two
plus
three
<end>

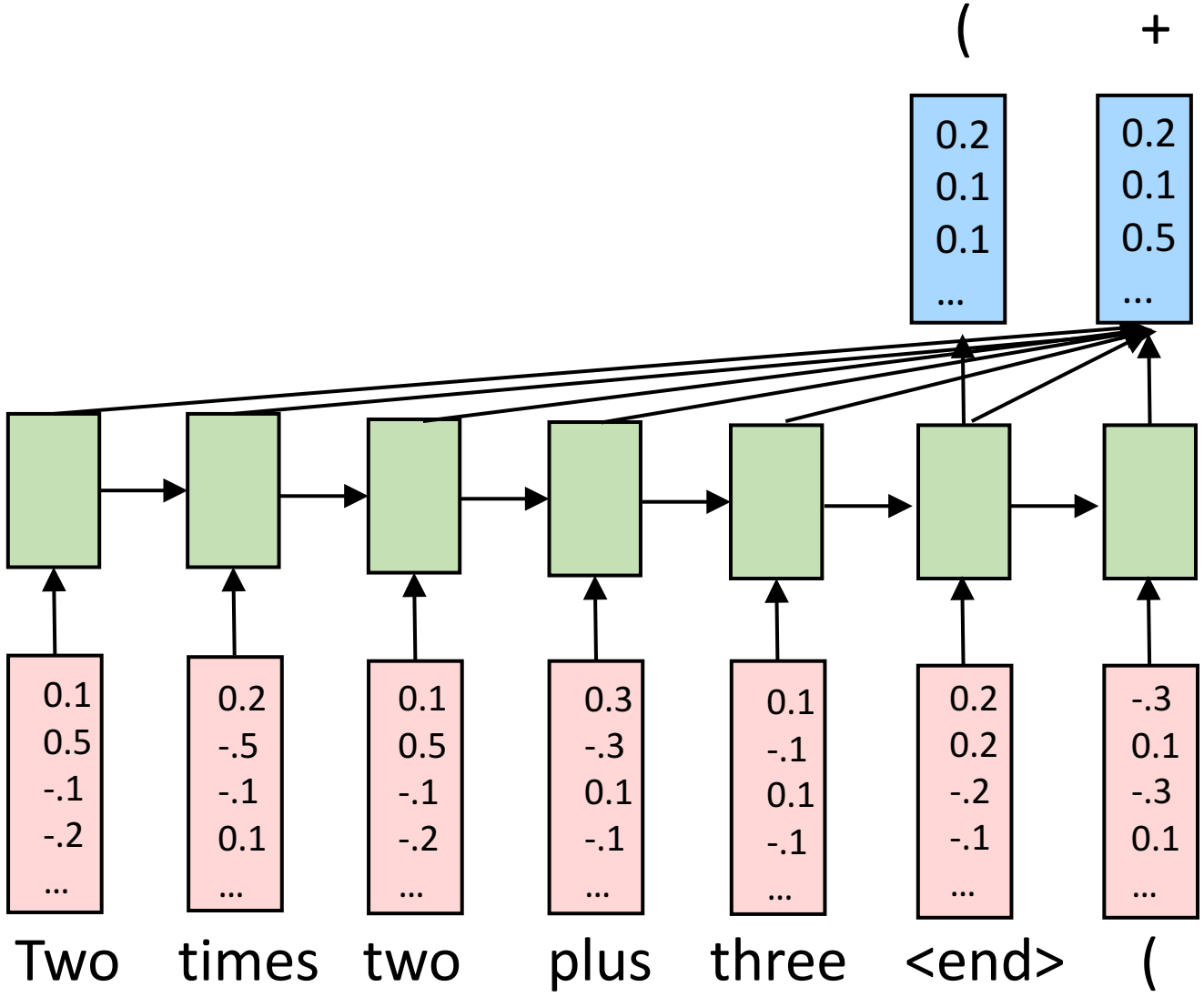

(
0.2
0.1
0.1
...
0.1
0.5
-.1
-.2
...
0.2
-.5
-.1
0.1
...
0.1
0.5
-.1
-.2
...
0.3
-.3
0.1
-.1
...
0.1
-.1
0.1
-.1
...
0.2
0.2
-.2
-.1
...
Two
times
two
plus
three
<end>


(
0.2
0.1
0.1
...
0.1
0.5
-.1
-.2
...
0.2
-.5
-.1
0.1
...
0.1
0.5
-.1
-.2
...
0.3
-.3
0.1
-.1
...
0.1
-.1
0.1
-.1
...
0.2
0.2
-.2
-.1
...
Two
times
two
plus
three
<end>



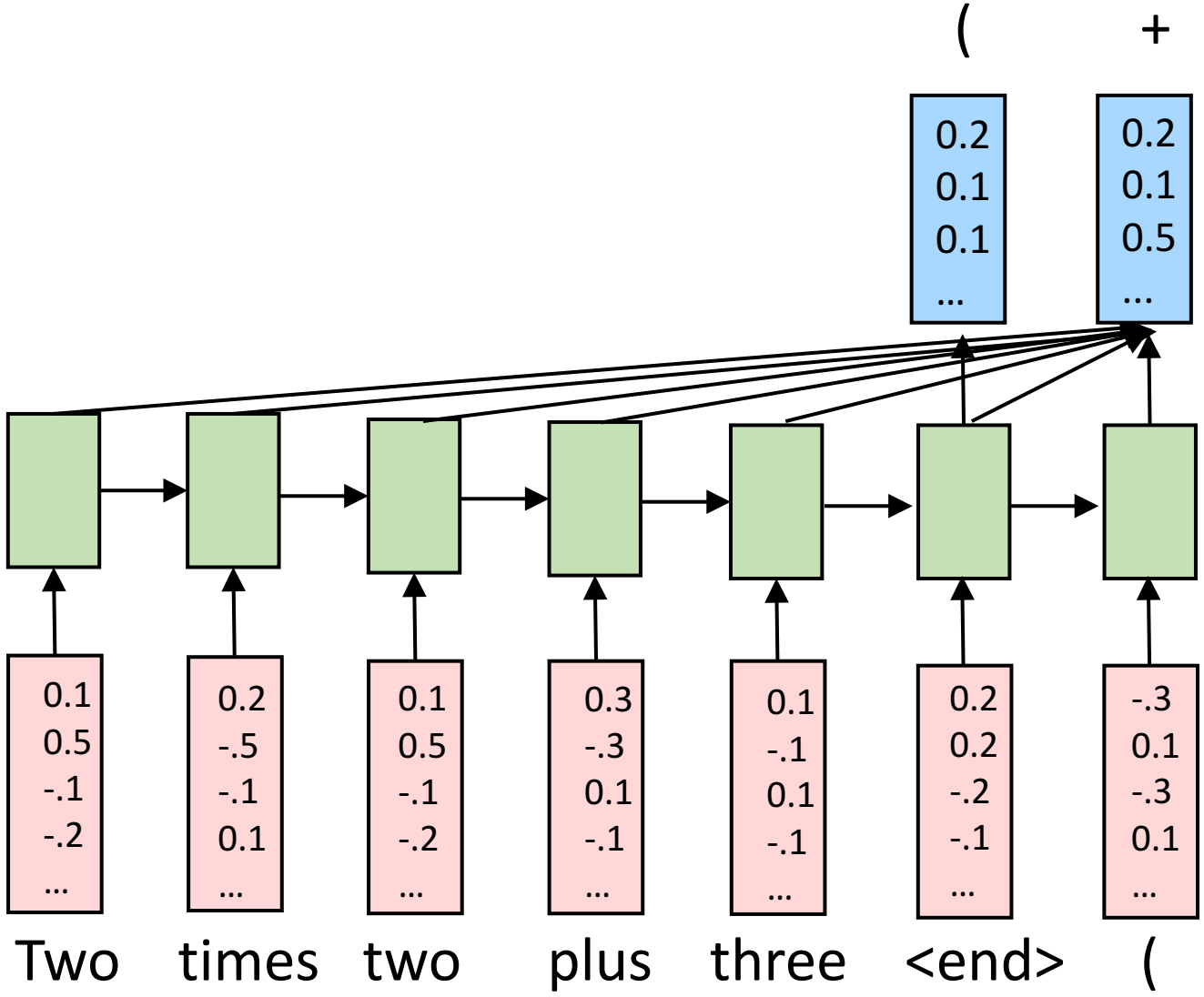
$$h_t^{context} = \sum_{i=1}^{n} \alpha_{t,i} h_i^{enc}$$

Learn to look back at the input

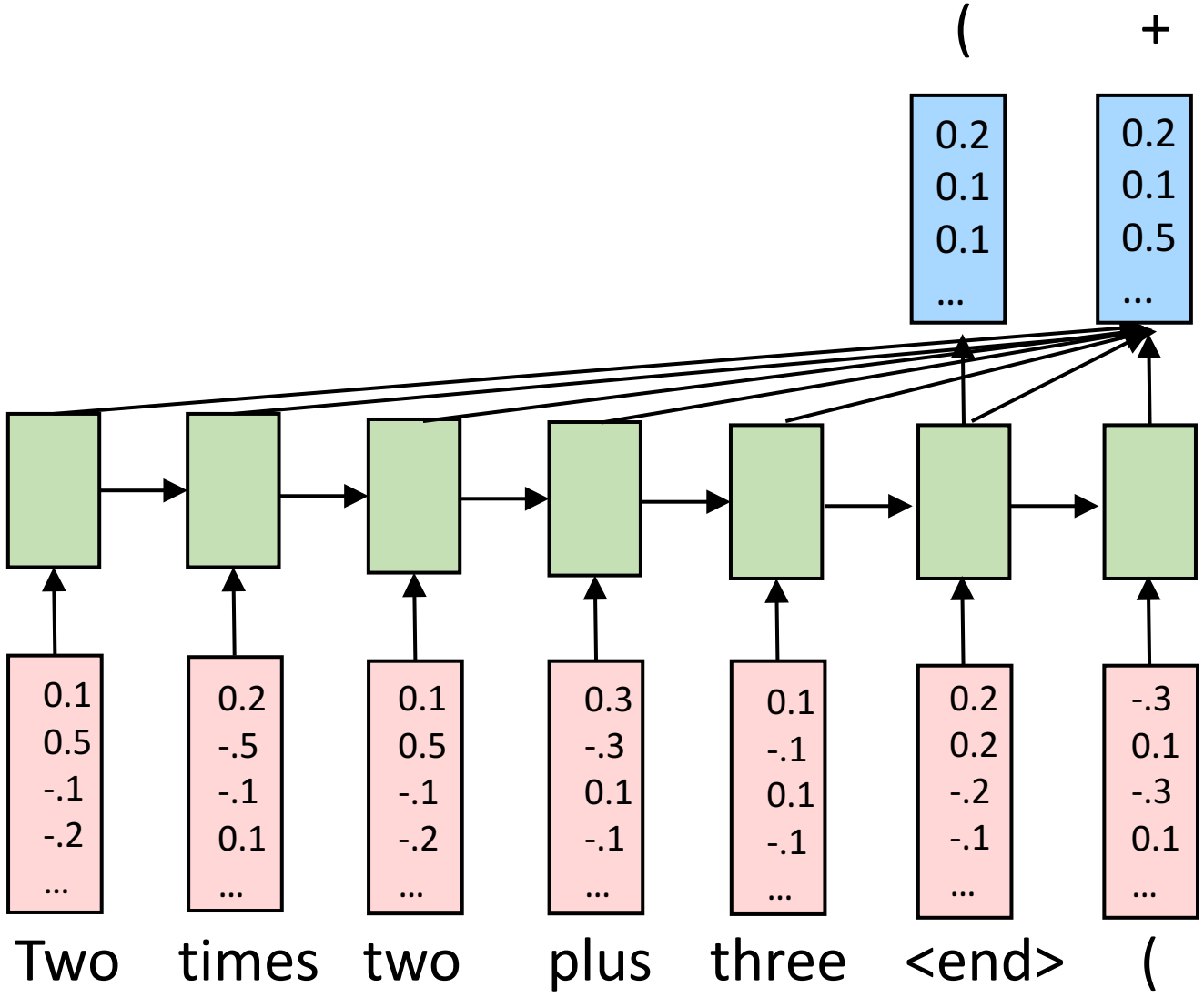


$$h_t^{context} = \sum_{i=1}^{n} \alpha_{t,i} h_i^{enc}$$

$$\alpha_{t,i} \propto (h_t^{dec} W_{proj})^T h_i^{enc}$$



$$p(y_t|y_{1:t-1}, x) = \text{softmax}(W[h_t^{dec}; h_t^{context}])$$



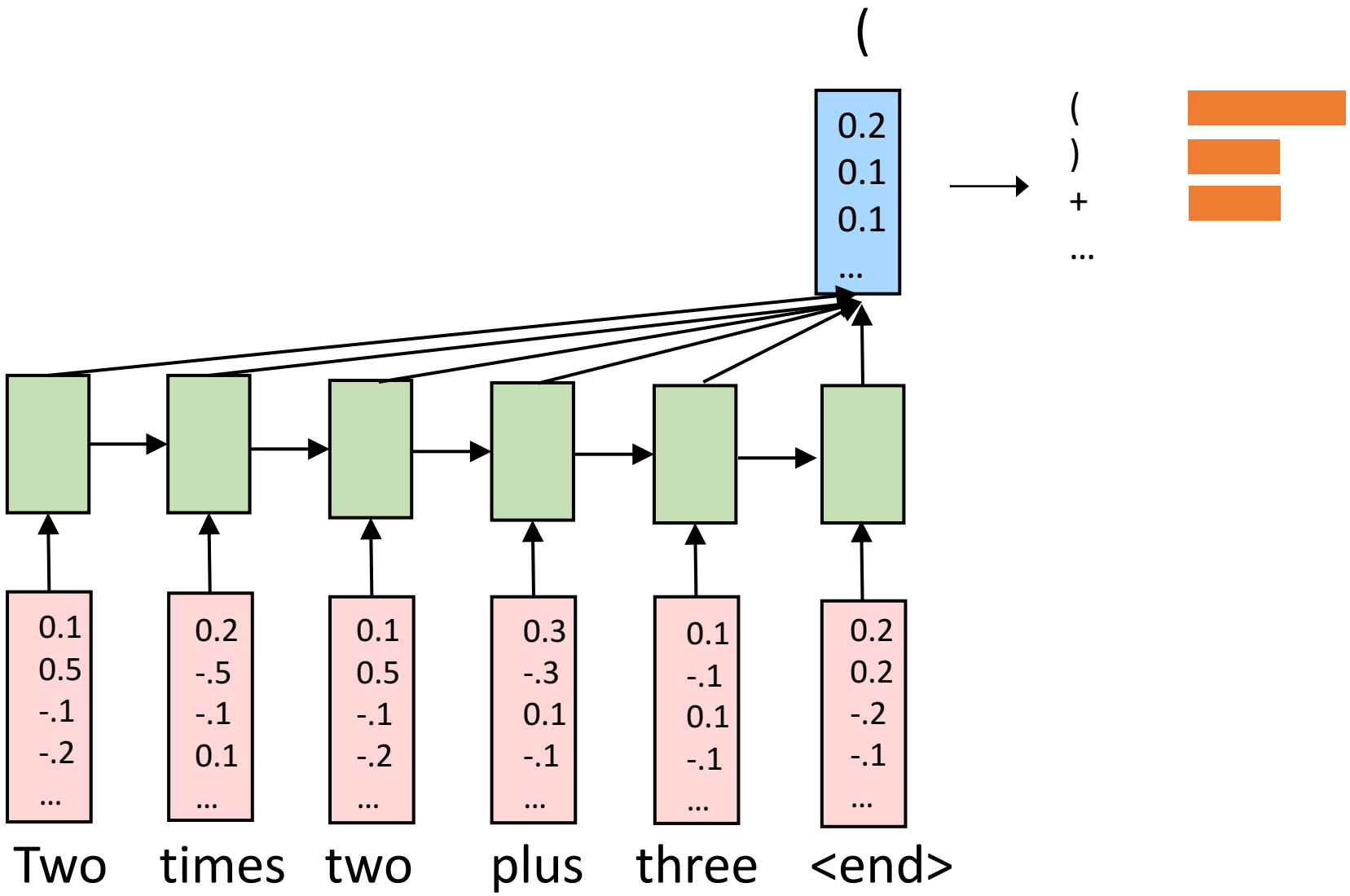
$$p(y_t|y_{1:t-1}, x) = \text{softmax}(W[h_t^{dec}; h_t^{context}])$$

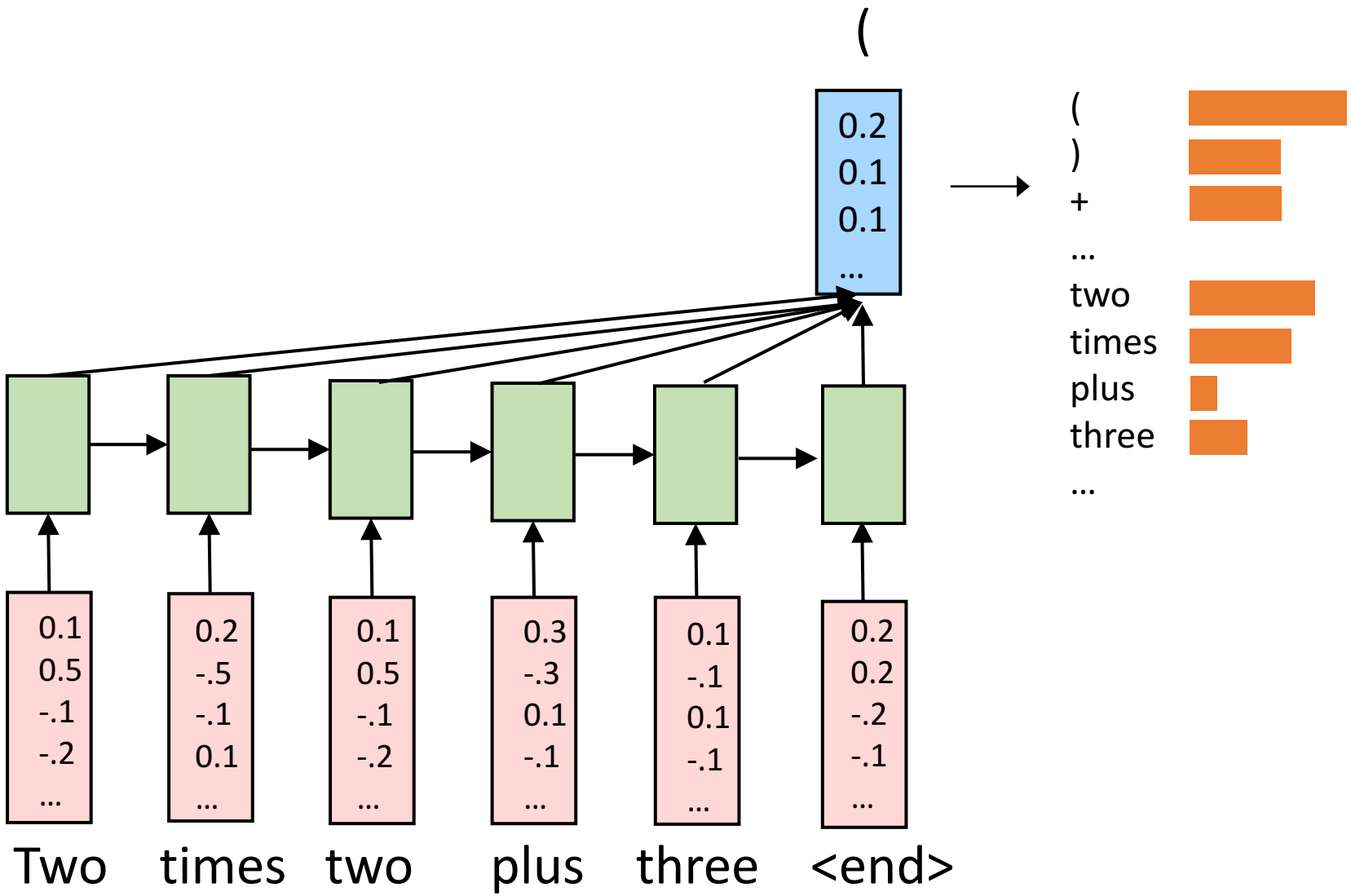
$$h_t^{dec} = f(h_{t-1}^{dec}, y_{t-1}, h_t^{context})$$

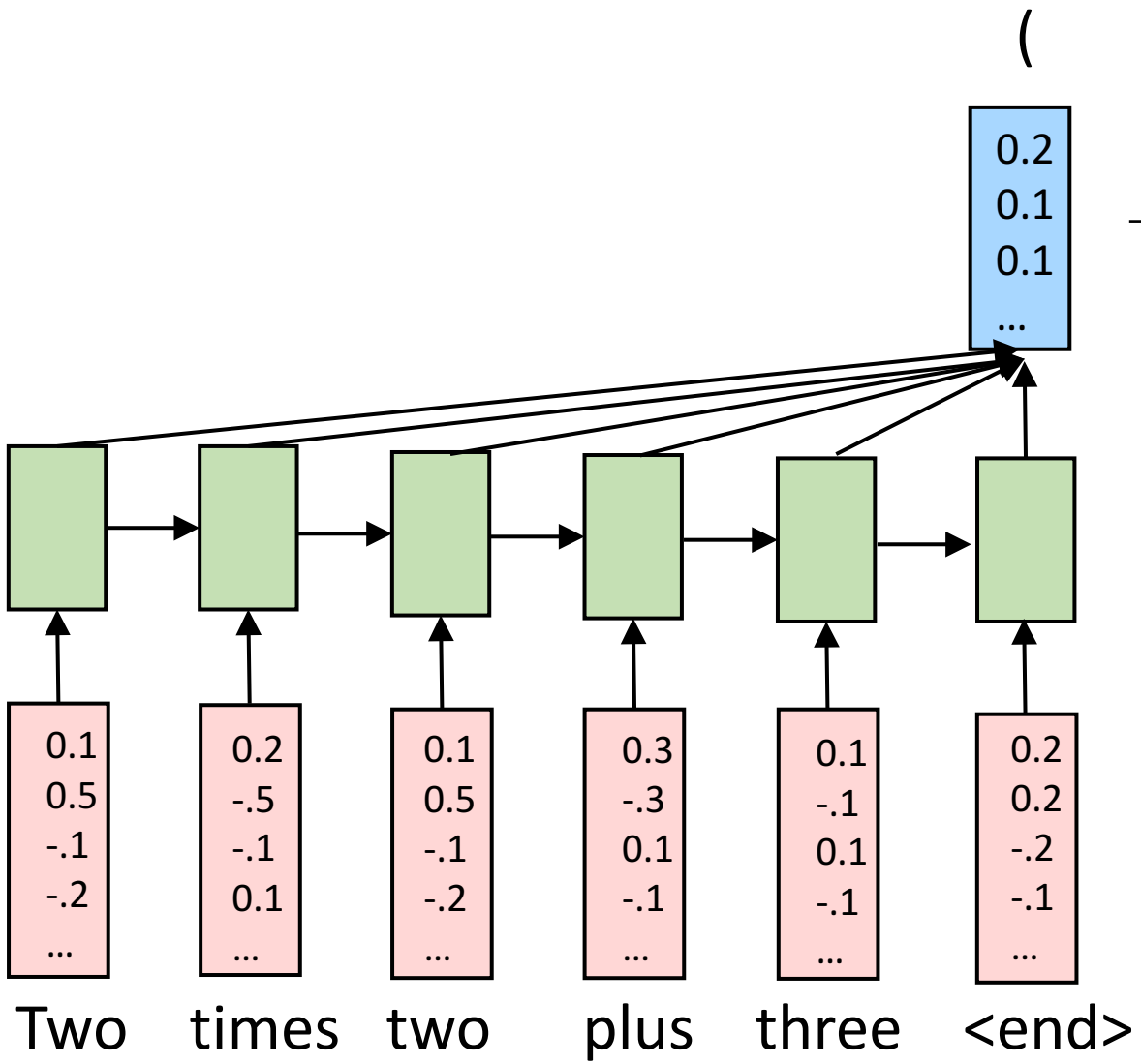
Improvements – Copying Mechanism

Issue: How to generate rare words? How to generate words that do not occur during training?

Solution: Attention-based copying


(
0.2
0.1
0.1
...
(
)
+
...
0.1
0.5
-.1
-.2
...
0.2
-.5
-.1
0.1
...
0.1
0.5
-.1
-.2
...
0.3
-.3
0.1
-.1
...
0.1
-.1
0.1
-.1
...
0.2
0.2
-.2
-.1
...
Two
times
two
plus
three
<end>


(
0.2
0.1
0.1
...
(
)
+
...
two
times
plus
three
...
0.1
0.5
-.1
-.2
...
0.2
-.5
-.1
0.1
...
0.1
0.5
-.1
-.2
...
0.3
-.3
0.1
-.1
...
0.1
-.1
0.1
-.1
...
0.2
0.2
-.2
-.1
...
Two
times
two
plus
three
<end>



$$p(y_t|y_{1:t-1}, x) =$$

$$\text{softmax}([W[h_t^{dec}; h_t^{context}]; \alpha])$$

Other tricks

- Increase depth
- Bidirectional LSTM
- Dropout
- Pre-train word embeddings
- Optimization algorithms: SGD, SGD+Momentum, AdaGrad, Adam, …

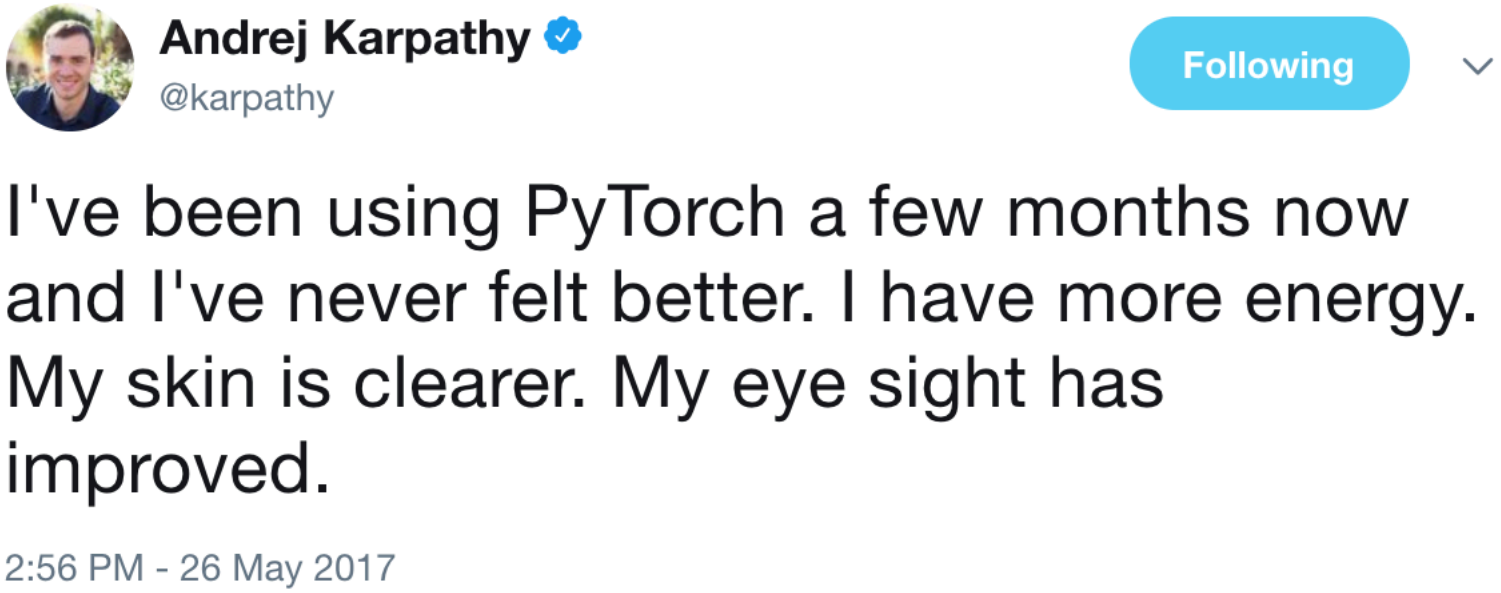
SEMPRE vs Seq2Seq

- Data preparation
  - Seq2Seq is simpler, no grammar
- Training
  - SEMPRE is more flexible in encoding structural features (structural loss vs. maximum likelihood loss)
  - With enough training data, seq2seq can learn complex hidden patterns
- Prediction
  - Both using beam search
  - SEMPRE may not need beam search when the grammar is constrained enough

PyTorch Tutorial

Why PyTorch?

Why PyTorch?


Andrej Karpathy
@karpathy
Following
I've been using PyTorch a few months now and I've never felt better. I have more energy. My skin is clearer. My eye sight has improved.
2:56 PM - 26 May 2017

Why PyTorch?

- Python code:
  - Easy to read and write
  - Use numpy and scipy

Why PyTorch?

- Python code:
  - Easy to read and write
  - Use numpy and scipy
- Fast:
  - Build-in data preprocessing
  - Multi-thread, multi-GPU support

Why PyTorch?

- Python code:
  - Easy to read and write
  - Use numpy and scipy
- Fast:
  - Build-in data preprocessing
  - Multi-thread, multi-GPU support
- Dynamic Computation Graphs:
  - Easier debugging
  - Process inputs of variable sizes

Why PyTorch?

- Python code:
  - Easy to read and write
  - Use numpy and scipy
- Fast:
  - Build-in data preprocessing
  - Multi-thread, multi-GPU support
- **Dynamic Computation Graphs**:
  - Easier debugging
  - Process inputs of variable sizes