Neural Networks in NLP

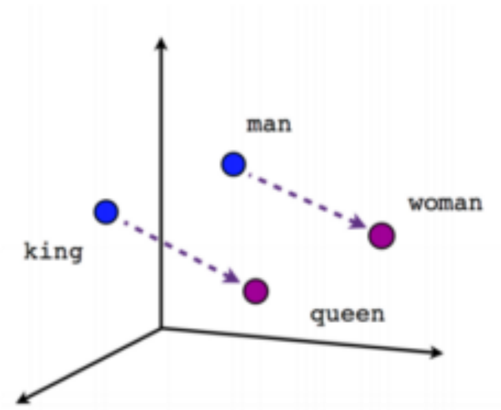
CS 485, Spring 2024
Applications of Natural Language Processing
https://people.cs.umass.edu/~brenocon/cs485_s24/

Brendan O'Connor
College of Information and Computer Sciences
University of Massachusetts Amherst

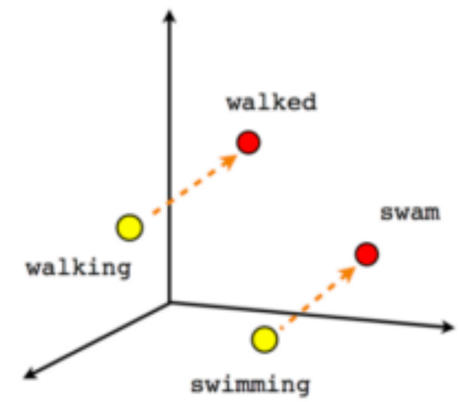
[Slides from Mohit Iyyer and Richard Socher]

- Progress report: due Monday 4/22
- HW4 after that - last homework!
- Final presentations: May 7 & 9

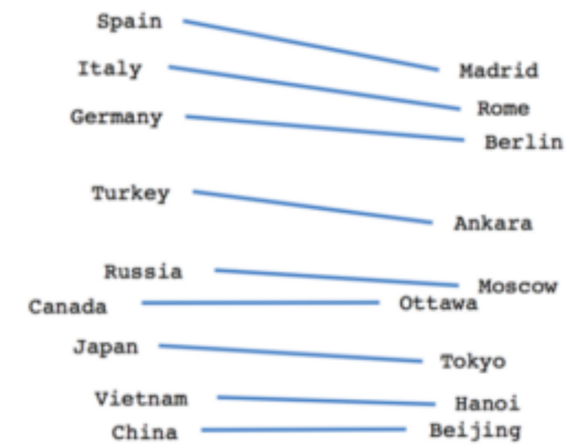
- We have word embeddings...



Male-Female



Verb tense



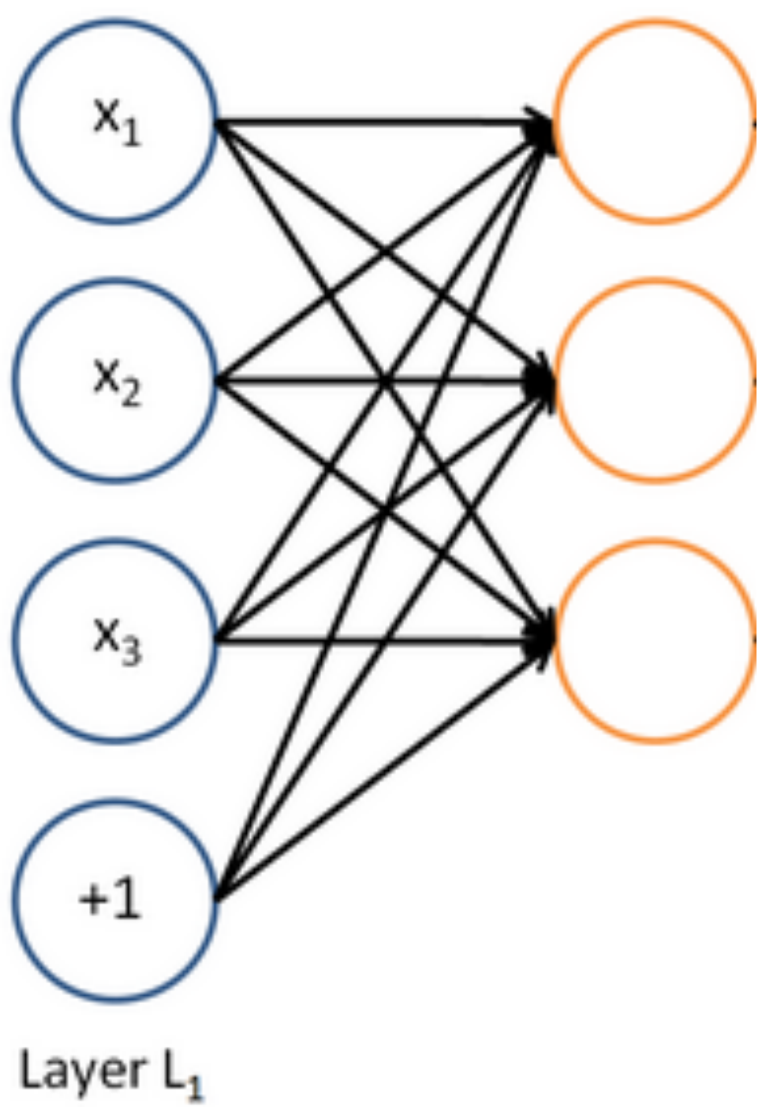
Country-Capital

- Can we compose meanings (embeddings) for phrases, sentences, etc.?

- Or, contextual meaning for *each* token?
- Key idea: *automatically* determine how to combine embedding from different tokens

NN: kind of like several intermediate logregs

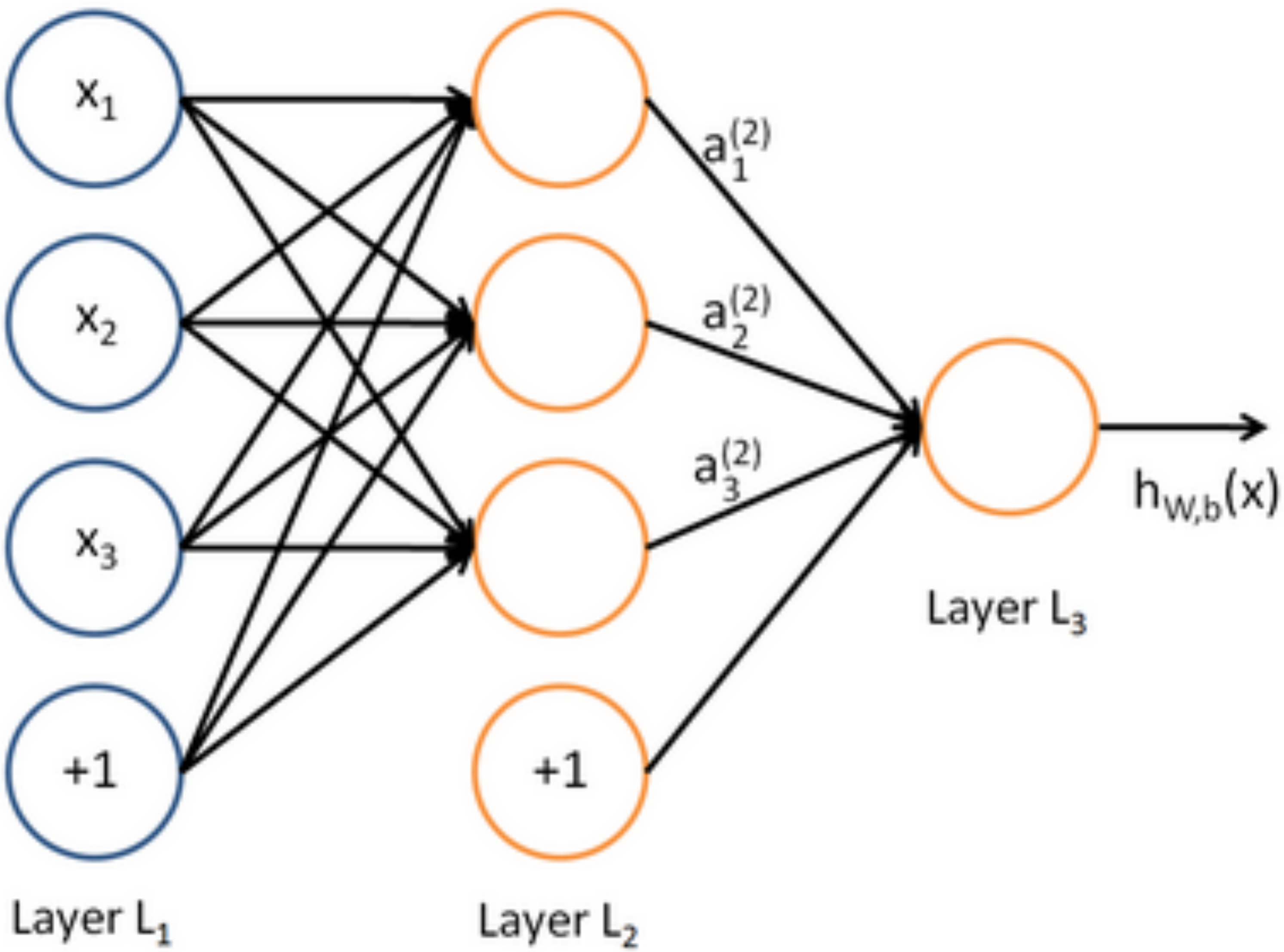
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs …



But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!

NN: kind of like several intermediate logregs

… which we can feed into another logistic regression function

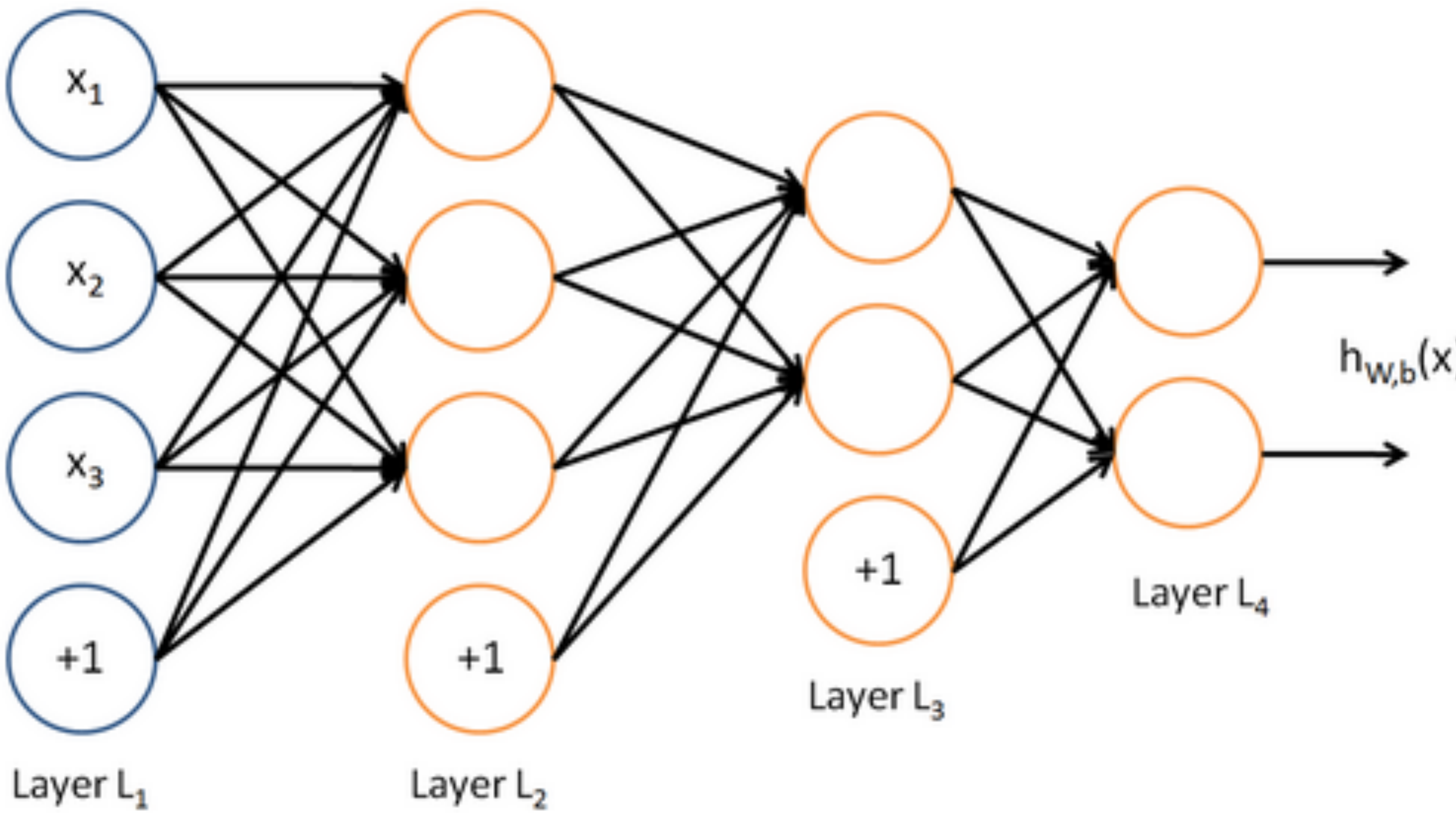


It is the loss function that will direct what the intermediate hidden variables should be, so as to do a good job at predicting the targets for the next layer, etc.

NN: kind of like several intermediate logregs

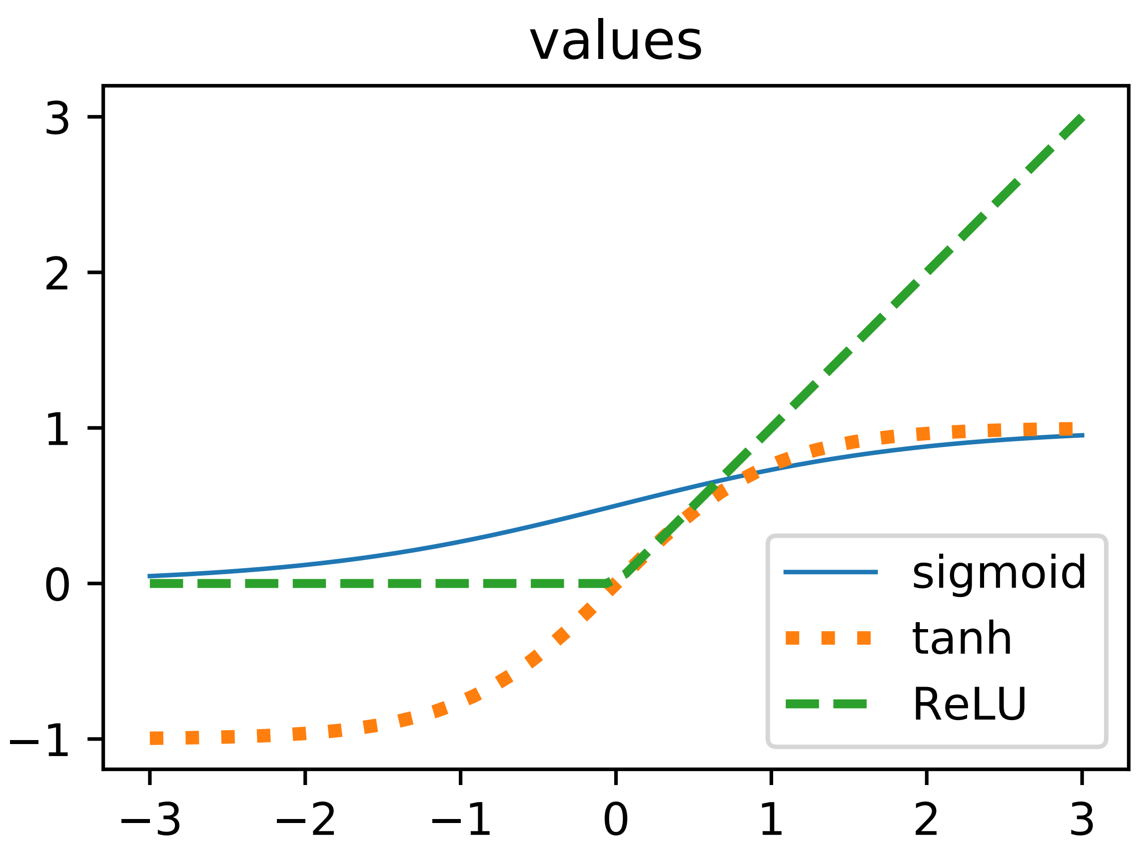
Before we know it, we have a multilayer neural network….

a.k.a. **feedforward network** (see INLP on terminology)



Nonlinear activations

- “Squash functions”!



- Logistic / Sigmoid

$$f(x) = \frac{1}{1+e^{-x}} \tag{1}$$

- tanh

$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1 \tag{2}$$

- ReLU

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{3}$$

is a multi-layer neural network with no nonlinearities
(i.e., f is the identity $f(\mathbf{x}) = \mathbf{x}$)
more powerful than a one-layer network?

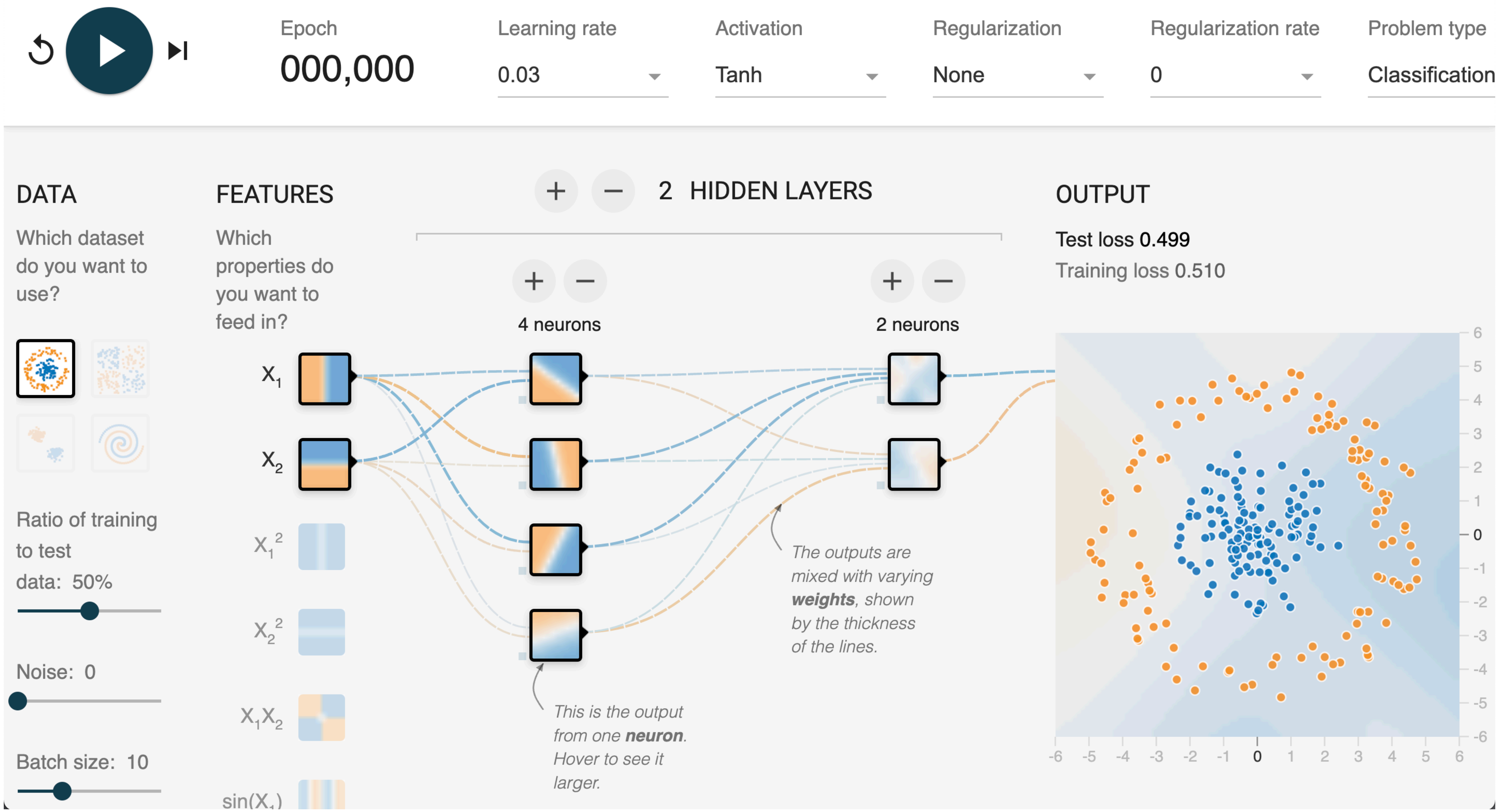
is a multi-layer neural network with no nonlinearities
(i.e., f is the identity $f(\mathbf{x}) = \mathbf{x}$)
more powerful than a one-layer network?

No! You can just compile all of the layers into a single transformation!

$$y = f(W_3 f(W_2 f(W_1 x))) = Wx$$

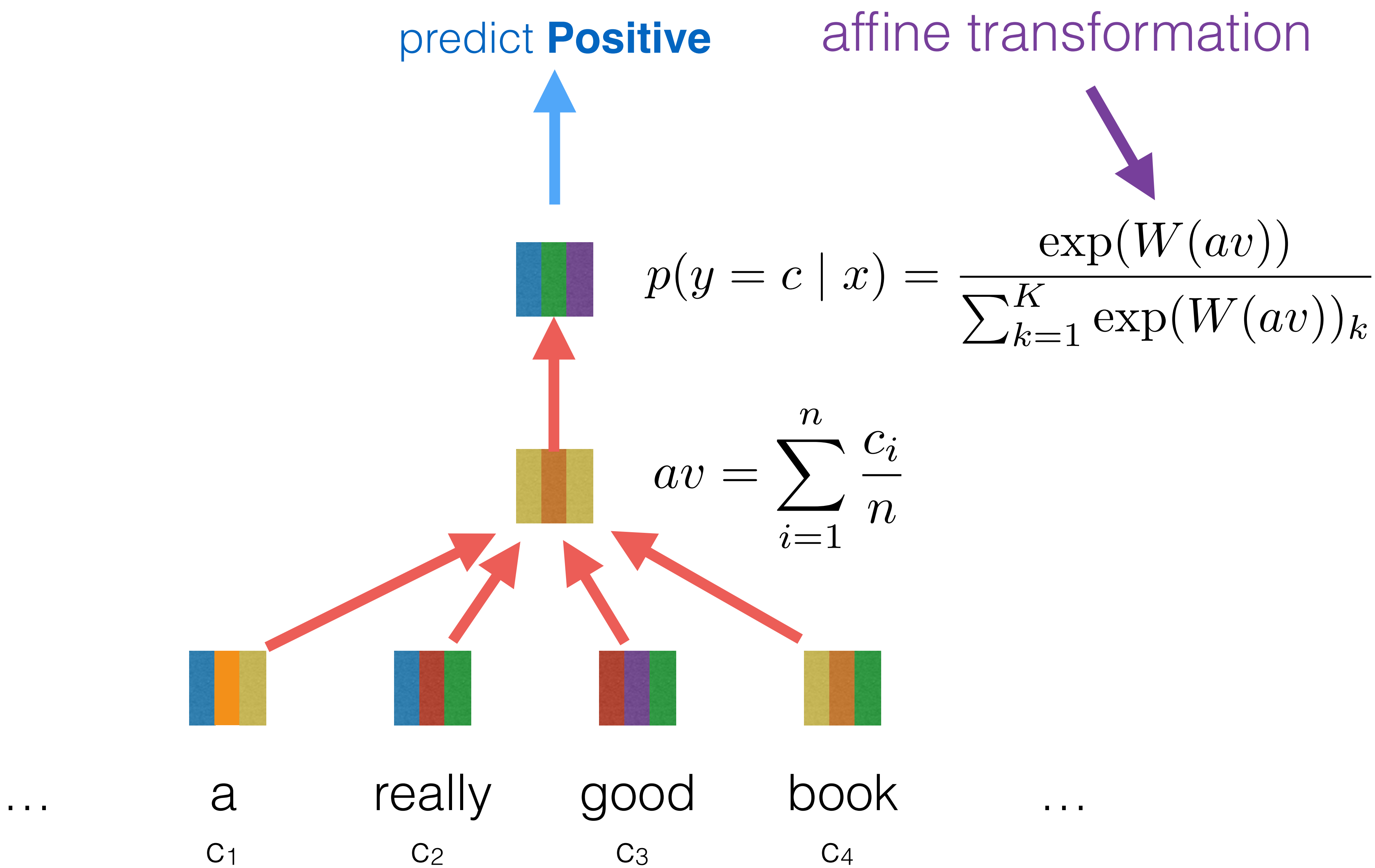
Demo

- https://playground.tensorflow.org/



- It's easy to create different neural network architectures, and execute gradient descent learning for arbitrary networks, via *backpropagation*
- e.g. the PyTorch library for Python
- Illustration: deep averaging models for text classification

“bag of embeddings”

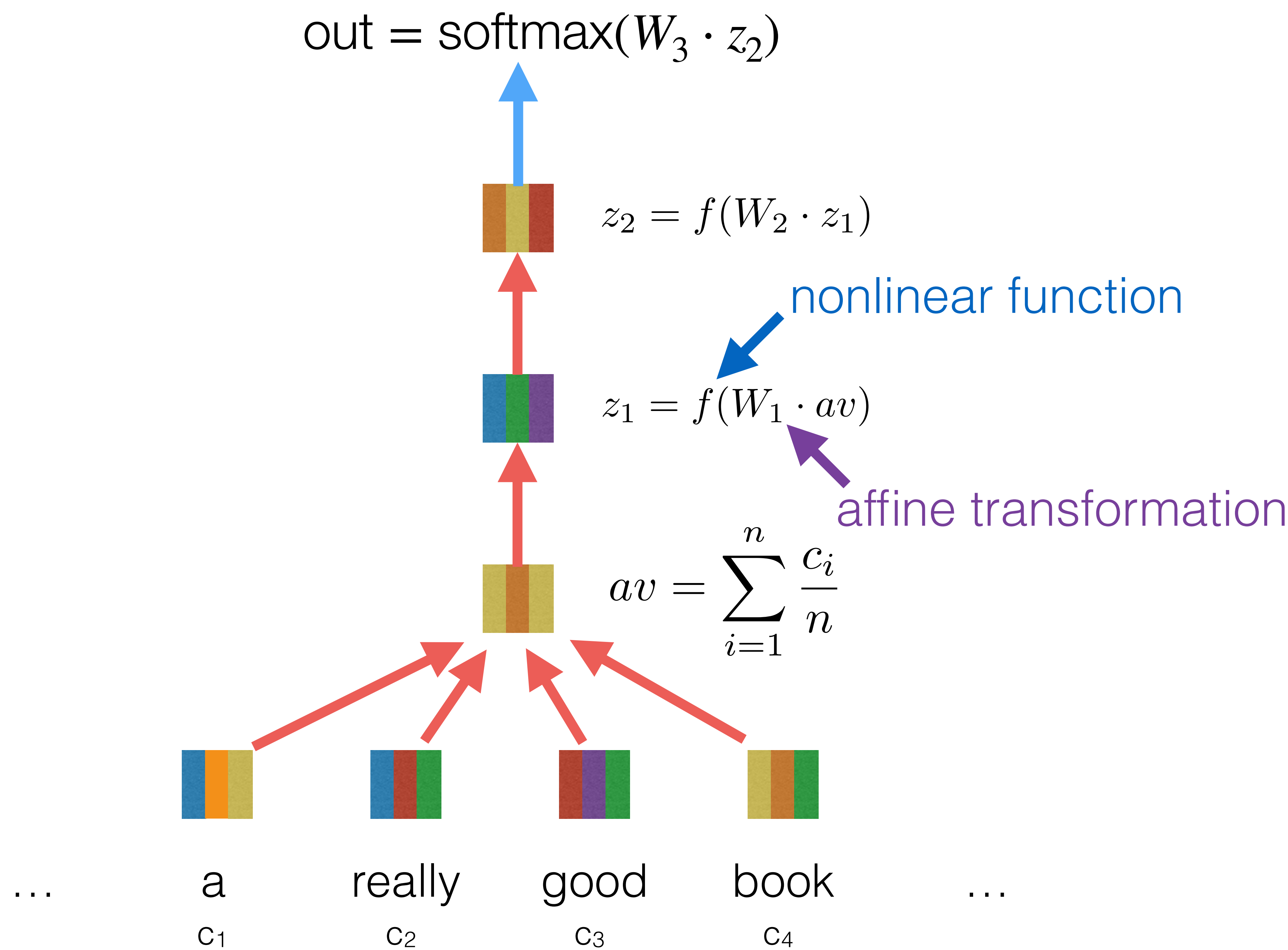


$$p(y = c \mid x) = \frac{\exp(W(av))}{\sum_{k=1}^{K} \exp(W(av))_k}$$

$$av = \sum_{i=1}^{n} \frac{c_i}{n}$$

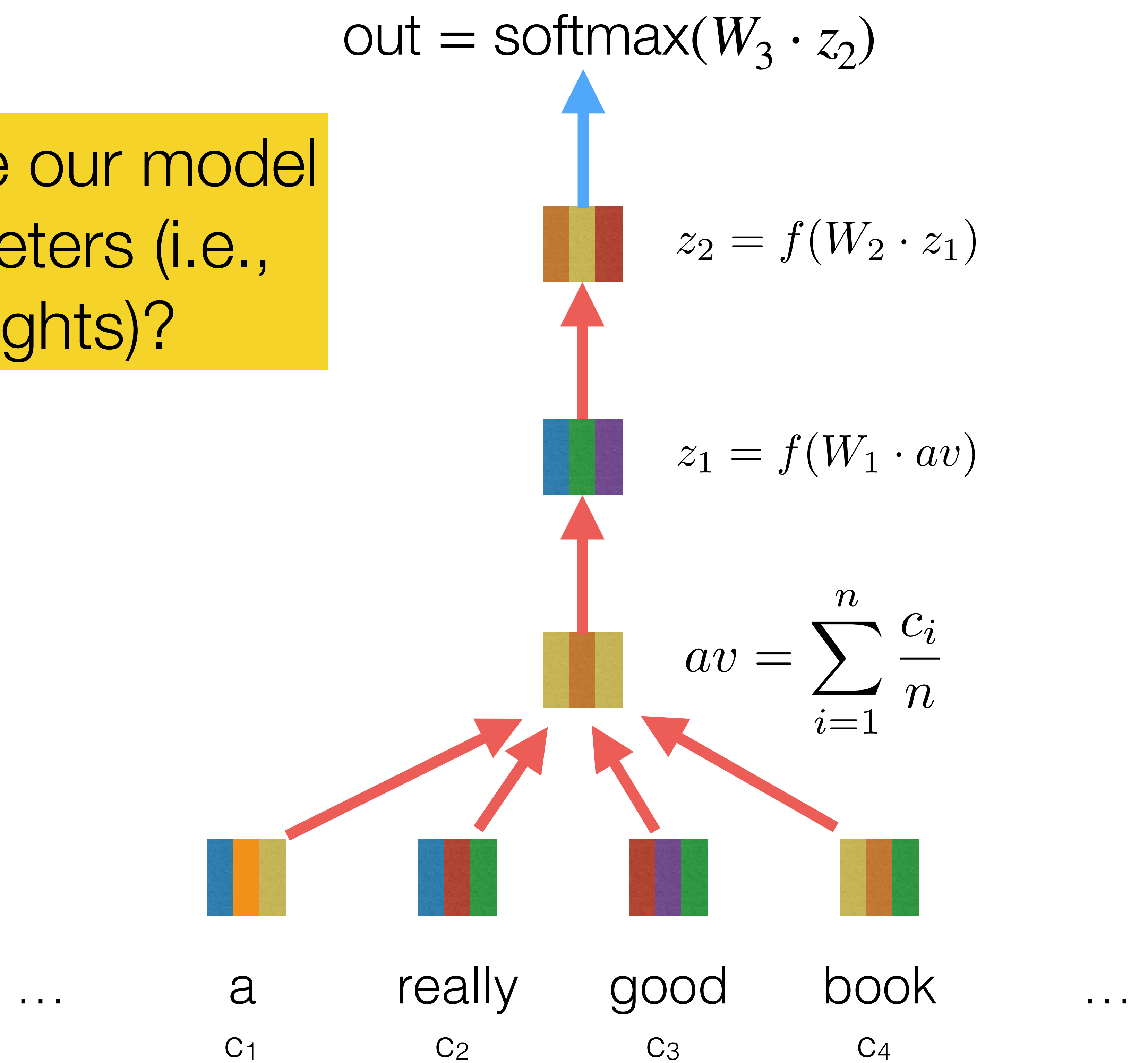
Iyyer et al., ACL 2015

deep averaging networks



deep averaging networks

what are our model parameters (i.e., weights)?



deep averaging networks

$$\text{out} = \text{softmax}(W_3 \cdot z_2)$$

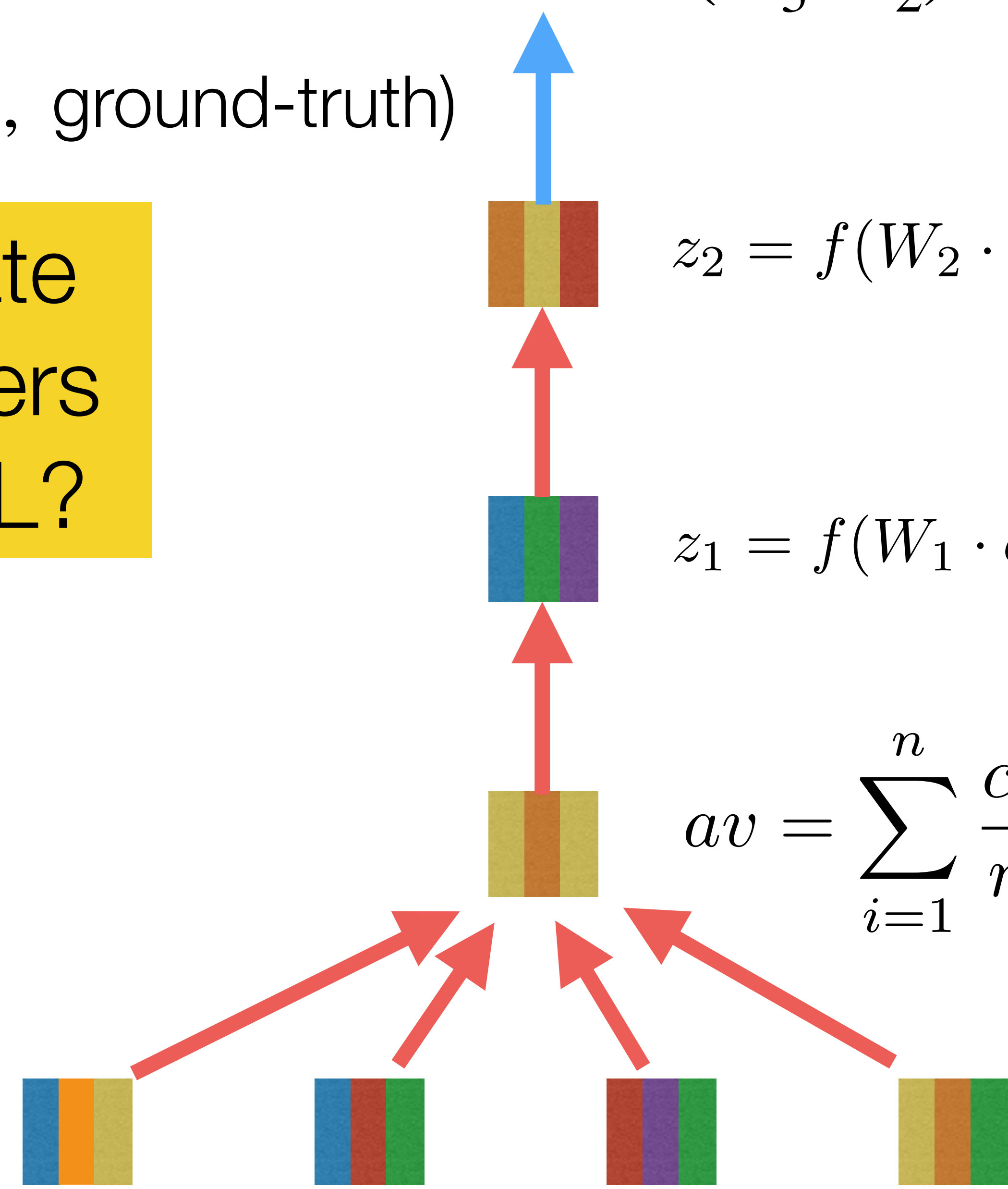
$$L = \text{cross-entropy}(\text{out, ground-truth})$$

how do i update these parameters given the loss L?

$$z_2 = f(W_2 \cdot z_1)$$

$$z_1 = f(W_1 \cdot av)$$

$$av = \sum_{i=1}^{n} \frac{c_i}{n}$$



… a (c_1) really (c_2) good (c_3) book (c_4) …

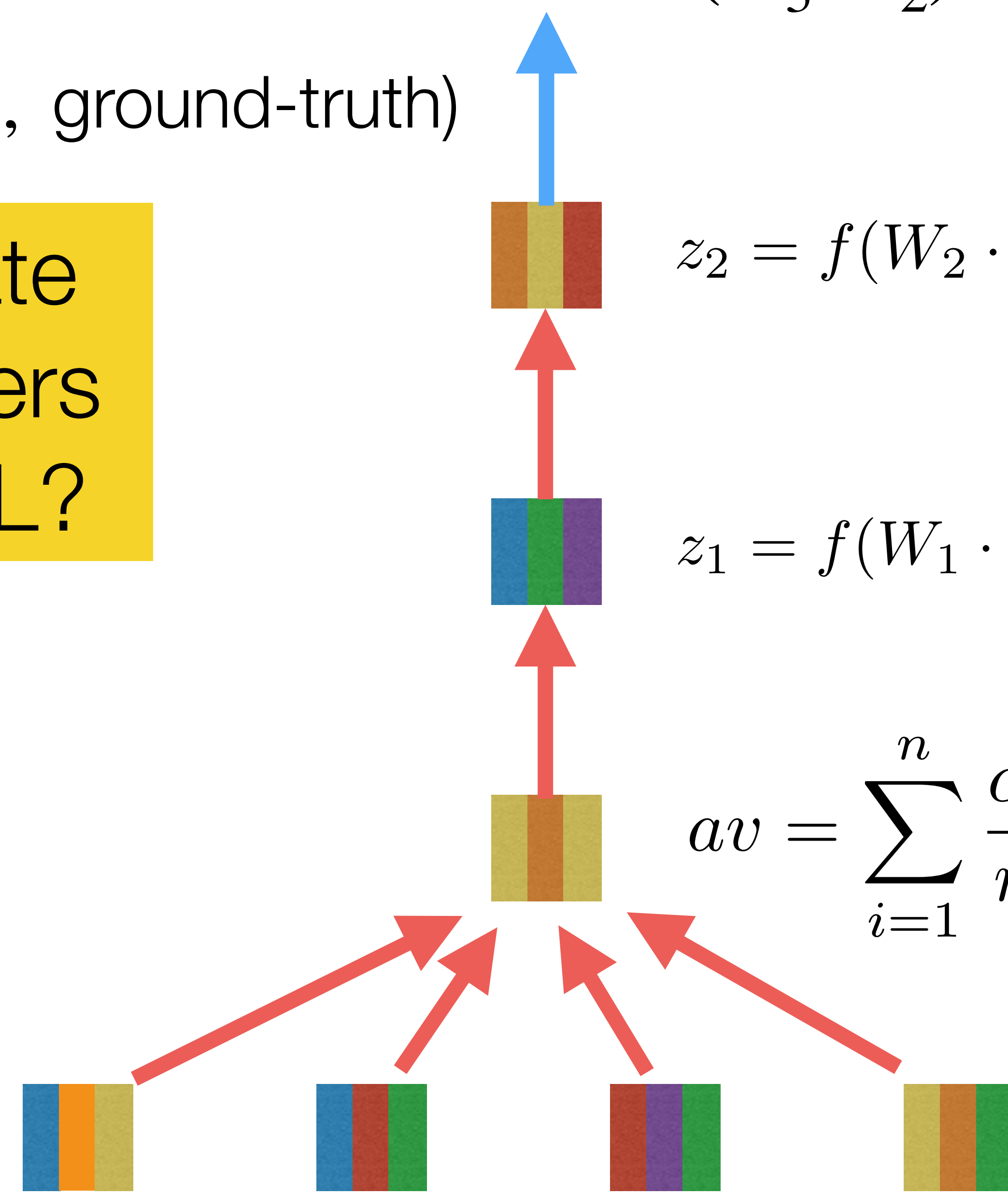
deep averaging networks

out = softmax$(W_3 \cdot z_2)$

L = cross-entropy(out, ground-truth)

how do i update these parameters given the loss L?

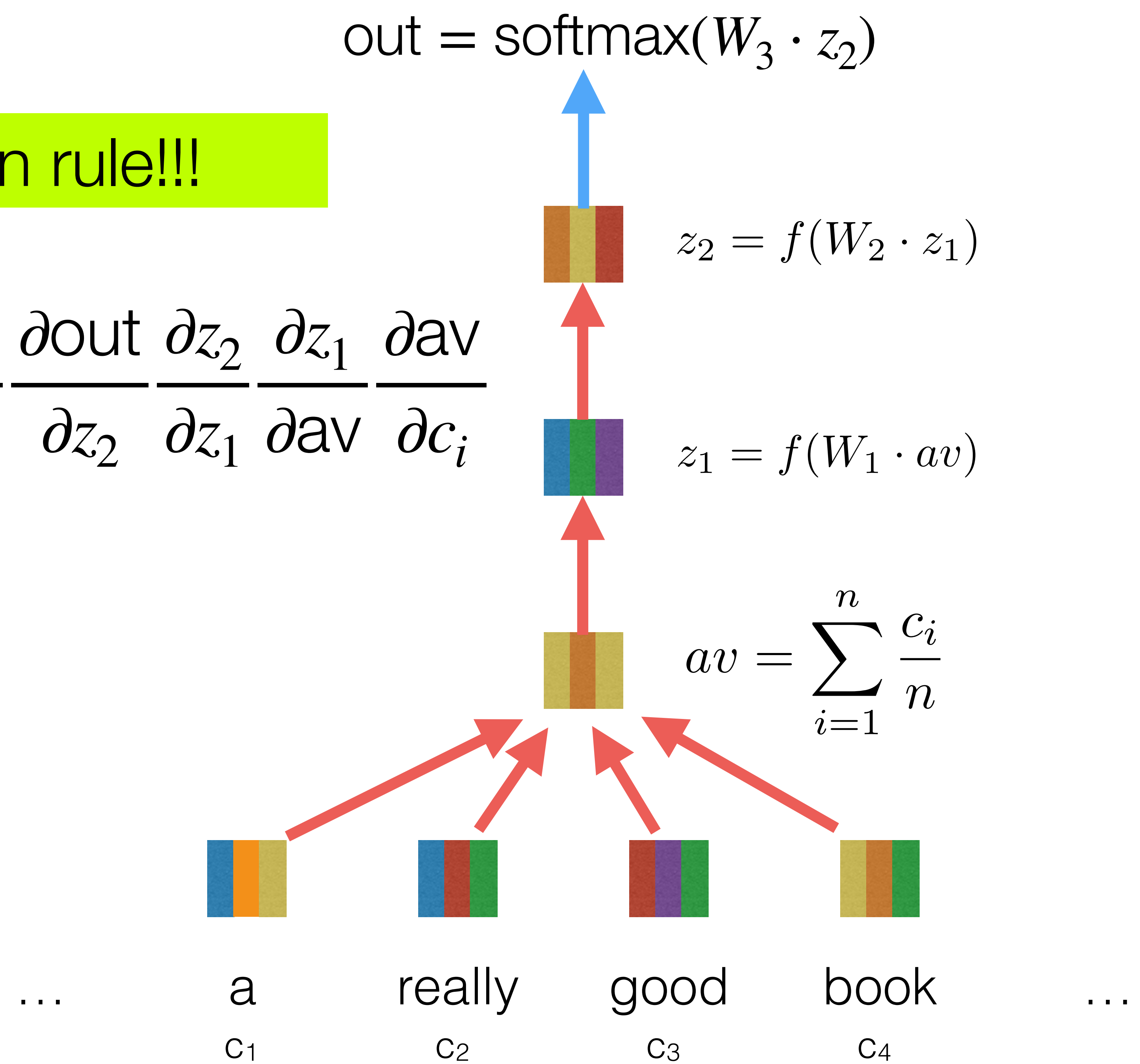
$$\frac{\partial L}{\partial c_i} = ???$$



deep averaging networks

chain rule!!!

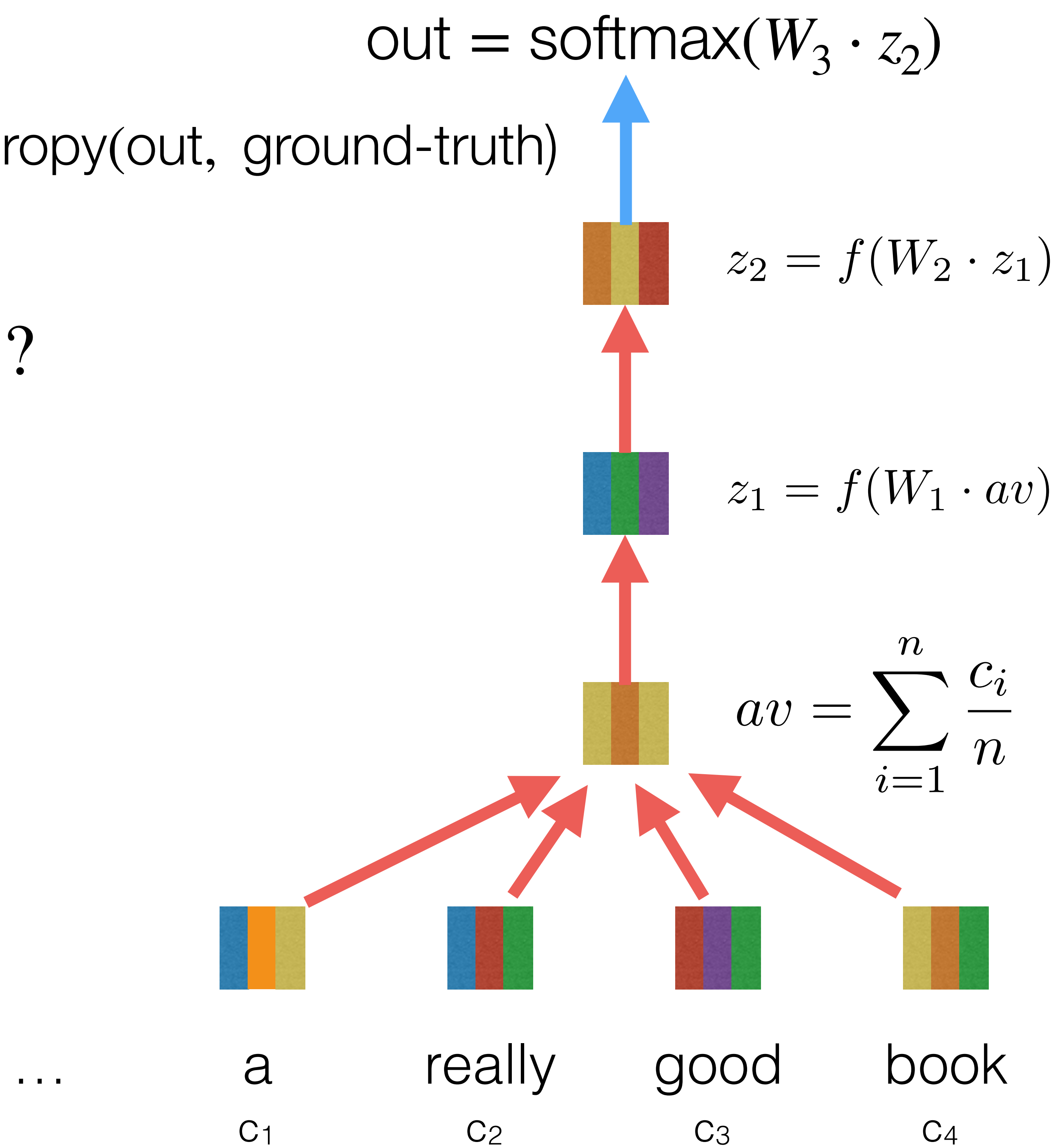
$$\frac{\partial L}{\partial c_i} = \frac{\partial L}{\partial \text{out}} \frac{\partial \text{out}}{\partial z_2} \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial \text{av}} \frac{\partial \text{av}}{\partial c_i}$$



deep averaging networks

$L = \text{cross-entropy(out, ground-truth)}$

$$\frac{\partial L}{\partial W_2} = ???$$

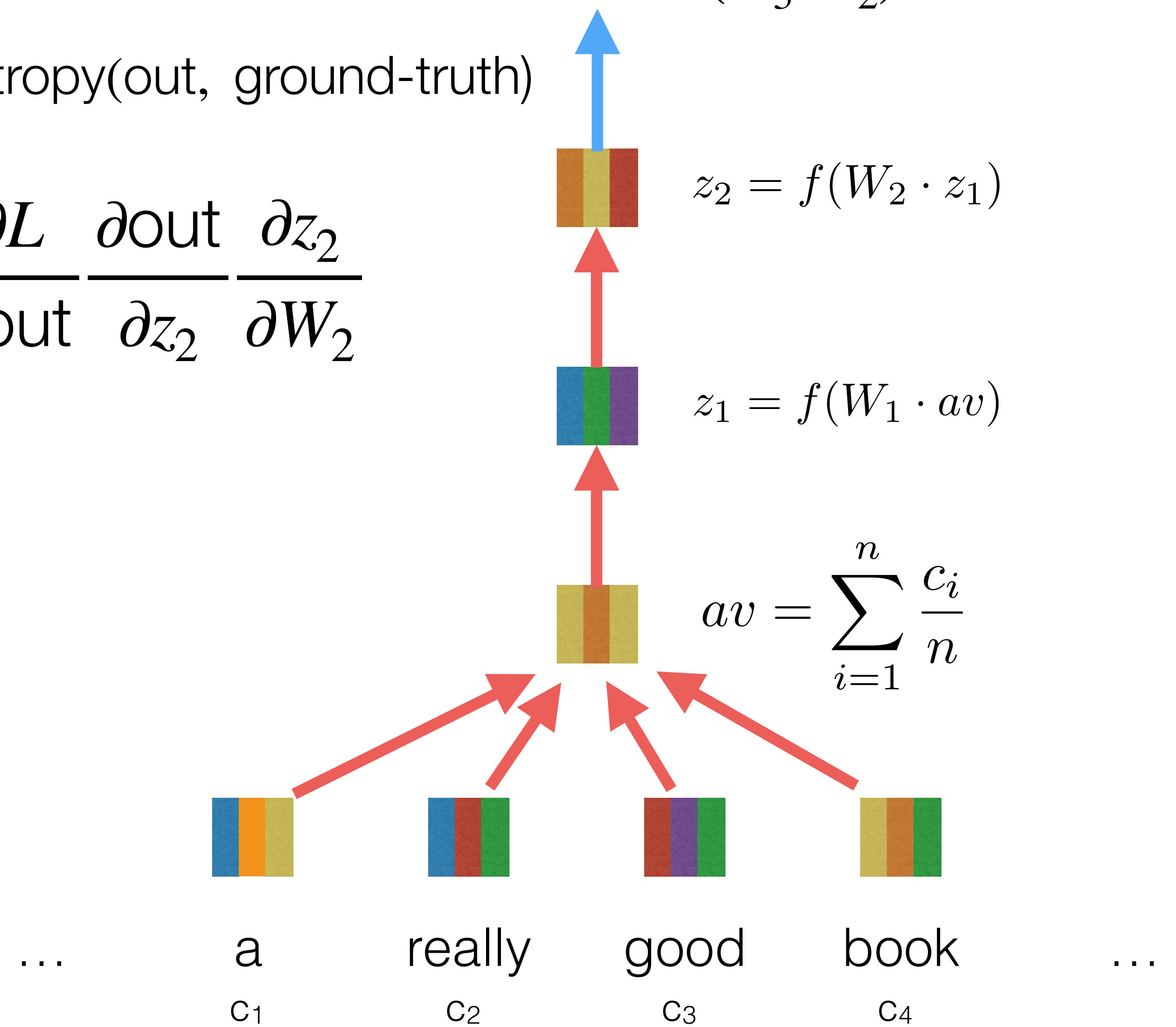


deep averaging networks

$$\text{out} = \text{softmax}(W_3 \cdot z_2)$$

$$L = \text{cross-entropy}(\text{out}, \text{ ground-truth})$$

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \text{out}} \frac{\partial \text{out}}{\partial z_2} \frac{\partial z_2}{\partial W_2}$$



backpropagation

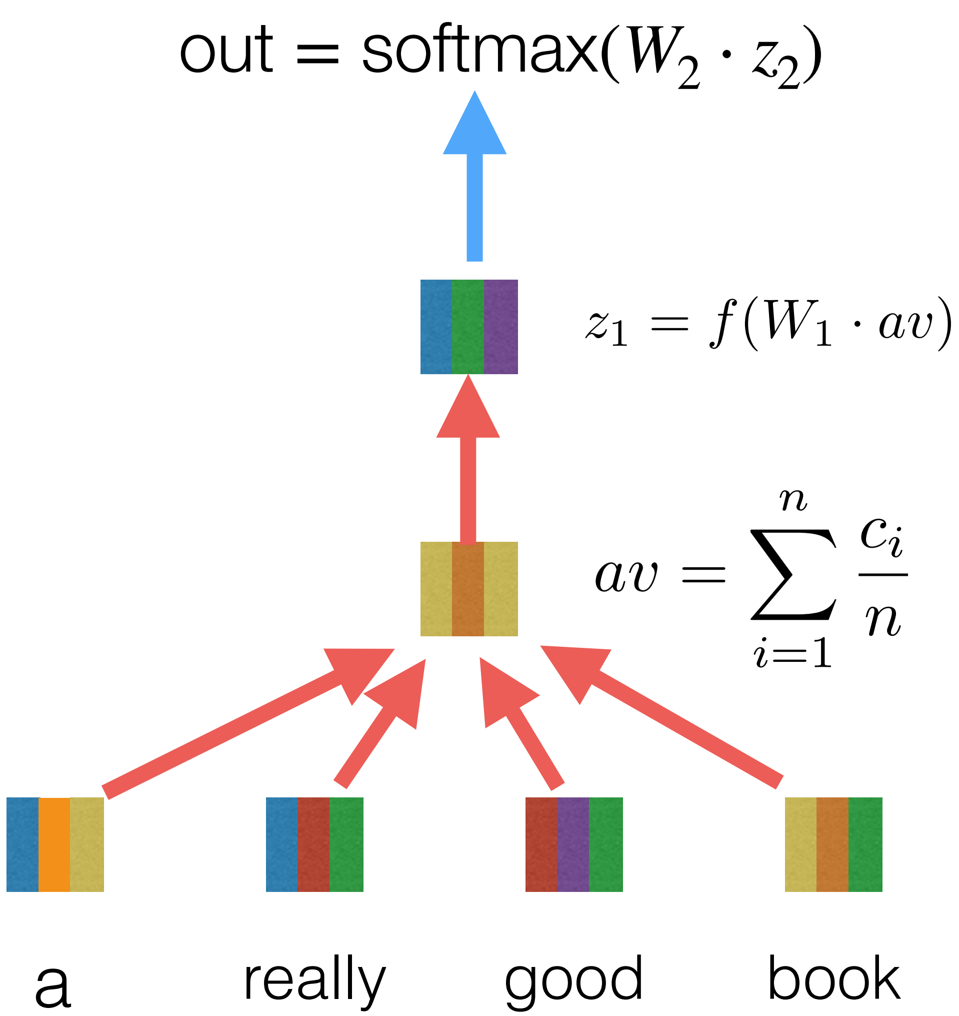
- use the chain rule to compute partial derivatives w/ respect to each parameter
- trick: re-use derivatives computed for higher layers to compute derivatives for lower layers!

$$\frac{\partial L}{\partial c_i} = \frac{\partial L}{\partial \text{out}} \frac{\partial \text{out}}{\partial z_2} \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial \text{av}} \frac{\partial \text{av}}{\partial c_i}$$

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \text{out}} \frac{\partial \text{out}}{\partial z_2} \frac{\partial z_2}{\partial W_2}$$

Rumelhart et al., 1986

deep learning frameworks make building NNs super easy!

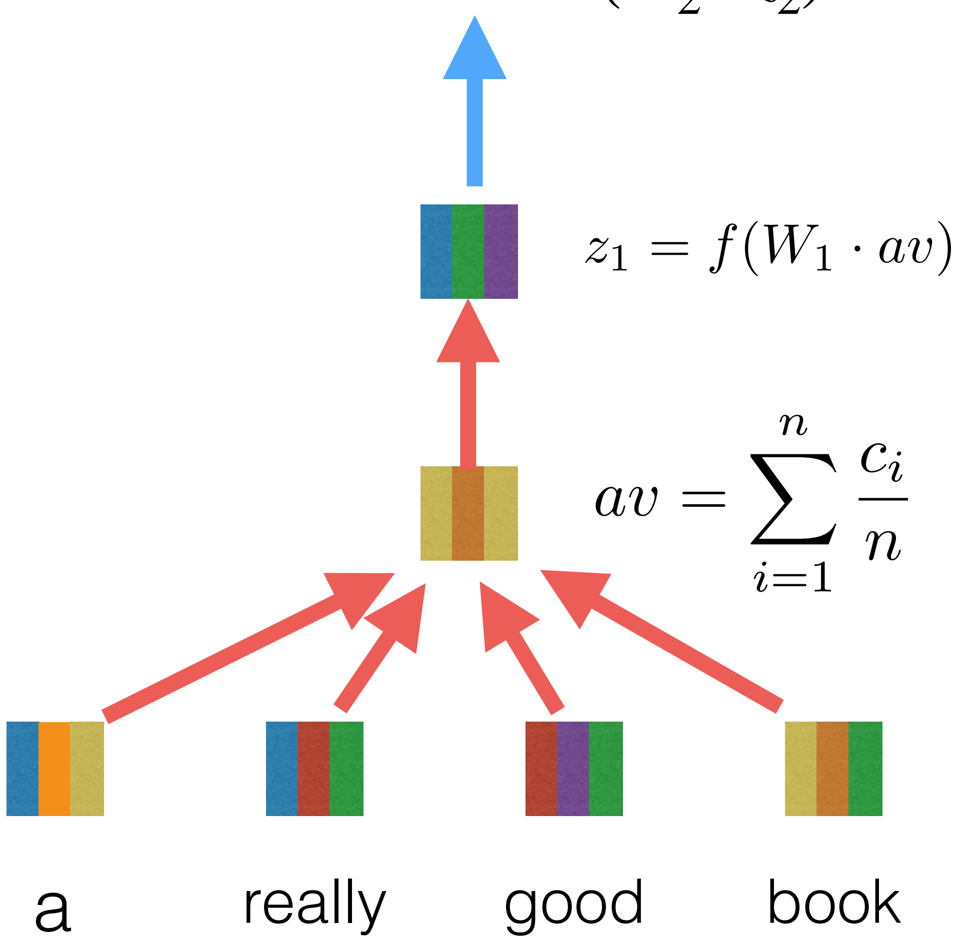


<u>set up the network</u>

```
def __init__(self, n_classes, vocab_size, emb_dim=300,
             n_hidden_units=300):
    super(DanModel, self).__init__()
    self.n_classes = n_classes
    self.vocab_size = vocab_size
    self.emb_dim = emb_dim
    self.n_hidden_units = n_hidden_units
    self.embeddings = nn.Embedding(self.vocab_size,
                                   self.emb_dim)
    self.classifier = nn.Sequential(
           nn.Linear(self.n_hidden_units,
                     self.n_hidden_units),
           nn.ReLU(),
           nn.Linear(self.n_hidden_units,
                     self.n_classes))
    self._softmax = nn.Softmax()
```

deep learning frameworks make building NNs super easy!

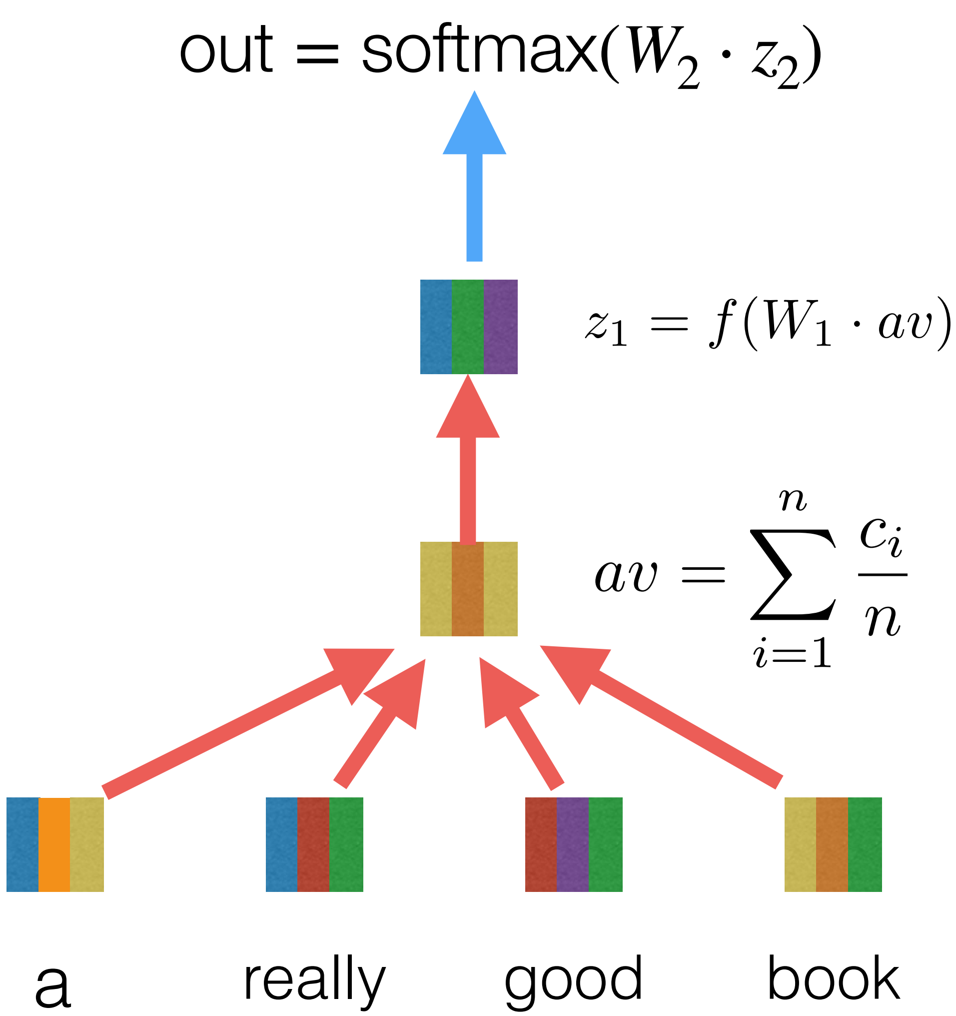
$\text{out} = \text{softmax}(W_2 \cdot z_2)$



<u>do a forward pass to compute prediction</u>

```
def forward(self, batch, probs=False):
    text = batch['text']['tokens']
    length = batch['length']
    text_embed = self._word_embeddings(text)
    # Take the mean embedding. Since padding results
    # in zeros its safe to sum and divide by length
    encoded = text_embed.sum(1)
    encoded /= lengths.view(text_embed.size(0), -1)

    # Compute the network score predictions
    logits = self.classifier(encoded)
    if probs:
        return self._softmax(logits)
    else:
        return logits
```

deep learning frameworks make building NNs super easy!

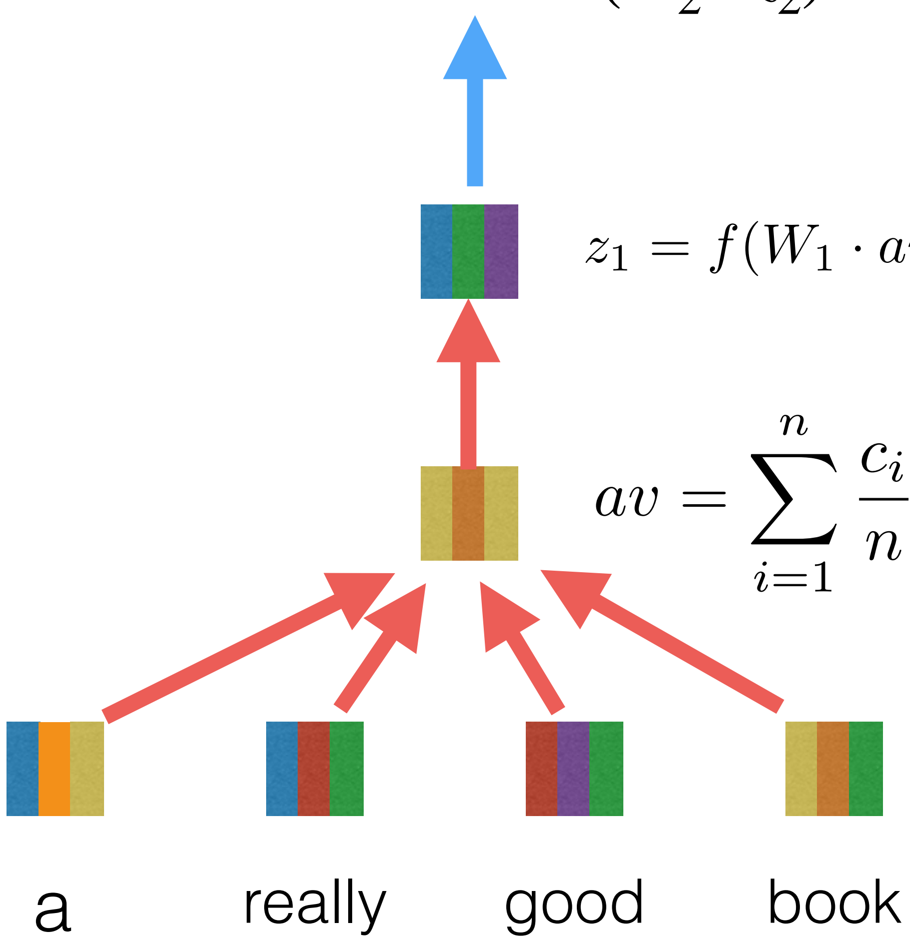


do a backward pass to update weights

```
def _run_epoch(self, batch_iter, train=True):
    self._model.train()
    for batch in batch_iter:
        model.zero_grad()
        out = model(batches)
        batch_loss = criterion(out,
                               batch['label'])
        batch_loss.backward()
        self.optimizer.step()
```

deep learning frameworks make building NNs super easy!

$\text{out} = \text{softmax}(W_2 \cdot z_2)$



$z_1 = f(W_1 \cdot av)$

$$av = \sum_{i=1}^{n} \frac{c_i}{n}$$

do a backward pass to update weights

```
def _run_epoch(self, batch_iter, train=True):
    self._model.train()
    for batch in batch_iter:
        model.zero_grad()
        out = model(batches)
        batch_loss = criterion(out,
                               batch['label'])
        batch_loss.backward()
        self.optimizer.step()
```

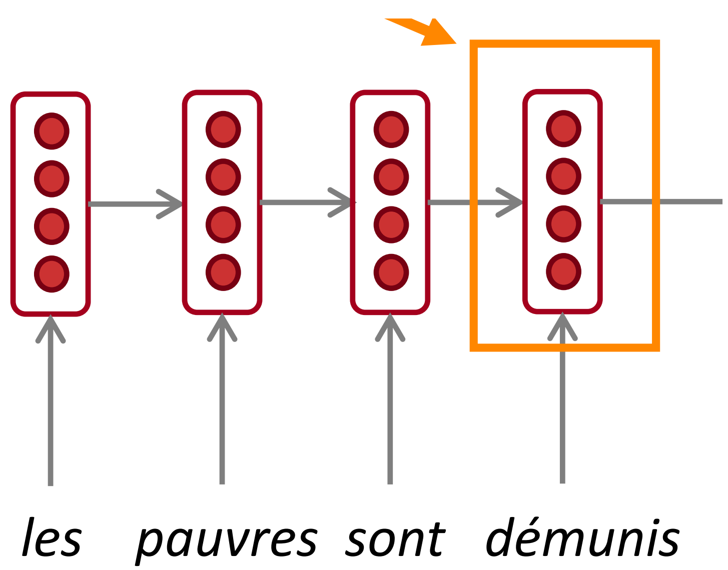
that’s it! no need to compute gradients by hand!

NN architectures

- We need neural network models that can process token-by-token
- Major components
  - Recurrent neural networks (RNNs)
  - Attention mechanism (softmax over tokens)
  - Self-attention ("Transformers"; next lecture)
    - This is basically the entire model behind BERT and GPT, the best general-purpose NN NLP models today!

Recurrent neural networks

- Idea: beyond the original word embedding, every token has its own hidden state vector h_t, influenced by the previous state!



- Many applications
  - Next word prediction
  - Text classification
  - Translation...

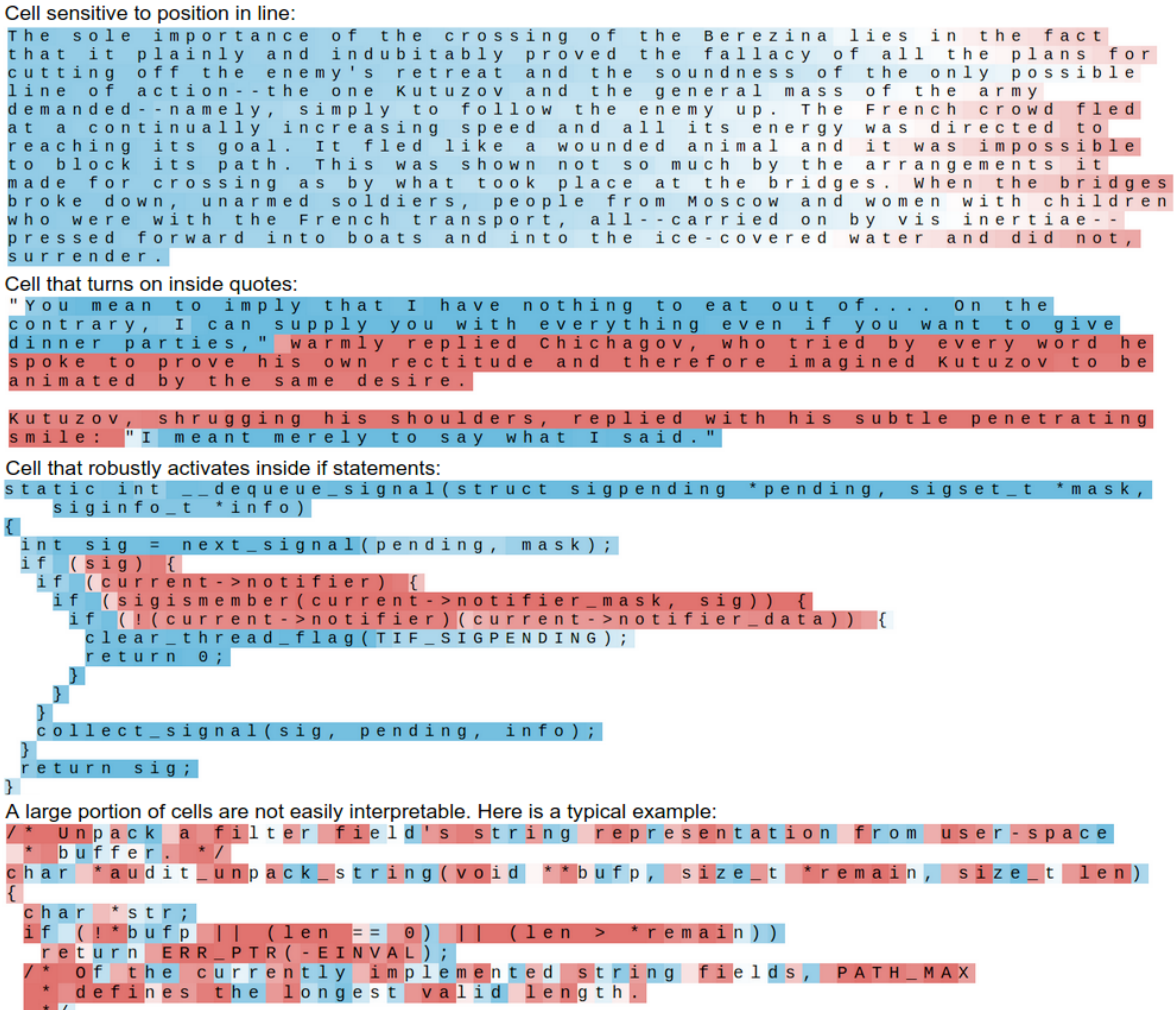
Character LMs comparison: RNN (LSTM) vs. N-Gram

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.
```

```
First Citizen:
Nay, then, that was hers,
It speaks against your other service:
But since the
youth of the circumstance be spoken:
Your uncle and one Baptista's daughter.

SEBASTIAN:
Do I stand till the break off.

BIRON:
Hide thy head.

VENTIDIUS:
He purposeth to Athens: whither, with the vow
I made to handle you.
```

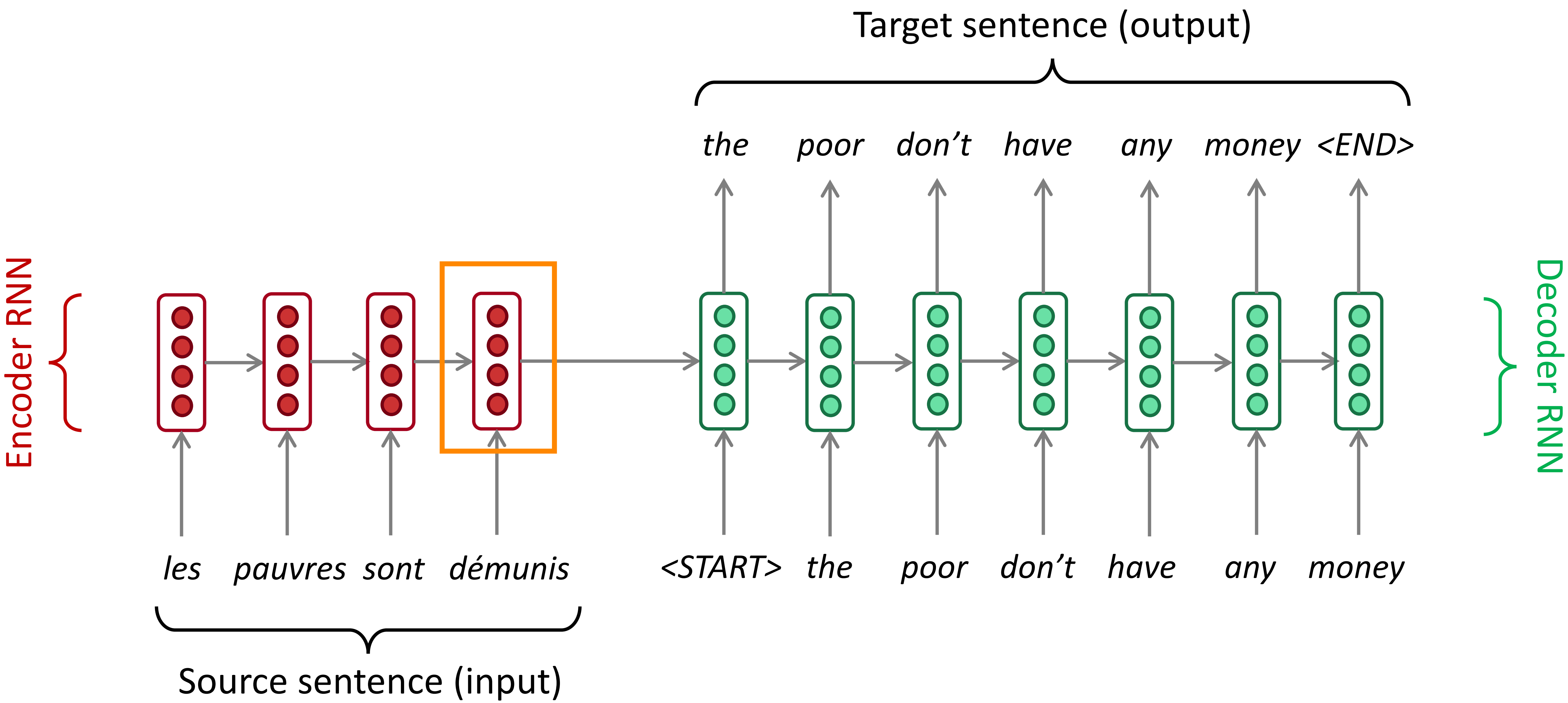
http://karpathy.github.io/2015/05/21/rnn-effectiveness/
http://nbviewer.jupyter.org/gist/yoavg/d76121dfde2618422139

Structure awareness (one particular RNN hidden state)

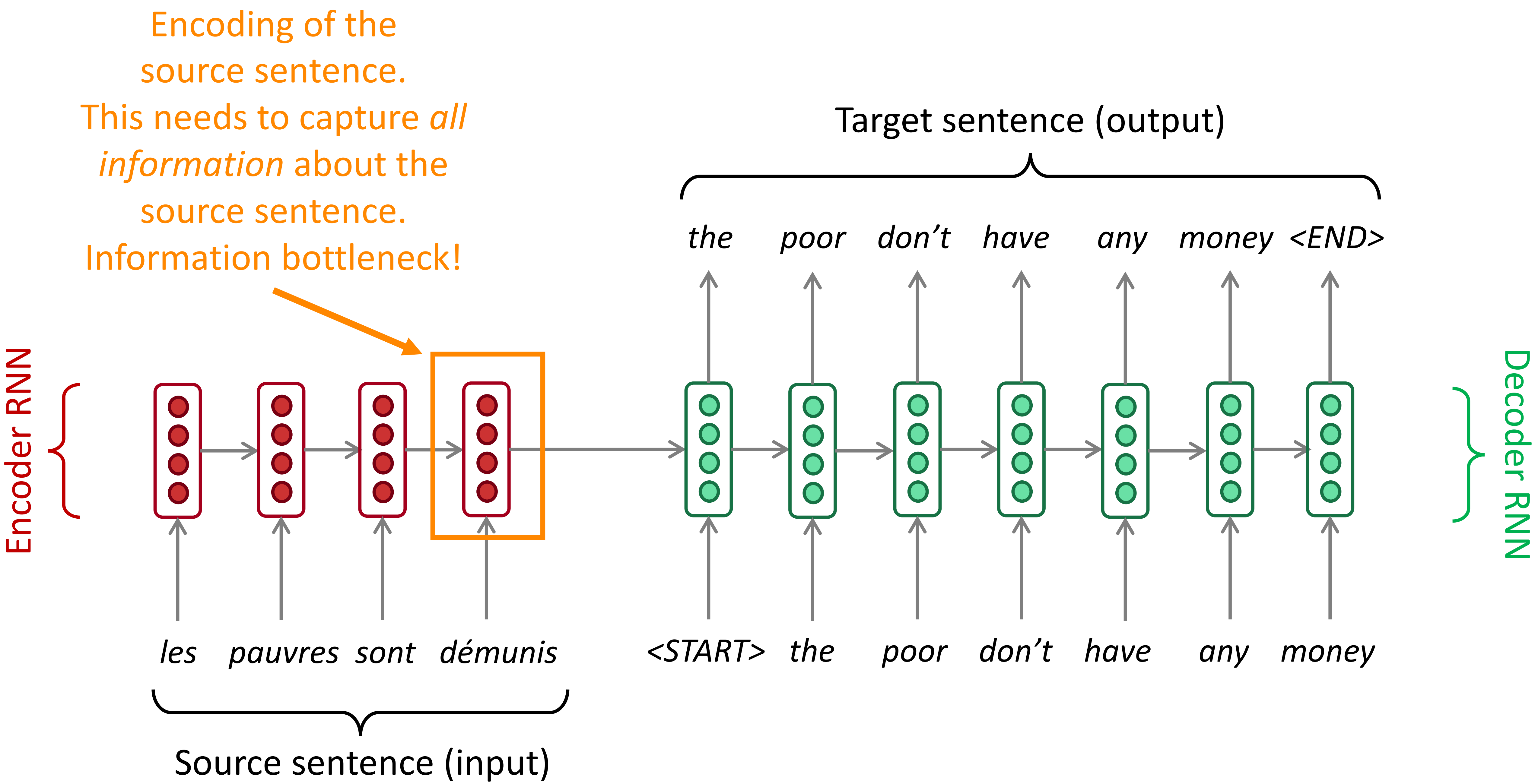


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Sequence-to-sequence: the bottleneck problem



Sequence-to-sequence: the bottleneck problem



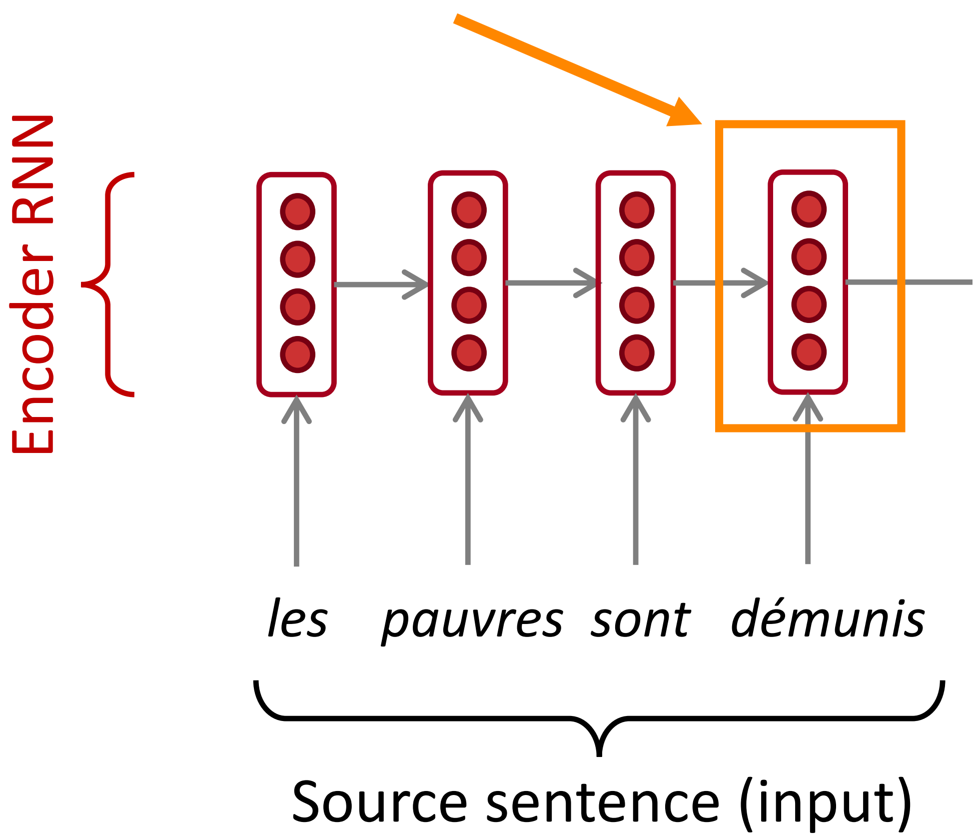
- stopped here

"you can't cram the meaning of a whole %&@#&ing sentence into a single $*(&@ing vector!"

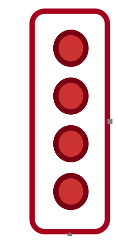
— Ray Mooney (famous NLP professor at UT Austin)

idea: what if we use multiple vectors?

Encoding of the source sentence. This needs to capture *all information* about the source sentence. Information bottleneck!

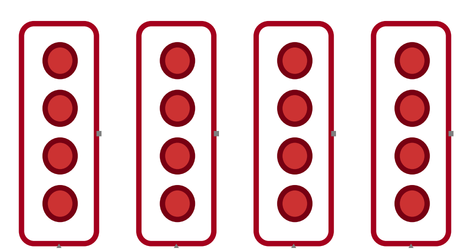


Instead of:

les pauvres sont démunis = 

Let's try:

les pauvres sont démunis =

 (all 4 hidden states!)

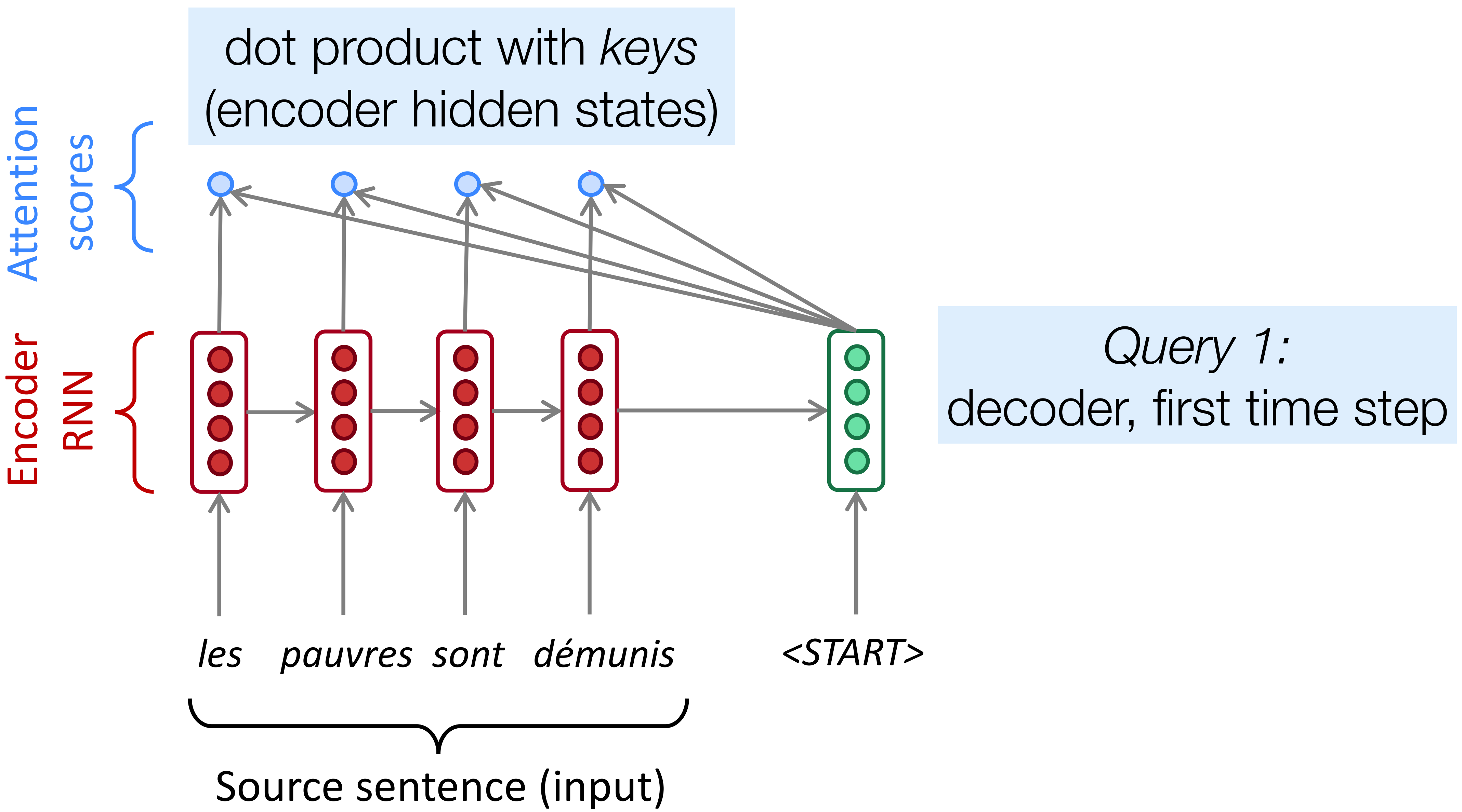
The solution: **attention**

- **Attention mechanisms** (Bahdanau et al., 2015) allow the decoder to focus on a particular part of the source sequence at each time step
  - Conceptually similar to *word alignments*

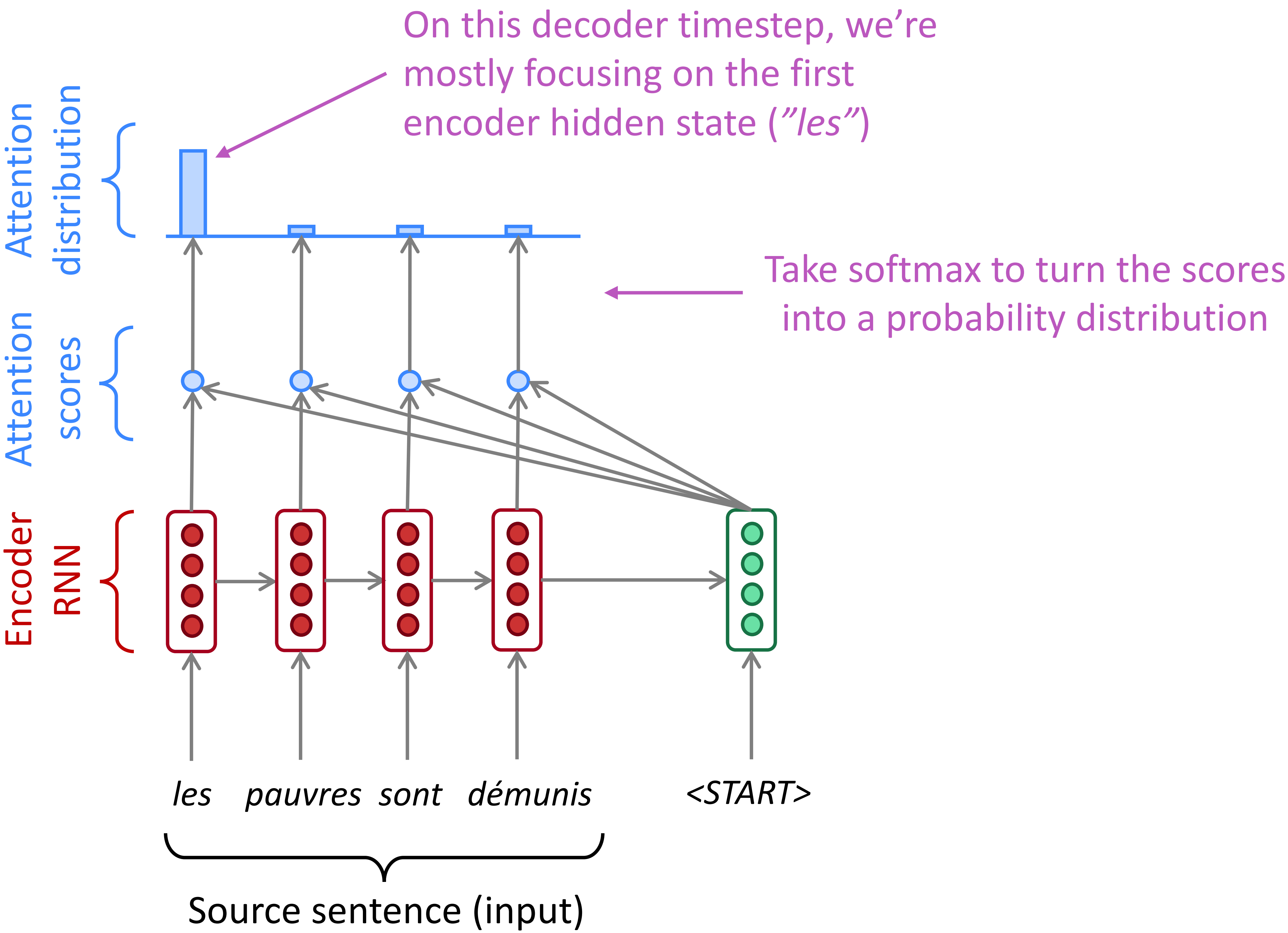
How does it work?

- in general, we have a single *query* vector and multiple *key* vectors. We want to score each query-key pair
  - Attention score based on query-key similarity
  - New representation = attention softmax-weighted average of token embeddings

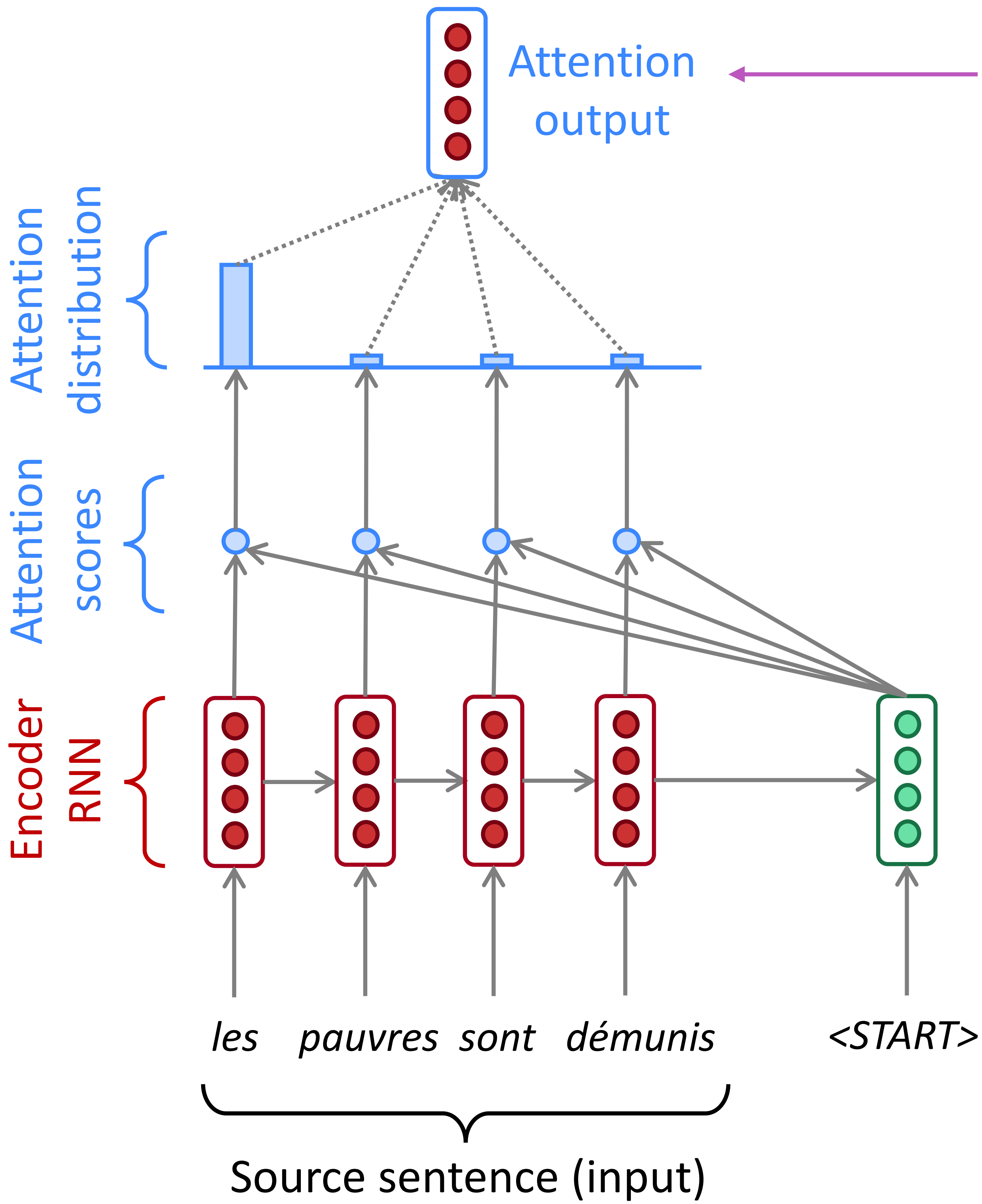
Sequence-to-sequence with attention



Sequence-to-sequence with attention



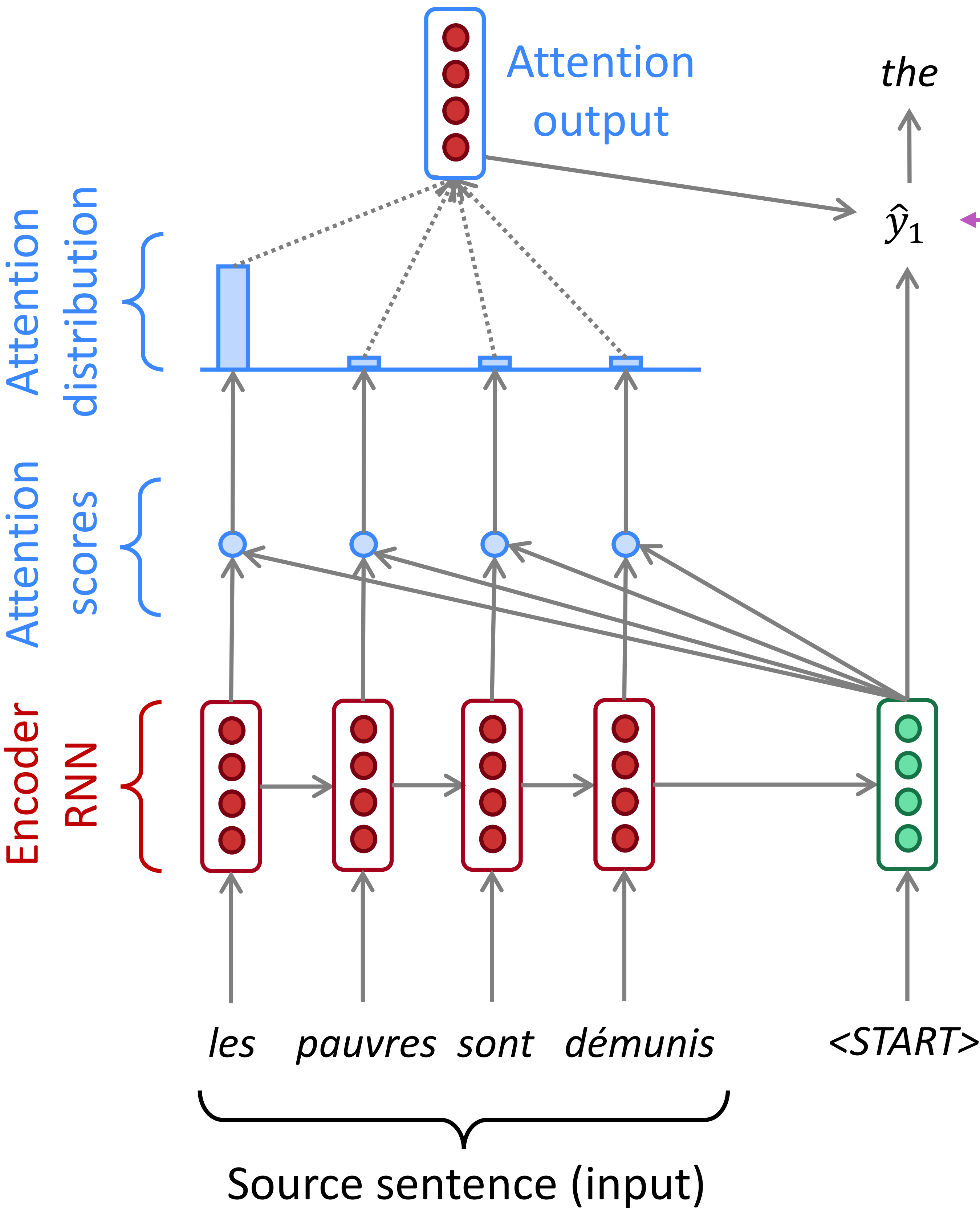
Sequence-to-sequence with attention



Use the attention distribution to take a **weighted sum** of the encoder hidden states.

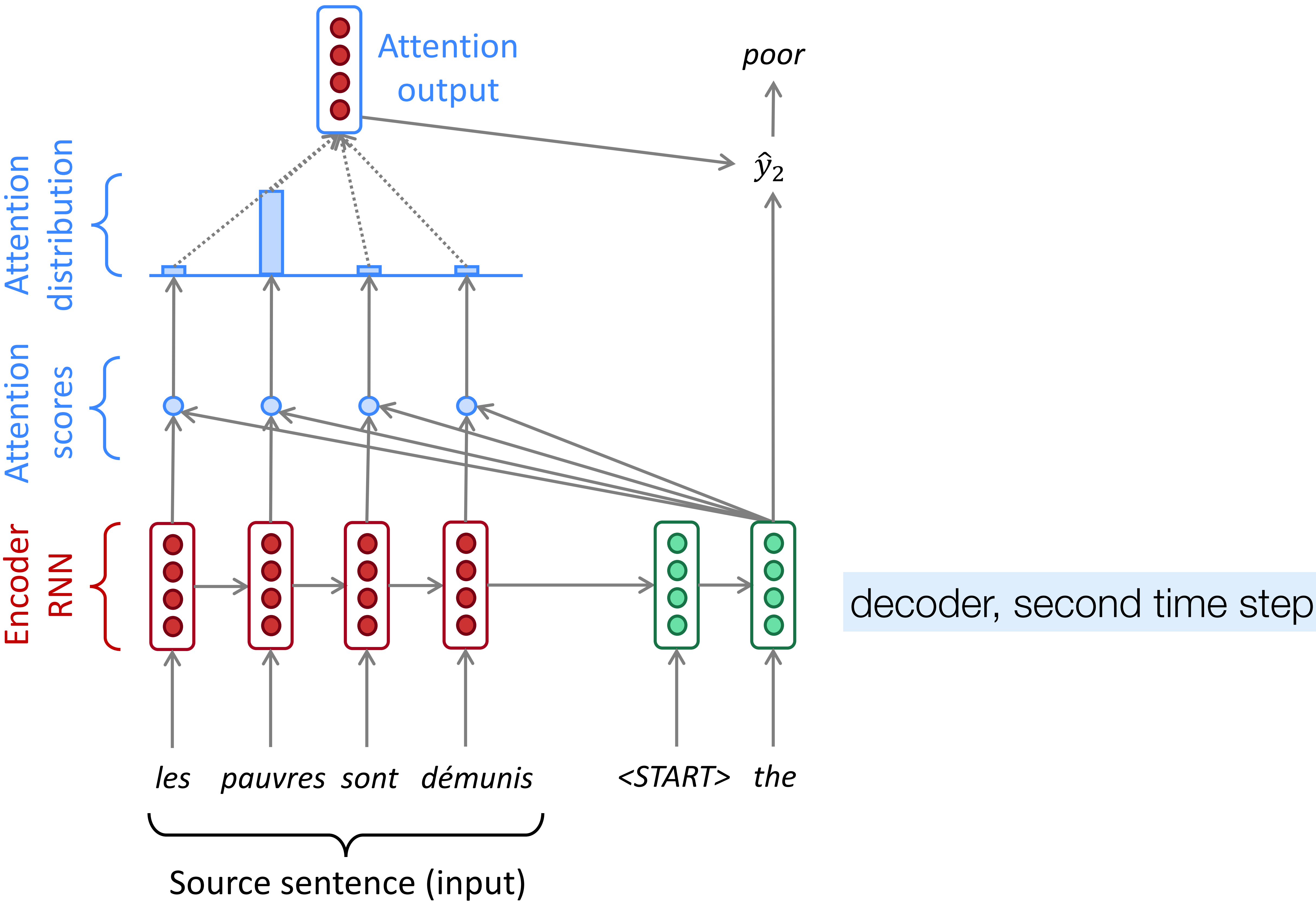
The attention output mostly contains information the hidden states that received high attention.

Sequence-to-sequence with attention



Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

Sequence-to-sequence with attention

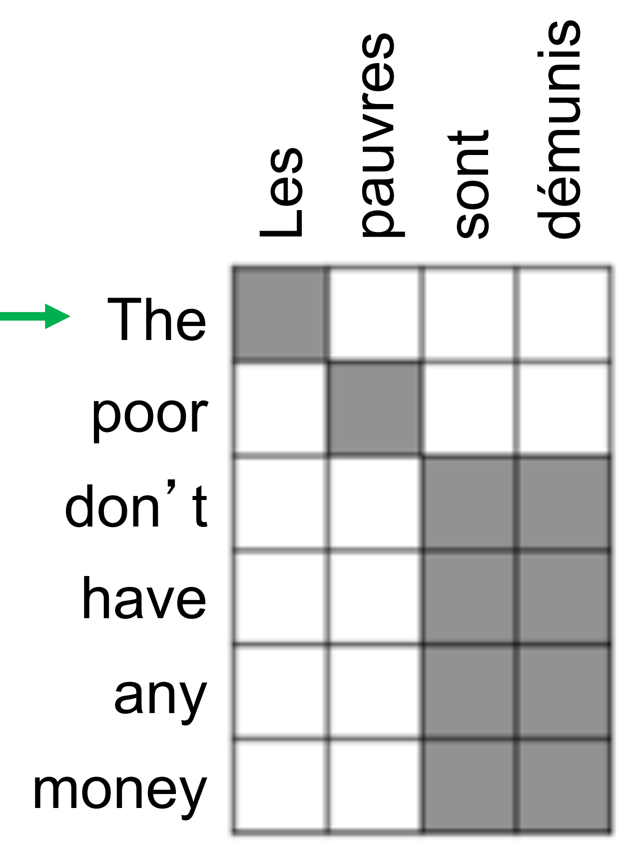


Many variants of attention

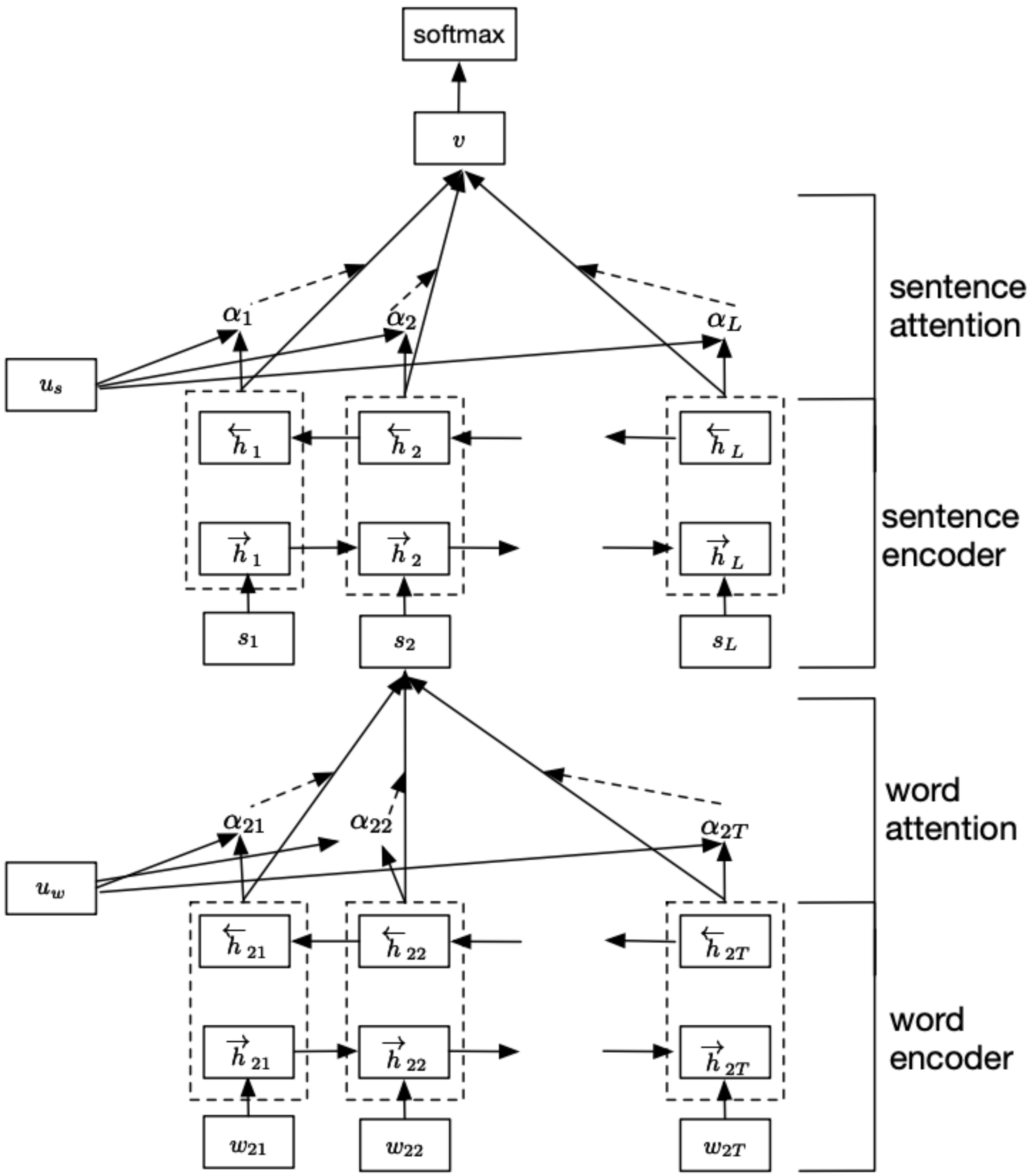
- Original formulation: $a(\mathbf{q}, \mathbf{k}) = w_2^T \tanh(W_1[\mathbf{q}; \mathbf{k}])$
- Bilinear product: $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T W \mathbf{k}$ Luong et al., 2015
- Dot product: $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T \mathbf{k}$ Luong et al., 2015
- Scaled dot product: $a(\mathbf{q}, \mathbf{k}) = \dfrac{\mathbf{q}^T \mathbf{k}}{\sqrt{|\mathbf{k}|}}$ Vaswani et al., 2017

Attention is great

- Attention significantly improves NMT performance
  - It’s very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself



Hierarchical attention



pork belly = delicious . || scallops? || I don't even like scallops, and these were a-m-a-z-i-n-g . || fun and tasty cocktails. || next time I in Phoenix, I will go back here. || Highly recommend.

Figure 1: A simple example review from Yelp 2013 that consists of five sentences, delimited by period, question mark. The first and third sentence delivers stronger meaning and inside, the word *delicious, a-m-a-z-i-n-g* contributes the most in defining sentiment of the two sentences.

- Yang et al., 2016: hierarchical attention for document classification