Neural Attention

Yoav Goldberg

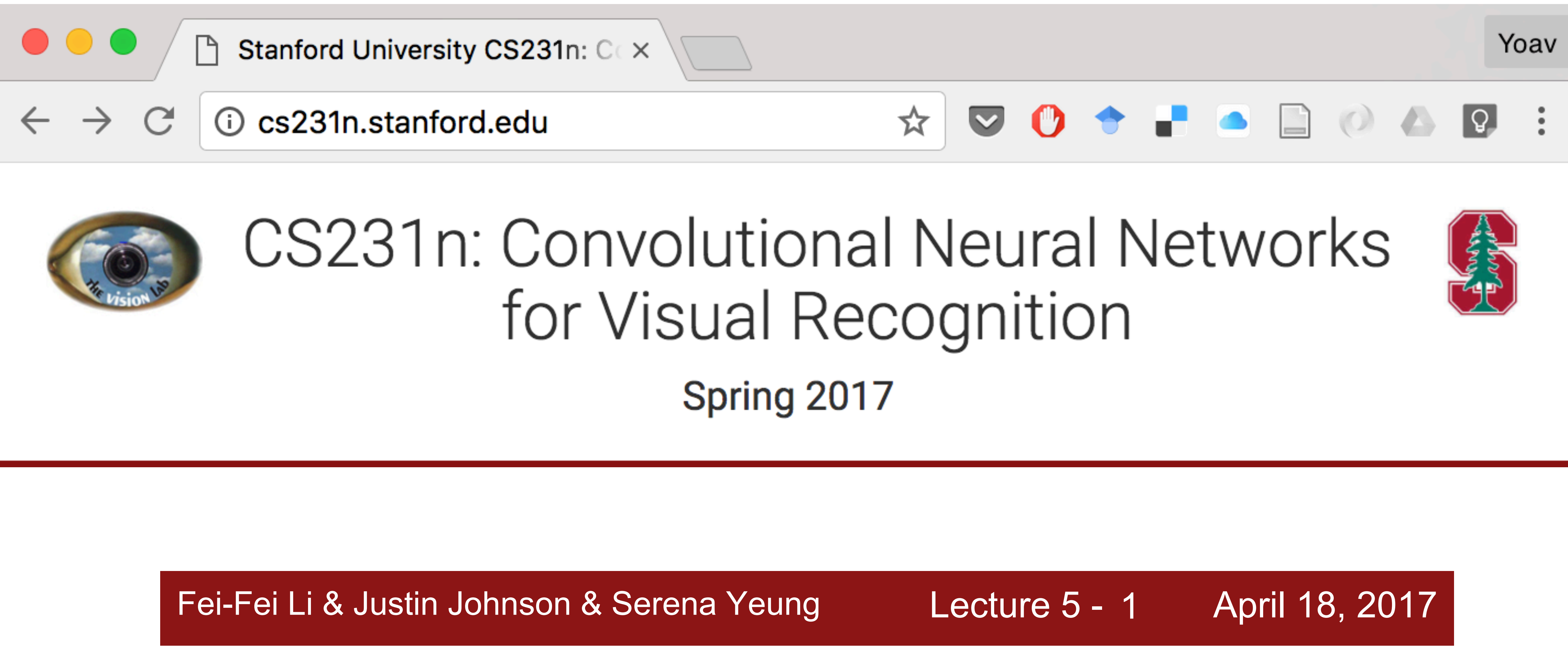
Before we start, some things to consider

- What makes a model slow?
- Which operations take more time?
- Which operations will be faster on GPU / CPU?
- Should I use CPU or GPU for this problem?
- Can I find an efficient parallel implementation for this architecture?
    - Data-parallel vs. Model-parallel

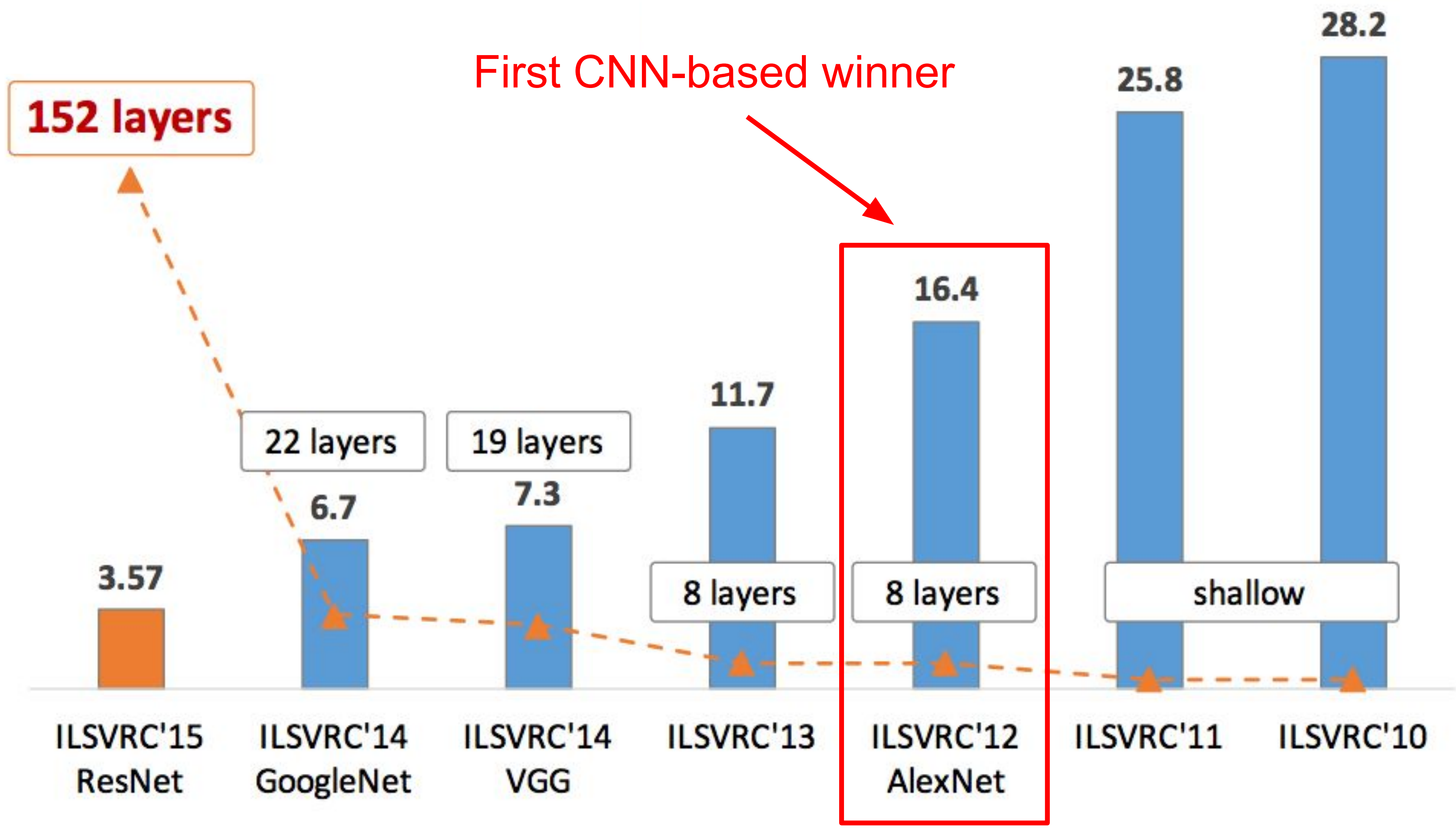
before we start:

Residual Connections

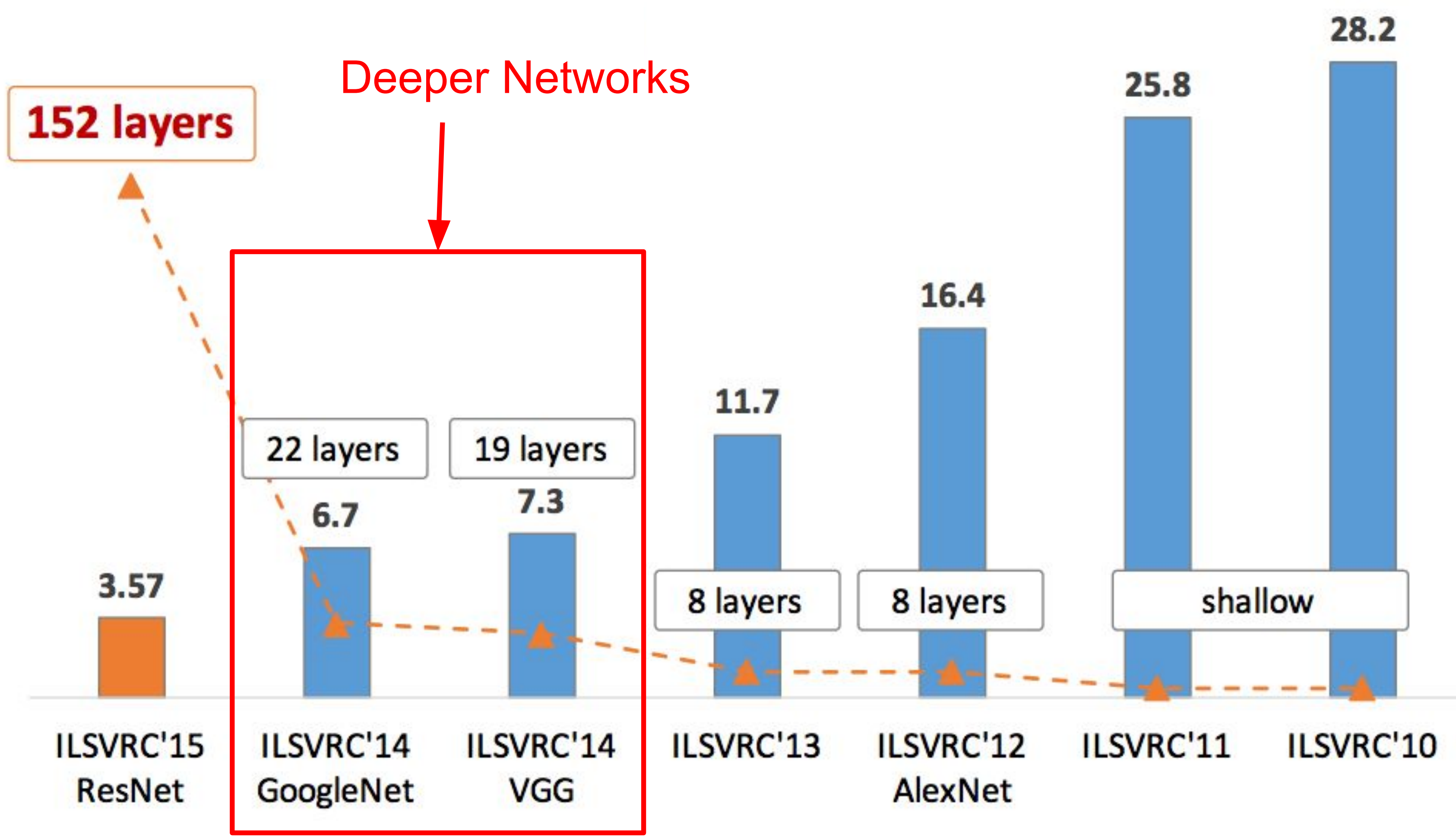
Neural Networks for Vision



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: VGGNet

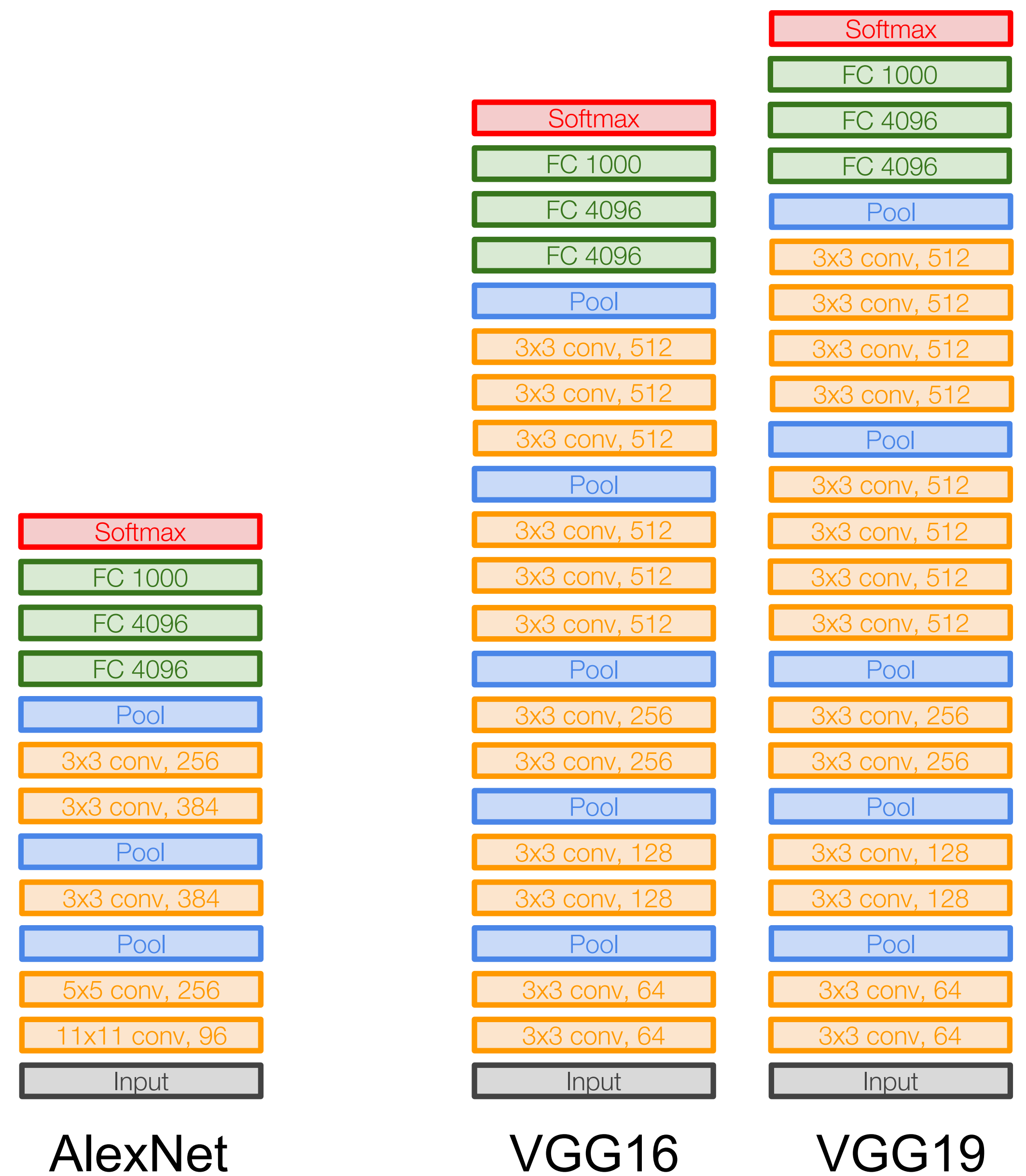
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

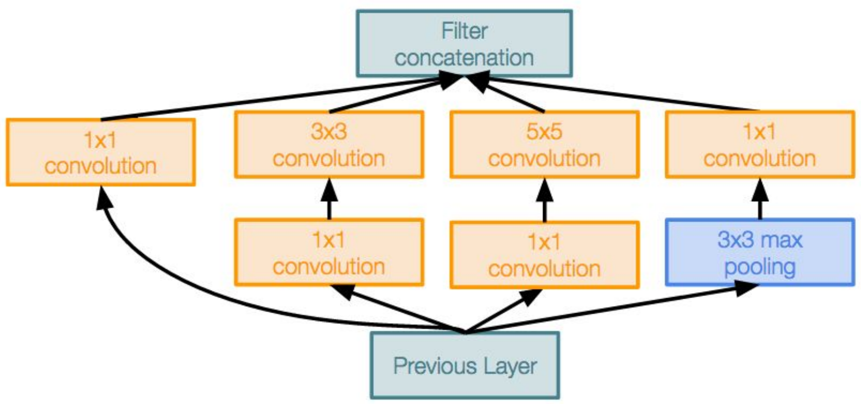
11.7% top 5 error in ILSVRC'13 (ZFNet)
-> 7.3% top 5 error in ILSVRC'14



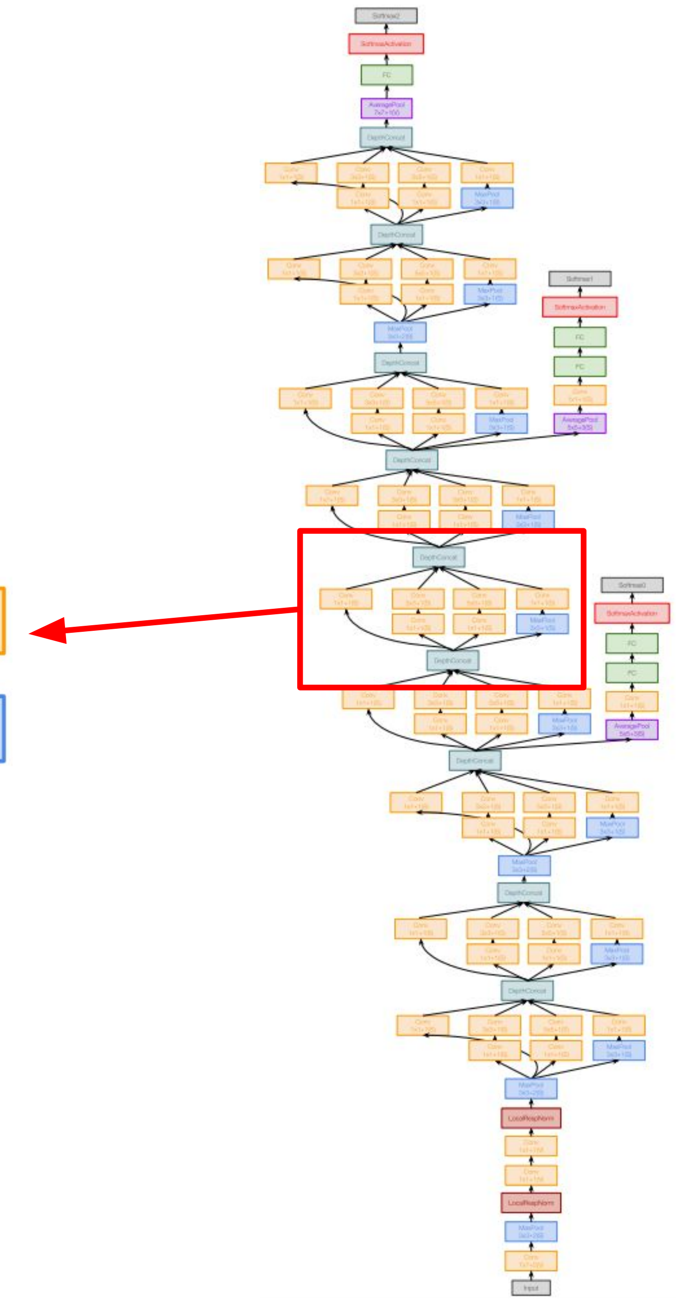
Case Study: GoogLeNet

[Szegedy et al., 2014]

"Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other



Inception module

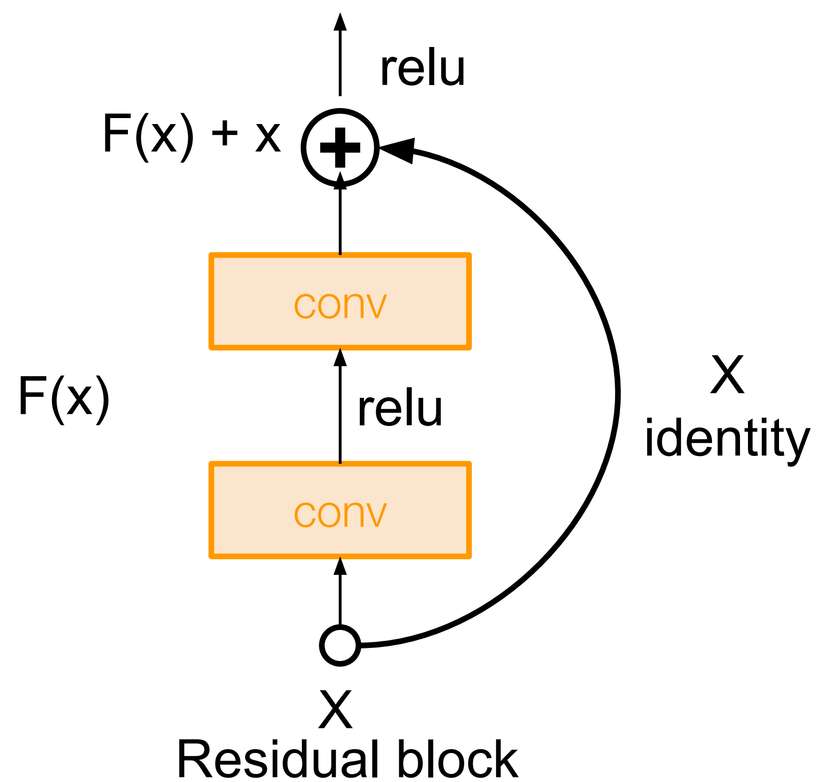


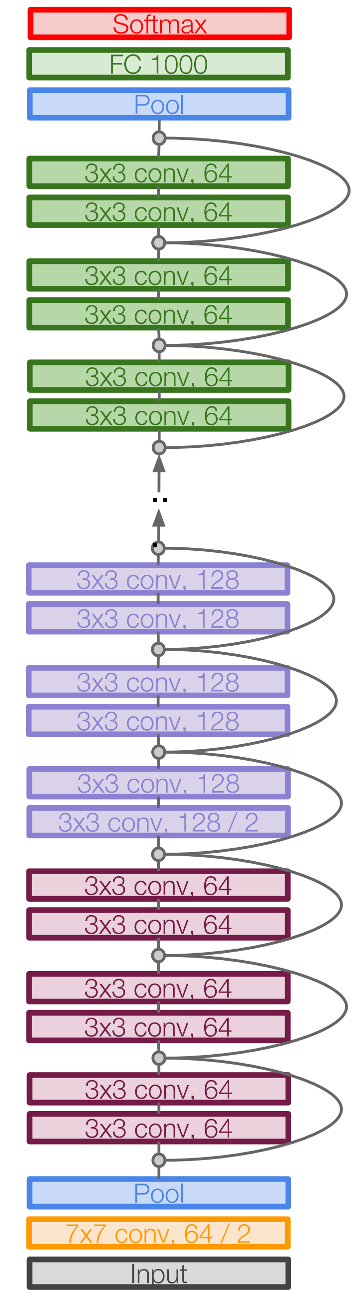
Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



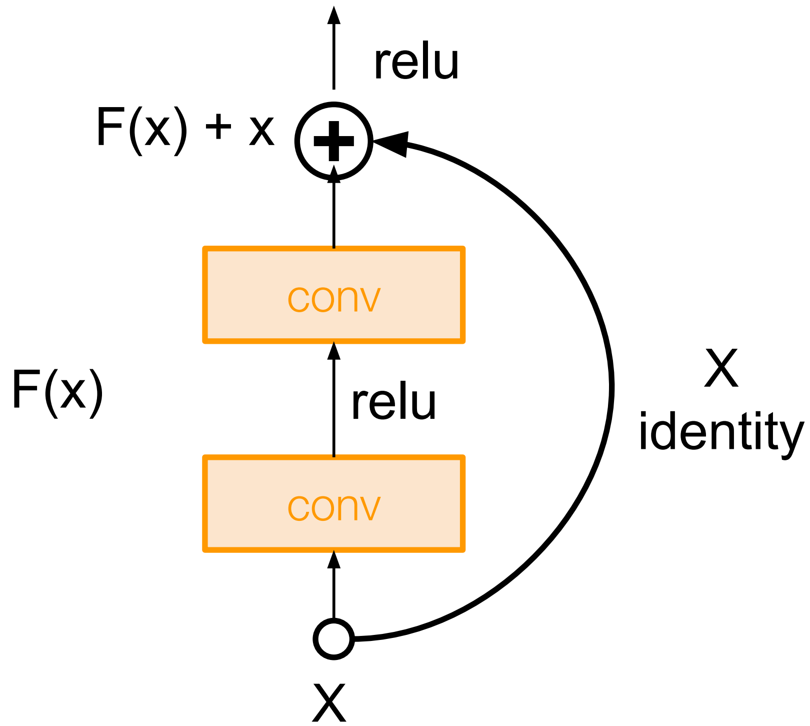


Case Study: ResNet

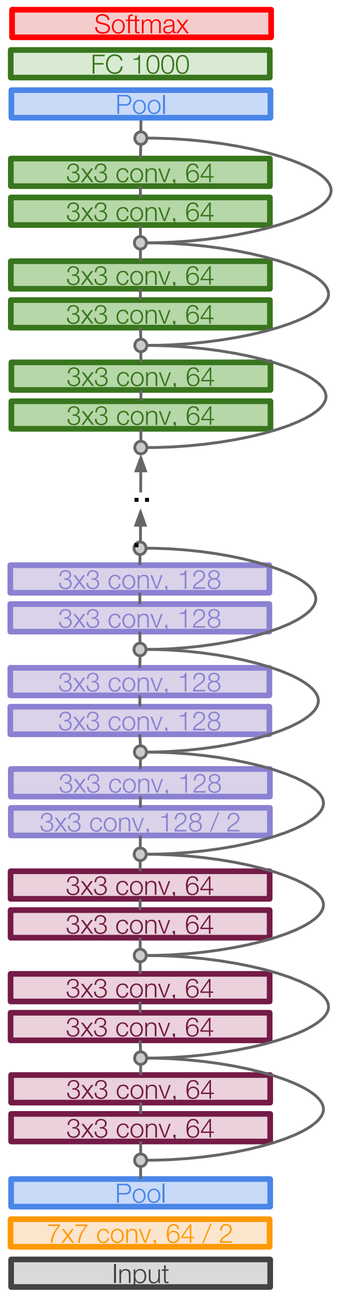
[He et al., 2015]

Very deep networks using residual connections

- $\boxed{\text{152-layer}}$ model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



Residual block



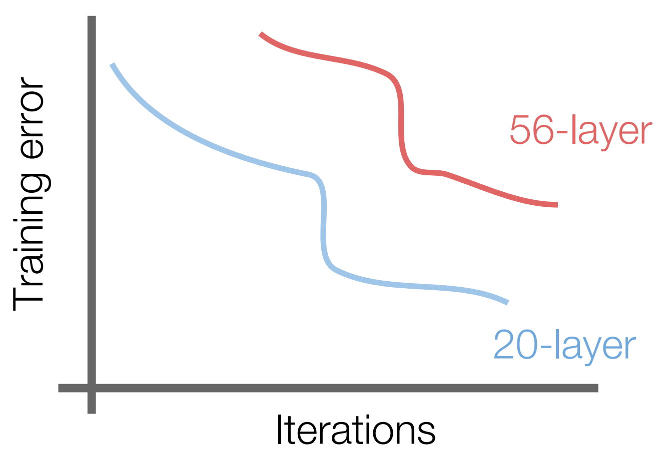
Deep Residual Learning for Image Recognition

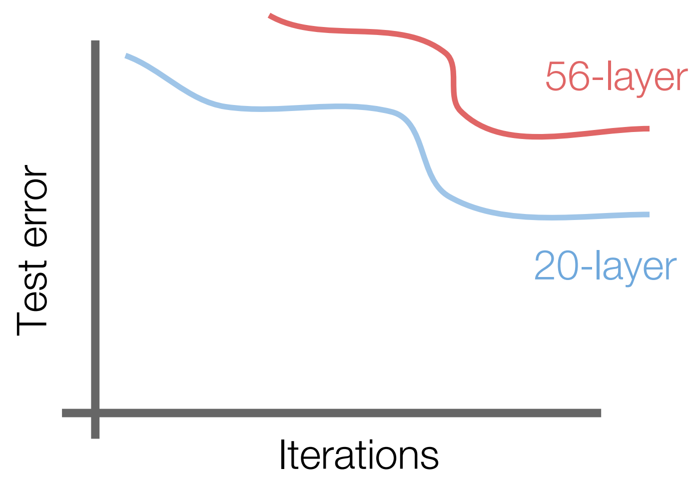
Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?





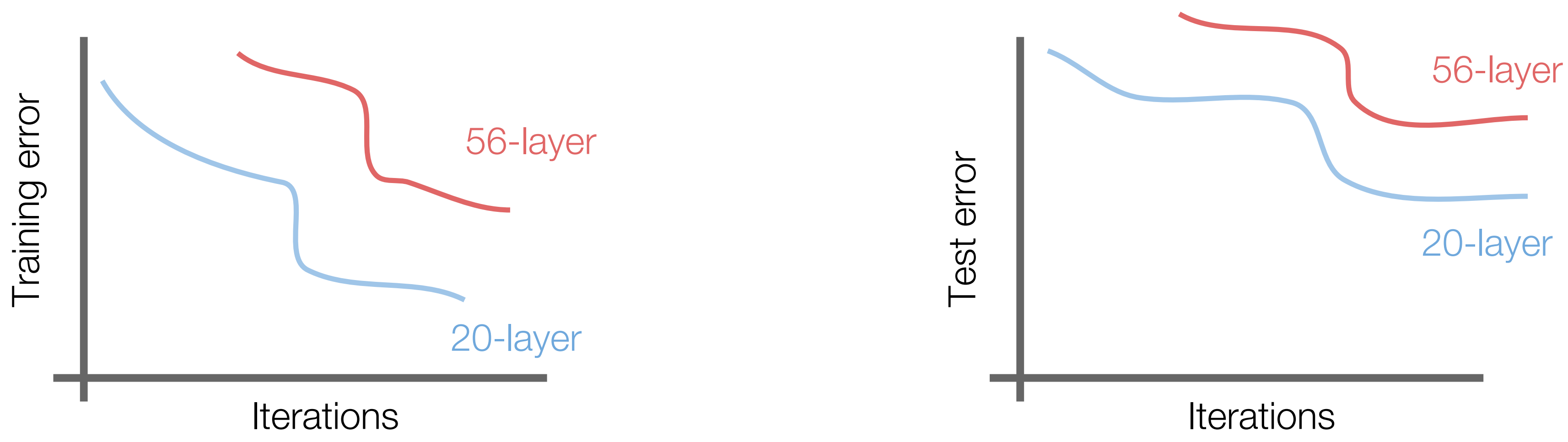
Q: What’s strange about these training and test curves?
[Hint: look at the order of the curves]

56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it's not caused by overfitting!

Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a "plain" convolutional neural network?



Q: What's strange about these training and test curves?
[Hint: look at the order of the curves]

Case Study: ResNet

[He et al., 2015]

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

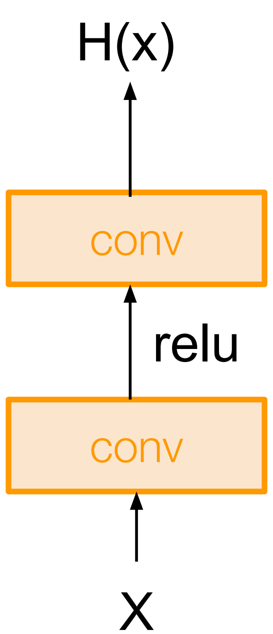
The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

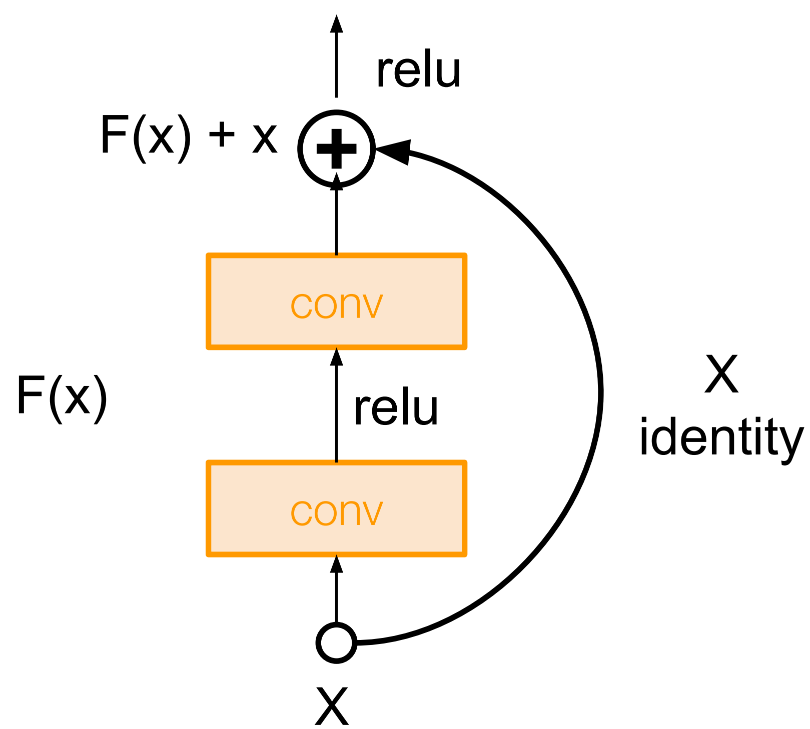
Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



"Plain" layers

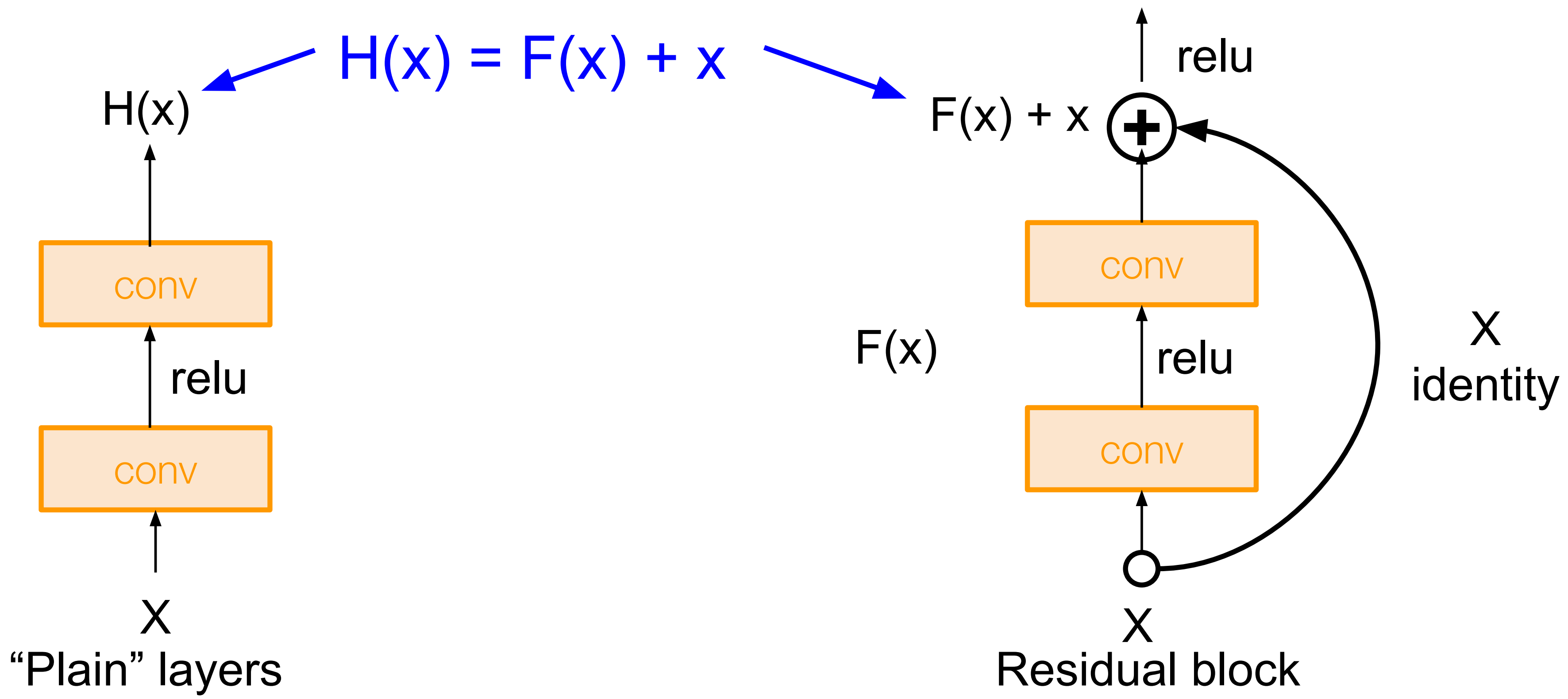


Residual block

Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



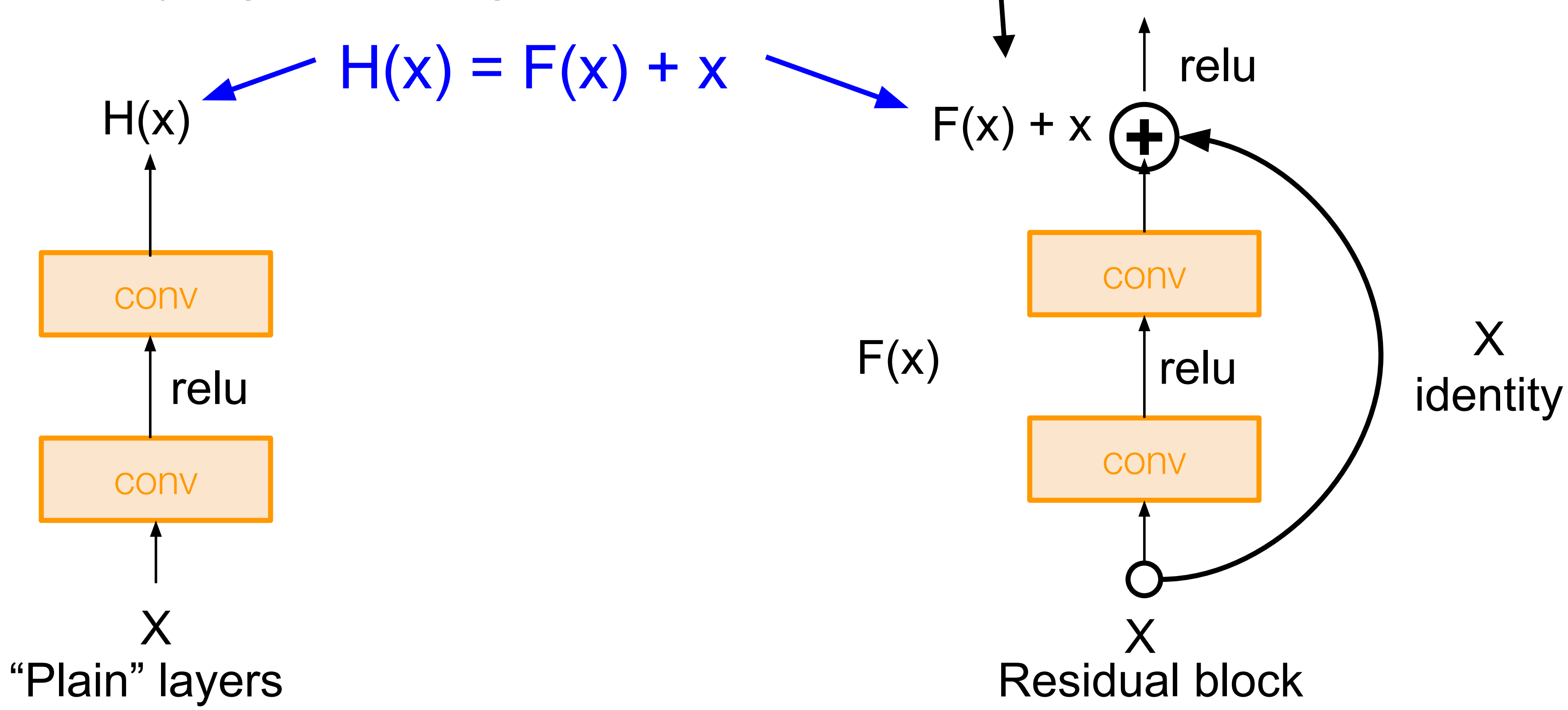
Use layers to fit residual $F(x) = H(x) - x$ instead of $H(x)$ directly

"copy x and add some change to it"

Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Use layers to fit residual $F(x) = H(x) - x$ instead of $H(x)$ directly

In this paper, we address the degradation problem by introducing a *deep residual learning* framework. Instead of hoping each few stacked layers directly fit a desired underlying mapping, we explicitly let these layers fit a residual mapping. Formally, denoting the desired underlying mapping as $\mathcal{H}(\mathbf{x})$, we let the stacked nonlinear layers fit another mapping of $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$. The original mapping is recast into $\mathcal{F}(\mathbf{x}) + \mathbf{x}$. We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers.

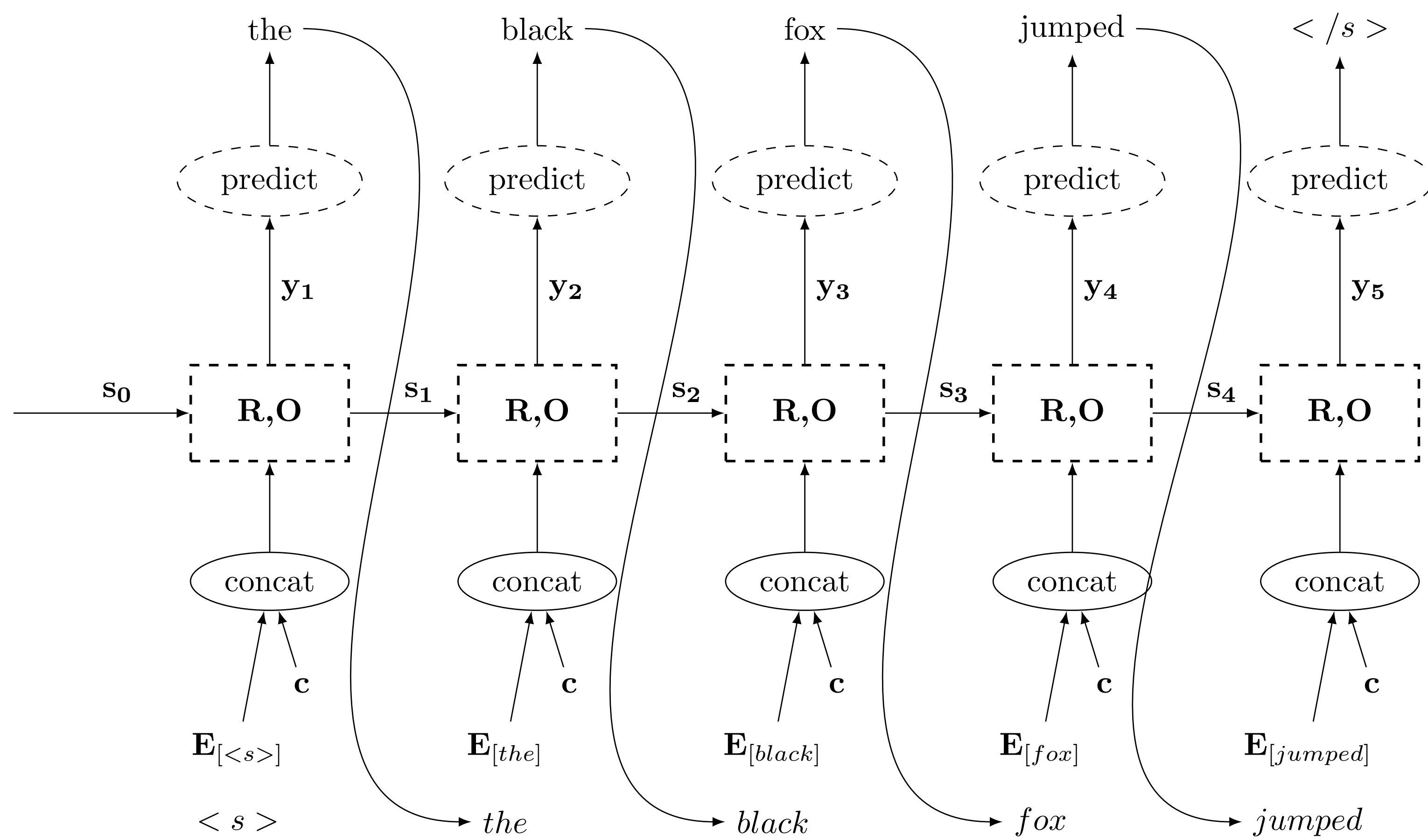
In this paper, we address the degradation problem by introducing a *deep residual learning* framework. Instead of hoping each few stacked layers directly fit a desired underlying mapping, we explicitly let these layers fit a residual mapping. Formally, denoting the desired underlying mapping as $\mathcal{H}(\mathbf{x})$, we let the stacked nonlinear layers fit another mapping of $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$. The original mapping is recast into $\mathcal{F}(\mathbf{x})+\mathbf{x}$. We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers.

back to sequences

Previously

- Mapping from a sequence to a single decision.
    - with RNN acceptors.
- Mapping from two sequences to a single decision.
    - with Siamese network.
- Mapping from a sequence to a sequence of same length.
    - with RNN transducer, or with biRNN.
- Mapping from length **m** to length **n** with a seq2seq (encoder decoder) architecture.
    - encoder RNN, decoder RNN.

RNN Language Model for **Conditioned generation**

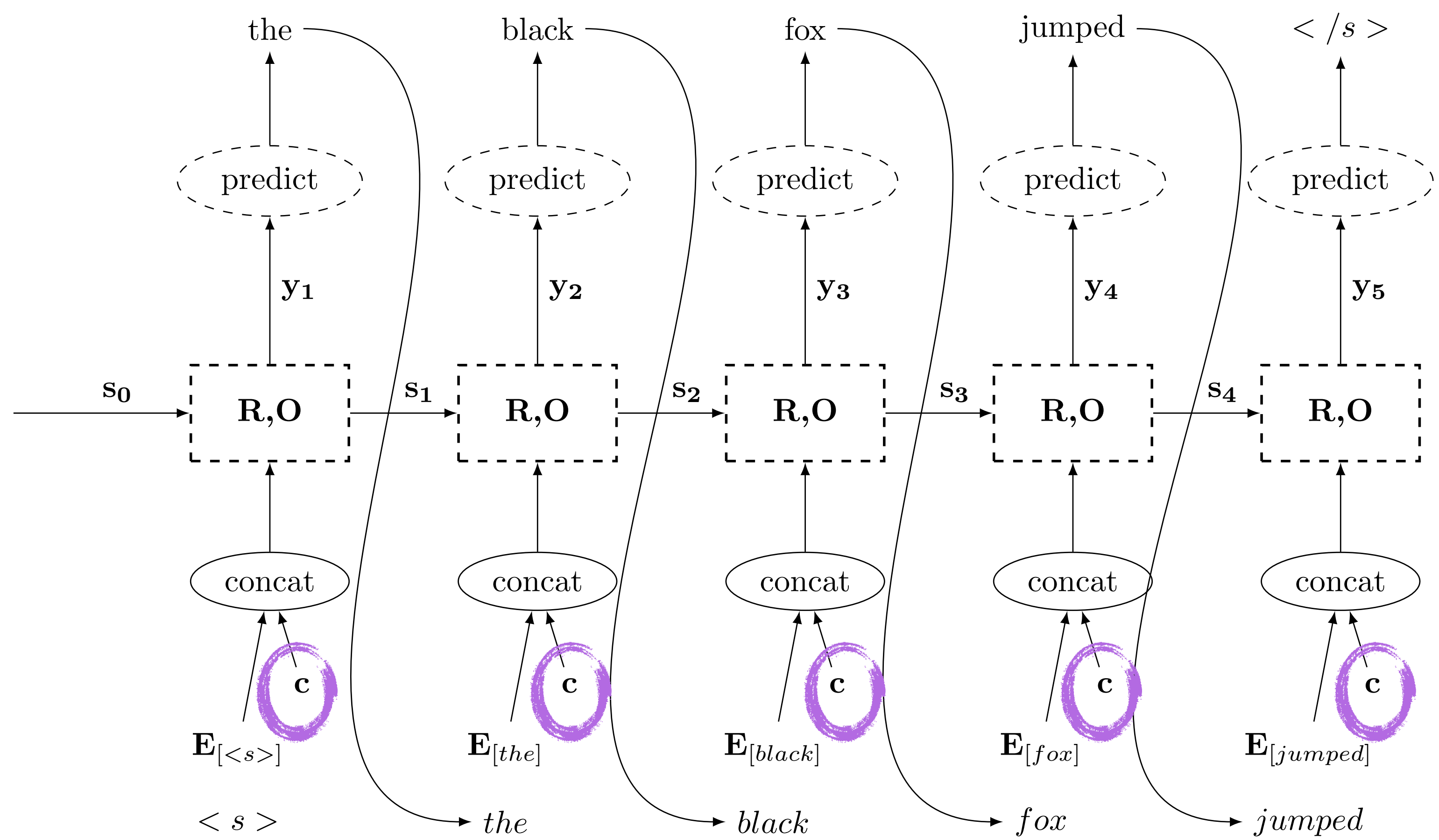


$$p(t_{j+1} = k \mid \hat{t}_{1:j}, c) = f(O(\mathbf{s_{j+1}}))$$

$$\mathbf{s_{j+1}} = R(\mathbf{s_j}, [\mathbf{\hat{t}_j}; \mathbf{c}])$$

$$\hat{t}_j \sim p(t_i \mid \hat{t}_{1:j-1}, c)$$

RNN Language Model for **Conditioned generation**

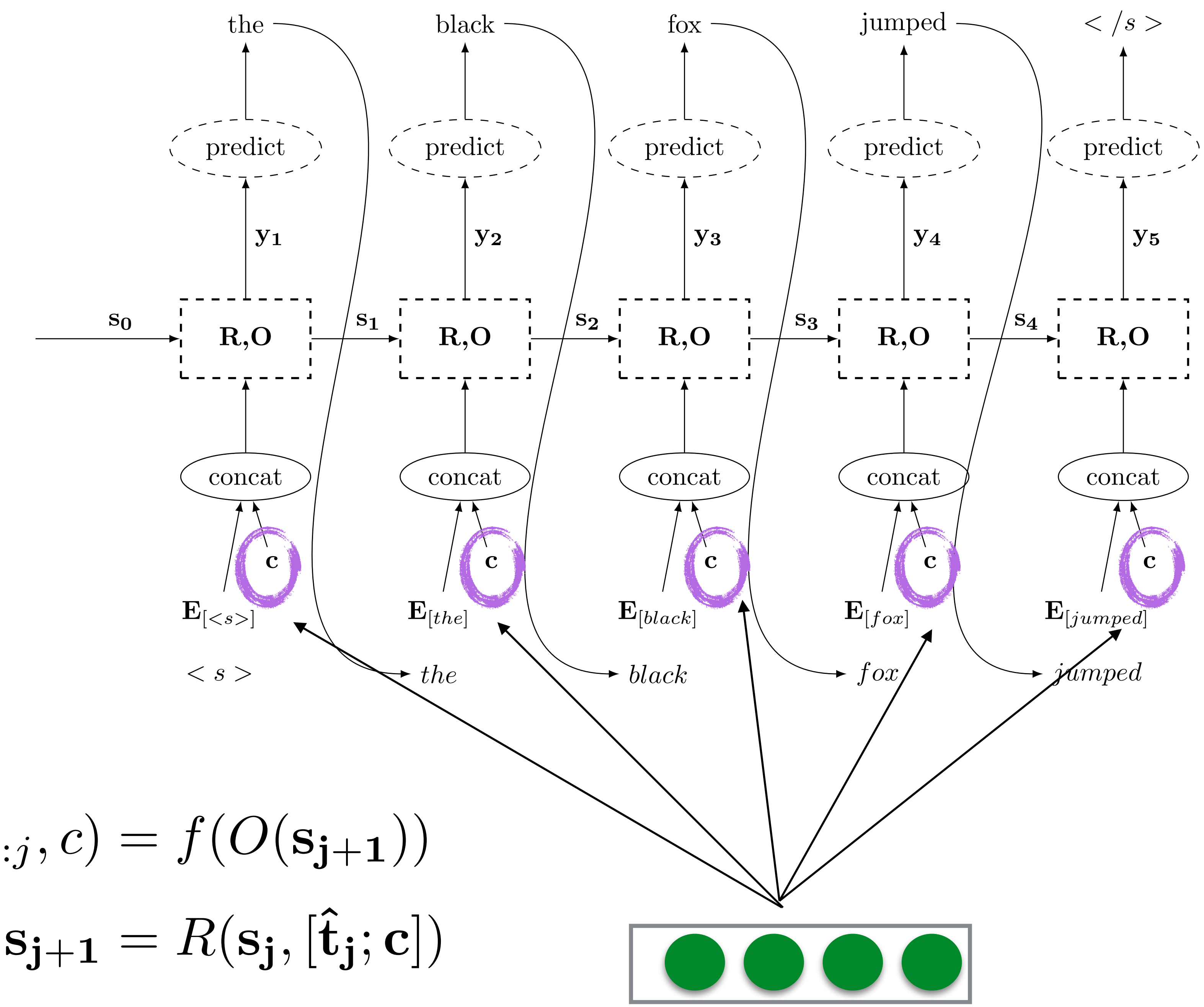


$$p(t_{j+1} = k \mid \hat{t}_{1:j}, c) = f(O(\mathbf{s_{j+1}}))$$

$$\mathbf{s_{j+1}} = R(\mathbf{s_j}, [\mathbf{\hat{t}_j}; \mathbf{c}])$$

$$\hat{t}_j \sim p(t_i \mid \hat{t}_{1:j-1}, c)$$

RNN Language Model for **Conditioned generation**

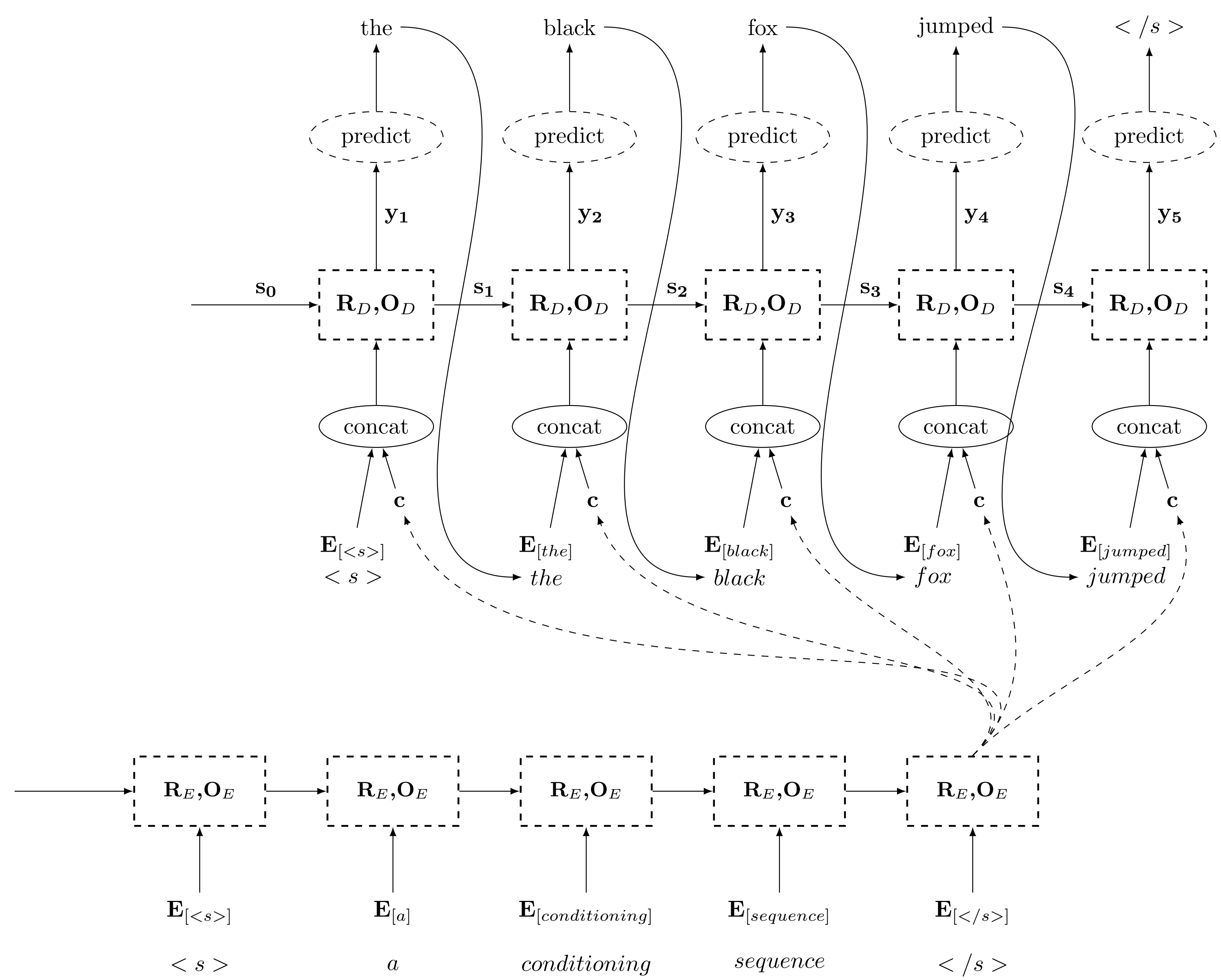


$$p(t_{j+1} = k \mid \hat{t}_{1:j}, c) = f(O(\mathbf{s_{j+1}}))$$

$$\mathbf{s_{j+1}} = R(\mathbf{s_j}, [\hat{\mathbf{t}}_\mathbf{j}; \mathbf{c}])$$

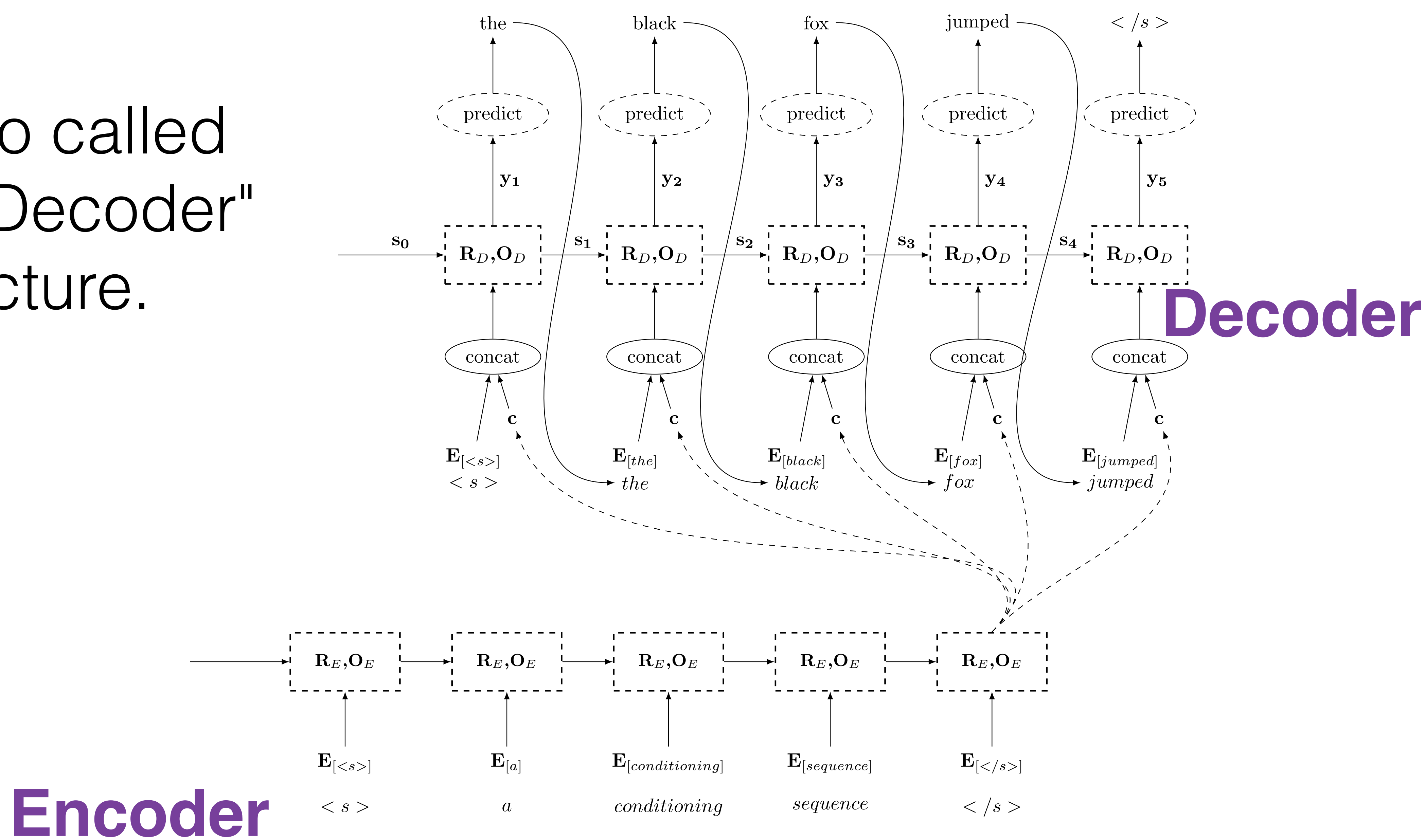
$$\hat{t}_j \sim p(t_i \mid \hat{t}_{1:j-1}, c)$$

Sequence to Sequence conditioned generation

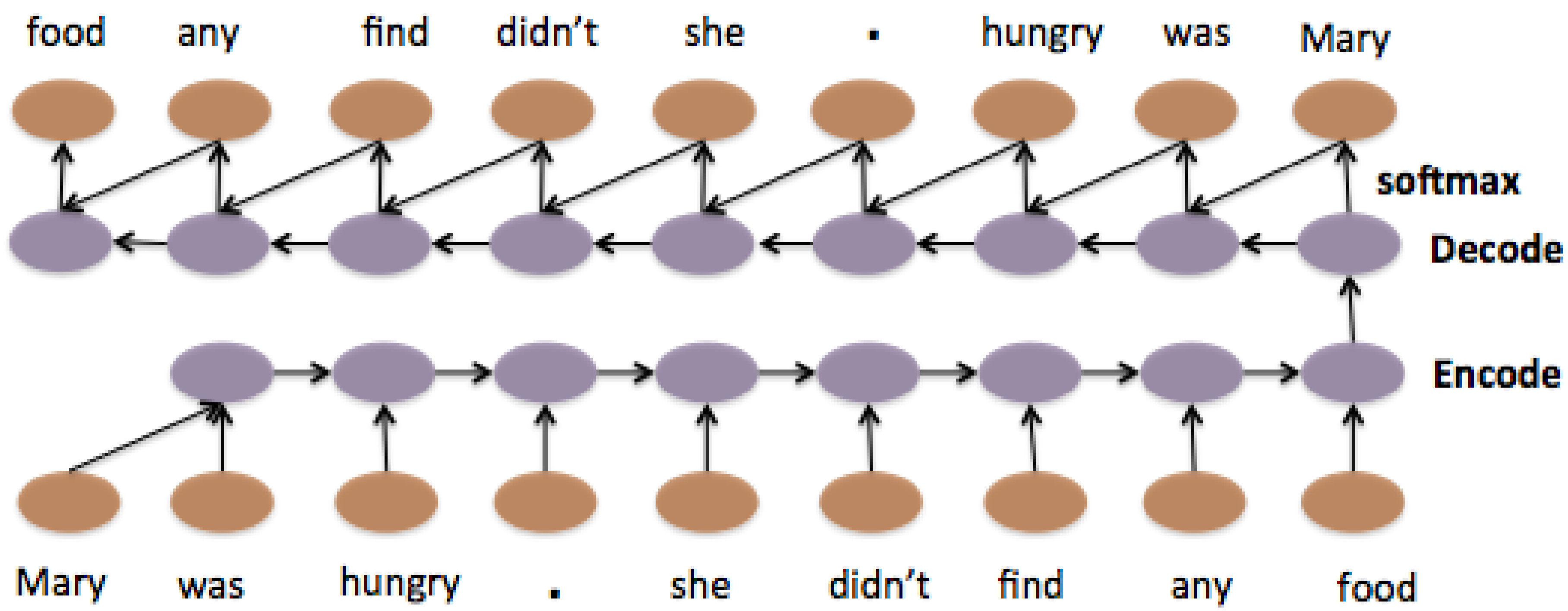


Sequence to Sequence conditioned generation

This is also called "Encoder Decoder" architecture.



Encoder-Decoder Example: Auto-encoder



A Hierarchical Neural Autoencoder for Paragraphs and Documents

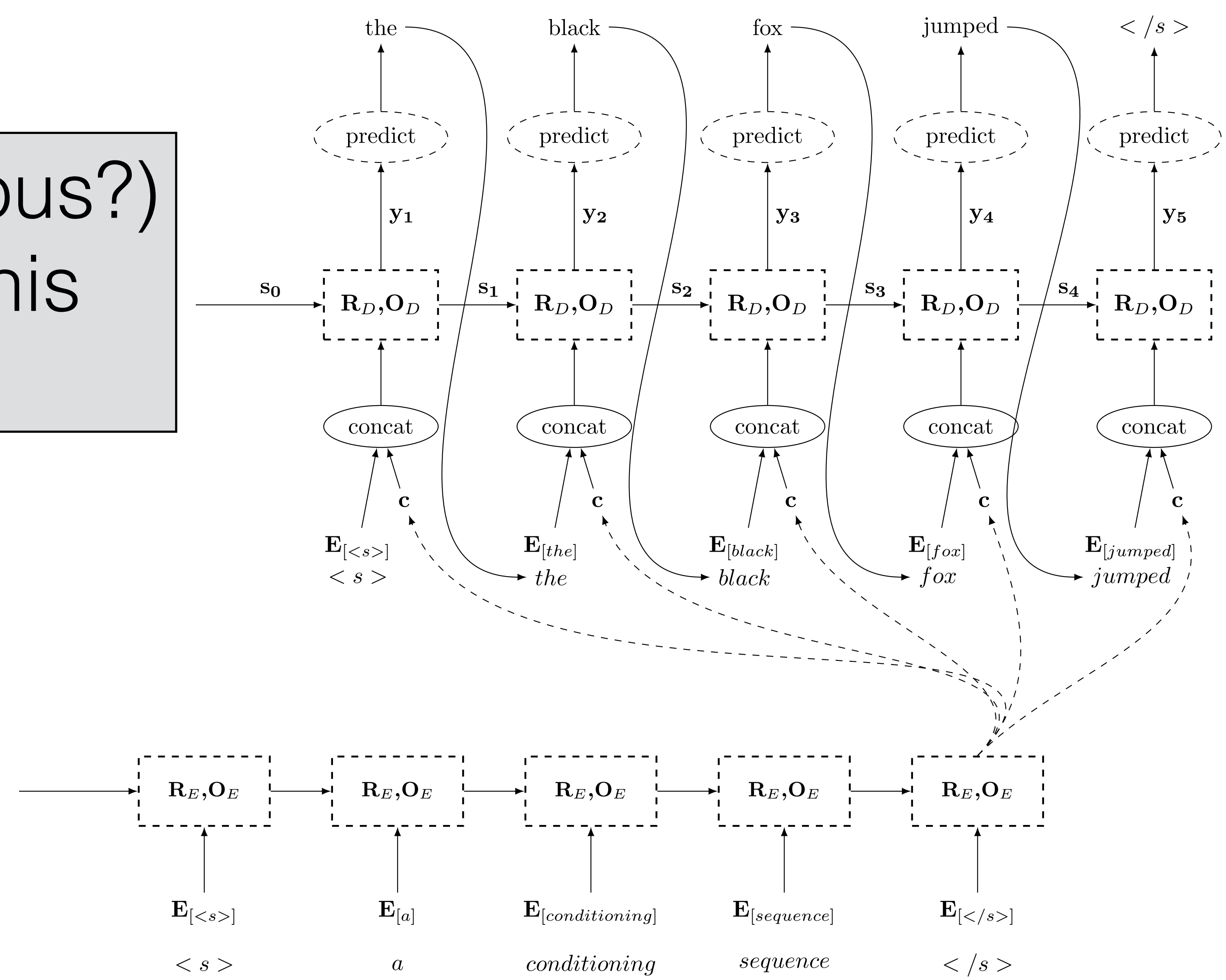
Jiwei Li, Minh-Thang Luong and Dan Jurafsky
Computer Science Department, Stanford University, Stanford, CA 94305, USA
jiweil, lmthang, jurafsky@stanford.edu

- Encode: English sentence.
  Decode: Same English sentence.

encoded vector is a "generic sentence representation"

Sequence to Sequence conditioned generation

What's the (obvious?) problem with this approach?



Attention

a.k.a "soft alignment"

NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

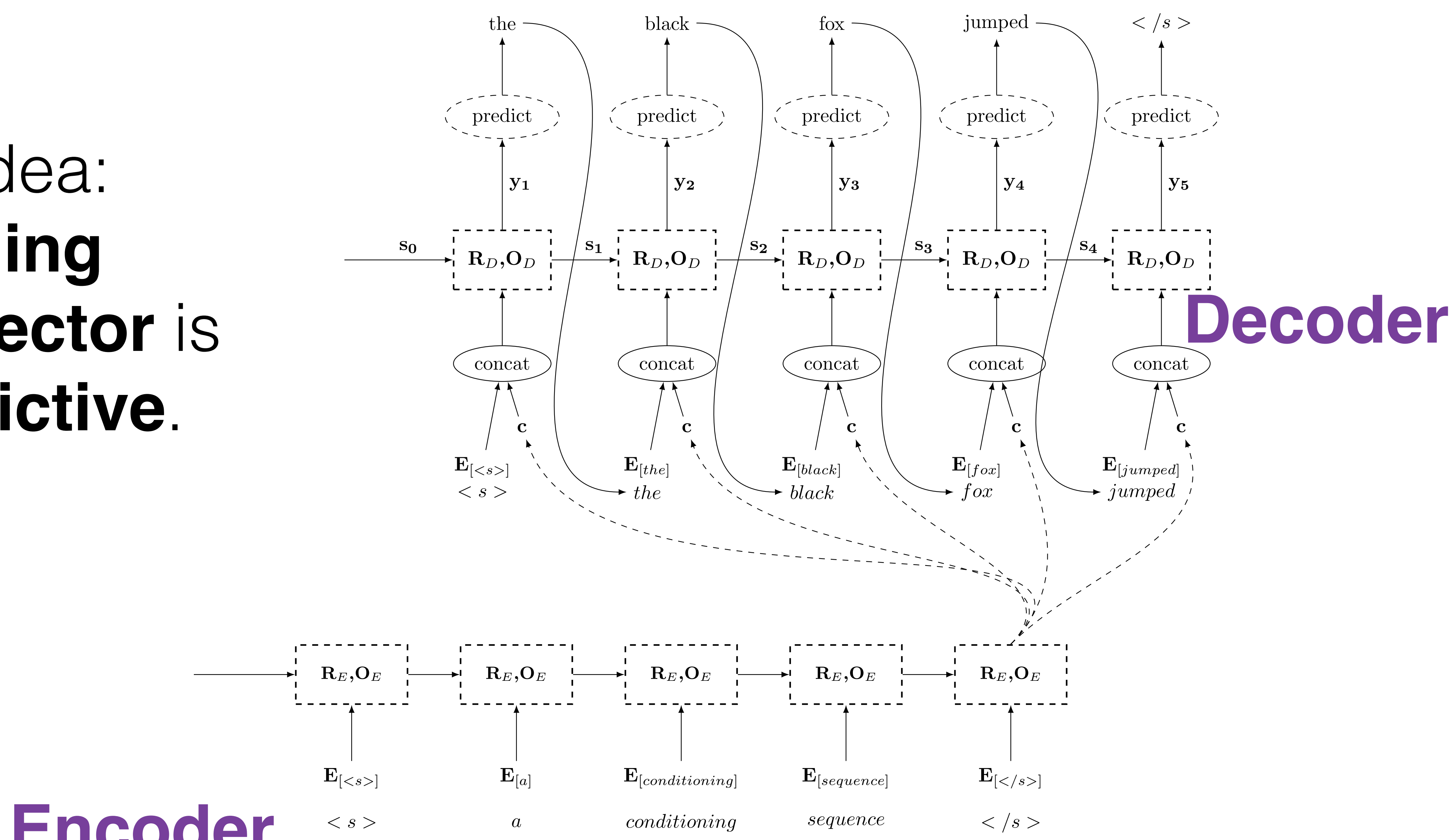
KyungHyun Cho **Yoshua Bengio***
Université de Montréal

Effective Approaches to Attention-based Neural Machine Translation

Minh-Thang Luong **Hieu Pham** **Christopher D. Manning**
Computer Science Department, Stanford University, Stanford, CA 94305
{lmthang,hyhieu,manning}@stanford.edu

Sequence to Sequence conditioned generation

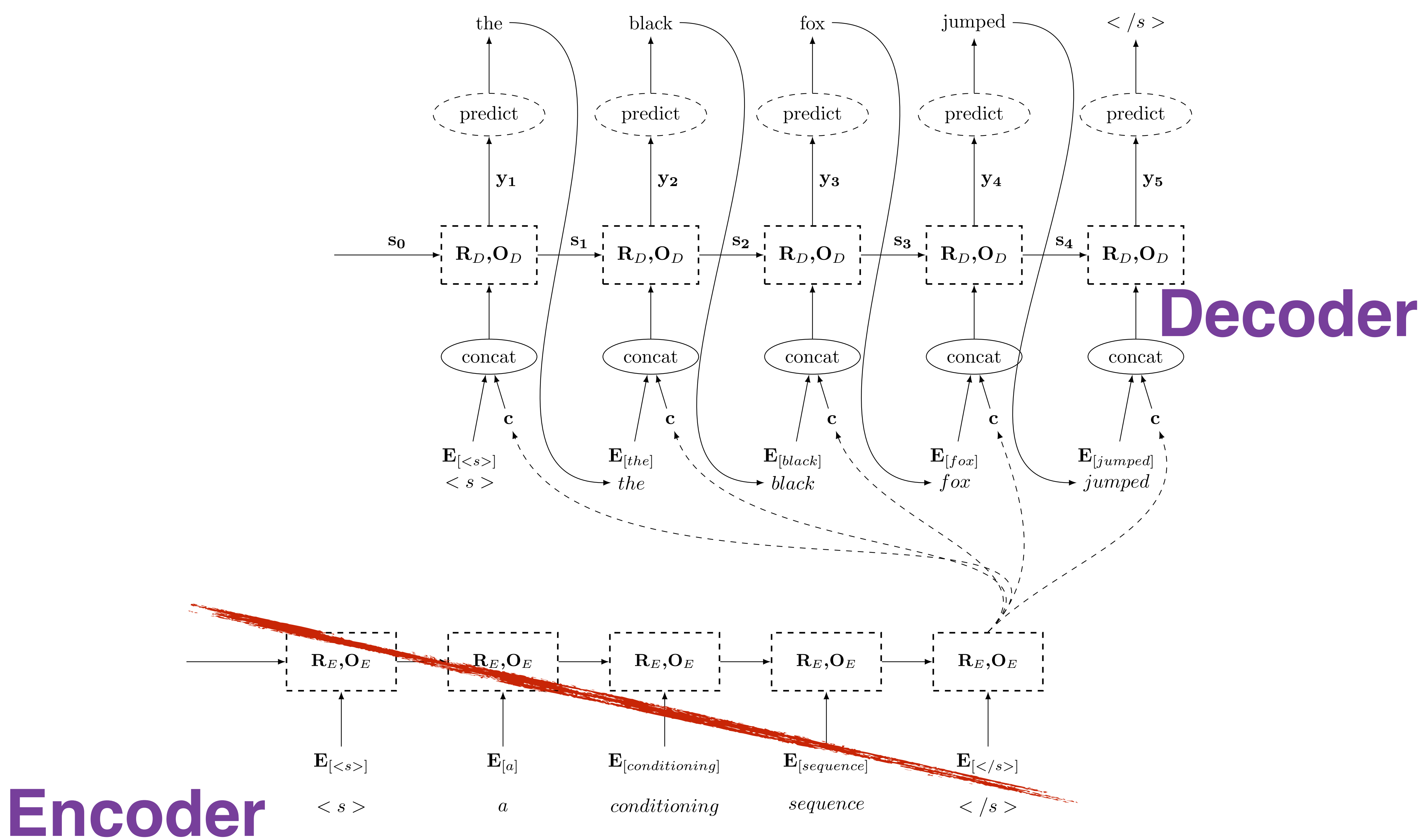
main idea:
encoding
a **single vector** is
too restrictive.



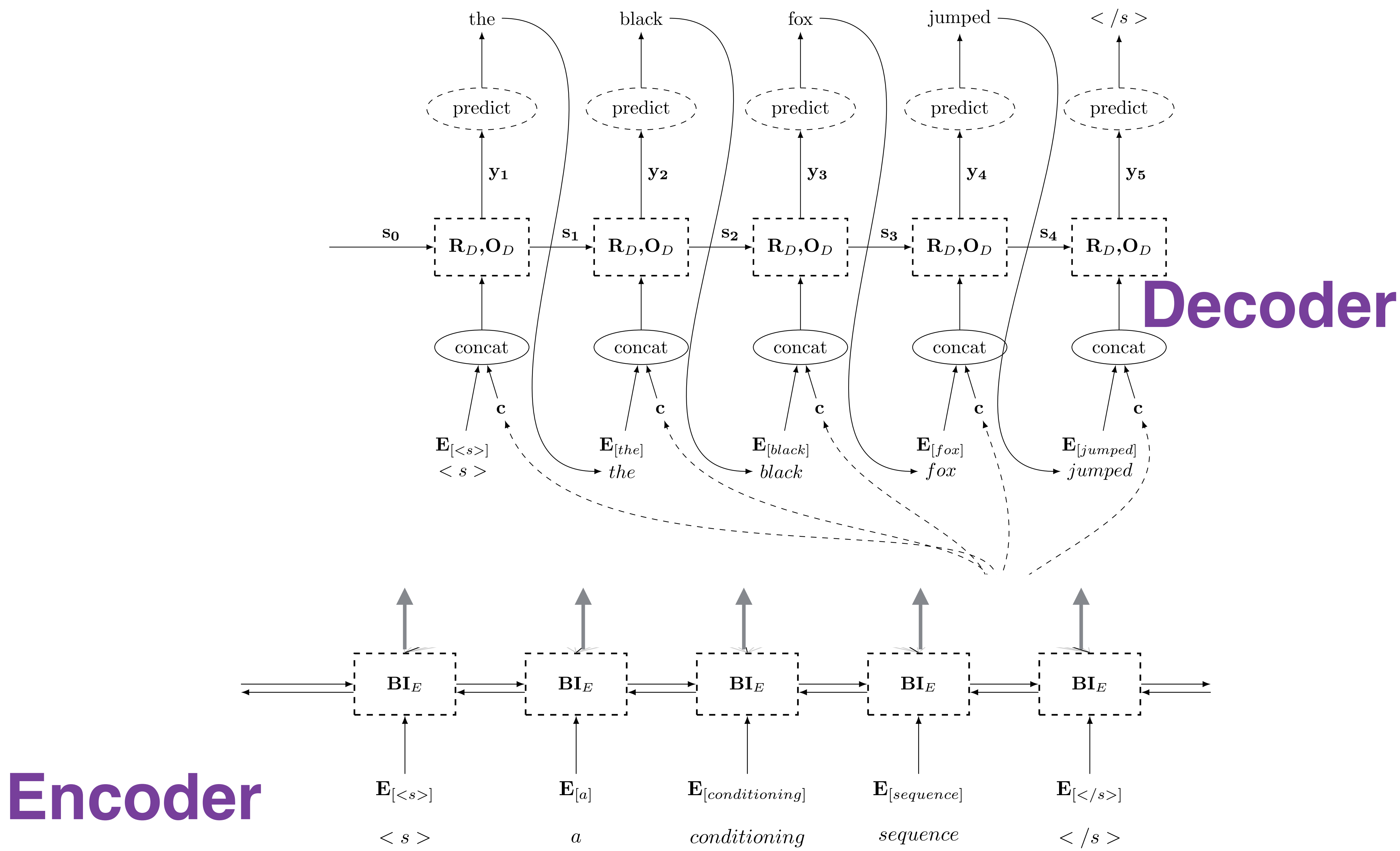
Attention

- Instead of the encoder producing a single vector for the sentence, it will produce a one vector **for each word**.

Sequence to Sequence conditioned generation

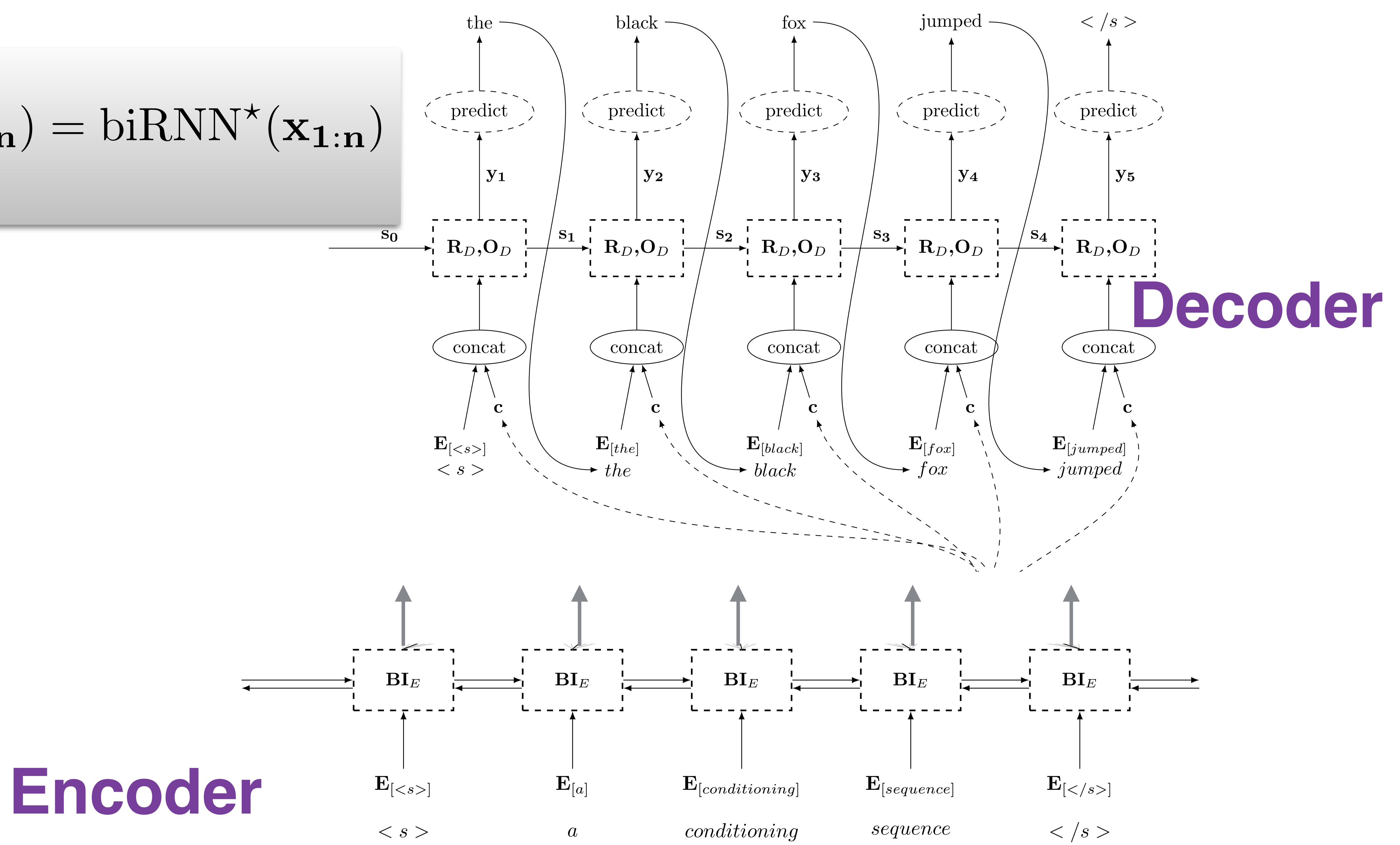


Sequence to Sequence conditioned generation



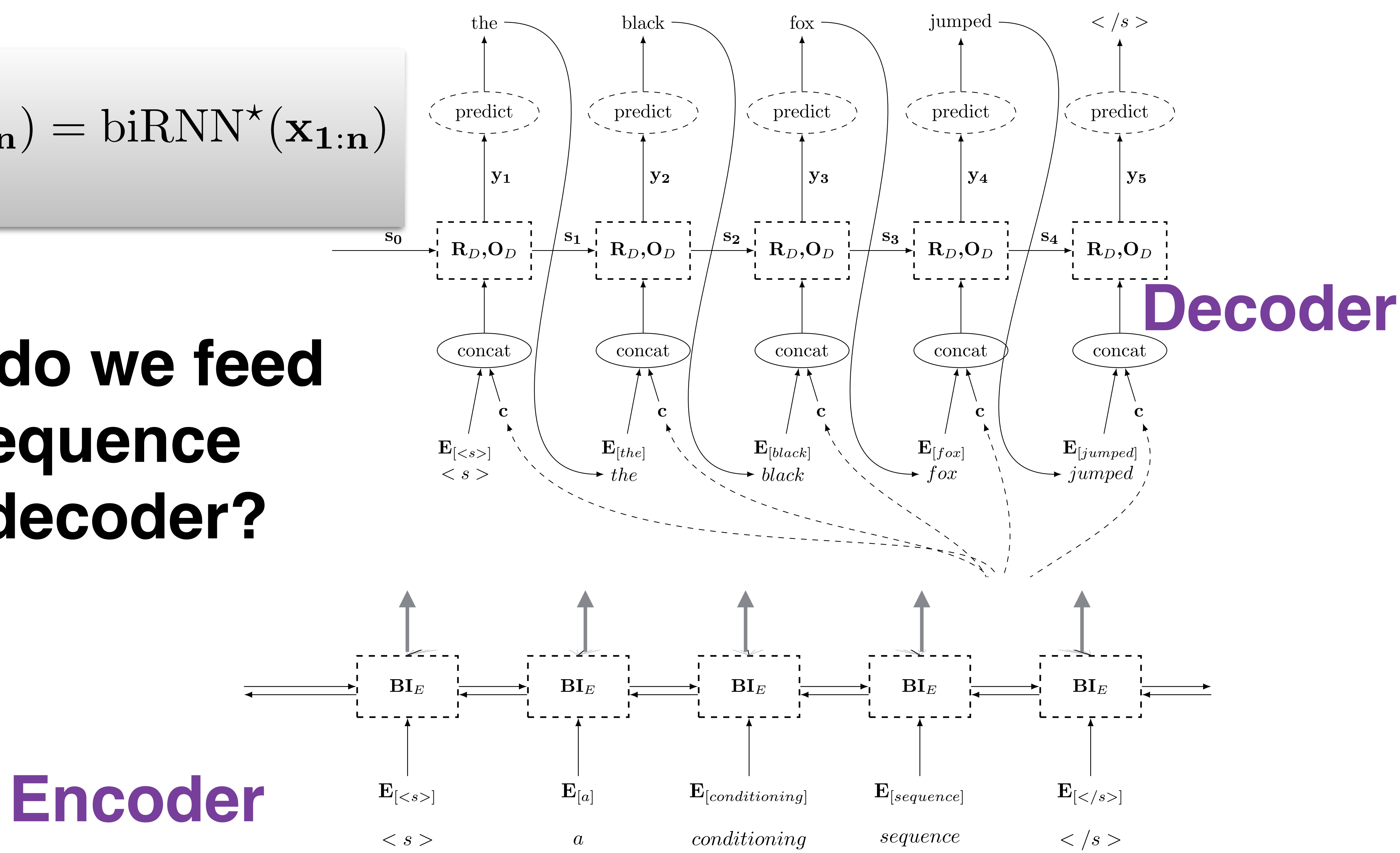
Sequence to Sequence conditioned generation

$$\mathbf{c_{1:n}} = \textsc{Enc}(\mathbf{x_{1:n}}) = \text{biRNN}^{\star}(\mathbf{x_{1:n}})$$



Sequence to Sequence conditioned generation

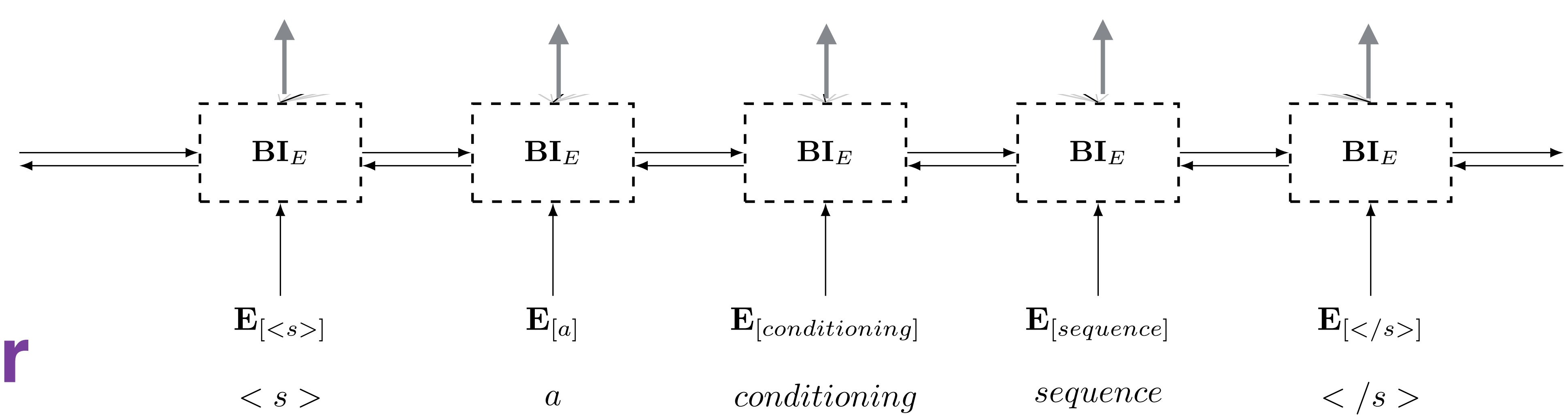
$$\mathbf{c_{1:n}} = \text{Enc}(\mathbf{x_{1:n}}) = \text{biRNN}^{\star}(\mathbf{x_{1:n}})$$



but how do we feed this sequence to the decoder?

Sequence to Sequence conditioned generation

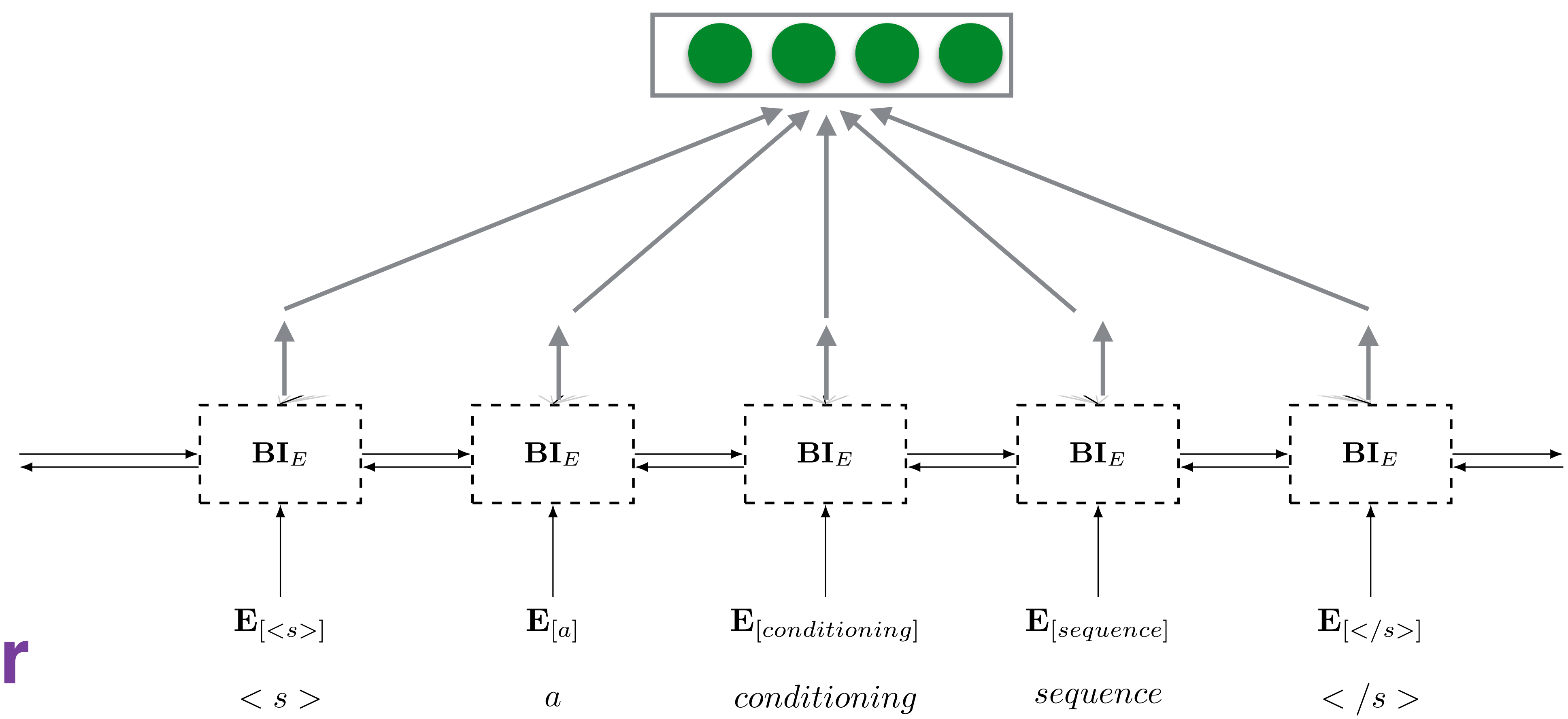
we can combine the different outputs
into a single vector



Encoder

Sequence to Sequence conditioned generation

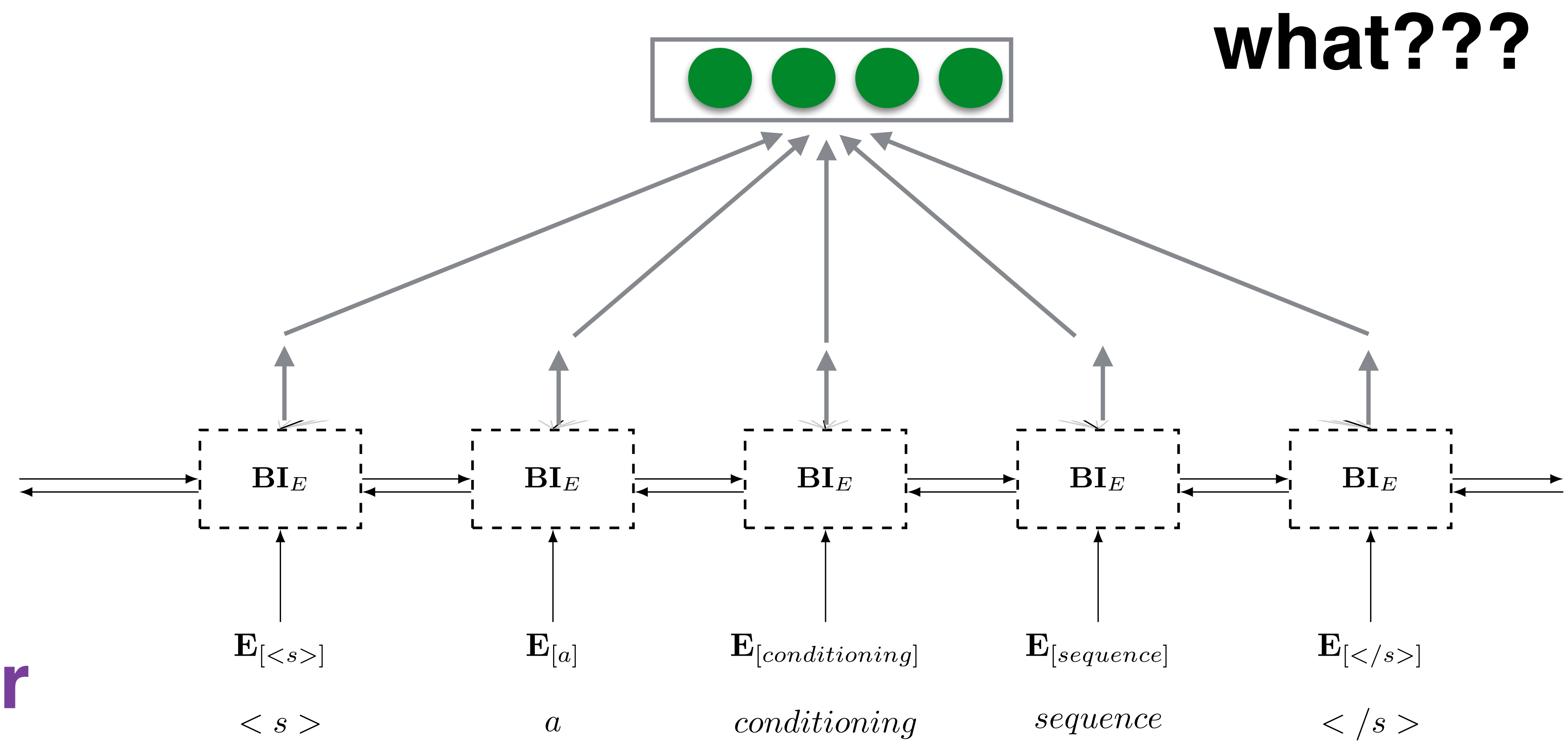
we can combine the different outputs
into a single vector



Encoder

Sequence to Sequence conditioned generation

we can combine the different outputs
into a single vector

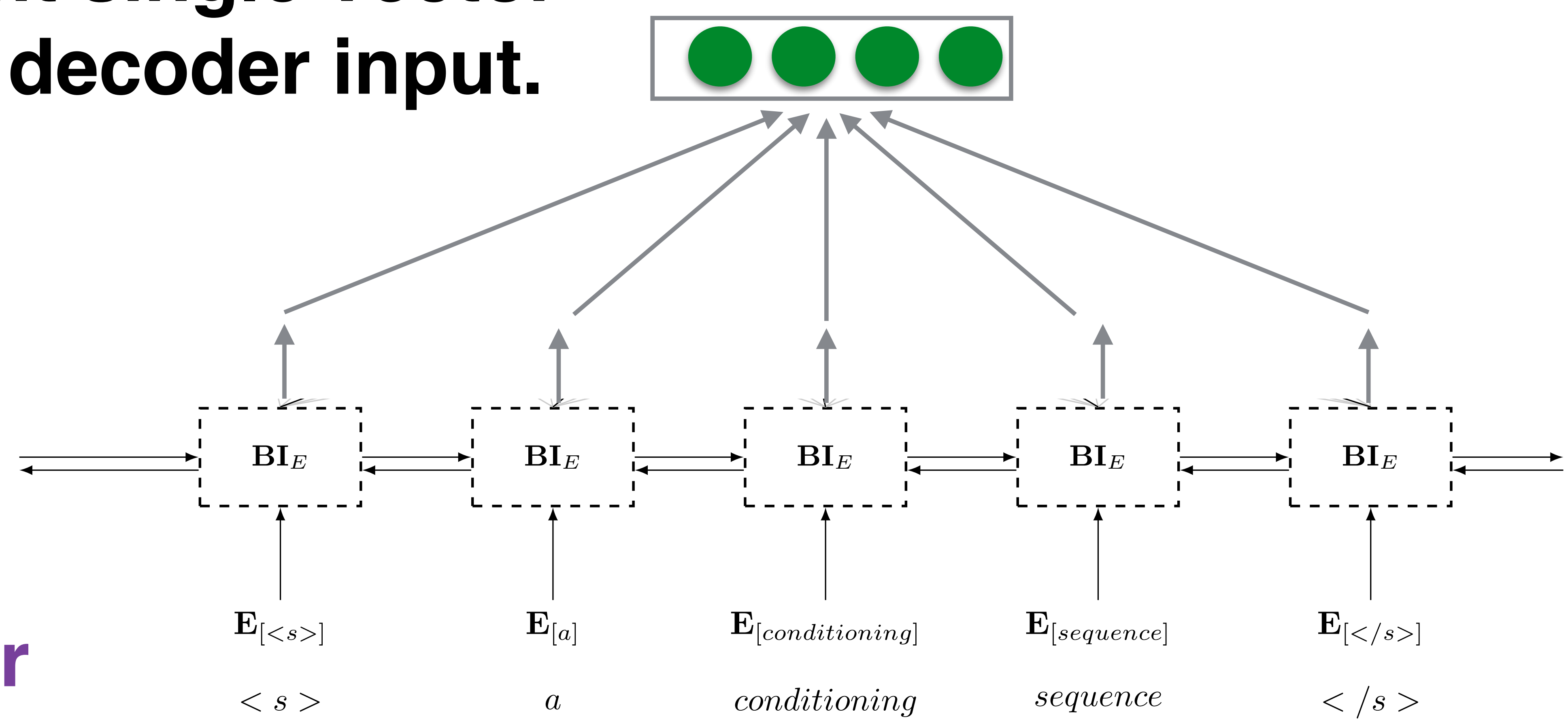


Encoder

Sequence to Sequence conditioned generation

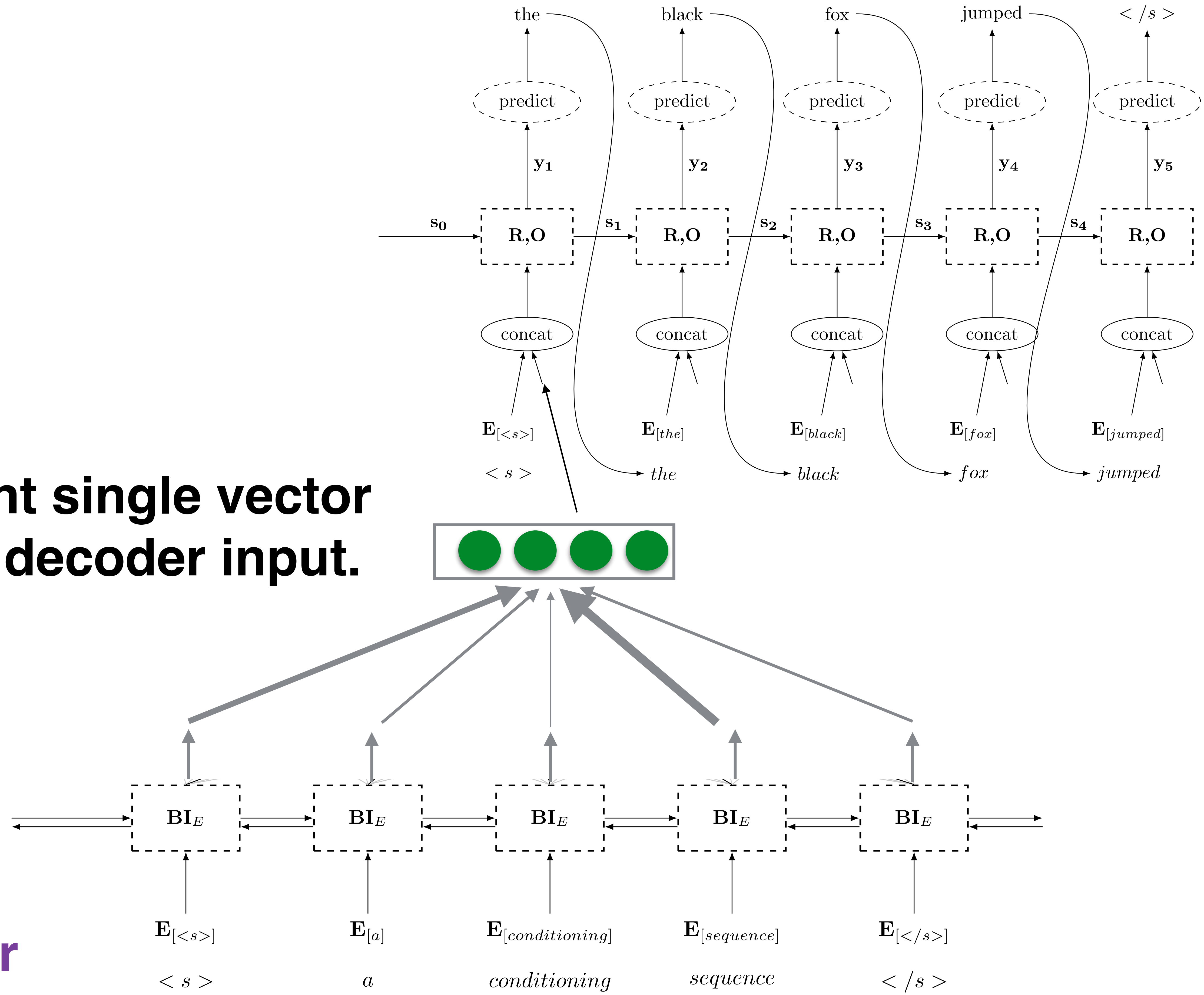
we can combine the different outputs
into a single vector

a different single vector at each decoder input.



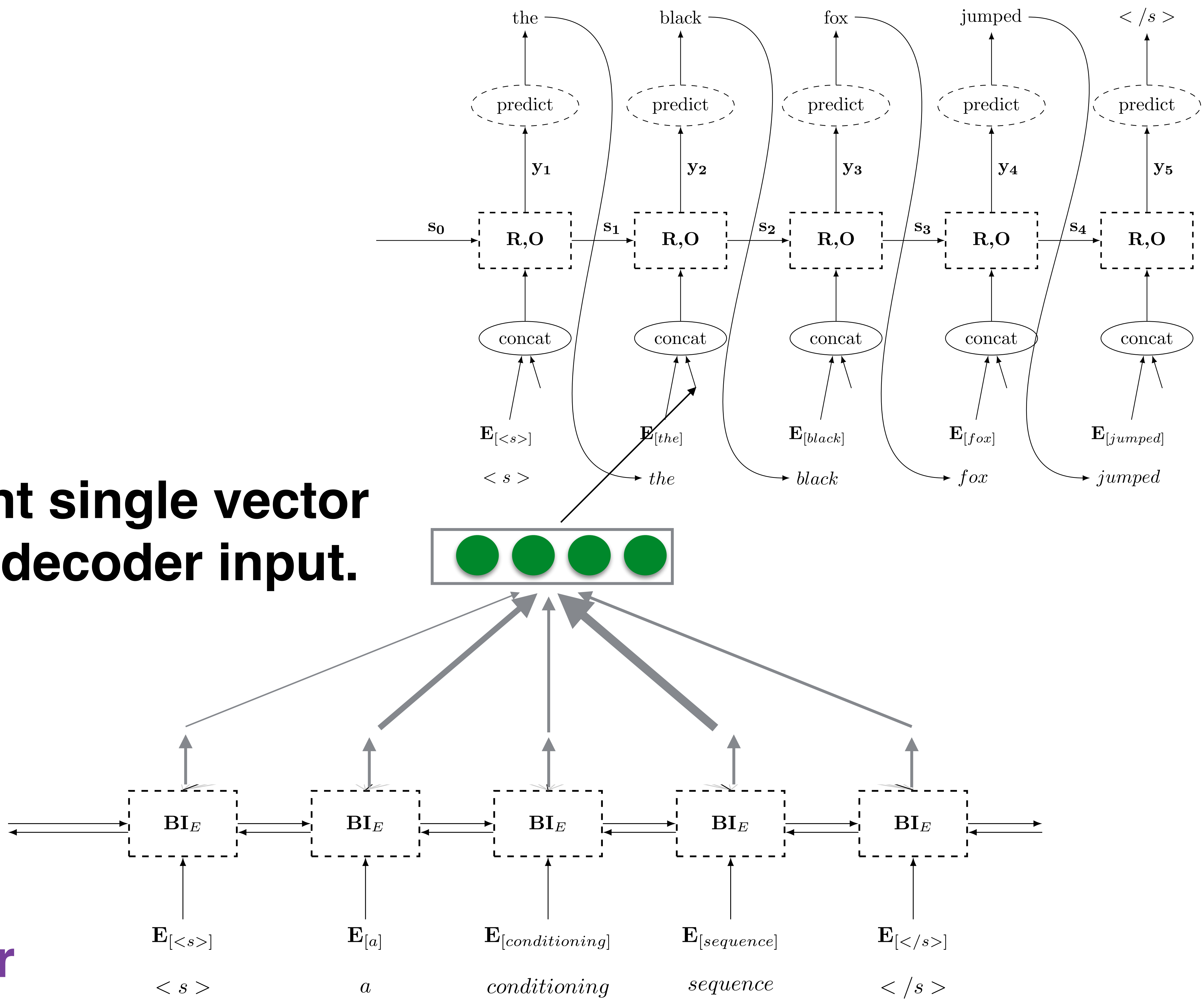
Encoder

a different single vector at each decoder input.



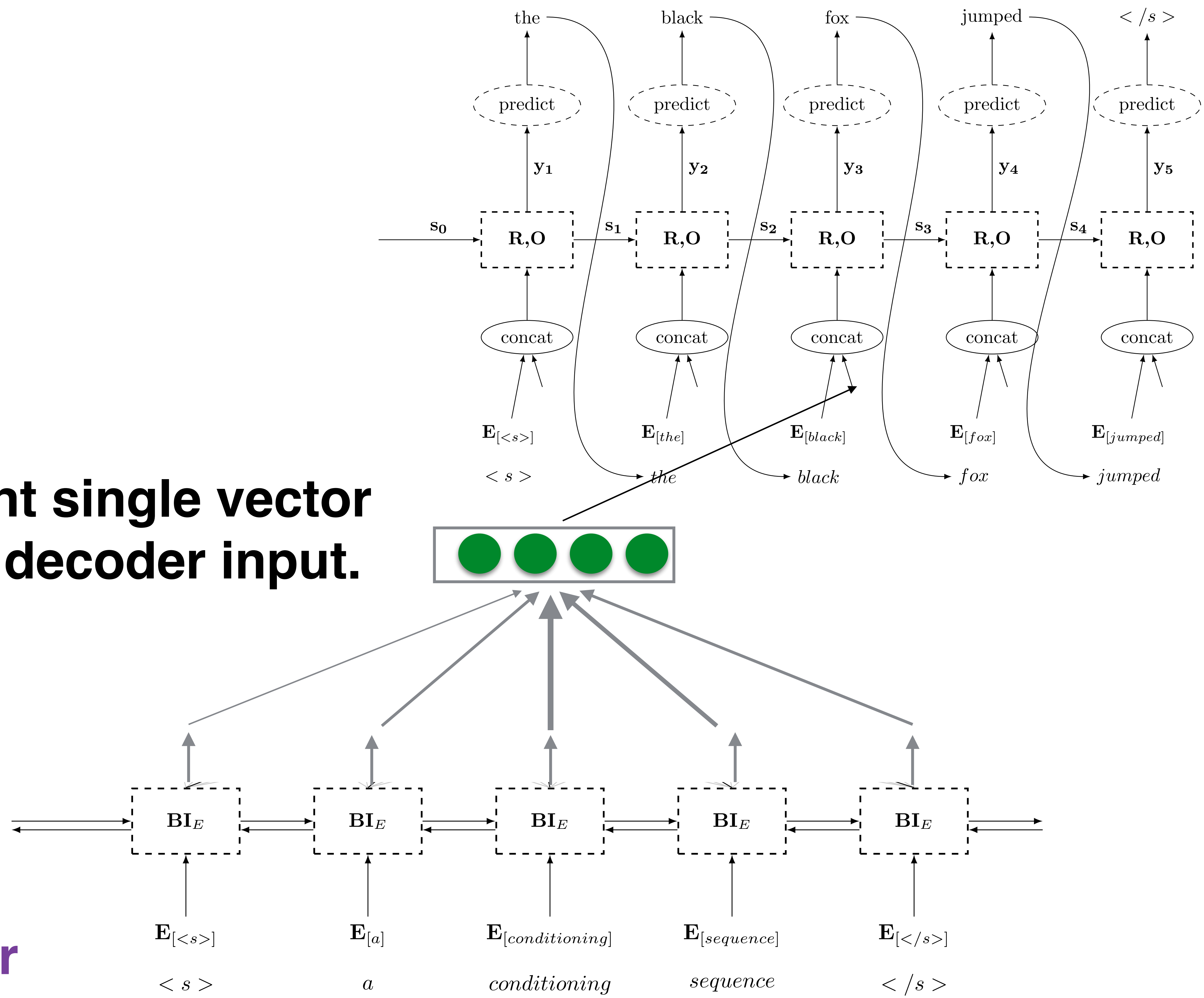
Encoder

a different single vector at each decoder input.



Encoder

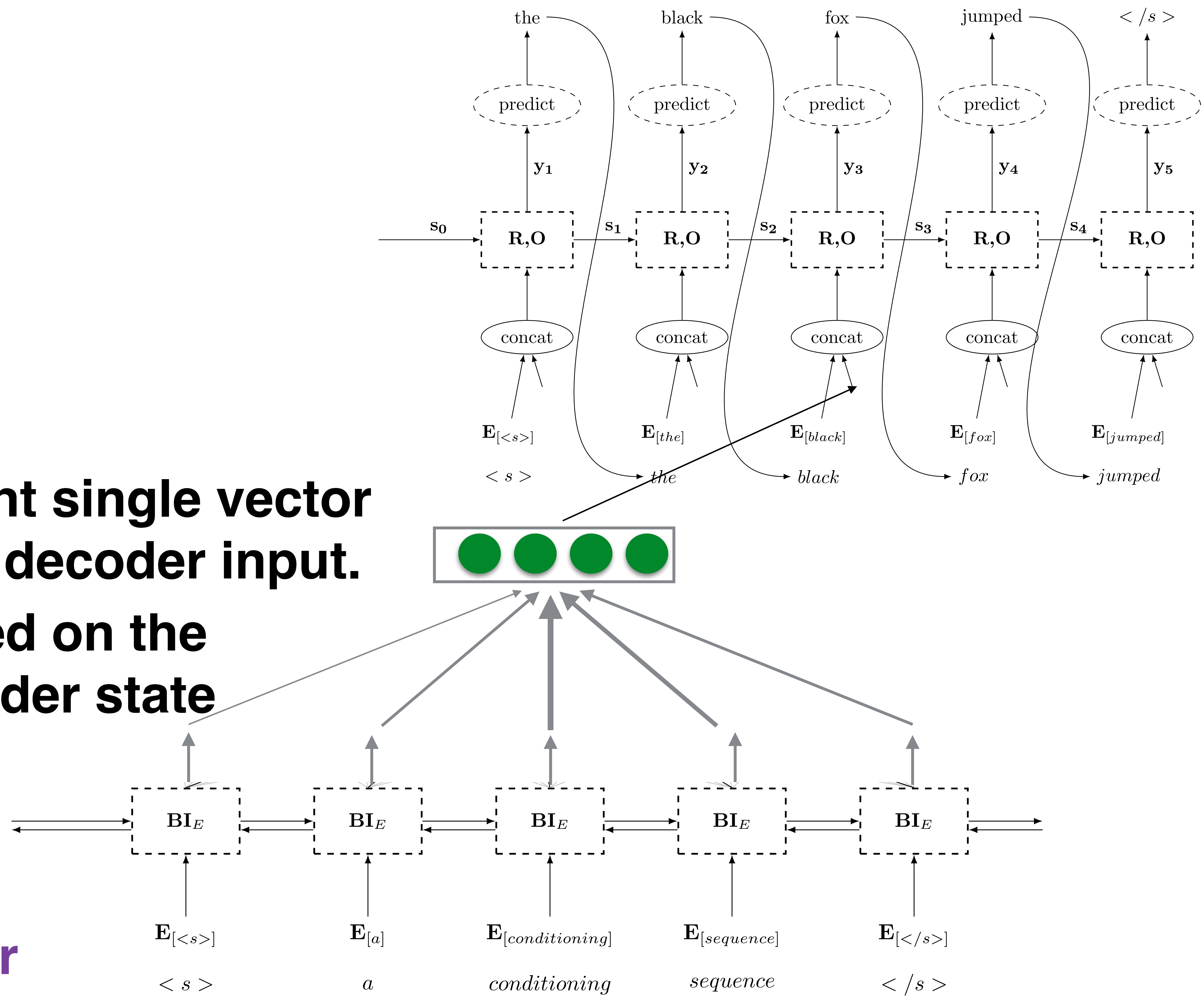
a different single vector at each decoder input.



Encoder

a different single vector at each decoder input.

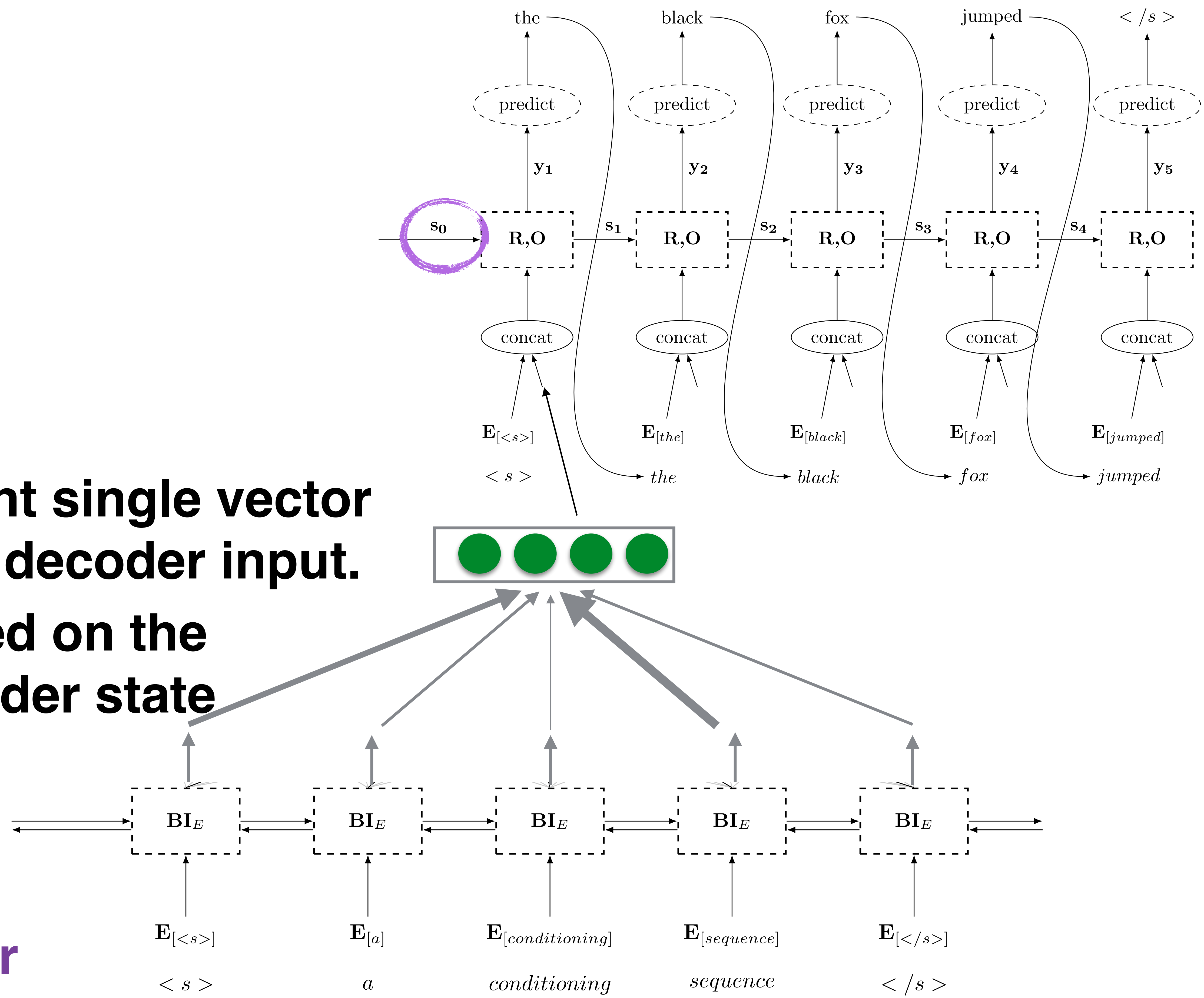
based on the decoder state



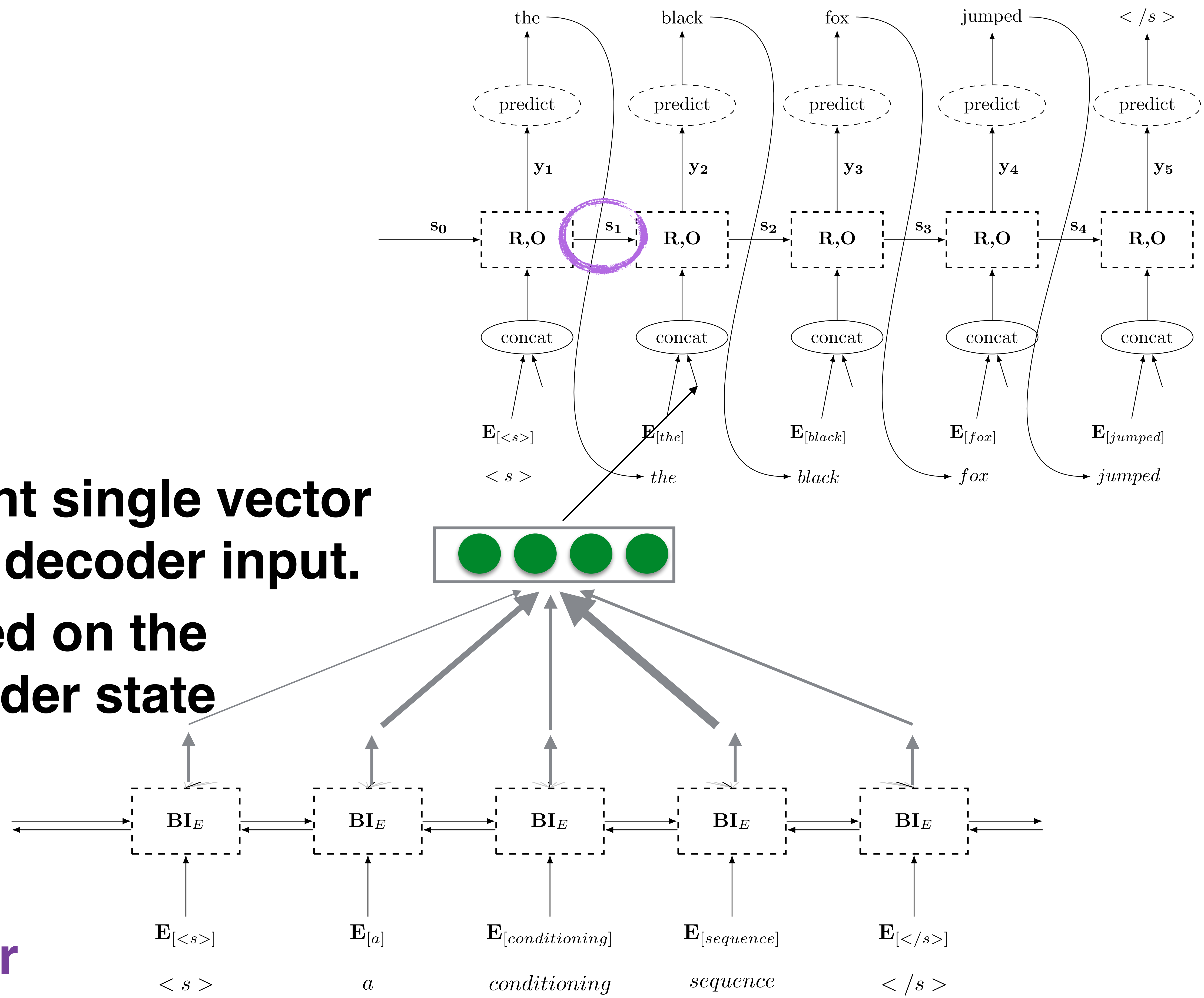
Encoder

a different single vector at each decoder input.

based on the decoder state



Encoder



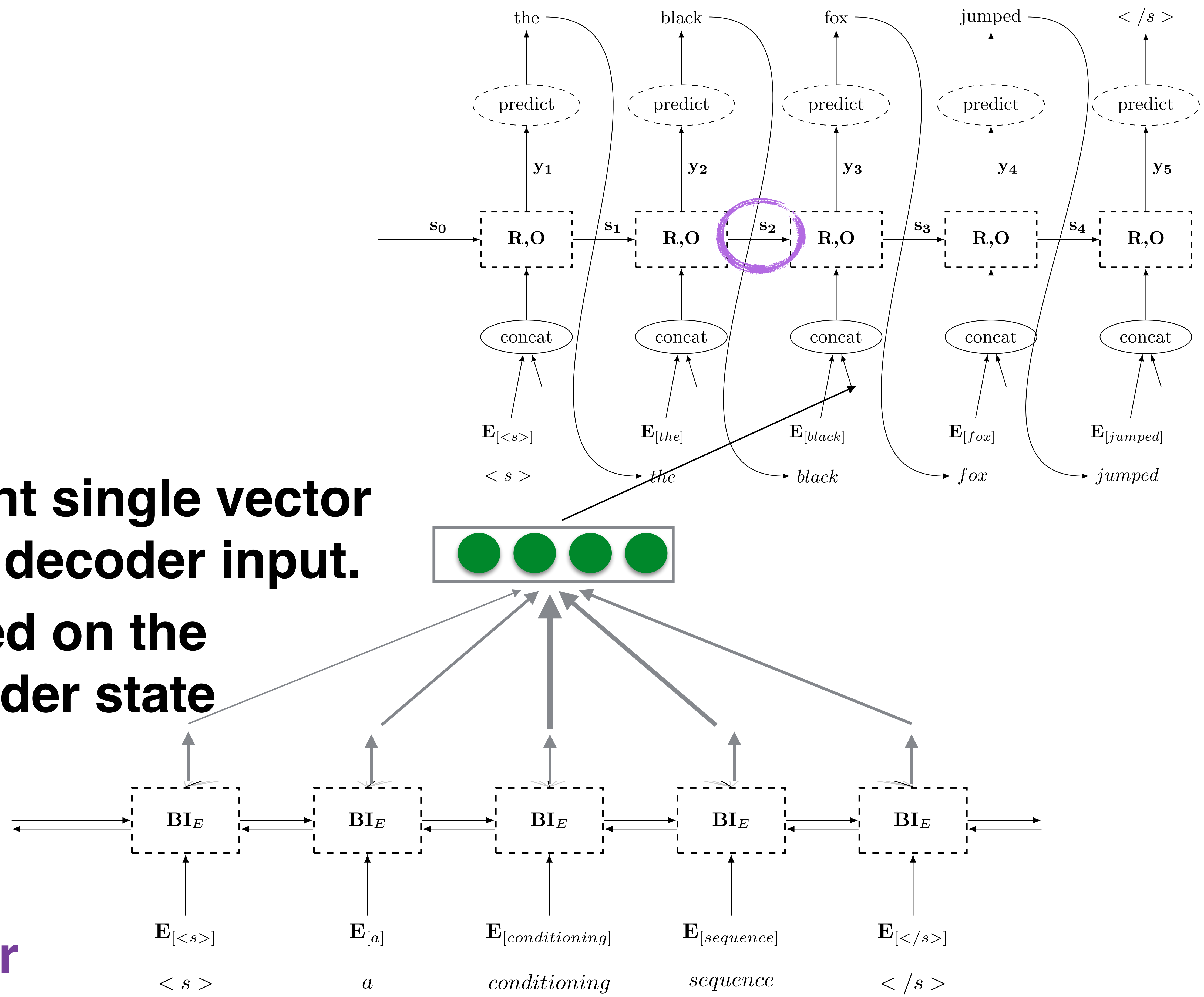
a different single vector at each decoder input.

based on the decoder state

Encoder

a different single vector at each decoder input.

based on the decoder state



Encoder

$$
\begin{aligned}
p(t_{j+1} = k \mid \hat{t}_{1:j}, \mathbf{x_{1:n}}) &= f(O(\mathbf{s_{j+1}})) \\
\mathbf{s_{j+1}} &= R(\mathbf{s_j}, [\mathbf{\hat{t}_j}; \mathbf{c^j}]) \\
\mathbf{c^j} &= \mathrm{attend}(\mathbf{c_{1:n}}, \hat{t}_{1:j}) \\
\hat{t}_j &\sim p(t_j \mid \hat{t}_{1:j-1}, \mathbf{x_{1:n}})
\end{aligned}
$$

$$p(t_{j+1} = k \mid \hat{t}_{1:j}, \mathbf{x_{1:n}}) = f(O(\mathbf{s_{j+1}}))$$

$$\mathbf{s_{j+1}} = R(\mathbf{s_j}, [\mathbf{\hat{t}_j}; \mathbf{c^j}])$$

$$\mathbf{c^j} = \text{attend}(\mathbf{c_{1:n}}, \hat{t}_{1:j})$$

$$\hat{t}_j \sim p(t_j \mid \hat{t}_{1:j-1}, \mathbf{x_{1:n}})$$

$$p(t_{j+1} = k \mid \hat{t}_{1:j}, \mathbf{x_{1:n}}) = f(O(\mathbf{s_{j+1}}))$$

$$\mathbf{s_{j+1}} = R(\mathbf{s_j}, [\hat{\mathbf{t}}_{\mathbf{j}}; \mathbf{c^j}])$$

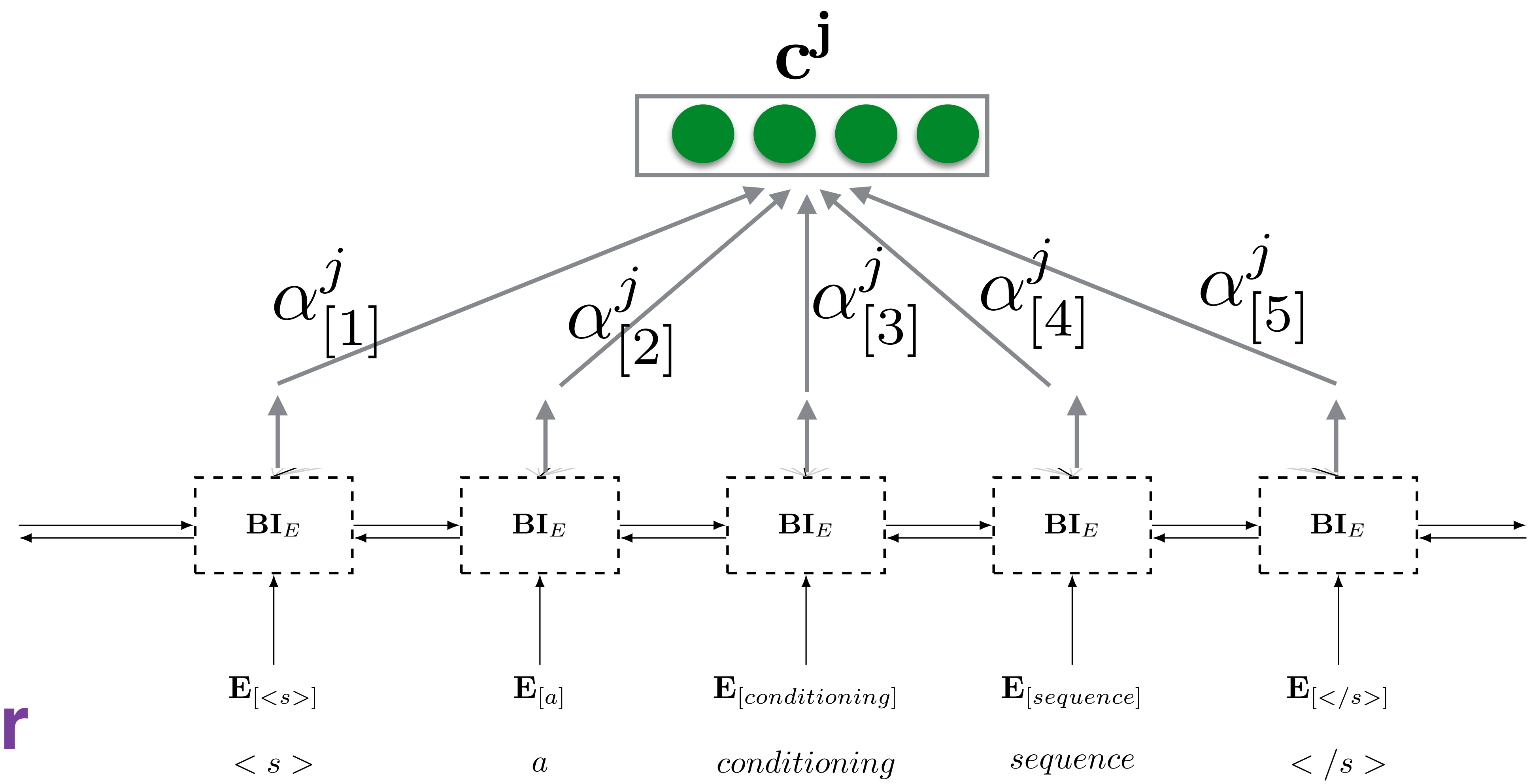
$$\mathbf{c^j} = \mathrm{attend}(\mathbf{c_{1:n}}, \hat{t}_{1:j})$$

$$\hat{t}_j \sim p(t_j \mid \hat{t}_{1:j-1}, \mathbf{x_{1:n}})$$

$$\mathbf{c^j} = \sum_{i=1}^{n} \alpha^{\mathbf{j}}_{[\mathbf{i}]} \cdot \mathbf{c_i}$$

Sequence to Sequence conditioned generation

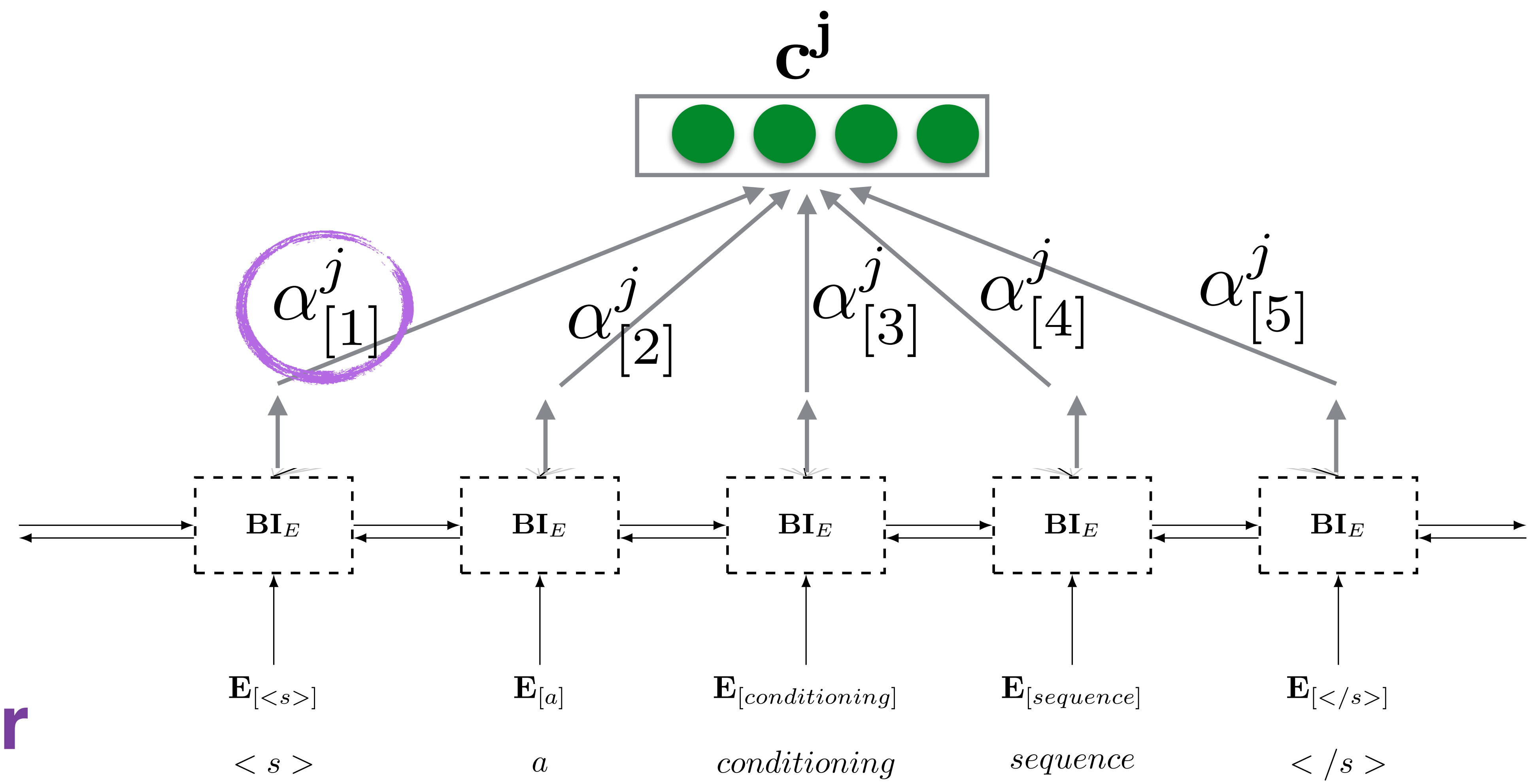
$$\mathbf{c^j} = \sum_{i=1}^{n} \alpha^{\mathbf{j}}_{[\mathbf{i}]} \cdot \mathbf{c_i}$$



Encoder

Sequence to Sequence conditioned generation

$$\mathbf{c^j} = \sum_{i=1}^{n} \alpha^{\mathbf{j}}_{[\mathbf{i}]} \cdot \mathbf{c_i}$$

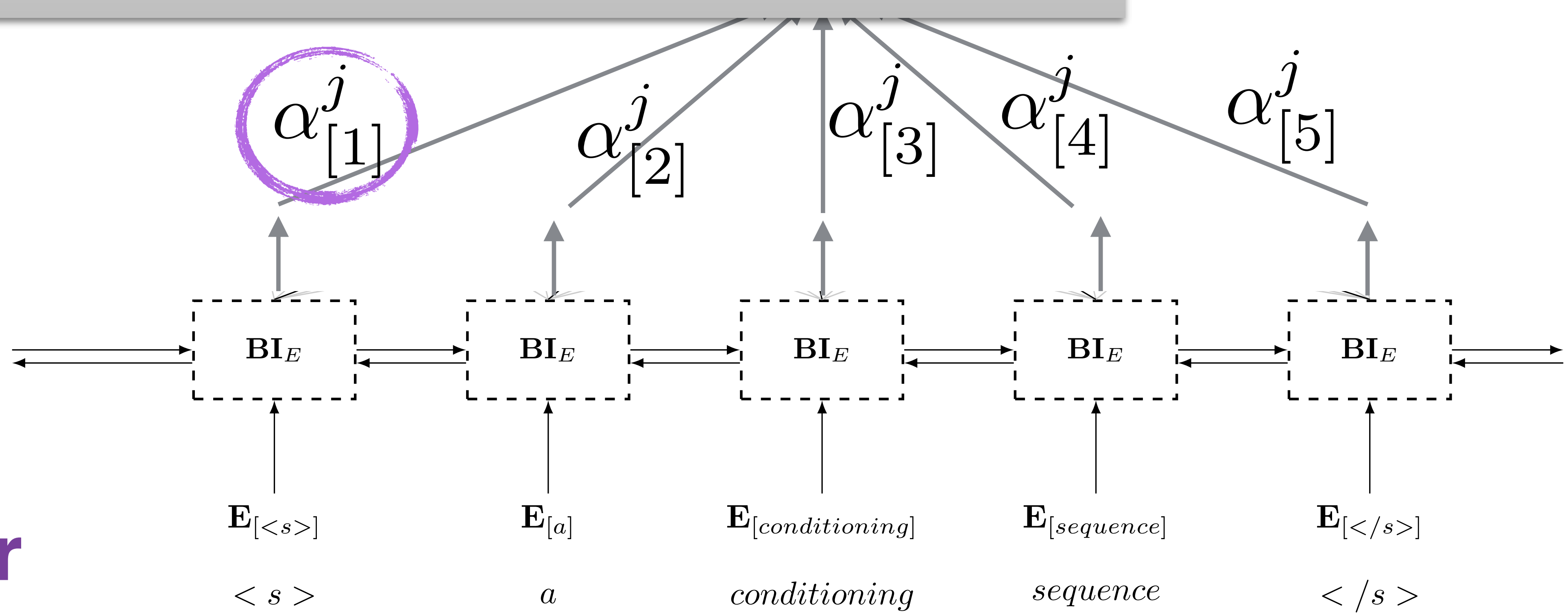


Encoder

Sequence to Sequence conditioned generation

$$\alpha^{\mathbf{j}} = \text{softmax}(\bar{\alpha}^{\mathbf{j}}_{[\mathbf{1}]}, \ldots, \bar{\alpha}^{\mathbf{j}}_{[\mathbf{n}]})$$

$$\mathbf{c^j} = \sum_{i=1}^{n} \alpha^{\mathbf{j}}_{[\mathbf{i}]} \mathbf{c_i}$$



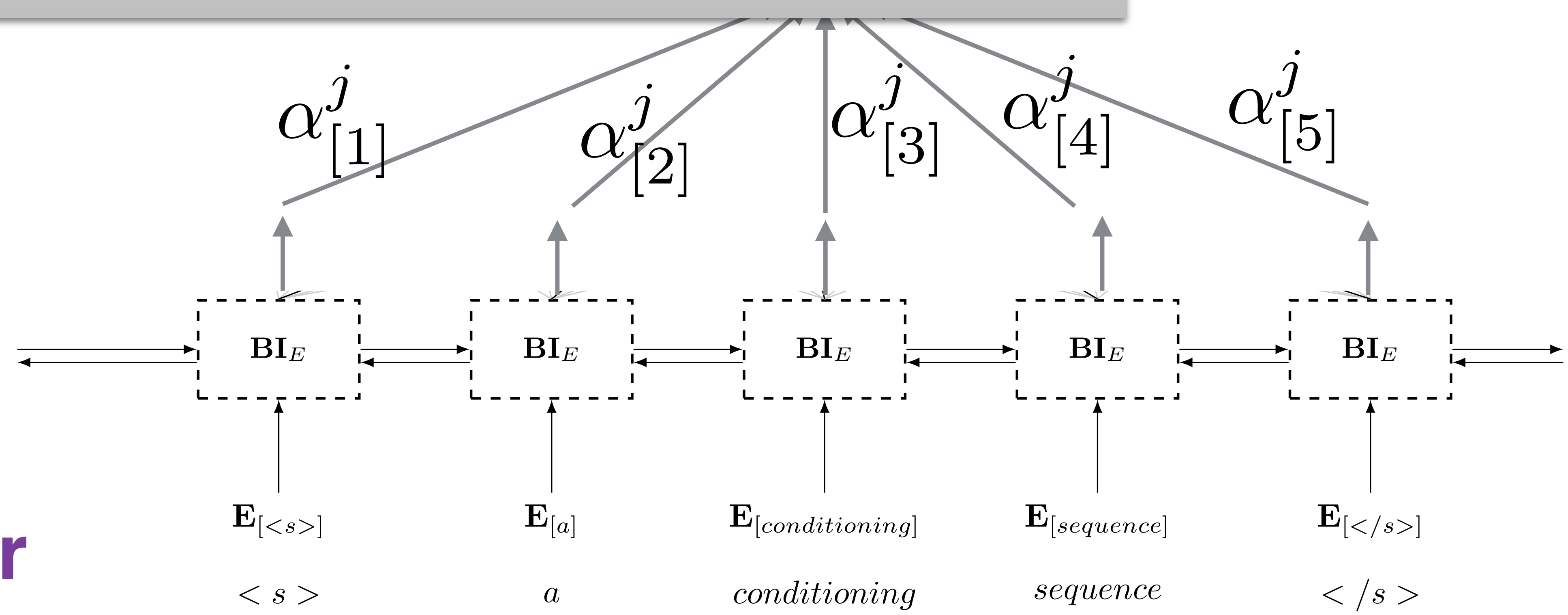
Encoder

Sequence to Sequence conditioned generation

$$\mathbf{c^j} = \sum_{i=1}^{n} \alpha^{\mathbf{j}}_{[\mathbf{i}]} \cdot \mathbf{c_i}$$

$$\alpha^{\mathbf{j}} = \mathrm{softmax}(\bar{\alpha}^{\mathbf{j}}_{[\mathbf{1}]}, \ldots, \bar{\alpha}^{\mathbf{j}}_{[\mathbf{n}]})$$

$$\begin{aligned}\bar{\alpha}^{\mathbf{j}} &= \bar{\alpha}^{\mathbf{j}}_{[\mathbf{1}]}, \ldots, \bar{\alpha}^{\mathbf{j}}_{[\mathbf{n}]} = \\ &= \mathrm{MLP}^{\mathrm{att}}([\mathbf{s_j}; \mathbf{c_1}]), \ldots, \mathrm{MLP}^{\mathrm{att}}([\mathbf{s_j}; \mathbf{c_n}])\end{aligned}$$



Encoder

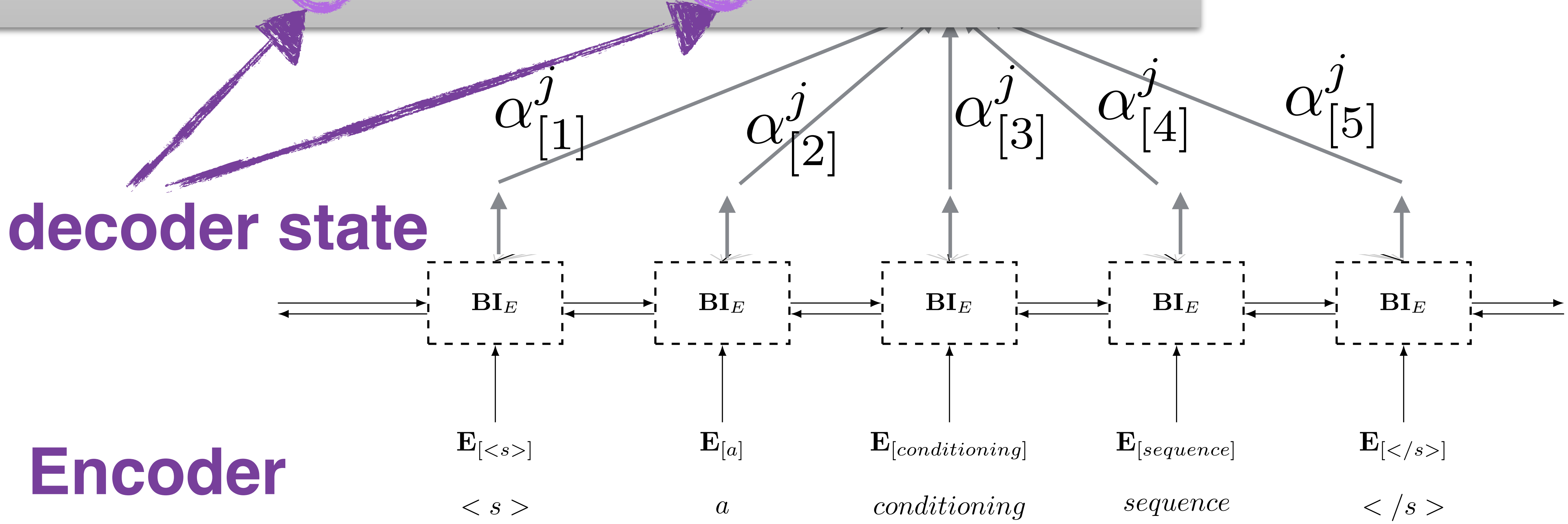
Sequence to Sequence conditioned generation

$$\mathbf{c^j} = \sum_{i=1}^{n} \alpha^{\mathbf{j}}_{[\mathbf{i}]} \cdot \mathbf{c_i}$$

$$\alpha^{\mathbf{j}} = \mathrm{softmax}(\bar{\alpha}^{\mathbf{j}}_{[\mathbf{1}]}, \ldots, \bar{\alpha}^{\mathbf{j}}_{[\mathbf{n}]})$$

$$\bar{\alpha}^{\mathbf{j}} = \bar{\alpha}^{\mathbf{j}}_{[\mathbf{1}]}, \ldots, \bar{\alpha}^{\mathbf{j}}_{[\mathbf{n}]} =$$

$$= \mathrm{MLP}^{\mathrm{att}}([\mathbf{s_j}; \mathbf{c_1}]), \ldots, \mathrm{MLP}^{\mathrm{att}}([\mathbf{s_j}; \mathbf{c_n}])$$

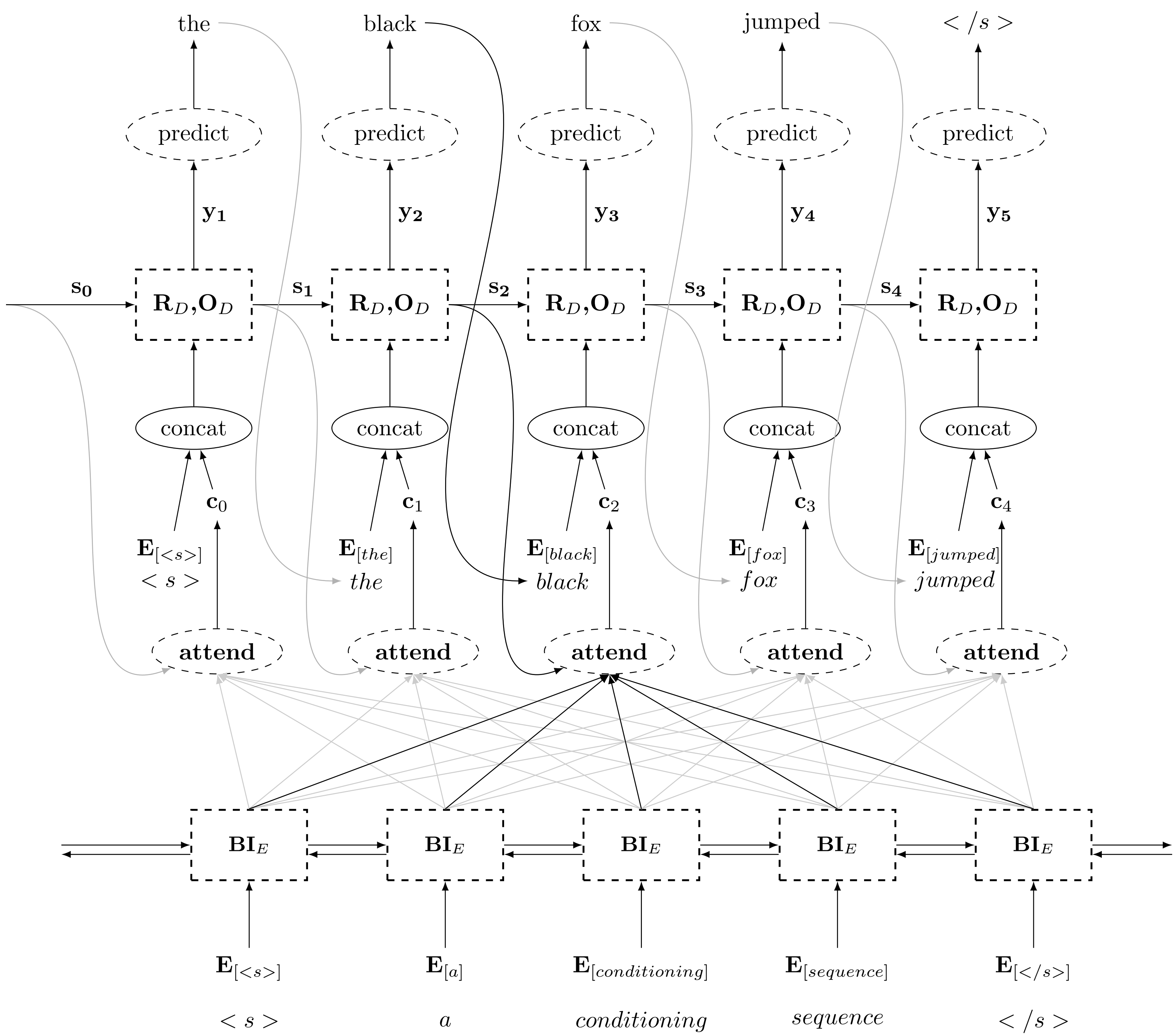


encoder-decoder with attention

$$
\begin{aligned}
p(t_{j+1} = k \mid \hat{t}_{1:j}, \mathbf{x_{1:n}}) &= f(O_{\text{dec}}(\mathbf{s_{j+1}})) \\
\mathbf{s_{j+1}} &= R_{\text{dec}}(\mathbf{s_j}, [\hat{\mathbf{t}}_\mathbf{j}; \mathbf{c^j}]) \\
\mathbf{c^j} &= \sum_{i=1}^{n} \alpha^{\mathbf{j}}_{[\mathbf{i}]} \cdot \mathbf{c_i} \\
\mathbf{c_{1:n}} &= \text{biRNN}^{\star}_{\text{enc}}(\mathbf{x_{1:n}}) \\
\alpha^{\mathbf{j}} &= \text{softmax}(\bar{\alpha}^{\mathbf{j}}_{[\mathbf{1}]}, \ldots, \bar{\alpha}^{\mathbf{j}}_{[\mathbf{n}]}) \\
\bar{\alpha}^{\mathbf{j}}_{[\mathbf{i}]} &= \text{MLP}^{\text{att}}([\mathbf{s_j}; \mathbf{c_i}]) \\
\hat{t}_j &\sim p(t_j \mid \hat{t}_{1:j-1}, \mathbf{x_{1:n}}) \\
f(\mathbf{z}) &= \text{softmax}(\text{MLP}^{\text{out}}(\mathbf{z}))
\end{aligned}
$$

$$
\text{MLP}^{\text{att}}([\mathbf{s_j}; \mathbf{c_i}]) = \mathbf{v} \tanh([\mathbf{s_j}; \mathbf{c_i}]\mathbf{U} + \mathbf{b})
$$

encoder-decoder with attention



encoder-decoder with attention

- Encoder encodes a sequence of vectors, $c_1,...,c_n$
- At each decoding stage, an MLP assigns a relevance score to each Encoder vector.
- The relevance score is based on c_i and the state s_j
- Weighted-sum (based on relevance) is used to produce the conditioning context for decoder step j.

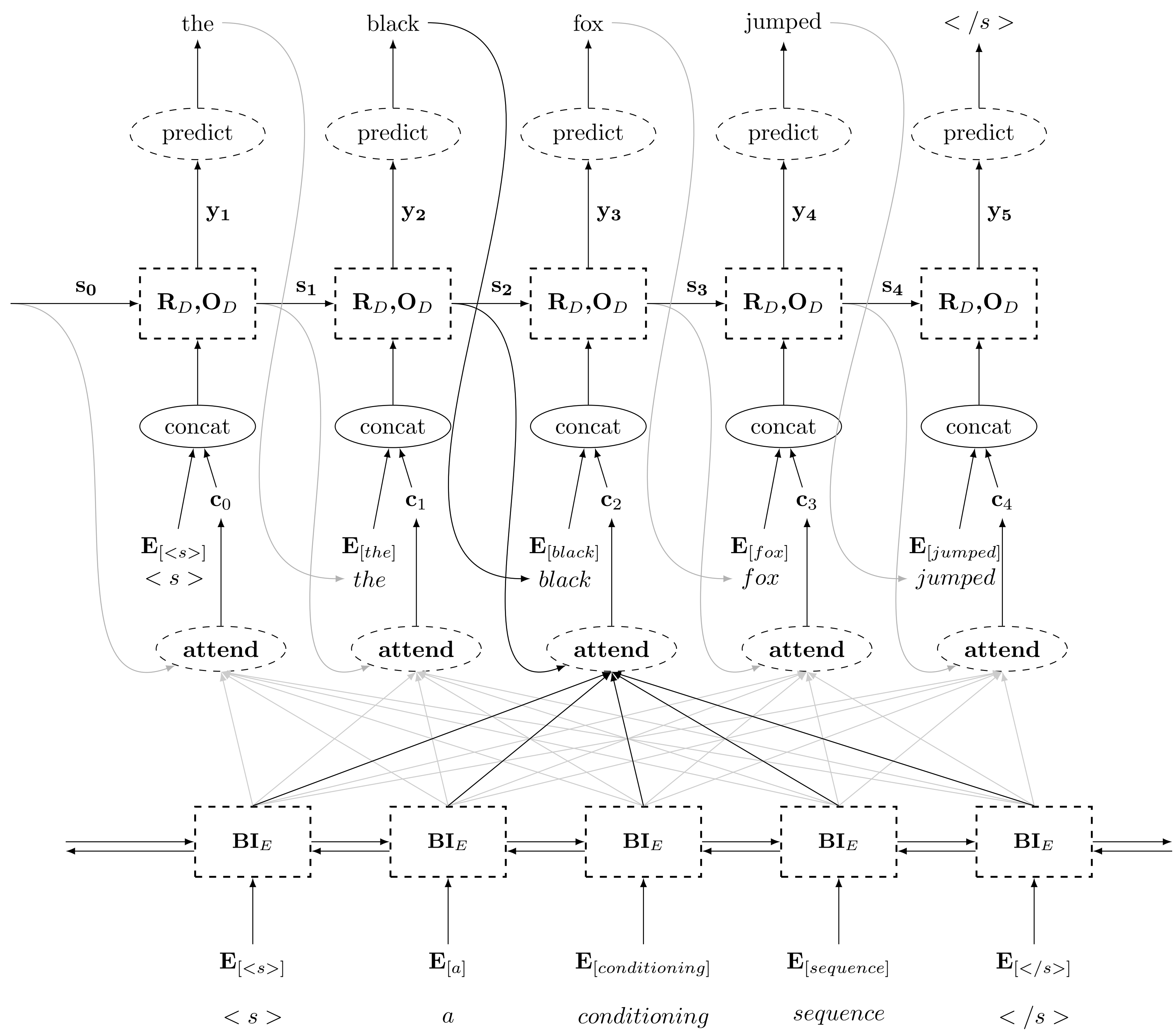
encoder-decoder with attention

- Decoder "pays attention" to different parts of the encoded sequence at each stage.
- The attention mechanism is "soft" -- it is a mixture of encoder states.

encoder-decoder with attention

- The encoder acts as a read-only memory for the decoder.
- The decoder chooses what to read at each stage.

encoder-decoder with attention



Attention

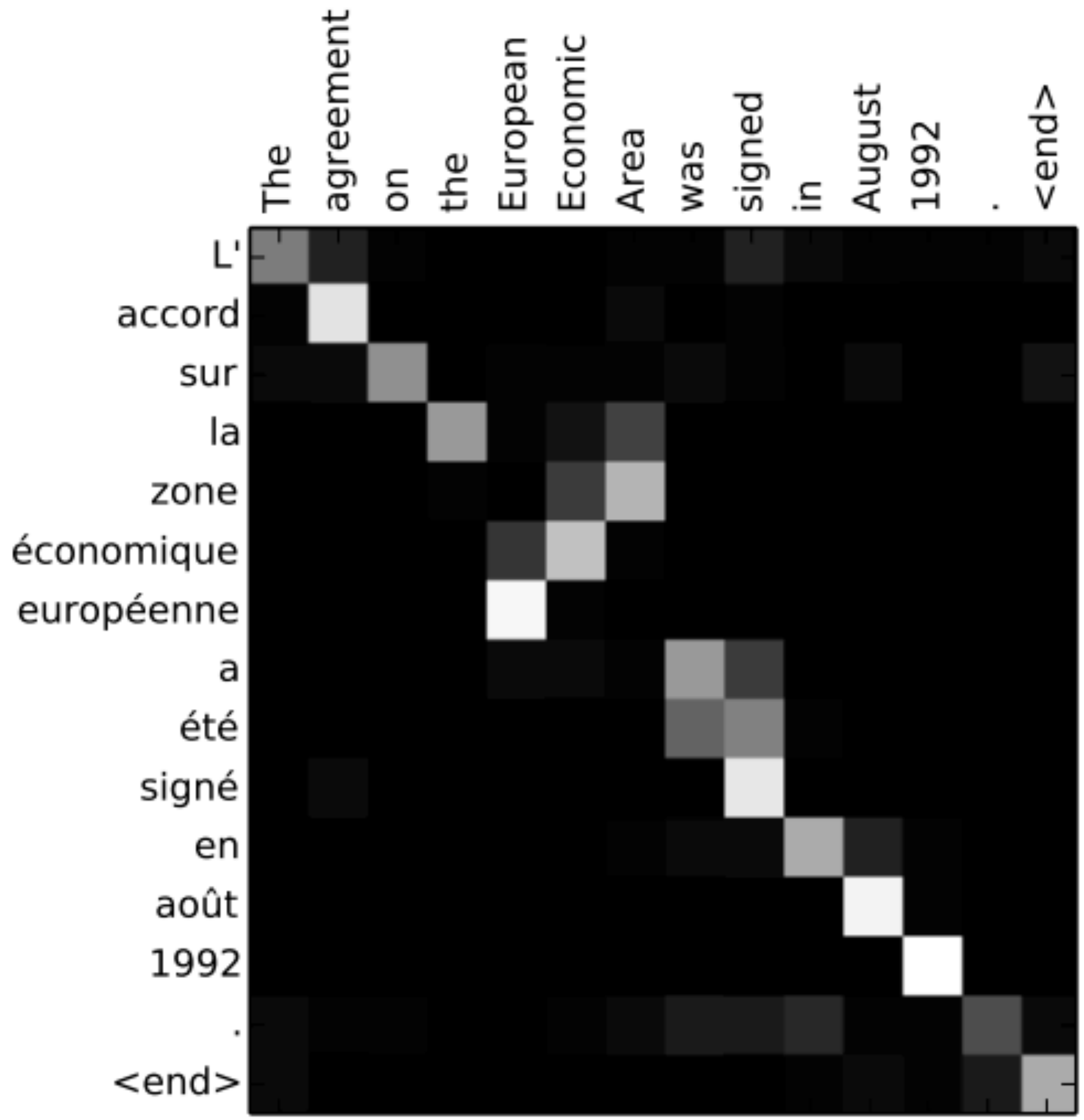
- Attention is very effective for sequence-to-sequence tasks.
- Current state-of-the-art systems all use attention. (this is basically how Machine Translation works)

Attention

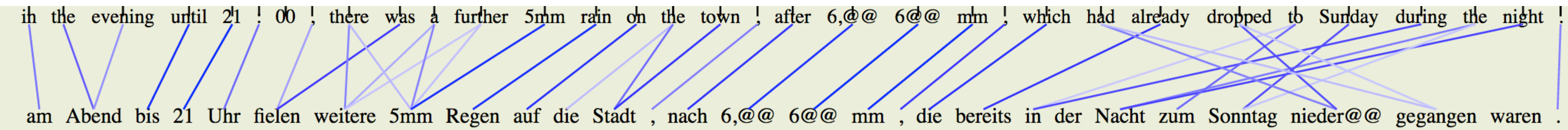
- Attention also makes models somewhat more interpretable.

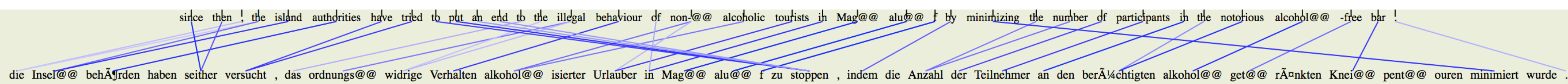
  (we can see where the model is "looking" at each stage of the prediction process)

Attention


The
agreement
on
the
European
Economic
Area
was
signed
in
August
1992
.
<end>
L'
accord
sur
la
zone
économique
européenne
a
été
signé
en
août
1992
.
<end>

Attention


in the evening until 21 ! 00 ! there was a further 5mm rain on the town ! after 6,@@ 6@@ mm ! which had already dropped to Sunday during the night !
am Abend bis 21 Uhr fielen weitere 5mm Regen auf die Stadt , nach 6,@@ 6@@ mm , die bereits in der Nacht zum Sonntag nieder@@ gegangen waren .


since then ! the island authorities have tried to put an end to the illegal behaviour of non-@@ alcoholic tourists in Mag@@ alu@@ f by minimizing the number of participants in the notorious alcohol@@ -free bar !
die Insel@@ behÃ¶rden haben seither versucht , das ordnungs@@ widrige Verhalten alkohol@@ isierter Urlauber in Mag@@ alu@@ f zu stoppen , indem die Anzahl der Teilnehmer an den berÃ¼chtigten alkohol@@ get@@ rÃ¤nkten Knei@@ pent@@ ouren minimiert wurde .

Complexity

- Encoder decoder:
- Encoder-decoder with attention:

Complexity

- Encoder decoder: $O(n + m)$
- Encoder-decoder with attention: $O(n \times m)$

Complexity

- Encoder decoder: $O(n + m)$
- Encoder-decoder with attention: $O(n \times m)$

Where/how can you parallelize?
in train time?
in test time?

Beyond Seq2Seq

- Can think of a general design pattern in neural nets:
  - **Input**: sequence, query
    - **Encode** the input into a sequence of vectors
    - **Attend** to the encoded vectors, based on query (weighted sum, determined by query)
    - **Predict** based on the attended vector

Attention More Abstractly

- Input sequence $\mathbf{x_1,...,x_n}$
- Query vector $\mathbf{q}$
- Attention weights $\mathbf{a}_{\mathbf{[1,...,n]}}$
  $= \text{softmax}(\text{score}(\mathbf{q},\mathbf{x_1}), ..., \text{score}(\mathbf{q},\mathbf{x_n}))$
- Result vector $\mathbf{v}$ = sum $\mathbf{a_i}$*$\mathbf{x_i}$
  $= \text{sum softmax}(\text{score}(\mathbf{q},\mathbf{x_1}), ..., \text{score}(\mathbf{q},\mathbf{x_n}))_{[i]} * \mathbf{x_i}$

How to Attend?

v: attended vec, **q**: query vec

- MLP:

$$\mathbf{u}g(\mathbf{W^1}\mathbf{v} + \mathbf{W^2}\mathbf{q})$$

- dot product:

- biaffine transform:

How to Attend?

v: attended vec, **q**: query vec

- MLP:

$$\mathbf{u}g(\mathbf{W^1}\mathbf{v} + \mathbf{W^2}\mathbf{q})$$

- dot product:

$$\mathbf{v} \cdot \mathbf{q}$$

- biaffine transform:

How to Attend?

$\mathbf{v}$: attended vec, $\mathbf{q}$: query vec

- MLP:

$$\mathbf{u}g(\mathbf{W^1}\mathbf{v} + \mathbf{W^2}\mathbf{q})$$

- dot product:

$$\mathbf{v} \cdot \mathbf{q}$$

Scaled dot product:

$$\mathbf{v} \cdot \mathbf{q}_{/\sqrt{\mathrm{dim}(\mathbf{v})}}$$

- biaffine transform:

How to Attend?

v: attended vec, **q**: query vec

- MLP:

$$\mathbf{u}g(\mathbf{W^1 v} + \mathbf{W^2 q})$$

- dot product:

$$\mathbf{v} \cdot \mathbf{q}$$

Scaled dot product:

$$\mathbf{v} \cdot \mathbf{q}_{/\sqrt{\dim(\mathbf{v})}}$$

- biaffine transform:

$$\mathbf{v}^\top \mathbf{W} \mathbf{q}$$

How to Attend?

v: attended vec, **q**: query vec

- MLP:

$$\mathbf{u}g(\mathbf{W^1v} + \mathbf{W^2q})$$

- dot product:

$$\mathbf{v} \cdot \mathbf{q}$$

Scaled dot product:

$$\mathbf{v} \cdot \mathbf{q}_{/\sqrt{\dim(\mathbf{v})}}$$

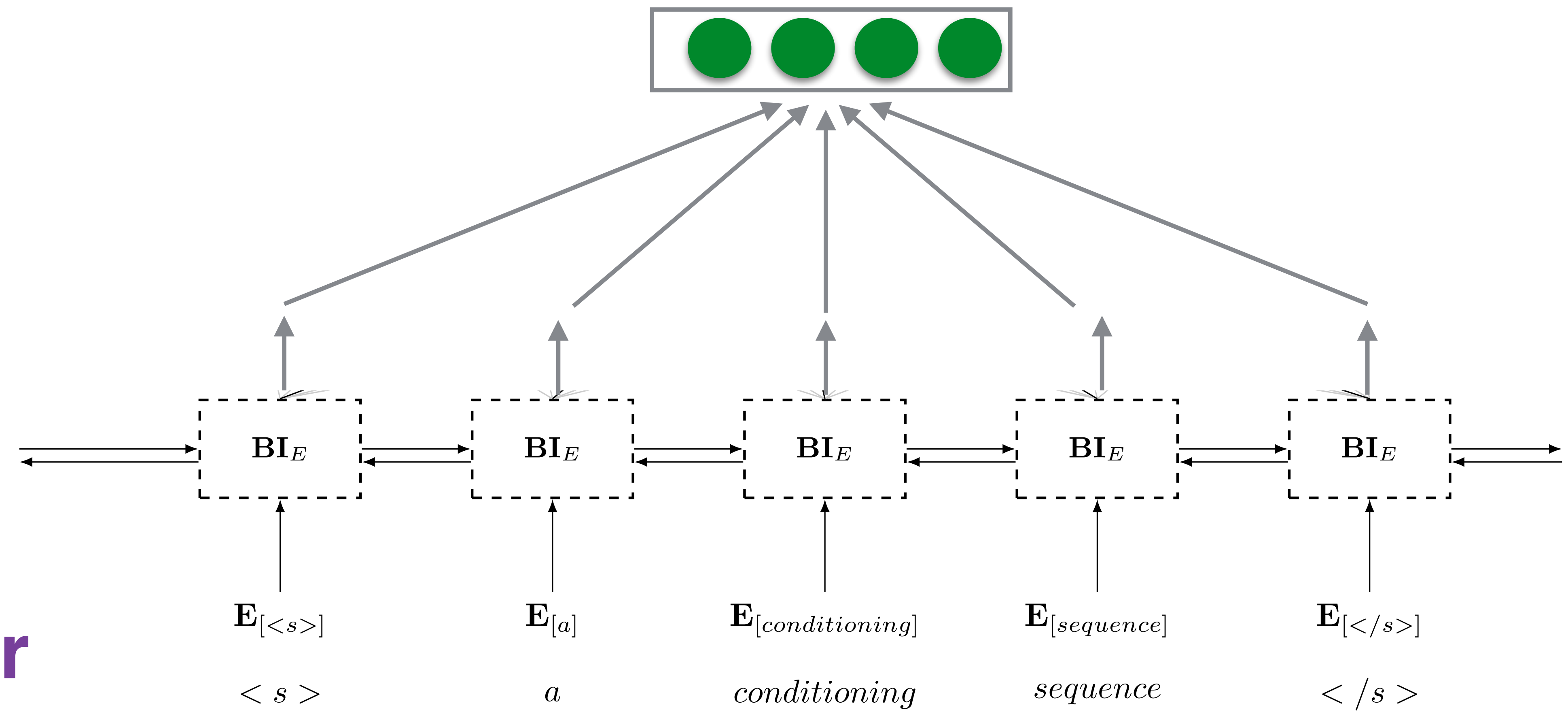
- biaffine transform:

$$\mathbf{v}^\top \mathbf{W} \mathbf{q}$$

Pros? Cons?

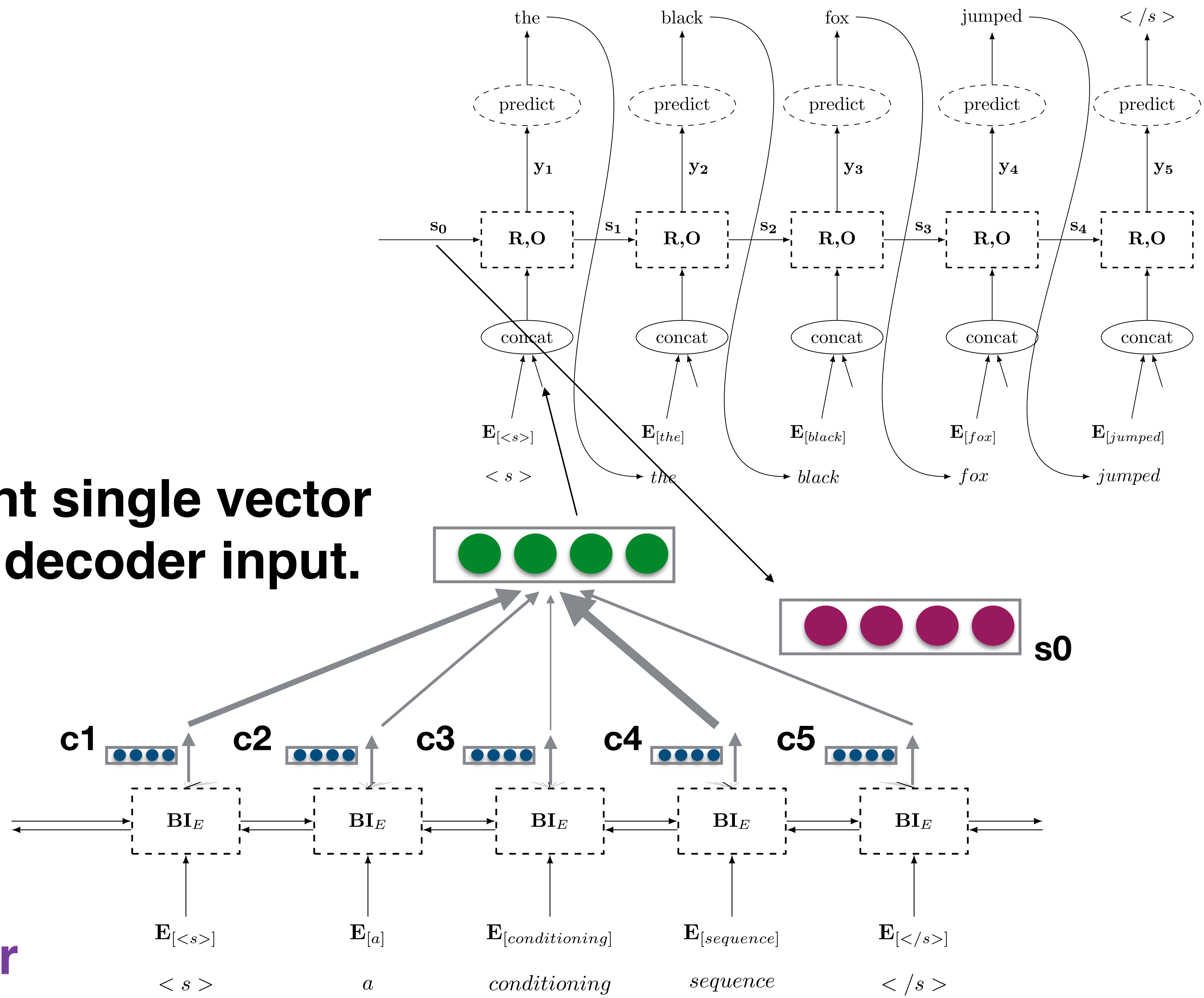
attention: more graphically

we can combine the different outputs into a single vector



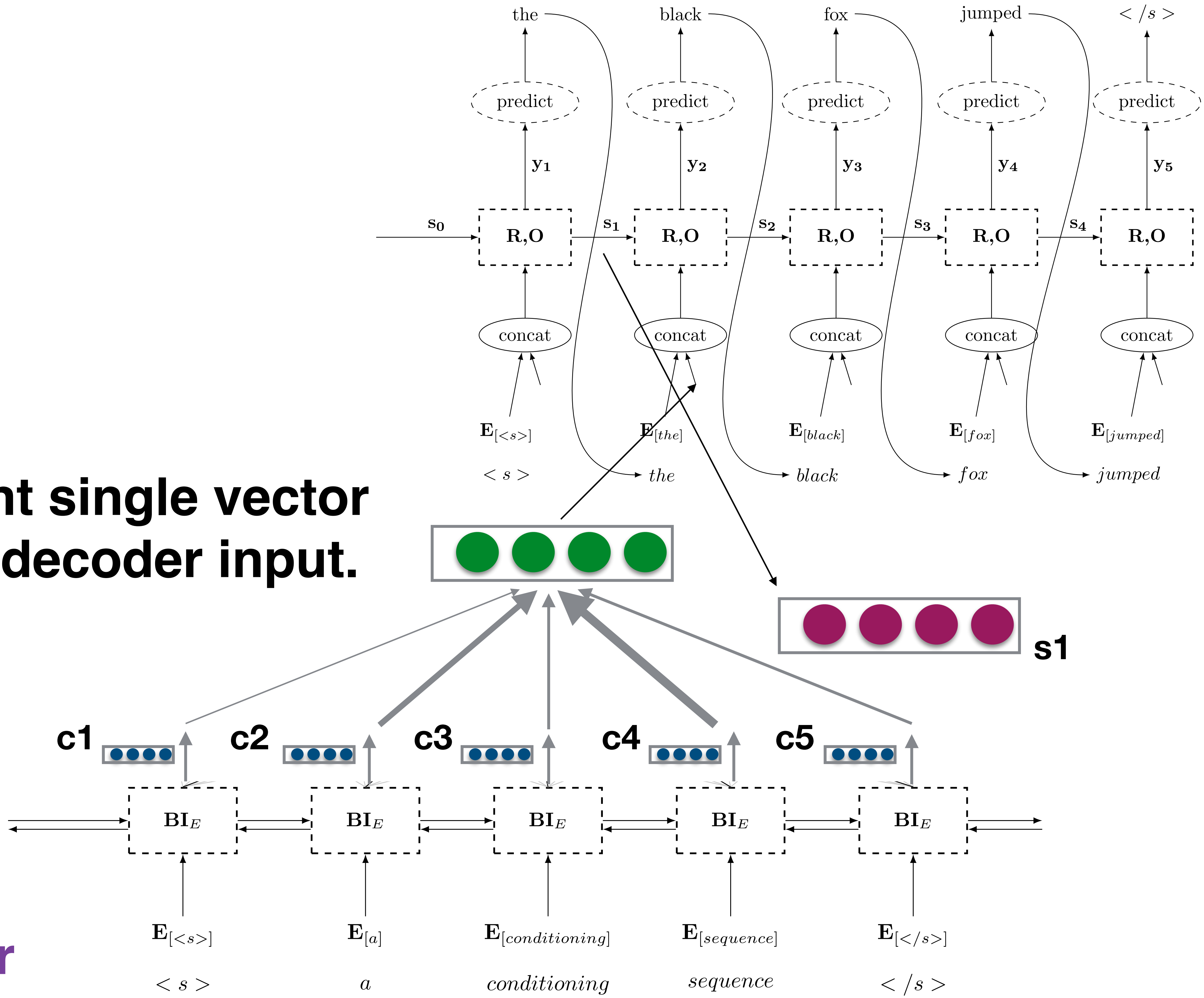
Encoder

a different single vector at each decoder input.



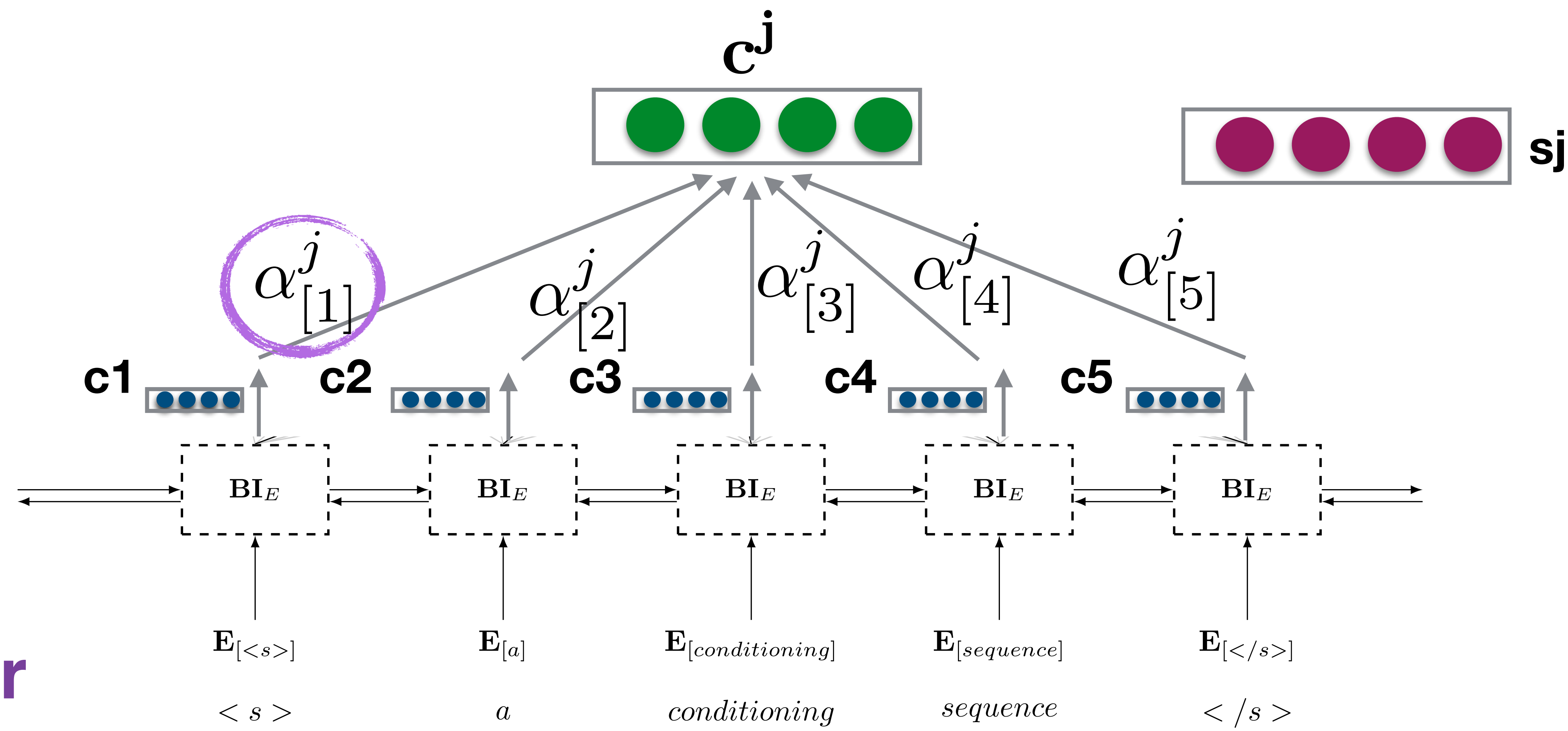
Encoder

a different single vector at each decoder input.



Encoder

$$\mathbf{c^j} = \sum_{i=1}^{n} \alpha^{\mathbf{j}}_{[\mathbf{i}]} \cdot \mathbf{c_i}$$

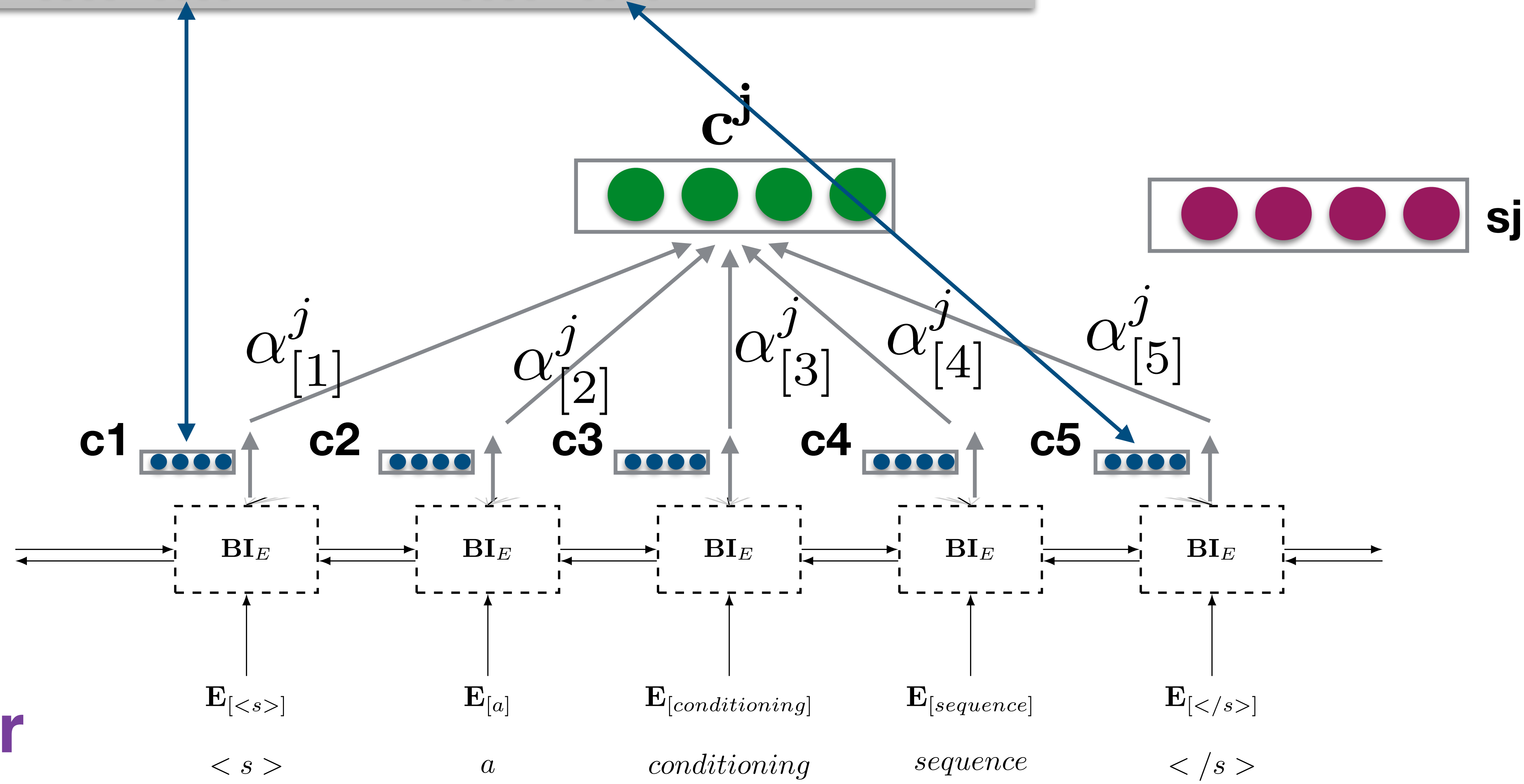


Encoder

$$\mathbf{c^j} = \sum_{i=1}^{n} \alpha^{\mathbf{j}}_{[\mathbf{i}]} \cdot \mathbf{c_i}$$

$$\alpha^{\mathbf{j}} = \mathrm{softmax}(\bar{\alpha}^{\mathbf{j}}_{[\mathbf{1}]}, \ldots, \bar{\alpha}^{\mathbf{j}}_{[\mathbf{n}]})$$

$$\begin{aligned}\bar{\alpha}^{\mathbf{j}} &= \bar{\alpha}^{\mathbf{j}}_{[\mathbf{1}]}, \ldots, \bar{\alpha}^{\mathbf{j}}_{[\mathbf{n}]} = \\ &= \mathrm{MLP}^{\mathrm{att}}([\mathbf{s_j}; \mathbf{c_1}]), \ldots, \mathrm{MLP}^{\mathrm{att}}([\mathbf{s_j}; \mathbf{c_n}])\end{aligned}$$



Encoder

Attention More Abstractly

- Input sequence $\mathbf{x}_1,\ldots,\mathbf{x}_n$
- Query vector $\mathbf{q}$
- Attention weights $\mathbf{a}_{[1,\ldots,n]}$
  $= \mathrm{softmax}(\mathrm{score}(\mathbf{q},\mathbf{x}_1), \ldots, \mathrm{score}(\mathbf{q},\mathbf{x}_n))$
- Result vector $\mathbf{v} = \mathrm{sum}\ \mathbf{a}_i * \mathbf{x}_i$
  $= \mathrm{sum}\ \mathrm{softmax}(\mathrm{score}(\mathbf{q},\mathbf{x}_1), \ldots, \mathrm{score}(\mathbf{q},\mathbf{x}_n))_{[i]} * \mathbf{x}_i$

How to Attend?

v: attended vec, **q**: query vec

- MLP:

$$\mathbf{u}g(\mathbf{W^1}\mathbf{v} + \mathbf{W^2}\mathbf{q})$$

- dot product:

$$\mathbf{v} \cdot \mathbf{q}$$

Scaled dot product:

$$\mathbf{v} \cdot \mathbf{q} / \sqrt{\dim(\mathbf{v})}$$

- biaffine transform:

$$\mathbf{v}^\top \mathbf{W} \mathbf{q}$$

Alternatives

- Soft vs. **Hard** attention
- Why use a biRNN encoder and not just the use word-vectors (embeddings) **directly**?
- What if the sequences are **mostly monotonic**?

Attention vs. No Attention

- When would you use an Encoder-Decoder without attention?

RNNs --> Transformers

Transformer

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Transformer

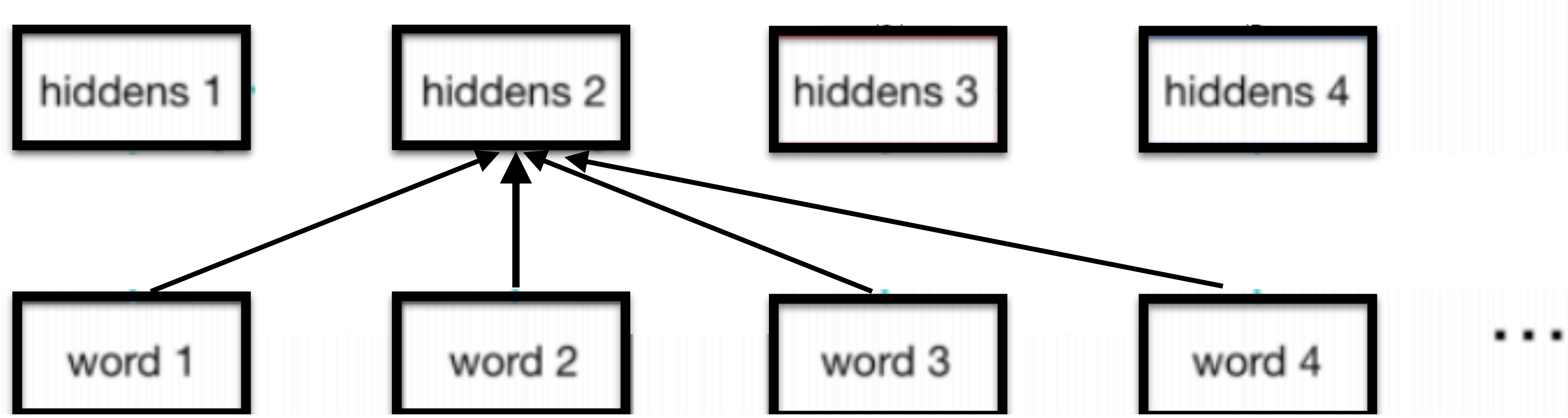
replace RNN with attention-based mechanism

- Main concepts to know:
  - Self-attention
  - Multi-head attention
- Also think about: why do this? what is the motivation?

Transformer

Self attention

each token attends to all tokens in previous layer

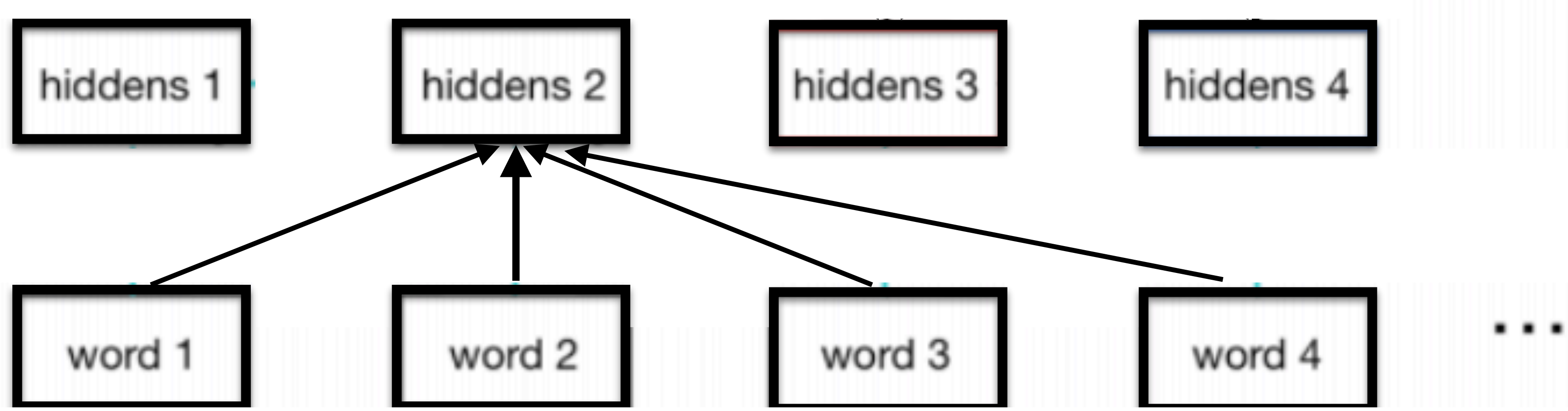


Transformer

Self attention

each token attends to all tokens in previous layer

$$att_{x_i} = softmax(dot(x_i, x_1), dot(x_i, x_2), ..., dot(x_i, x_n))_{[i]} x_i$$



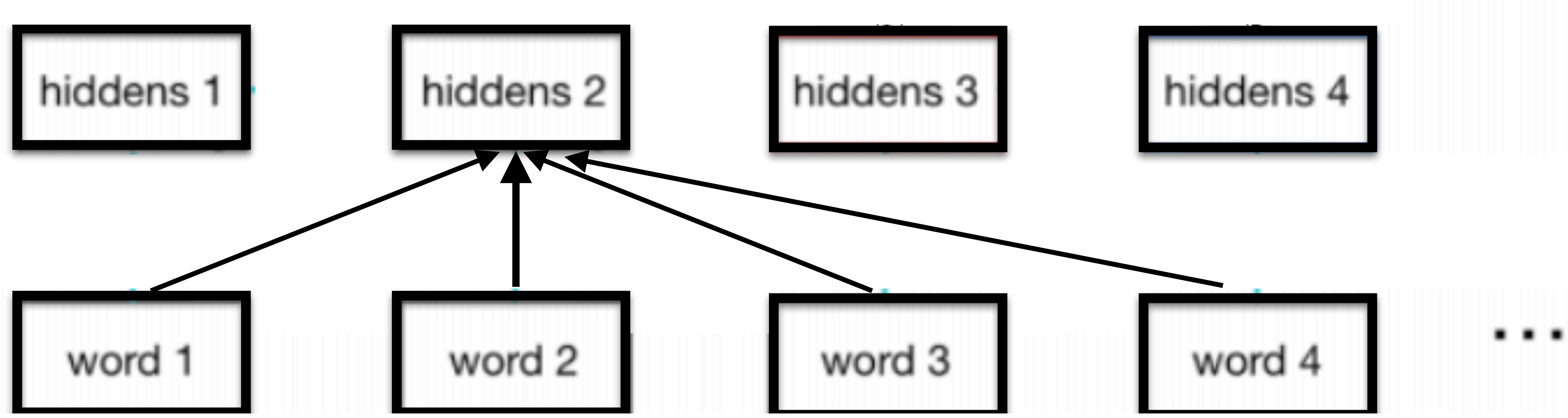
Transformer

Self attention

each token attends to all tokens in previous layer

transform x into Q, K, V

$$att_{x_i} = softmax(dot(W^Q x_i, W^K x_1), dot(W^Q x_i, W^K x_2), ..., dot(W^Q x_i, W^K x_n))_{[i]} W^V x_i$$



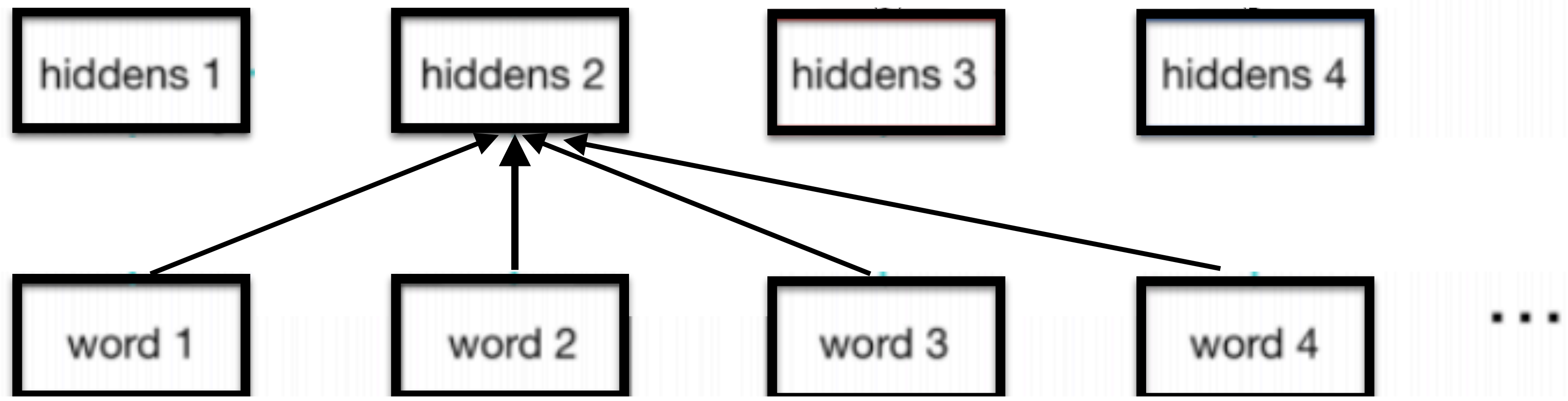
Transformer

Self attention

each token attends to all tokens in previous layer

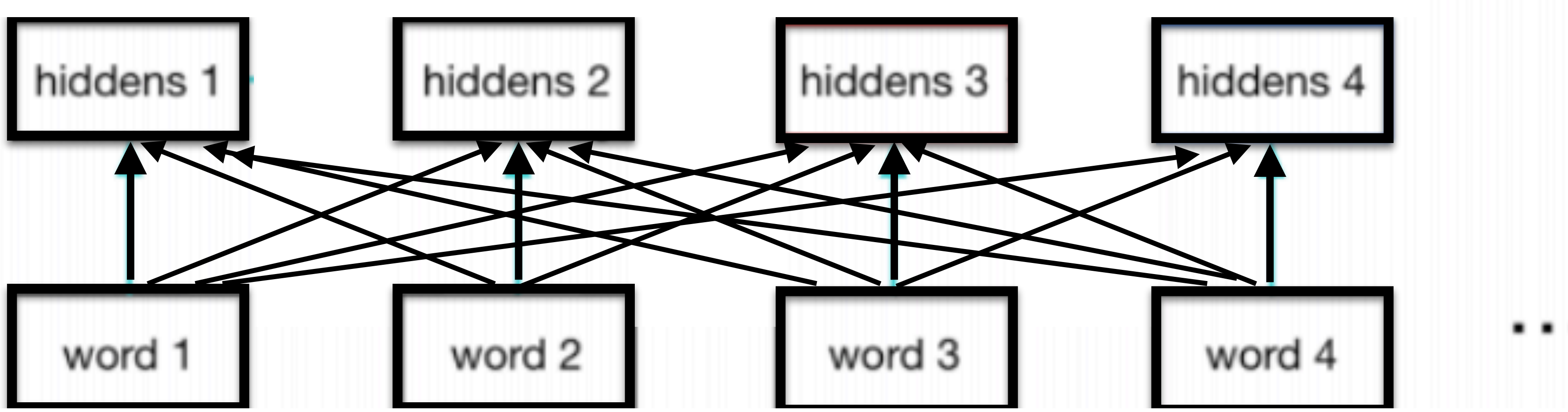
transform x into Q, K, V

$$att_{x_i} = softmax(dot(\underbrace{W^Q x_i}_{q_i}, \underbrace{W^K x_1}_{k_1}), dot(\underbrace{W^Q x_i}_{q_i}, \underbrace{W^K x_2}_{k_2}), ..., dot(\underbrace{W^Q x_i}_{q_i}, \underbrace{W^K x_n}_{k_n}))_{[i]} \underbrace{W^V x_i}_{v_i}$$



Transformer

Self attention

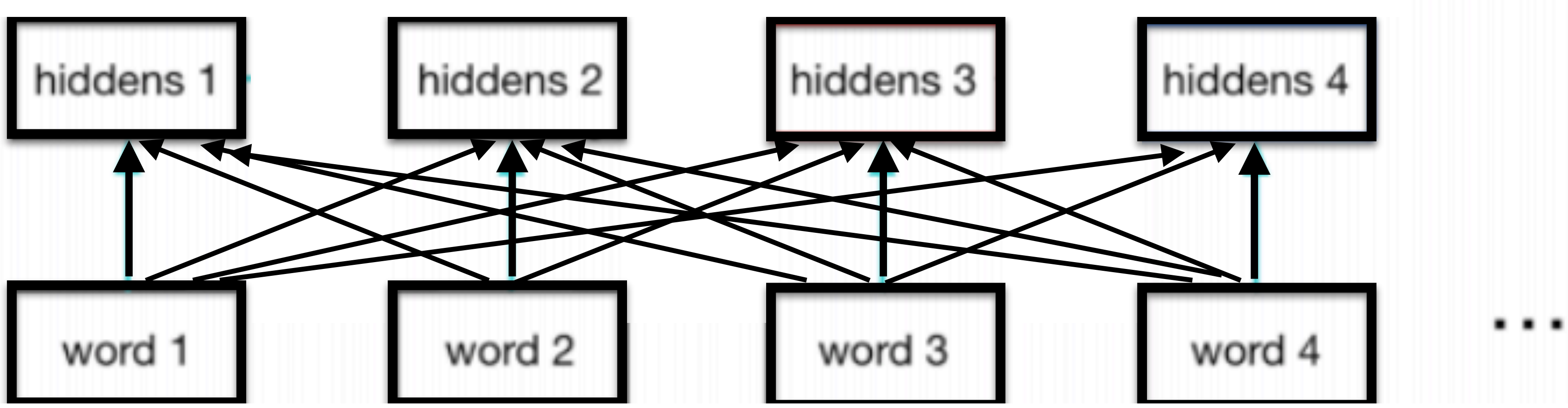


Transformer

Self attention

matrix form:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(QK^T)V$$

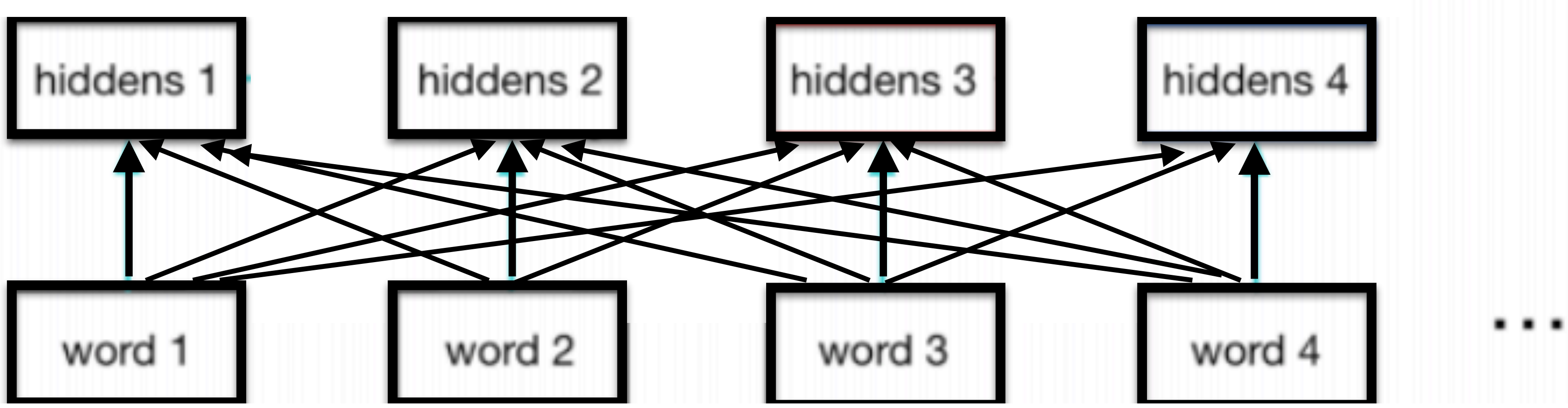


Transformer

Self attention

matrix form + scaled attention:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

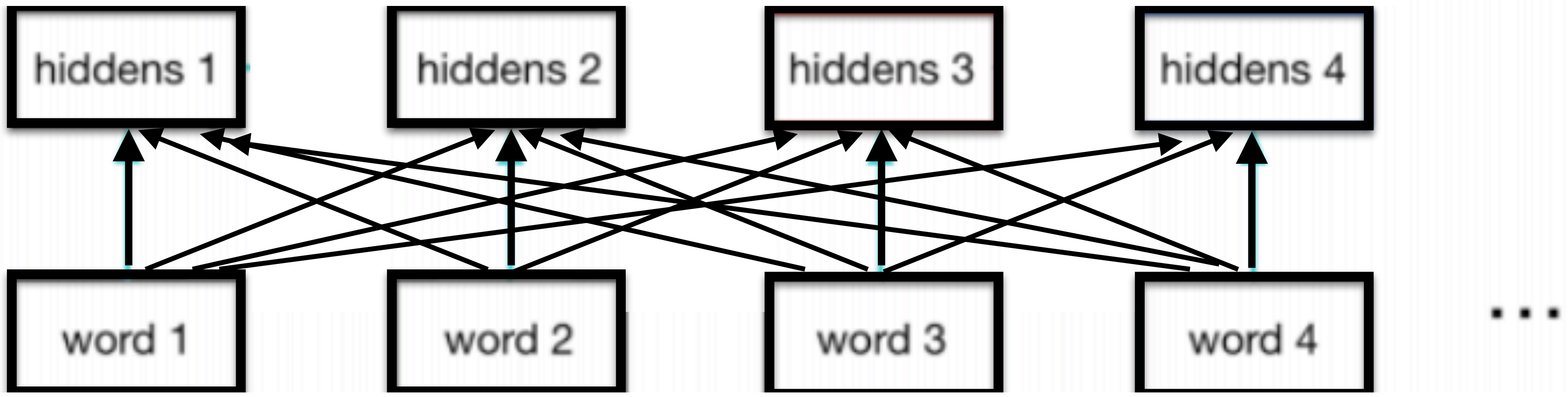


Transformer

We suspect that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients [4]. To counteract this effect, we scale the dot products by $\frac{1}{\sqrt{d_k}}$.

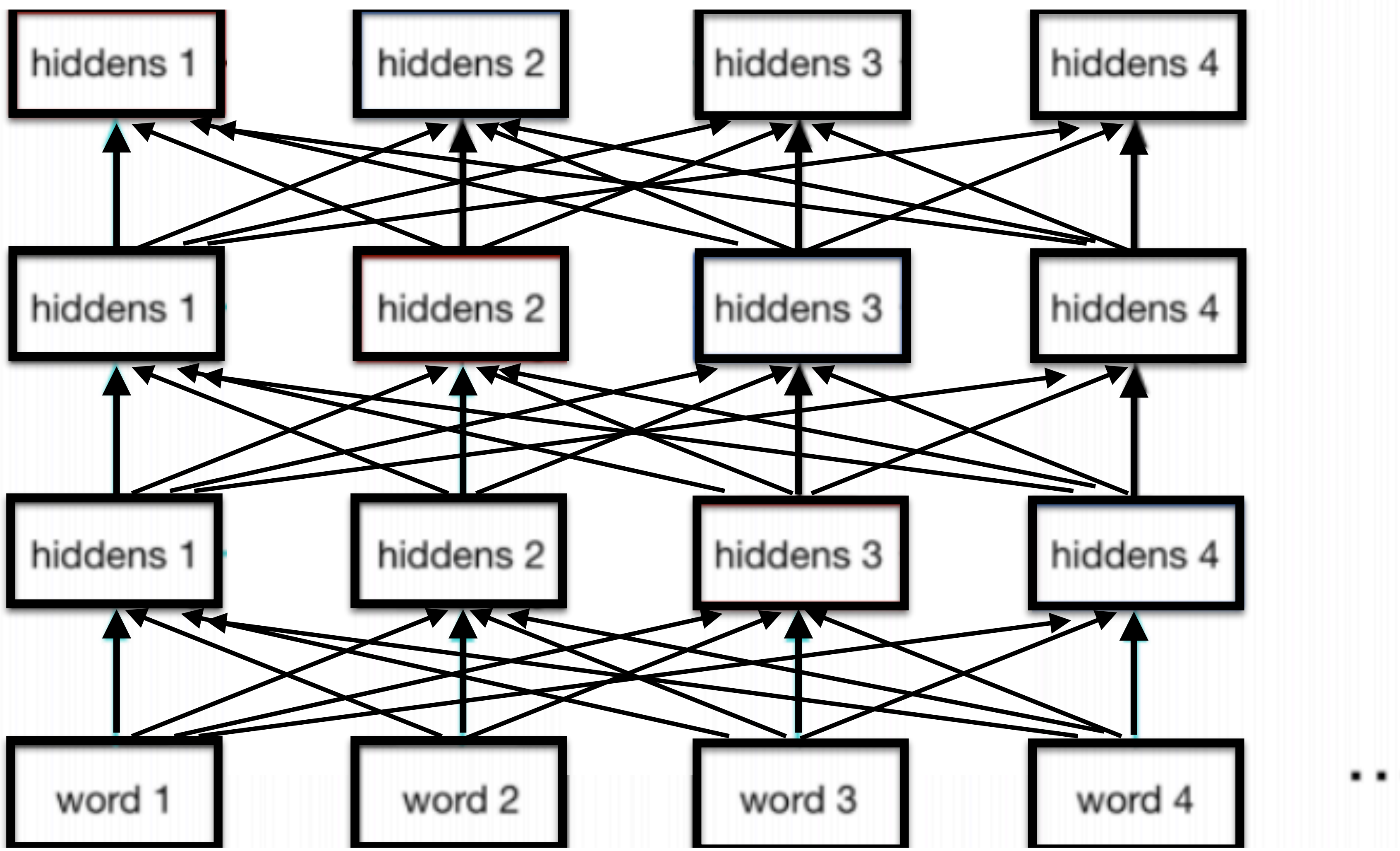
matrix form + scaled attention:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



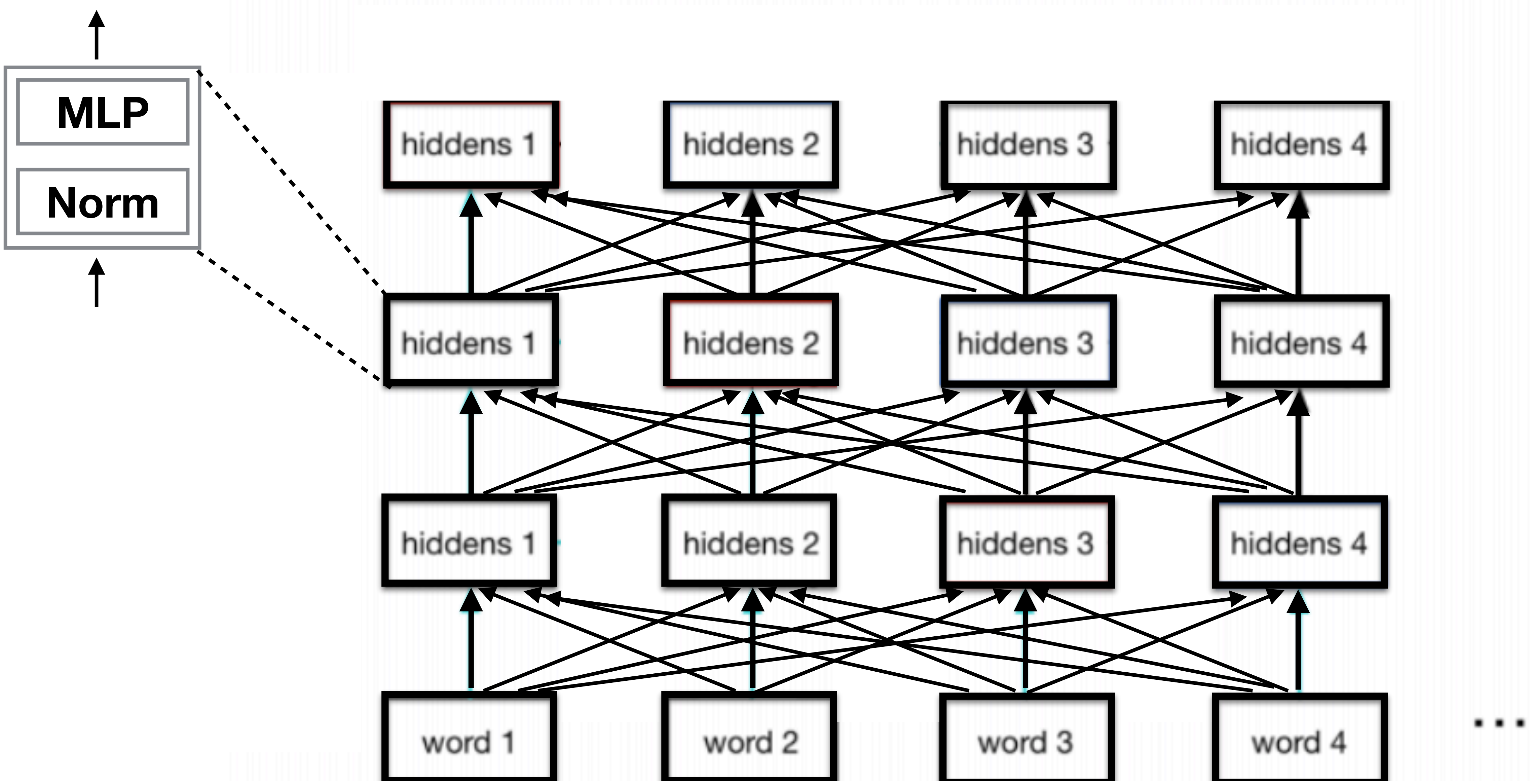
Transformer

Self attention



Transformer

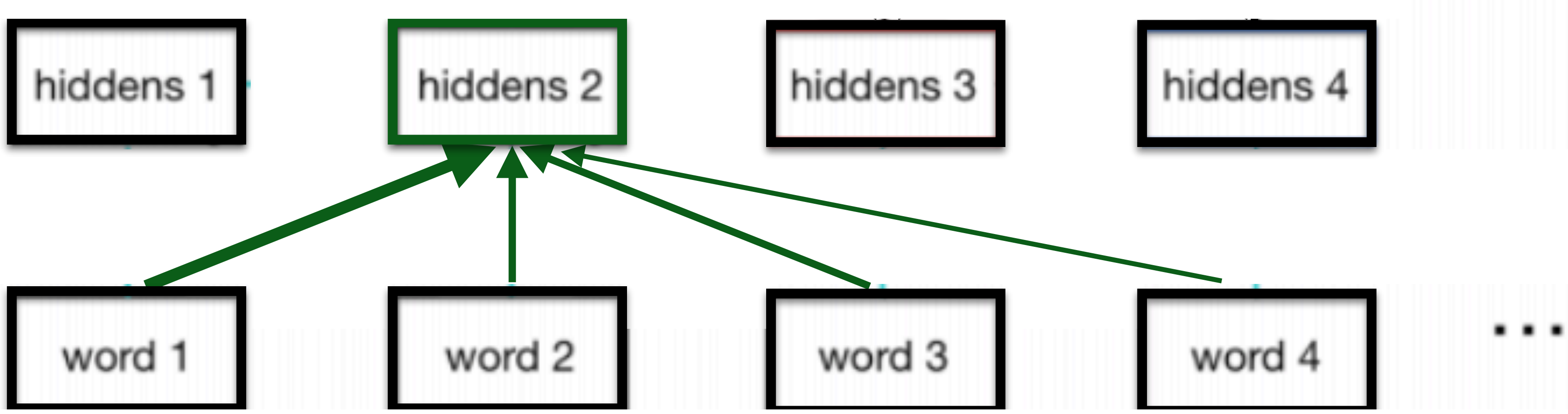
Each attention layer is followed by Layer-norm and MLP



Transformer

multi-head attention

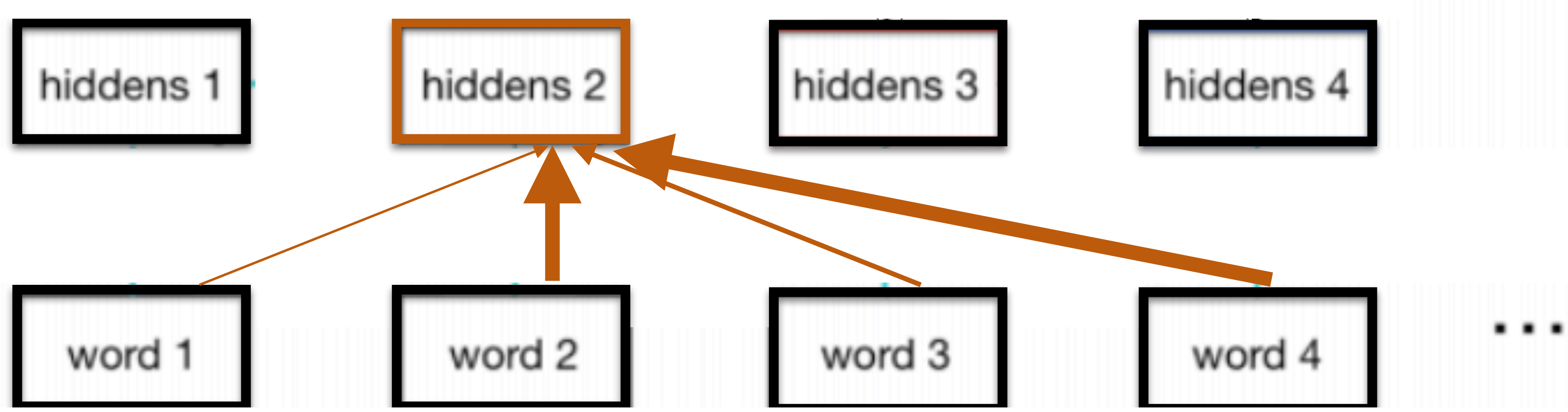
one attention pattern



Transformer

multi-head attention

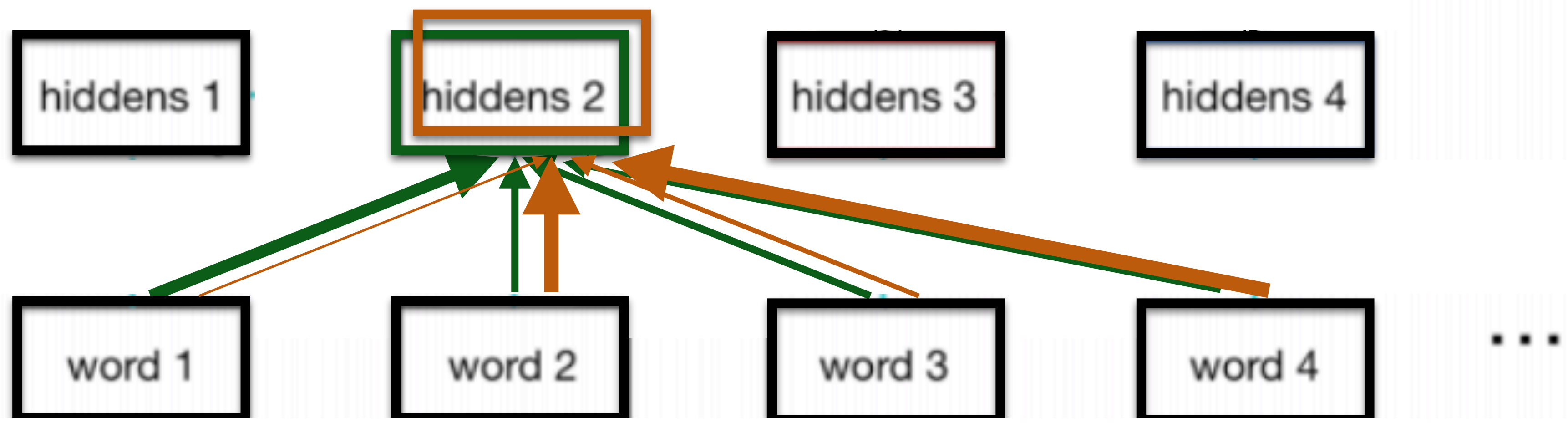
another attention pattern



Transformer

multi-head attention

why chose if we can just have several?

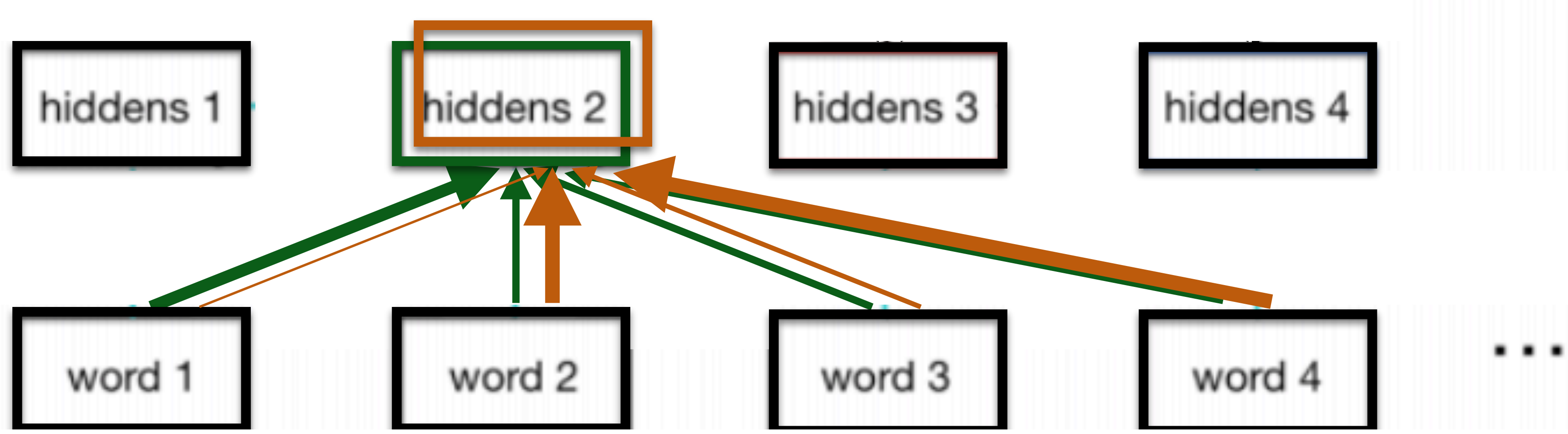


Transformer

multi-head attention

why chose if we can just have several?

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$



Transformer

multi-head attention

why chose if we can just have several?

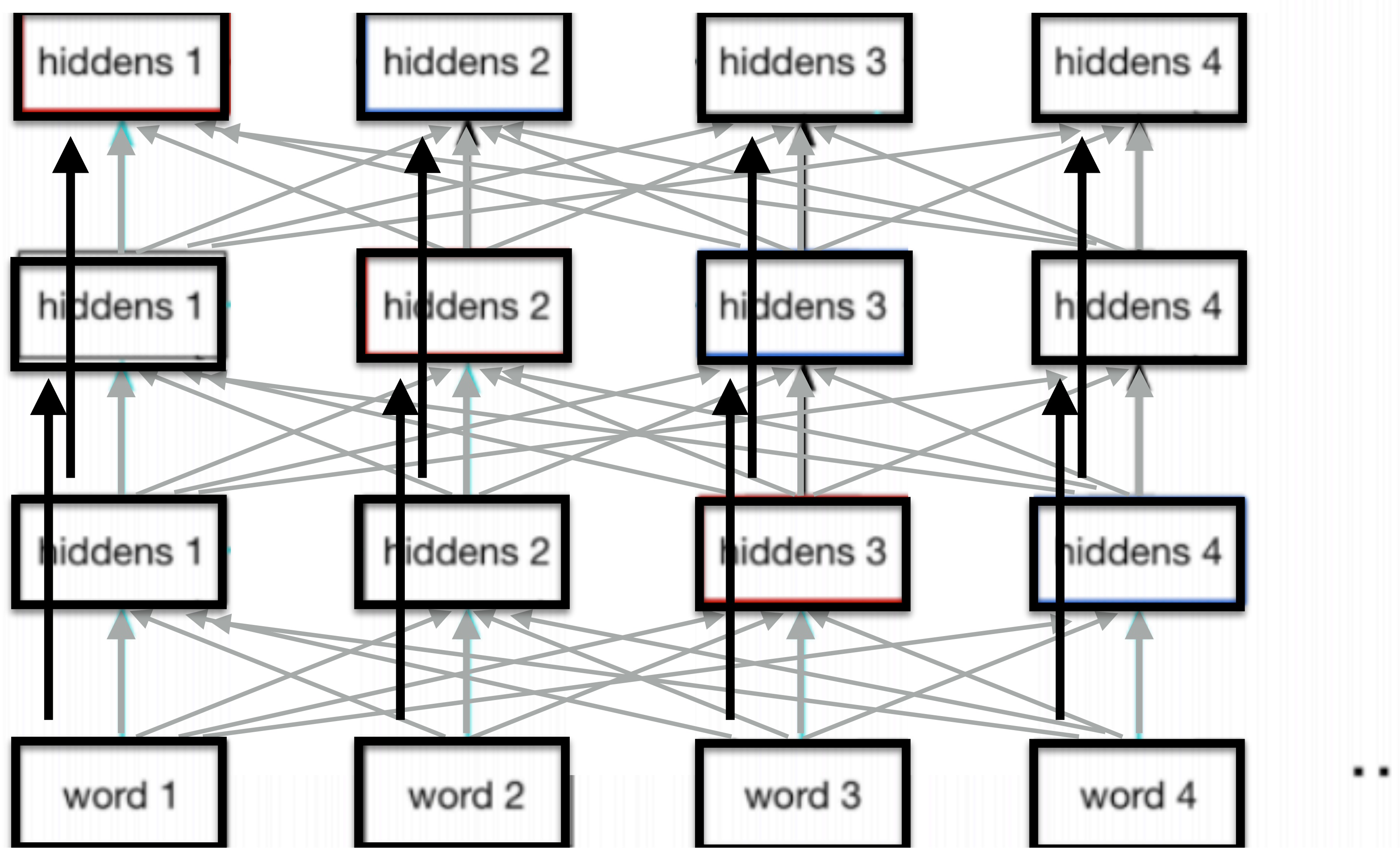
$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where}\ \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{\text{model}}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

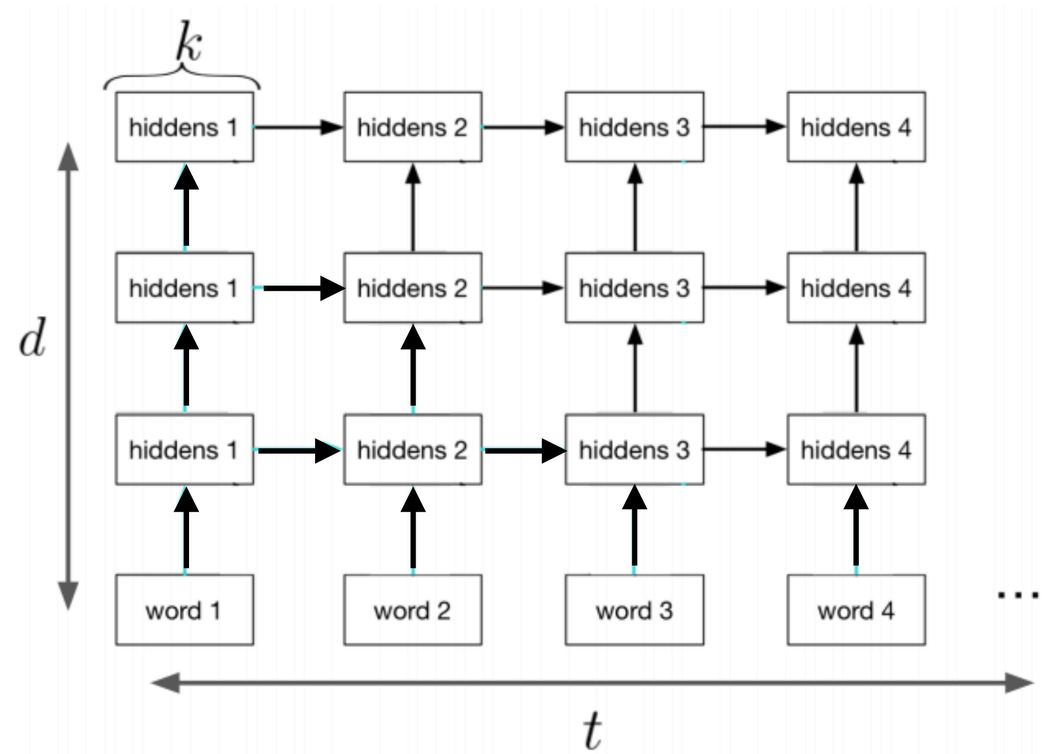
Transformer

Skip connections



Cost vs Opportunity

- Consider a standard d layer RNN from Lecture 13 with k hidden units, training on a sequence of length t.

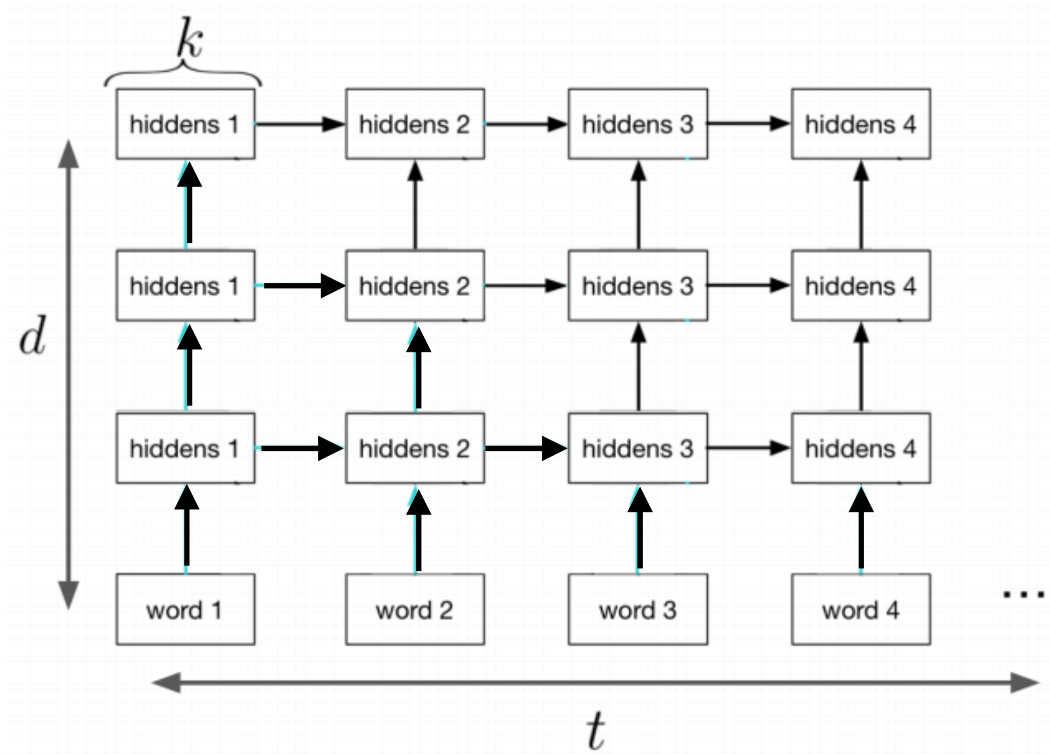


- There are k^2 connections for each hidden-to-hidden connection. A total of $t \times k^2 \times d$ connections.
- We need to store all $t \times k \times d$ hidden units during training.
- Only $k \times d$ hidden units need to be stored at test time.

http://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/slides/lec16.pdf

Cost vs Opportunity

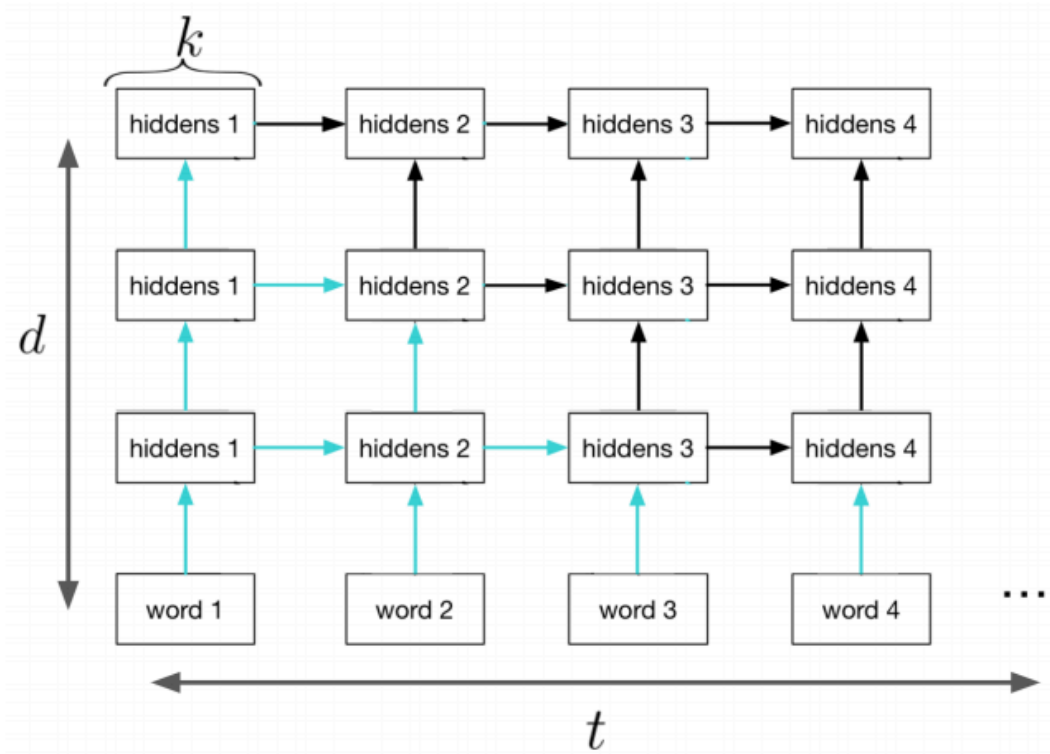
- Consider a standard d layer RNN from Lecture 13 with k hidden units, training on a sequence of length t.



- Which hidden layers can be computed in parallel in this RNN?

Cost vs Opportunity

- Consider a standard d layer RNN from Lecture 13 with k hidden units, training on a sequence of length t.

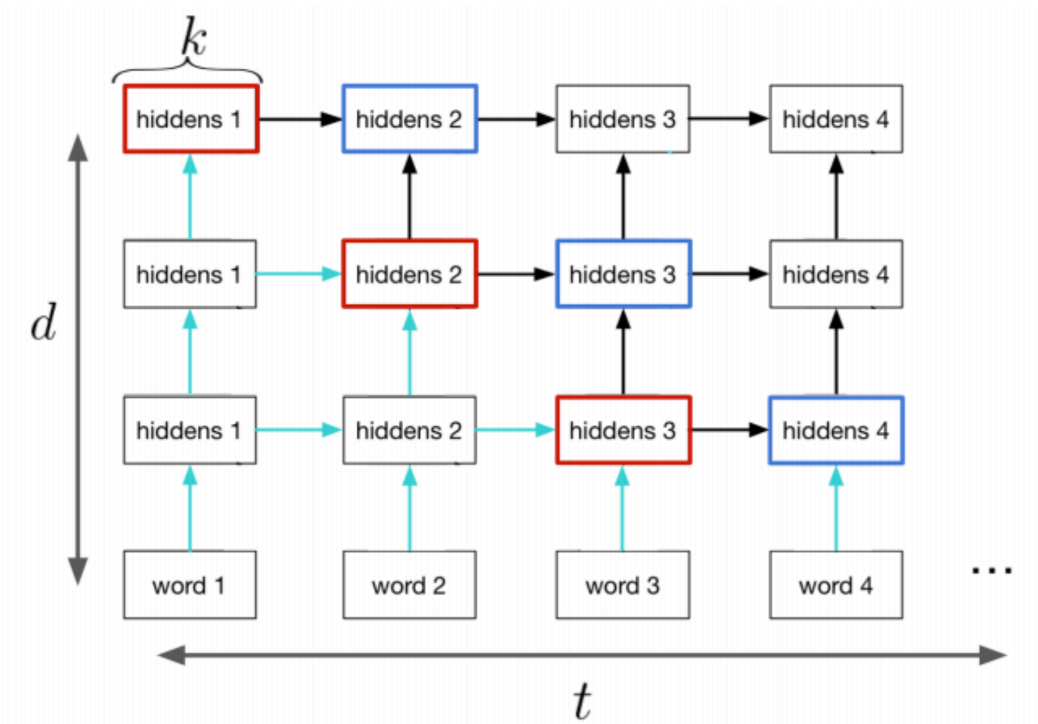


- Which hidden layers can be computed in parallel in this RNN?

http://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/slides/lec16.pdf

Cost vs Opportunity

- Consider a standard d layer RNN from Lecture 13 with k hidden units, training on a sequence of length t.

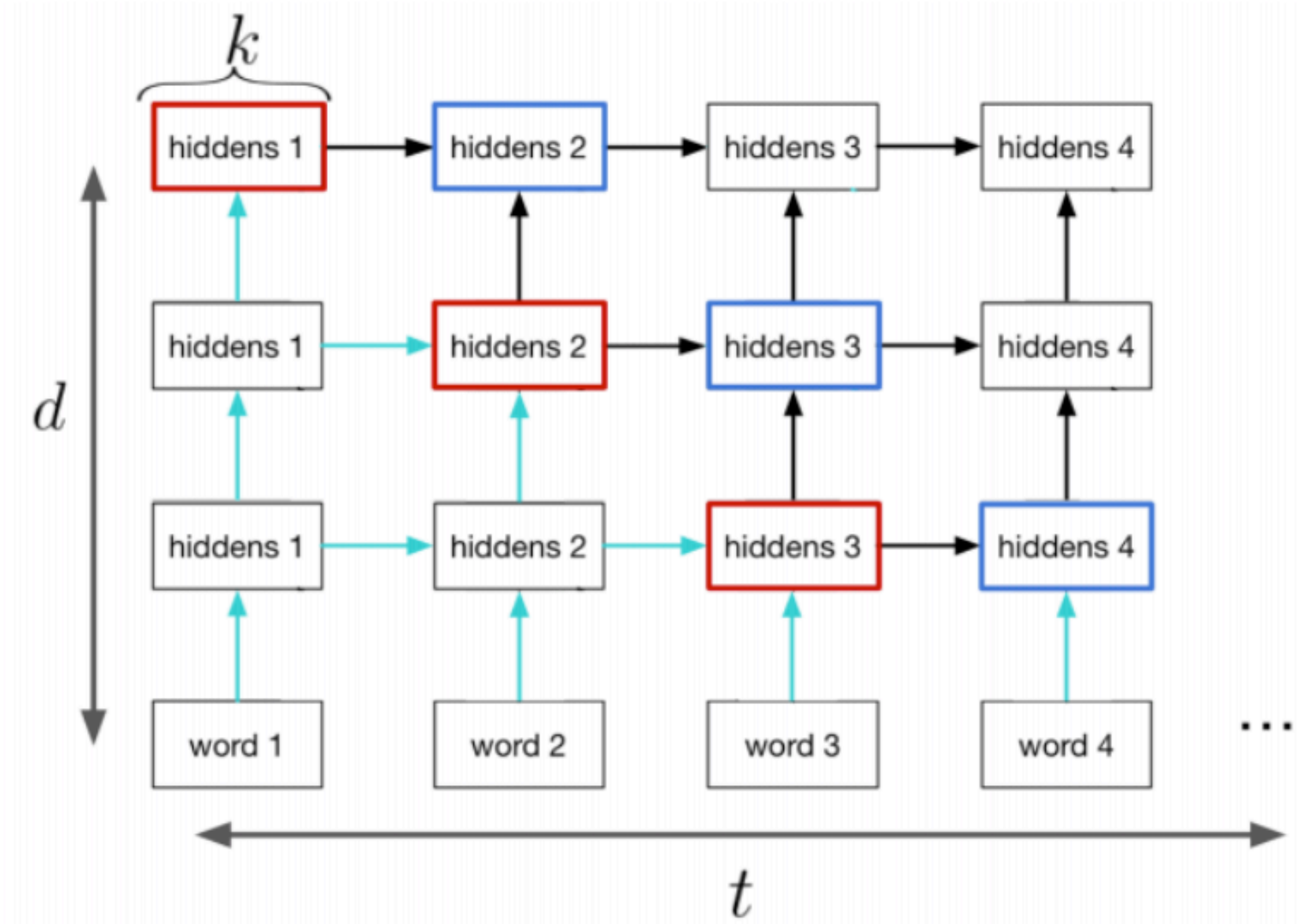


- Both the input embeddings and the outputs of an RNN can be computed in parallel.
- The blue hidden units are independent given the red.
- The numer of sequential operation is still propotional to t.

http://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/slides/lec16.pdf

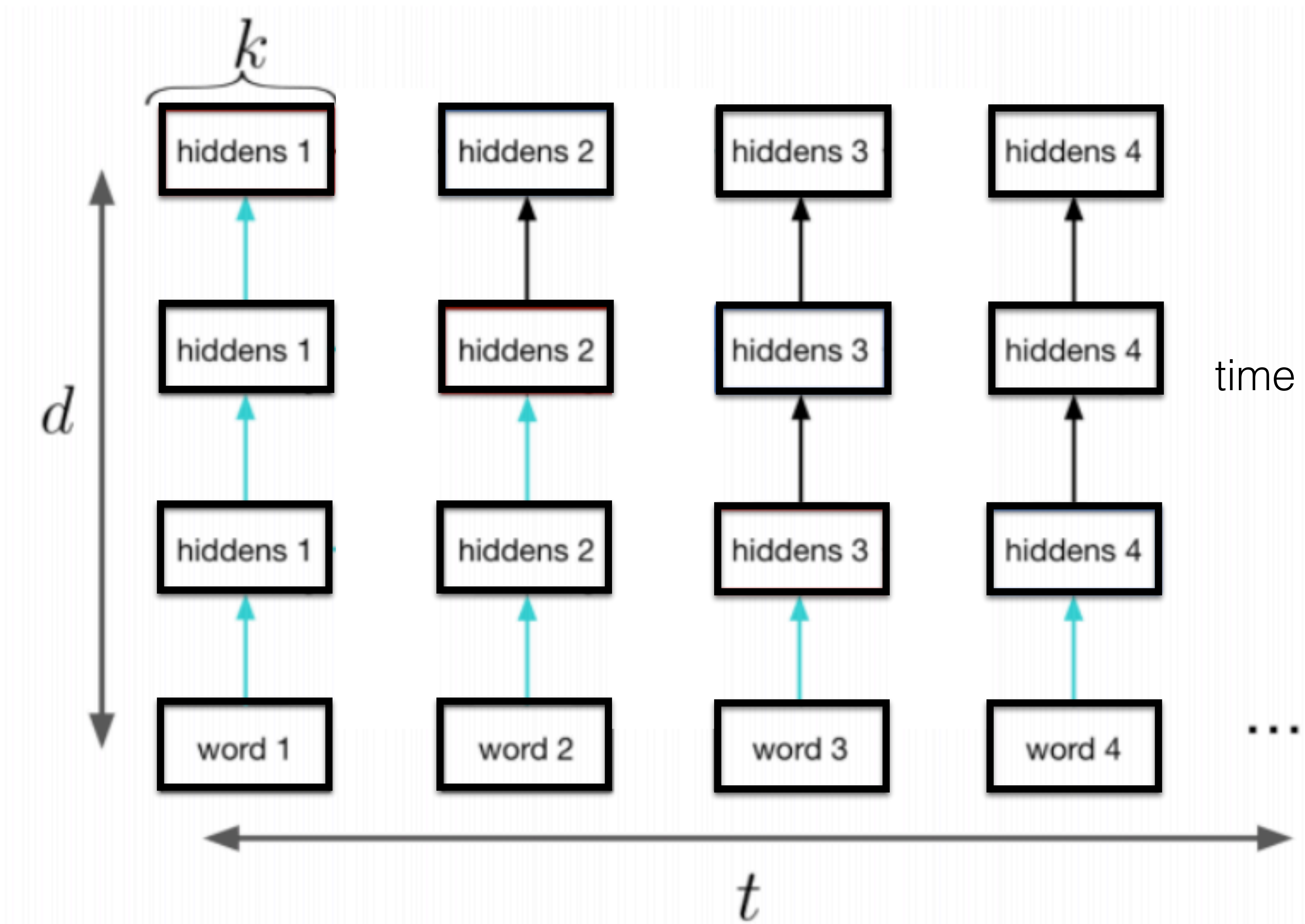
Cost vs Opportunity

RNN to Self-attention



Cost vs Opportunity

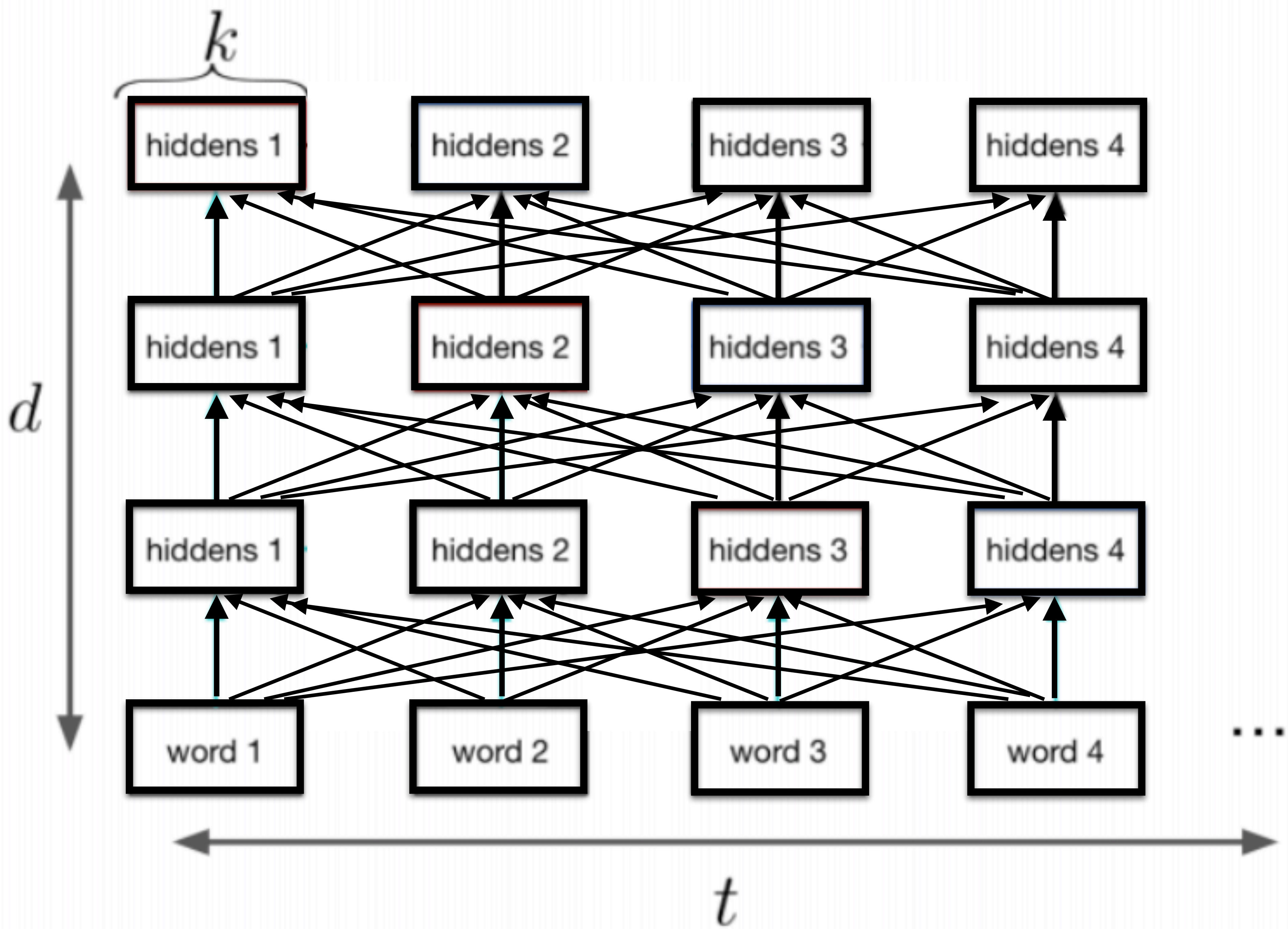
RNN to Self-attention



drop
time dependence

Cost vs Opportunity

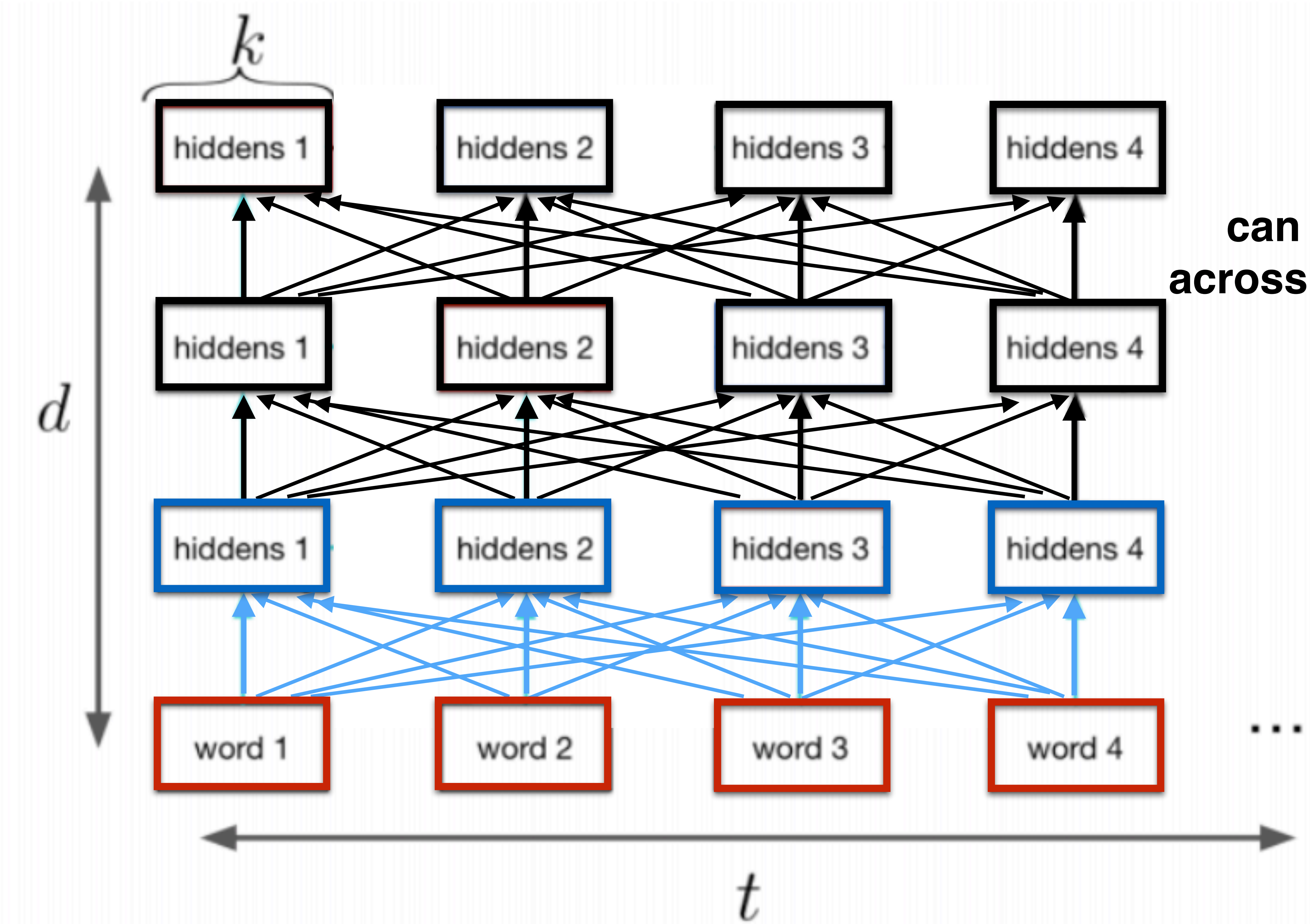
RNN to Self-attention



add attention

Cost vs Opportunity

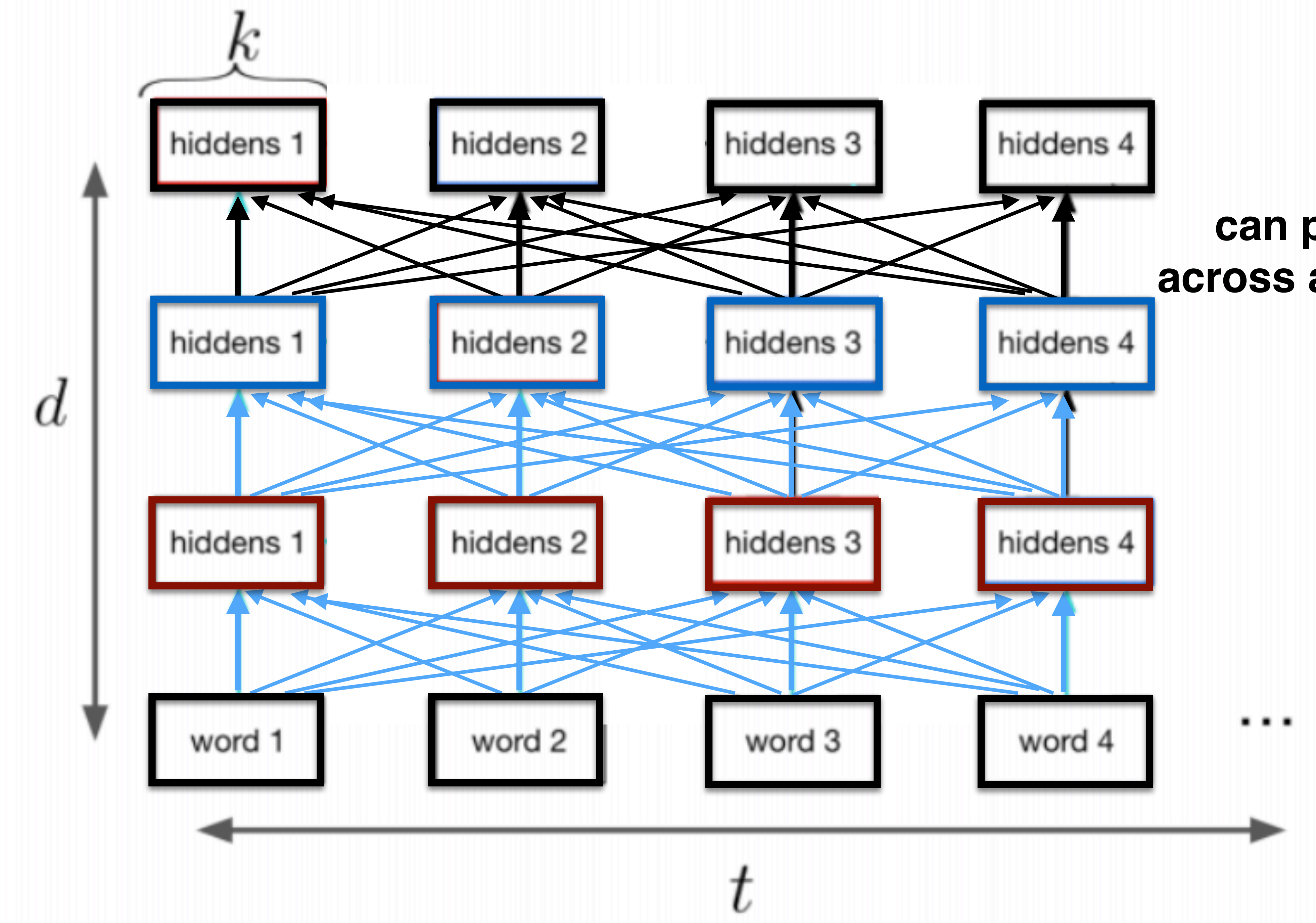
RNN to Self-attention



can parallelize across all sequence

Cost vs Opportunity

RNN to Self-attention

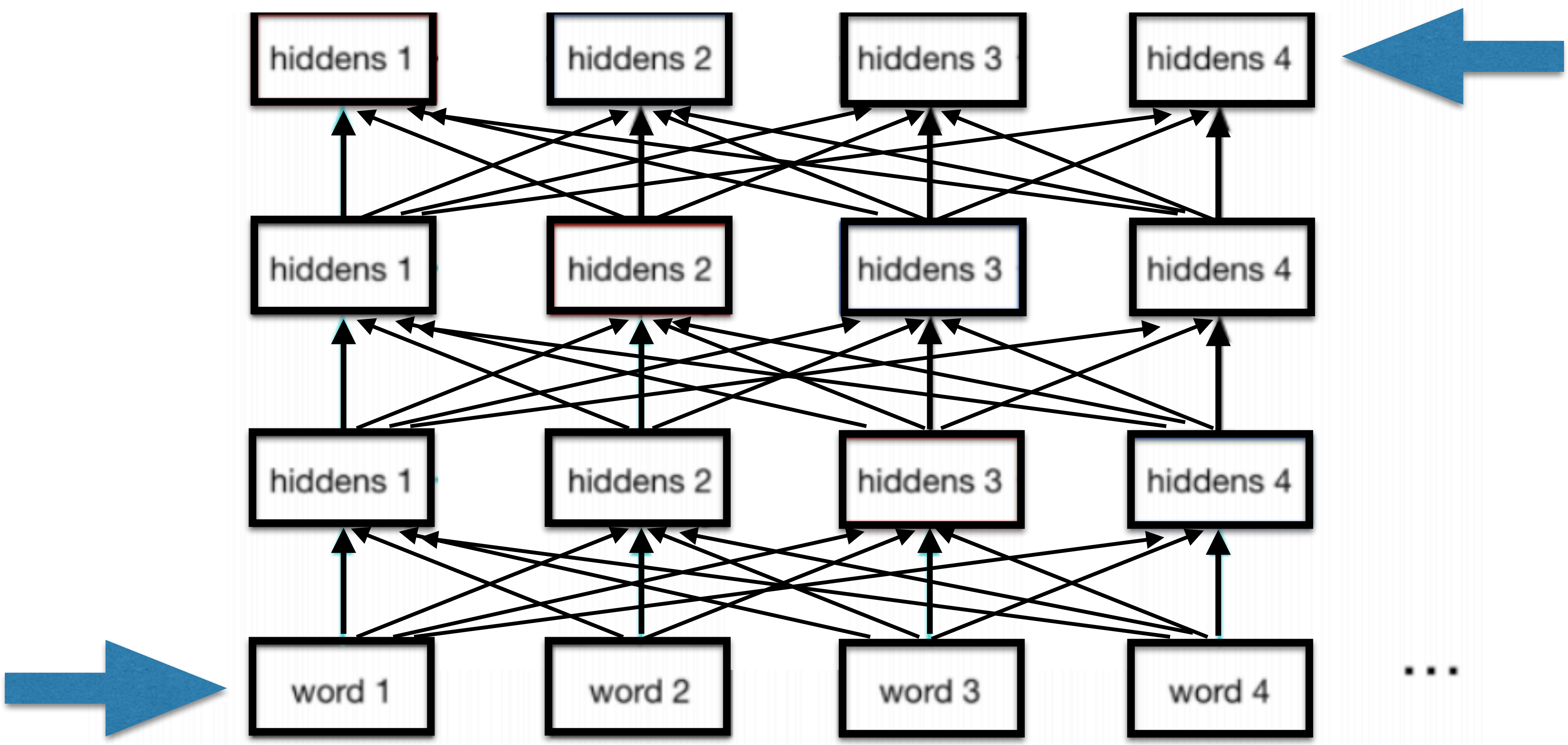


can parallelize across all sequence

Transformer

Information flow

how do we pass information between the blue arrows?

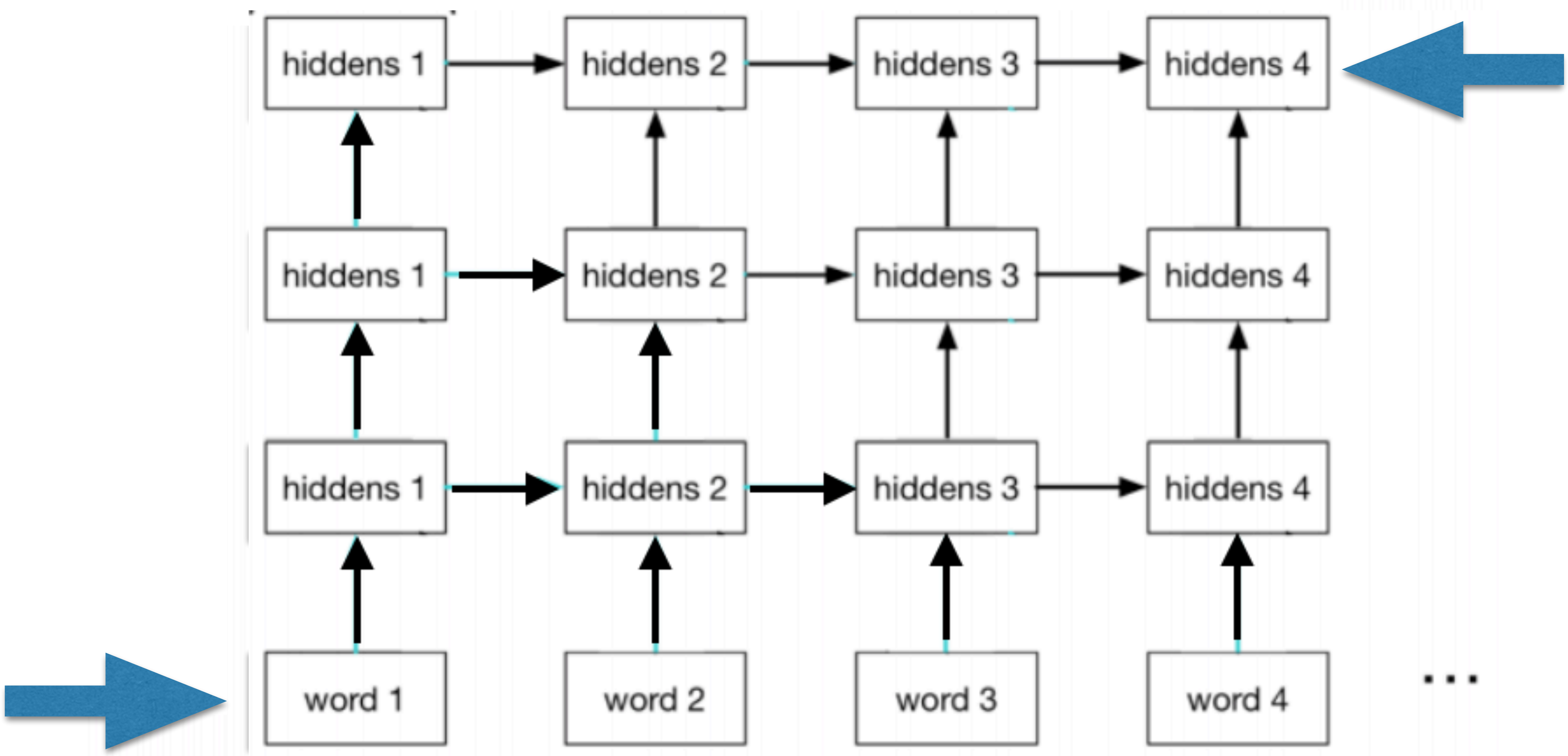


Transformer

vs RNN case

Information flow

how do we pass information between the blue arrows?



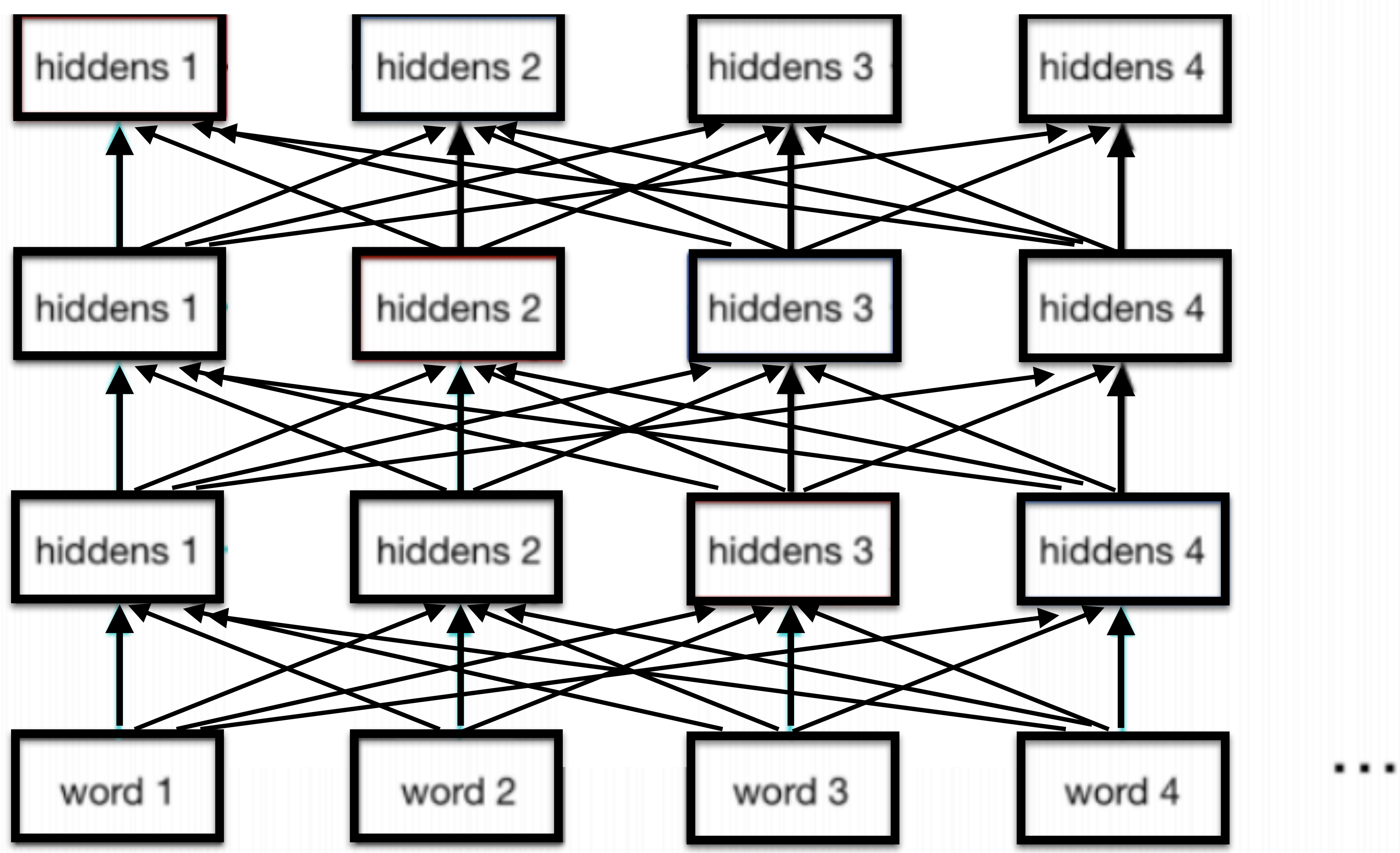
Transformer

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

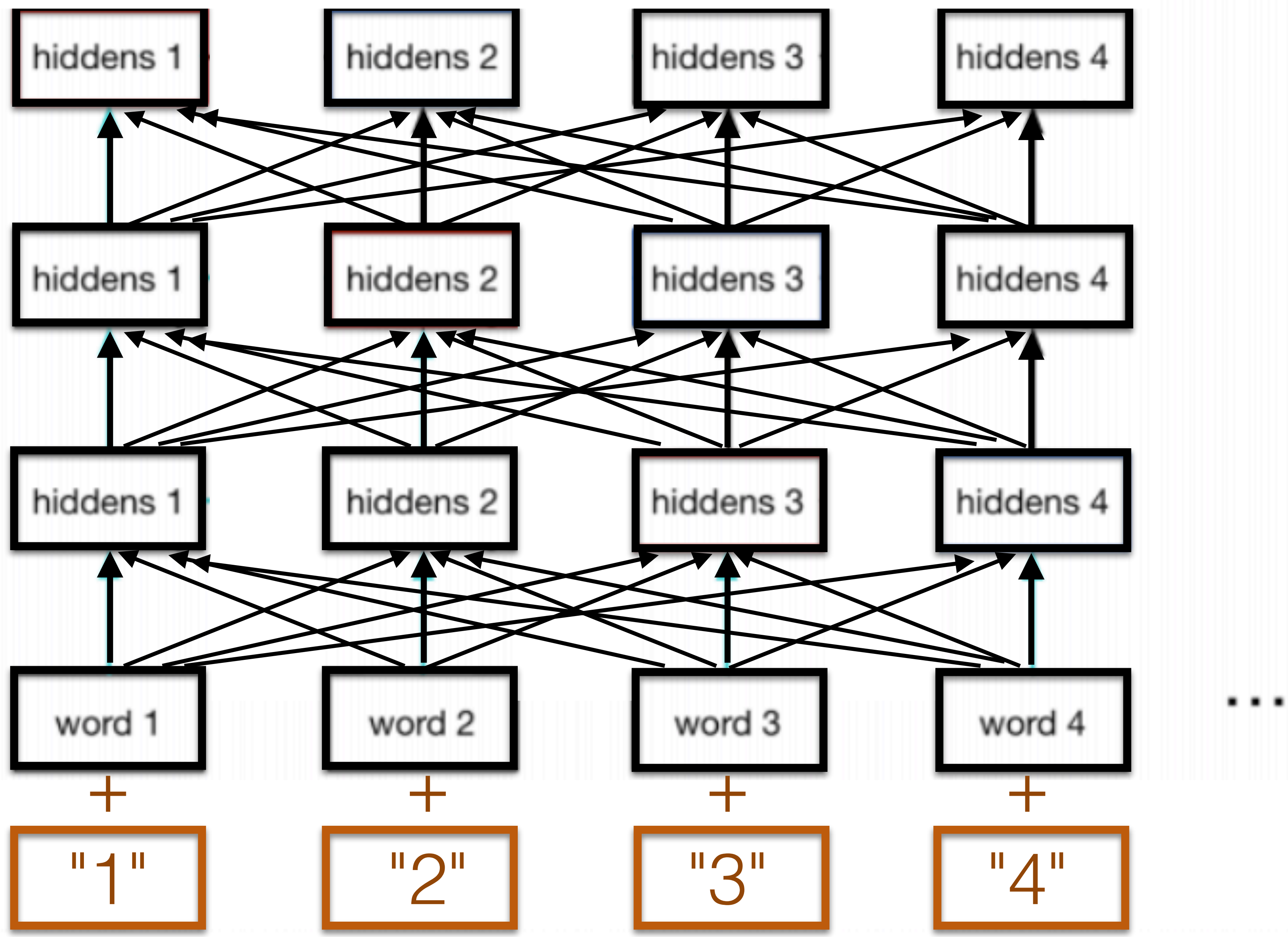
Transformer

Positional information



Transformer

Positional information



Transformer: more details

http://nlp.seas.harvard.edu/2018/04/03/attention.html

harvardnlp Members PI Code Publications

The Annotated Transformer

Apr 3, 2018

http://jalammar.github.io/illustrated-transformer/

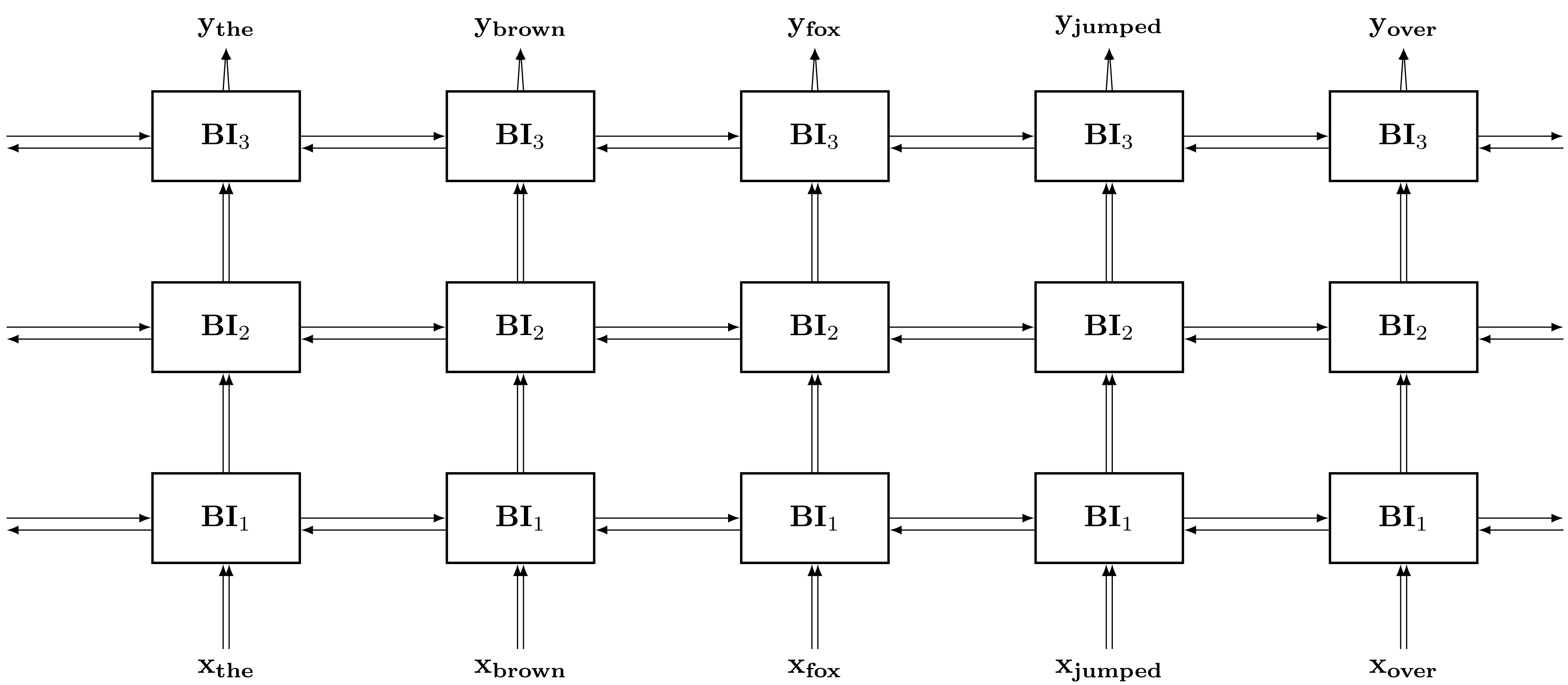
Jay Alammar

Visualizing machine learning one concept at a time

The Illustrated Transformer

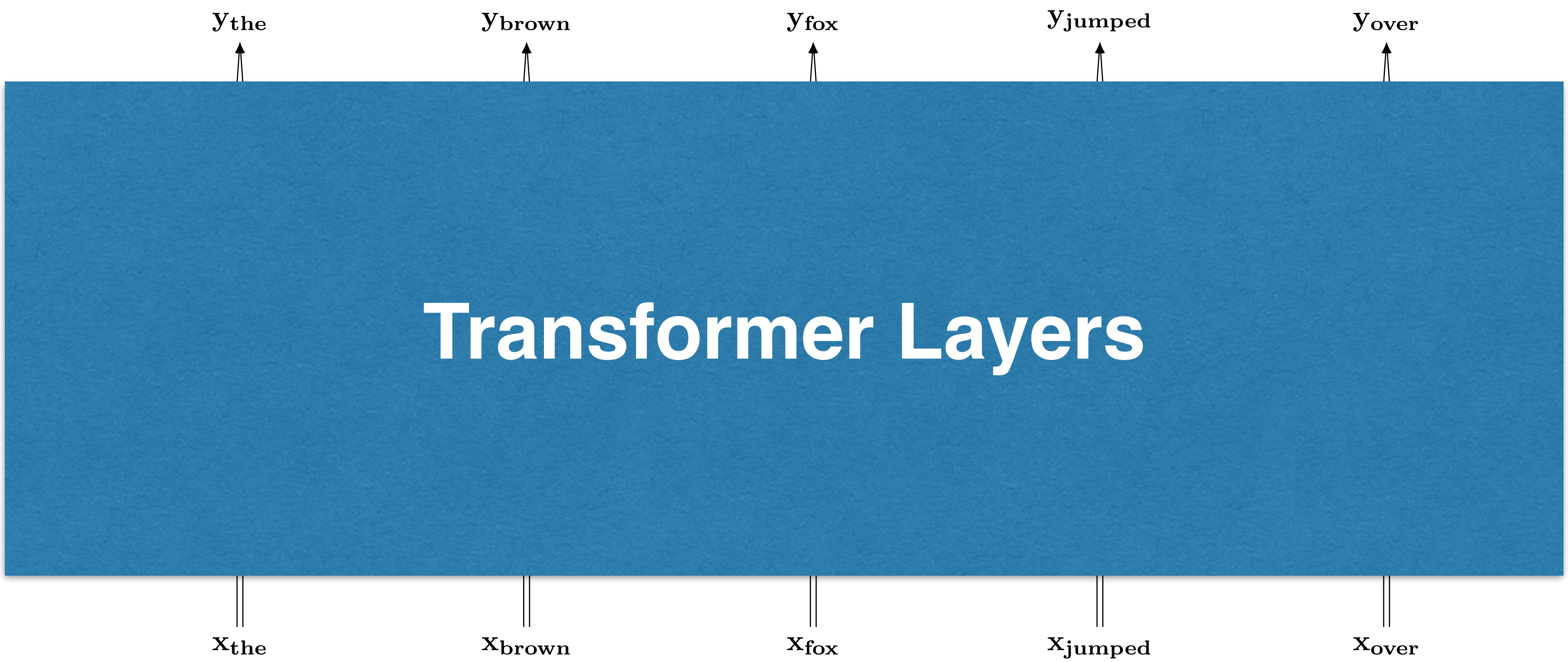
good explanations, have a look.

BiRNN -> Transformer



Deep biRNN

BiRNN -> Transformer



Transformer

- An alternative to RNN
- Replace recurrence with attention to all tokens.
- More computation. More parallelism.
  Shorter connections between data points.

To summarize

- RNNs are very capable learners of sequential data.
- n -> 1: RNN acceptor
- n -> n : biRNN (transducer)
- 1 -> m : conditioned generation (conditioned LM)
- n -> m : conditioned generation (encoder-decoder)
- n -> m : encoder-decoder with attention