Sequence to Sequence Models with Attention

Mausam

(Slides by Yoav Goldberg, Graham Neubig, Prabhakar Raghavan)

BACKGROUND

A Primer on D.L. Building Blocks

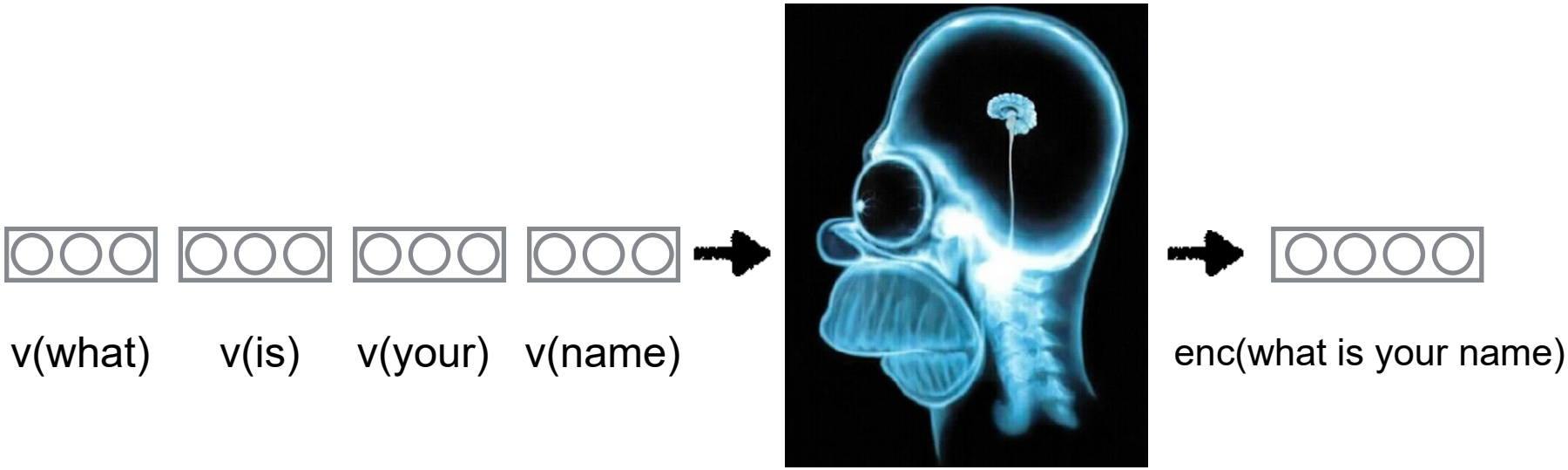
- A single vector for an ordered pair of vectors?
- A single vector for a variable-sized bag of vectors?
- Project a vector to a new space?
- Are two vectors (from same space) similar?
- Are two vectors (from different space) similar?
- A new vector that depends on some vector input?

- x;y
- $\sum_i x_i$
- Wx
- x.y
- xWy
- g(Wx+b)

A Primer on D.L. Building Blocks

• Output a probability	• σ
• Output one of two classes	• σ
• Output one of many classes	• softmax
• A feature w/ positive & negative influence	• tanh
• A feature w/ positive influence for “deep” nets	• ReLu

Recurrent Neural Networks



- Very strong models of sequential data.
- **Trainable** function from *n* vectors to a single vector.

Recurrent Neural Networks

$$RNN(\mathbf{s_0}, \mathbf{x_{1:n}}) = \mathbf{s_n}, \mathbf{y_n}$$

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}},\ \mathbf{y_i} \in \mathbb{R}^{d_{out}},\ \mathbf{s_i} \in \mathbb{R}^{f(d_{out})}$$

- Very strong models of sequential data.
- **Trainable** function from *n* vectors to a single* vector.

Recurrent Neural Networks

$$RNN(\mathbf{s_0}, \mathbf{x_{1:n}}) = \mathbf{s_n}, \mathbf{y_n}$$

*this one is internal. we only care about the **y**

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}}, \ \mathbf{y_i} \in \mathbb{R}^{d_{out}}, \ \mathbf{s_i} \in \mathbb{R}^{f(d_{out})}$$

- Very strong models of sequential data.
- **Trainable** function from *n* vectors to a single* vector.

Recurrent Neural Networks

$$RNN(\mathbf{s_0}, \mathbf{x_{1:n}}) = \mathbf{s_n}, \mathbf{y_n}$$

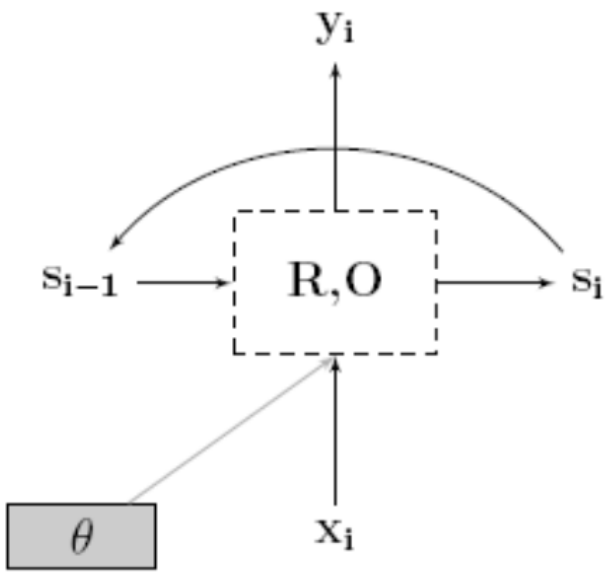
$$\mathbf{s_i} = R(\mathbf{s_{i-1}}, \mathbf{x_i})$$

$$\mathbf{y_i} = O(\mathbf{s_i})$$

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}}, \ \mathbf{y_i} \in \mathbb{R}^{d_{out}}, \ \mathbf{s_i} \in \mathbb{R}^{f(d_{out})}$$

- **Recursively defined.**
- There's a vector $\mathbf{y_i}$ for every prefix $\mathbf{x_{1:i}}$

Recurrent Neural Networks

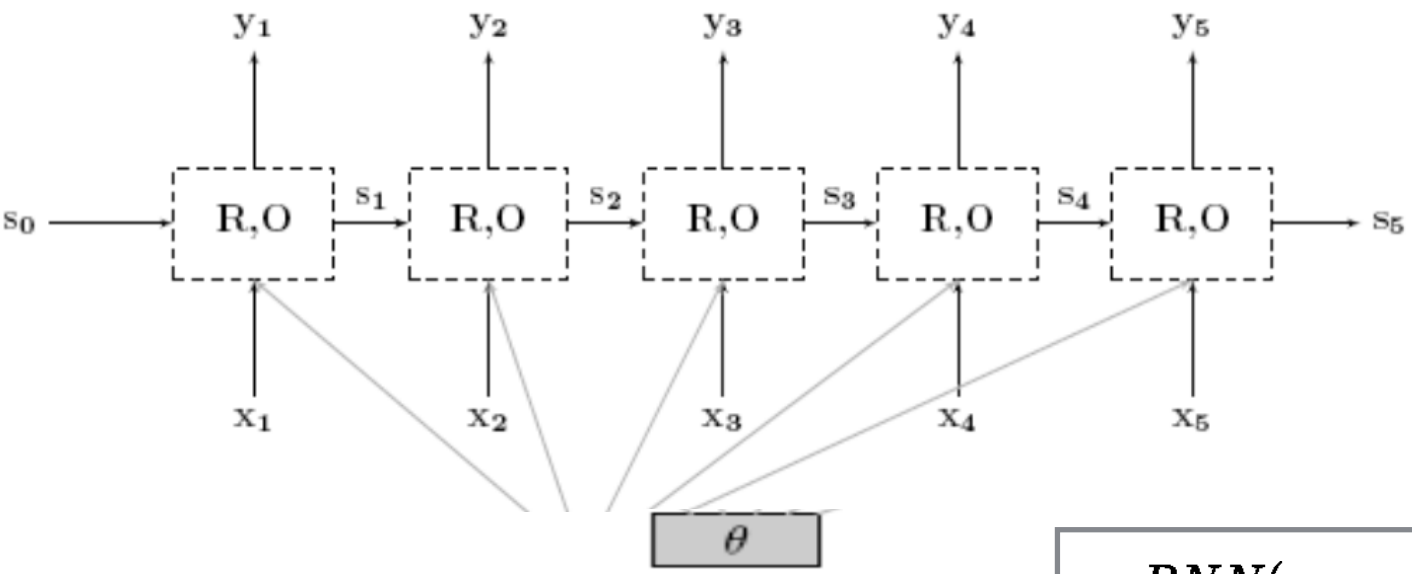


$$RNN(\mathbf{s_0}, \mathbf{x_{1:n}}) = \mathbf{s_n}, \mathbf{y_n}$$
$$\mathbf{s_i} = R(\mathbf{s_{i-1}}, \mathbf{x_i})$$
$$\mathbf{y_i} = O(\mathbf{s_i})$$

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}},\ \mathbf{y_i} \in \mathbb{R}^{d_{out}},\ \mathbf{s_i} \in \mathbb{R}^{f(d_{out})}$$

- **Recursively defined**.
- There's a vector $\mathbf{y_i}$ for every prefix $\mathbf{x_{1:i}}$

Recurrent Neural Networks



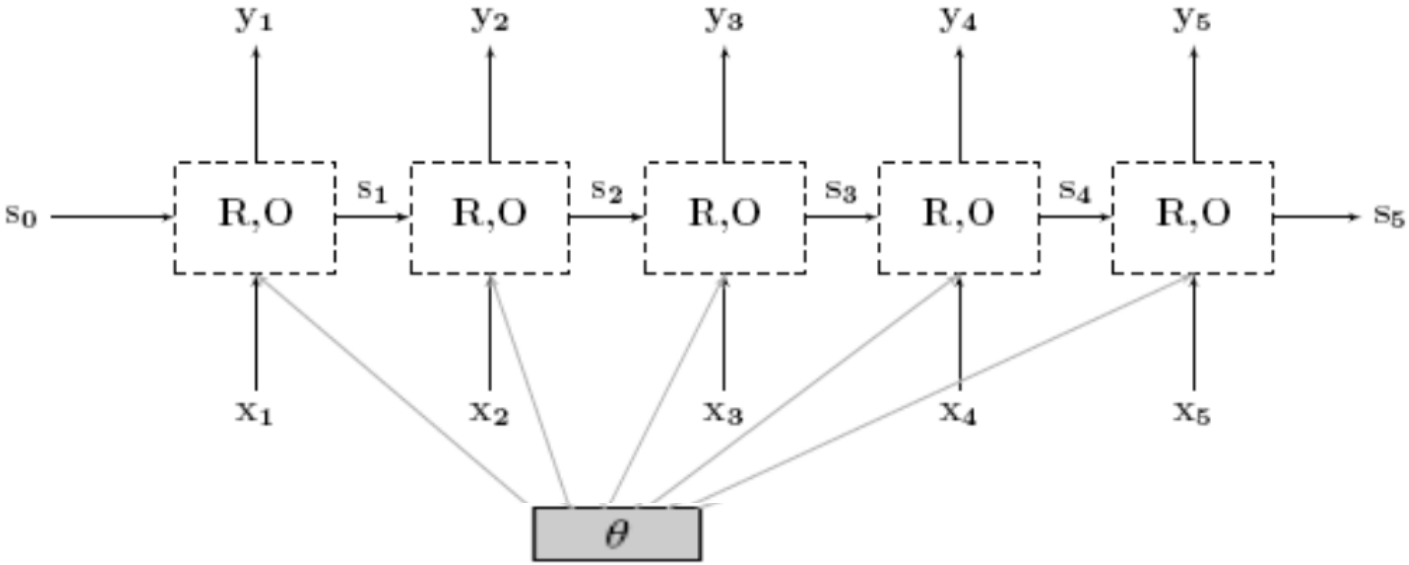
for every finite input sequence,
can unroll the recursion.

$$RNN(\mathbf{s_0}, \mathbf{x_{1:n}}) = \mathbf{s_n}, \mathbf{y_n}$$
$$\mathbf{s_i} = R(\mathbf{s_{i-1}}, \mathbf{x_i})$$
$$\mathbf{y_i} = O(\mathbf{s_i})$$

$$\mathbf{x_i} \in \mathbb{R}^{d_{in}},\ \mathbf{y_i} \in \mathbb{R}^{d_{out}},\ \mathbf{s_i} \in \mathbb{R}^{f(d_{out})}$$

- Recursively defined.
- There's a vector $\mathbf{y_i}$ for every prefix $\mathbf{x_{1:i}}$

Recurrent Neural Networks



for every finite input sequence, can unroll the recursion.

State s_i encodes **history** till this point

Simple RNN (Elman RNN)

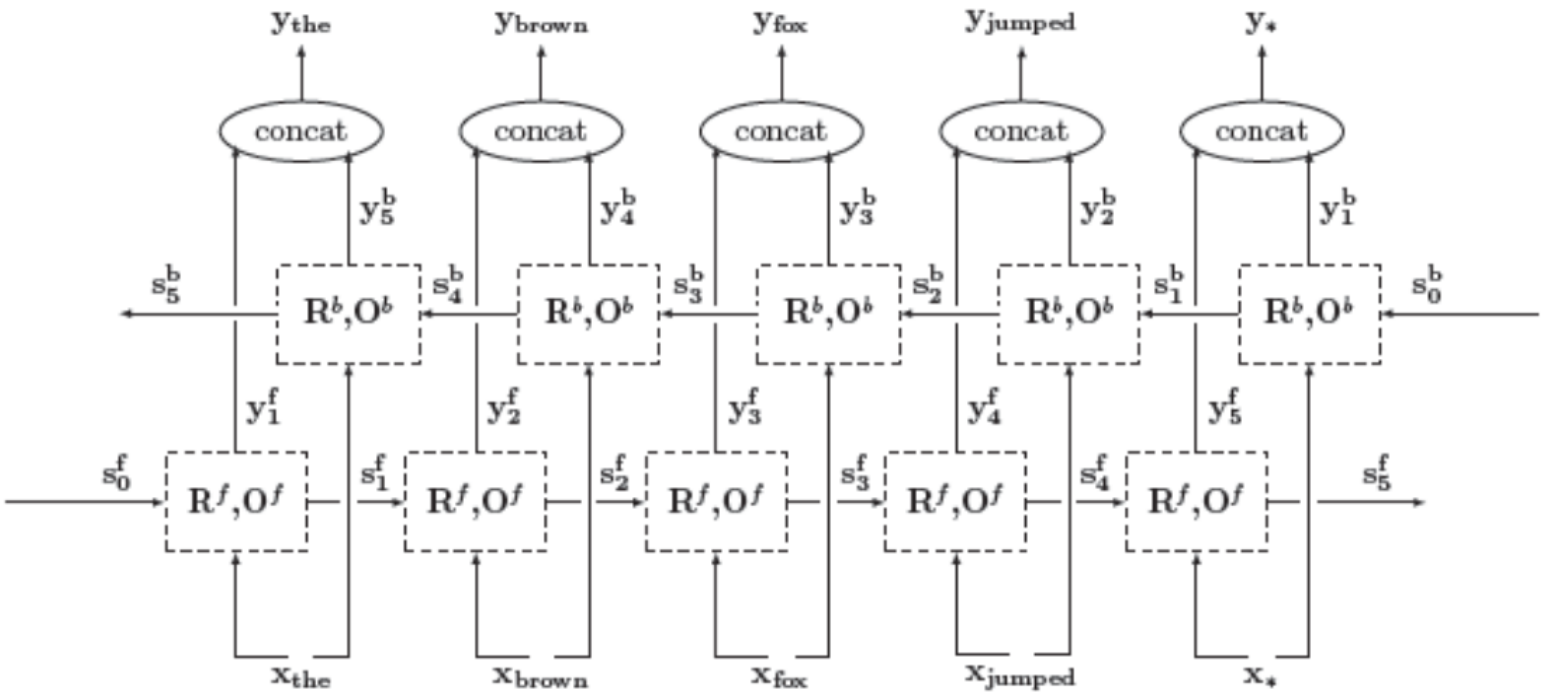
$$R_{SRNN}(\mathbf{s_{i-1}}, \mathbf{x_i}) = tanh(\mathbf{W^s} \cdot \mathbf{s_{i-1}} + \mathbf{W^x} \cdot \mathbf{x_i})$$

In principle: capture infinite history upto this point
In practice: have issues with long sequences

RNN → LSTM

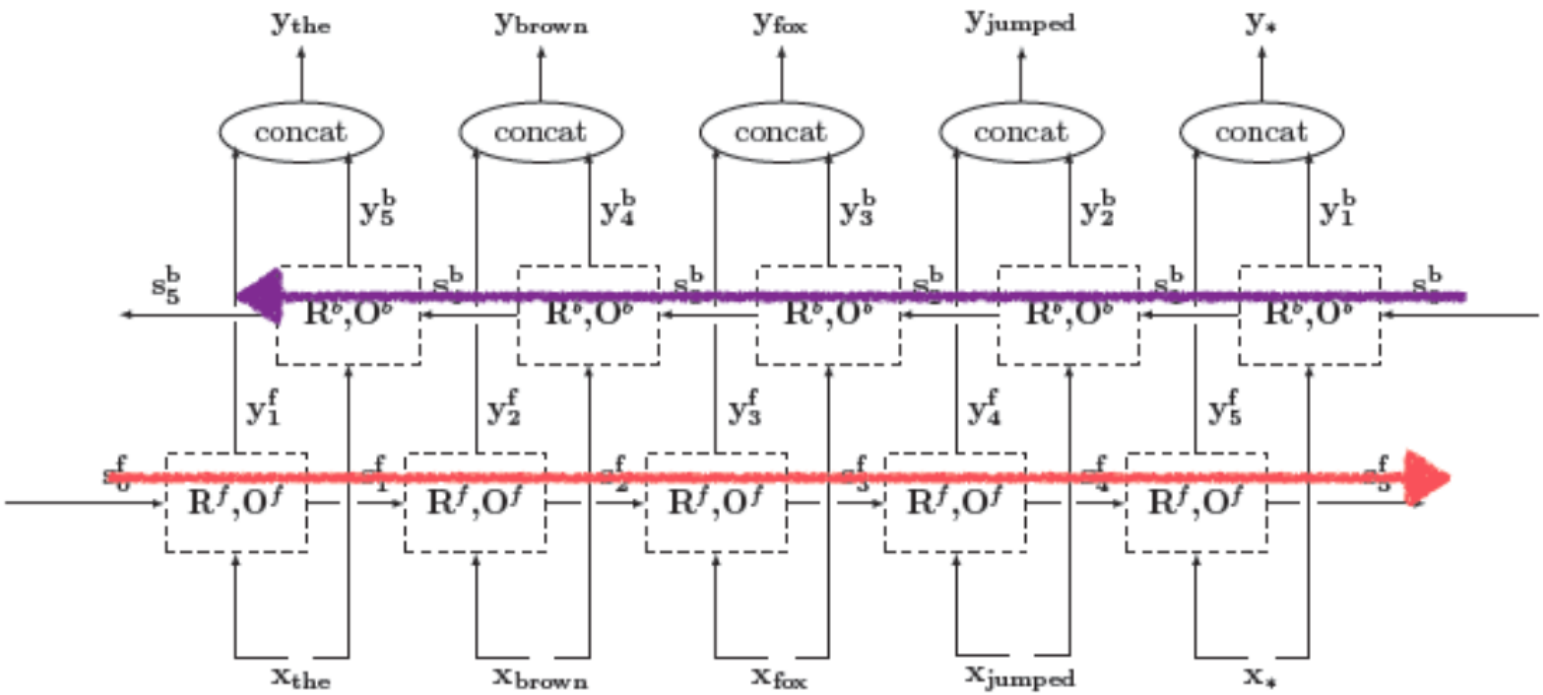
Good for backpropagating through long chain sequences

Bidirectional RNNs



One RNN runs left to right.
Another runs right to left.
Encode **both future and history** of a word.

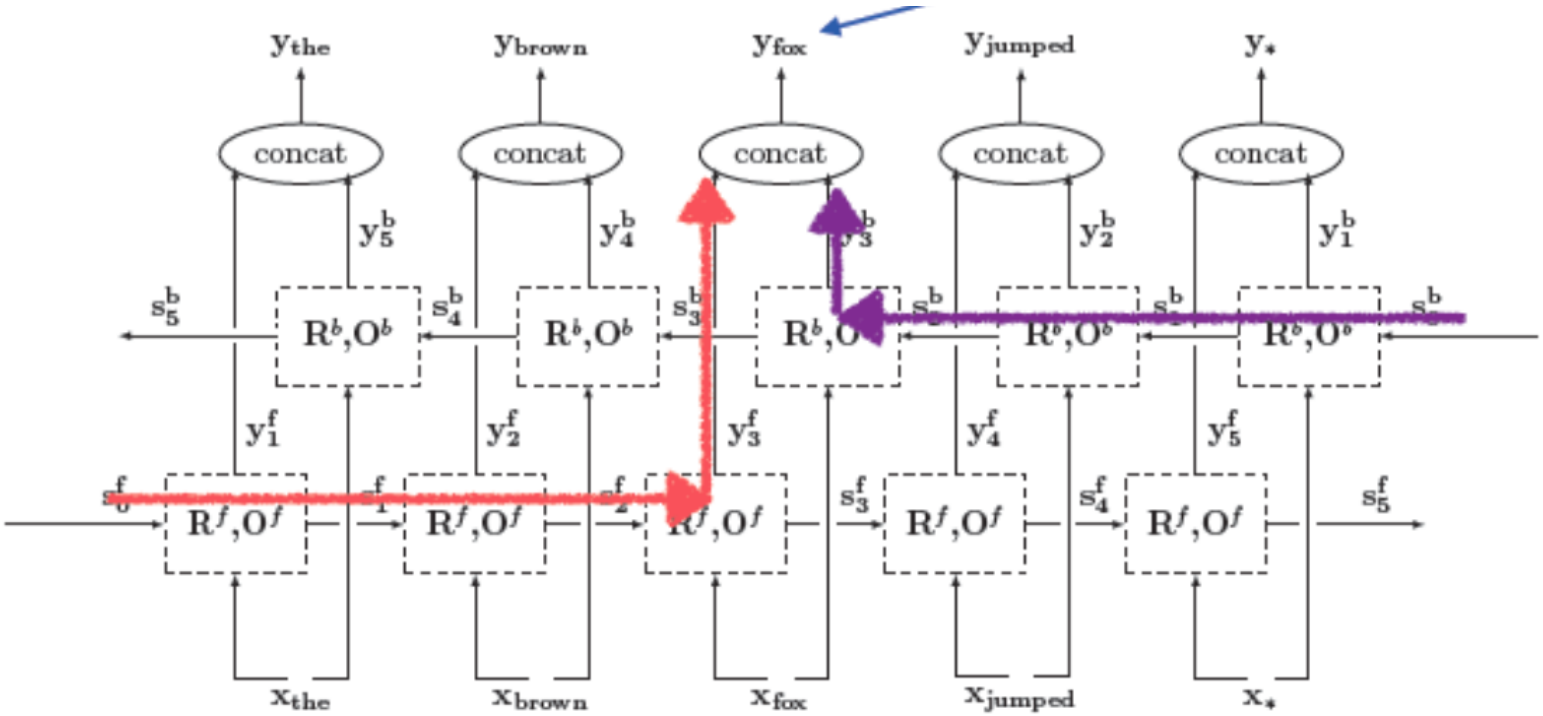
Bidirectional RNNs



One RNN runs left to right.
Another runs right to left.
Encode **both future and history** of a word.

Bidirectional RNNs

Infinite window around the word



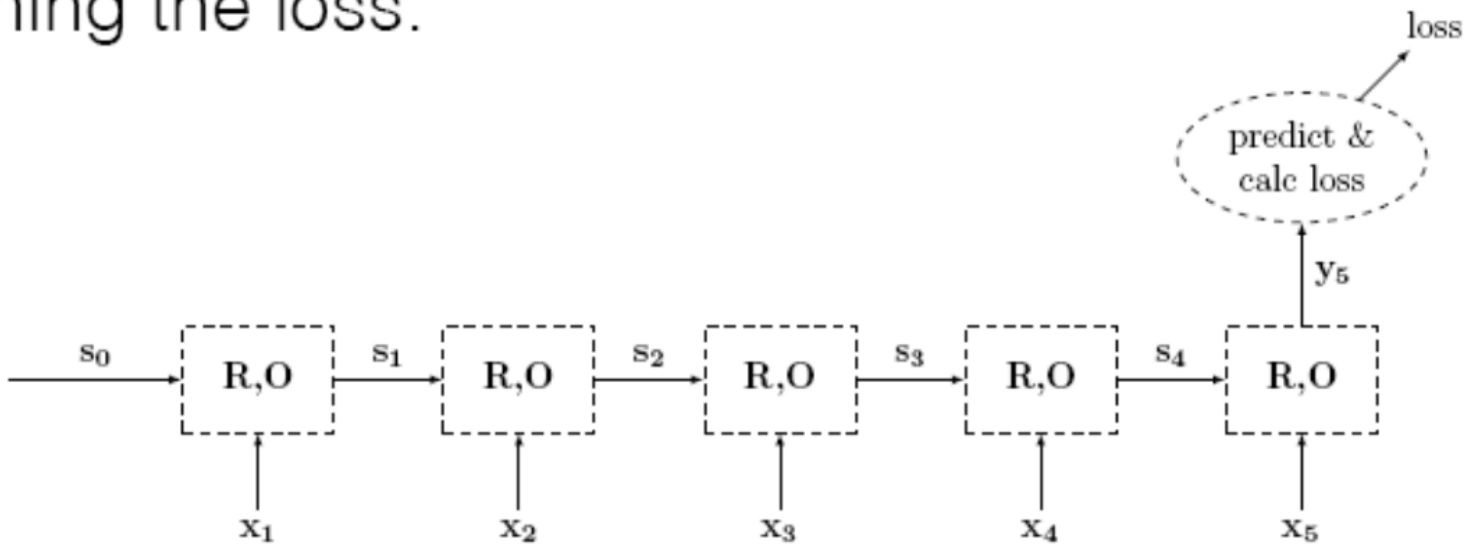
One RNN runs left to right.
Another runs right to left.
Encode **both future and history** of a word.

Neural Architectures

- Mapping from a sequence to a single decision.
  - with RNN acceptor.
- Mapping from a sequence to a sequence of same length.
  - with RNN transducer

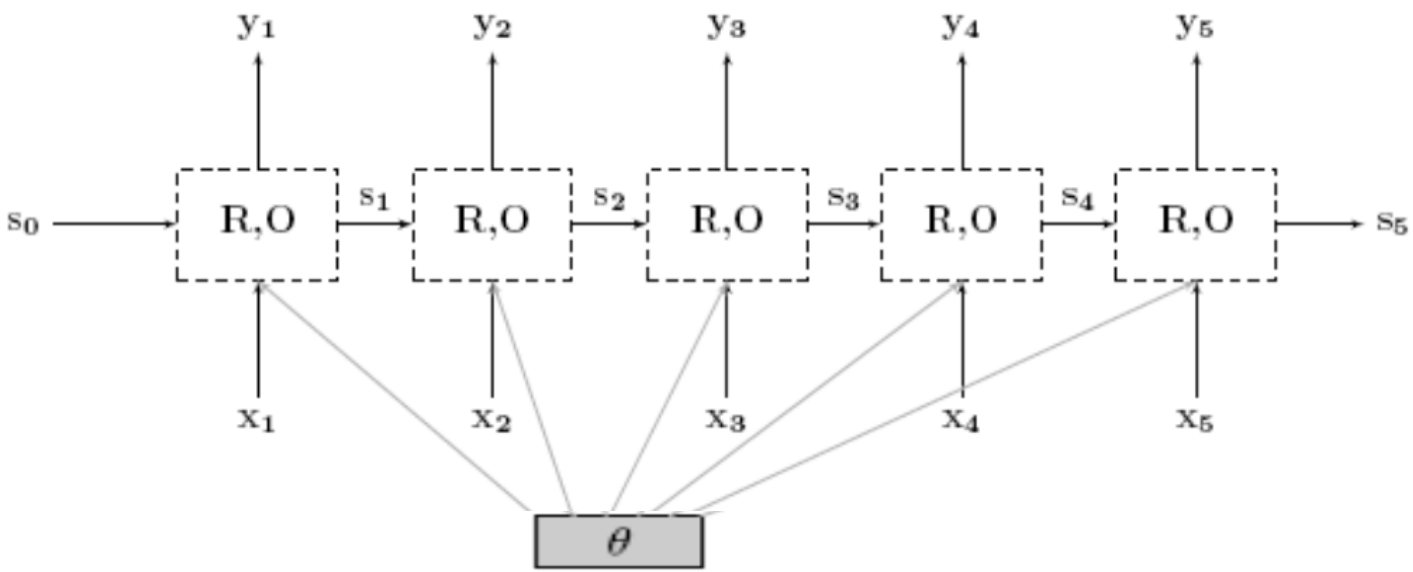
RNN Acceptor

Defining the loss.



Acceptor: predict something from end state.
Backprop the error all the way back.
Train the network to capture meaningful information

RNN Transducer



what do we do if the input and output sequences are of **different lengths**?

Sequence Generation

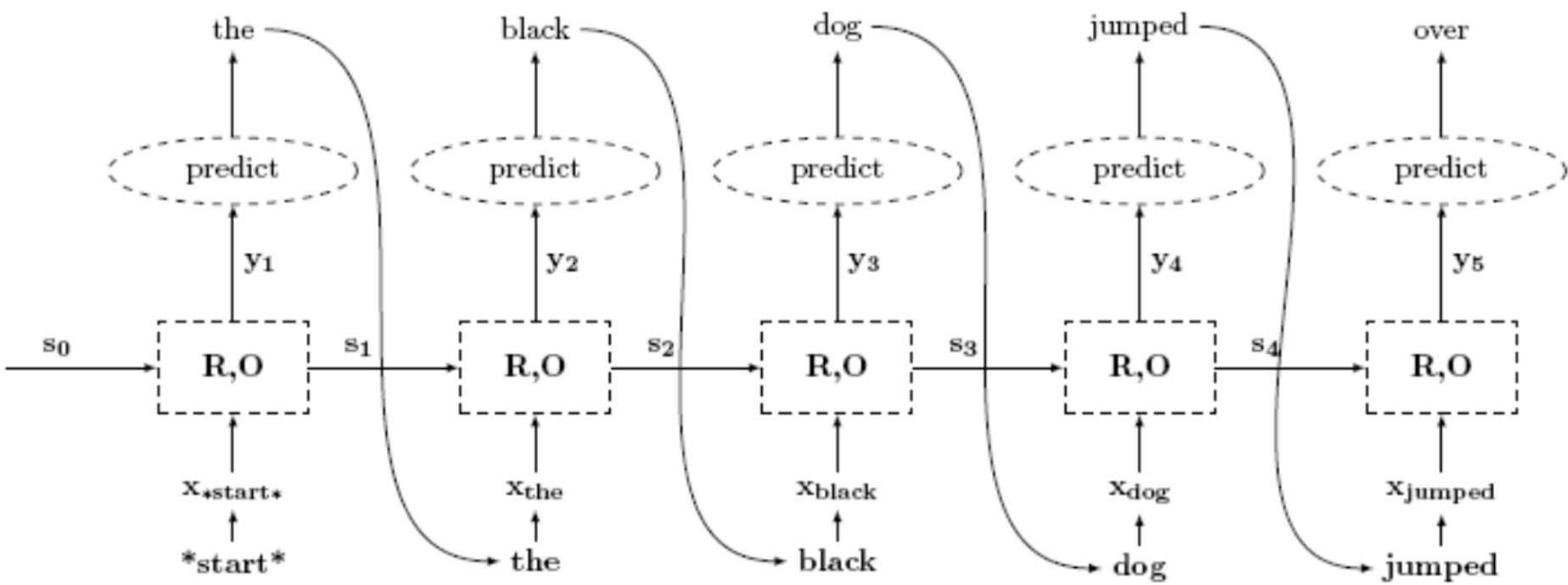
Aside

How about an architecture for
0 to n mapping.

(Neural Language Model)

RNN Language Models

- *Training*: similar to an RNN Transducer.
- *Generation*: the output of step i is input to step i+1.



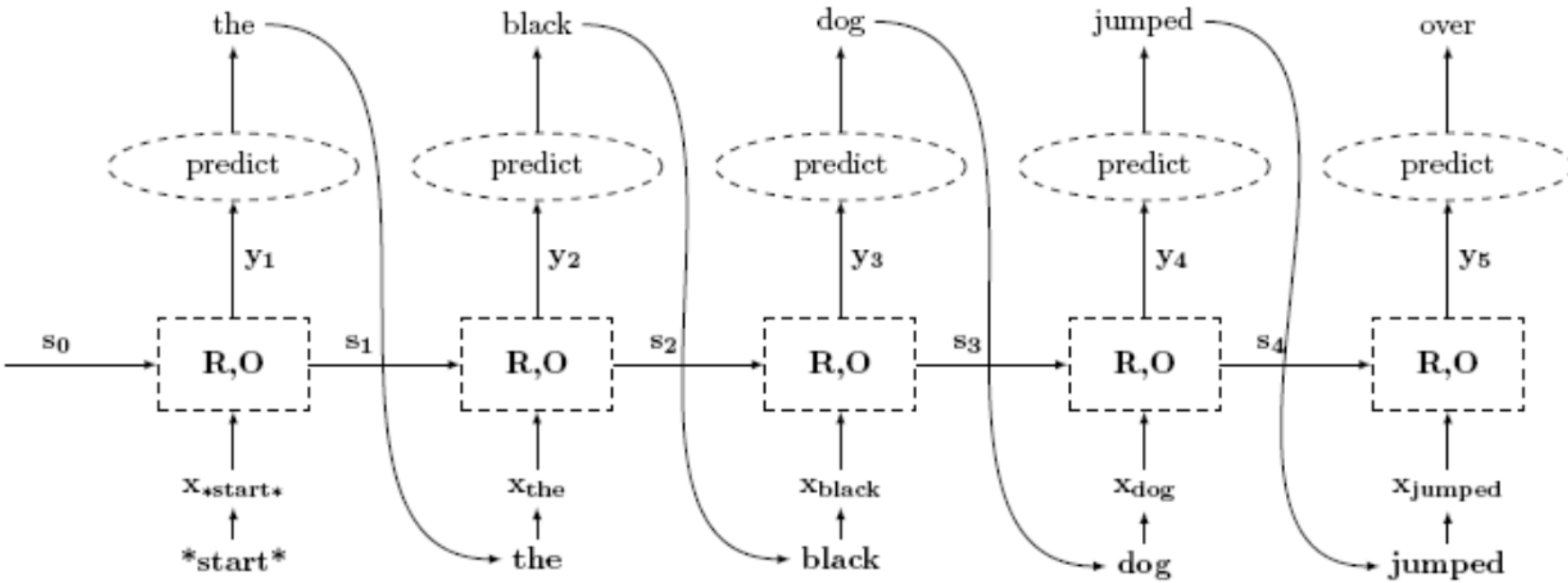
RNN Language Model for generation

- Define the probability distribution over the next item in a sequence (and hence the probability of a sequence).

$$P(w_{1:n}) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_{1:2})P(w_4 \mid w_{1:3}) \ldots P(w_n \mid w_{1:n-1})$$

$$P(w_1, ..., w_n) = \prod_{i=1}^{n} P(t_i = w_i | w_1, ..., w_{i-1})$$

RNN Language Models



$$p(t_{j+1} = k \mid \hat{t}_{1:j}) = f(\mathrm{RNN}(\hat{\mathbf{t}}_{\mathbf{1:j}}))$$

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

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

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

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

Sequence 2 Sequence

Part I: No attention

Back to Original question

How about an architecture for
m to n mapping.

Generating sentences is nice, but what if we want to add some additional conditioning contexts?

Conditioned Language Model

- Not just generate text, generate text according to some specification

Input *X*	Output *Y* (**Text**)	Task
Structured Data	NL Description	NL Generation
English	Japanese	Translation
Document	Short Description	Summarization
Utterance	Response	Response Generation
Image	Text	Image Captioning
Speech	Transcript	Speech Recognition

RNN Language Model for **Conditioned generation**

Let's add the condition variable to the equation.

$$P(\mathrm{T}) = \prod_{i=1}^{I} P(\underbrace{t_i}_{\text{Next Word}} \mid \underbrace{t_1, \ldots, t_{i-1}}_{\text{Context}})$$

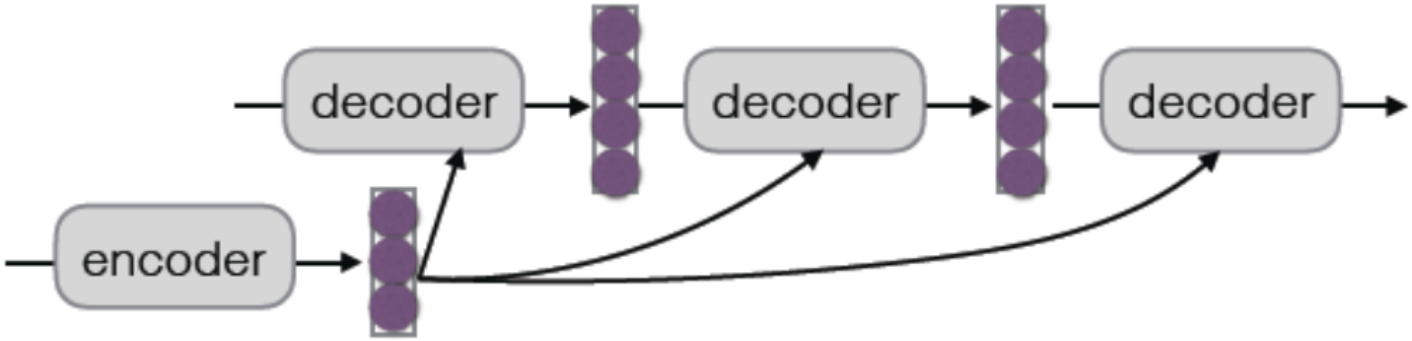
$$P(\mathrm{T} \mid \mathrm{C}) = \prod_{j=1}^{J} P(t_i \mid \underbrace{c}_{\text{Added Context! (a vector)}}, t_1, \ldots, t_{i-1})$$

How to Pass Context

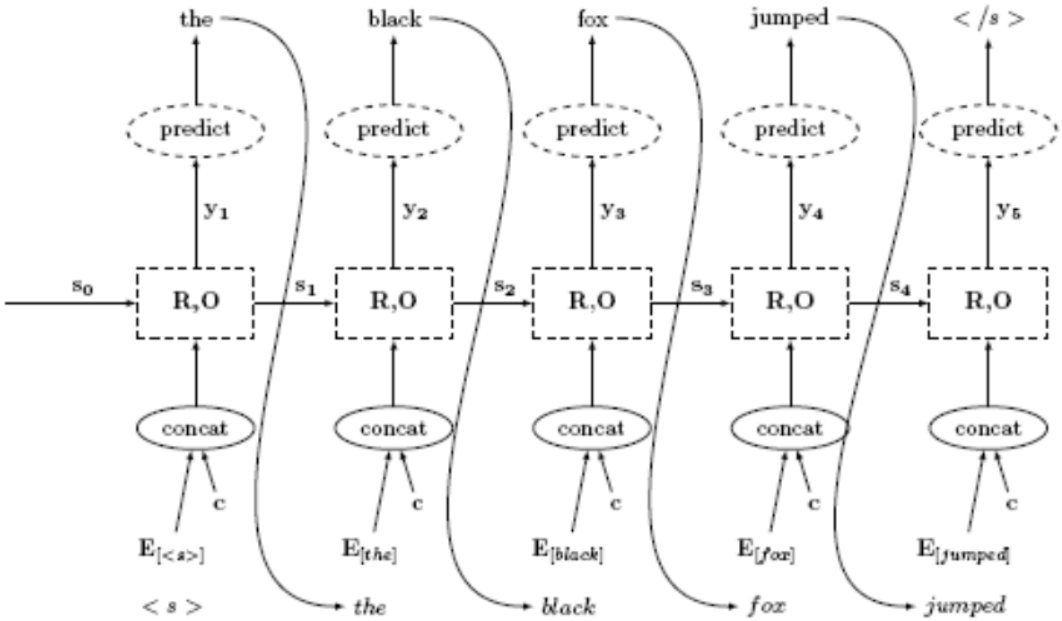
- Initialize decoder w/ encoder (Sutskever et al. 2014)

- Transform (can be different dimensions)

- Input at every time step (Kalchbrenner & Blunsom 2013)



RNN Language Model for **Conditioned generation**



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

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

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

RNN Language Model for **Conditioned generation**

what if we want to condition on an entire sentence?

just encode it as a vector...

$$\mathbf{c} = \mathrm{RNN}^{\mathrm{enc}}(\mathbf{x_{1:n}})$$

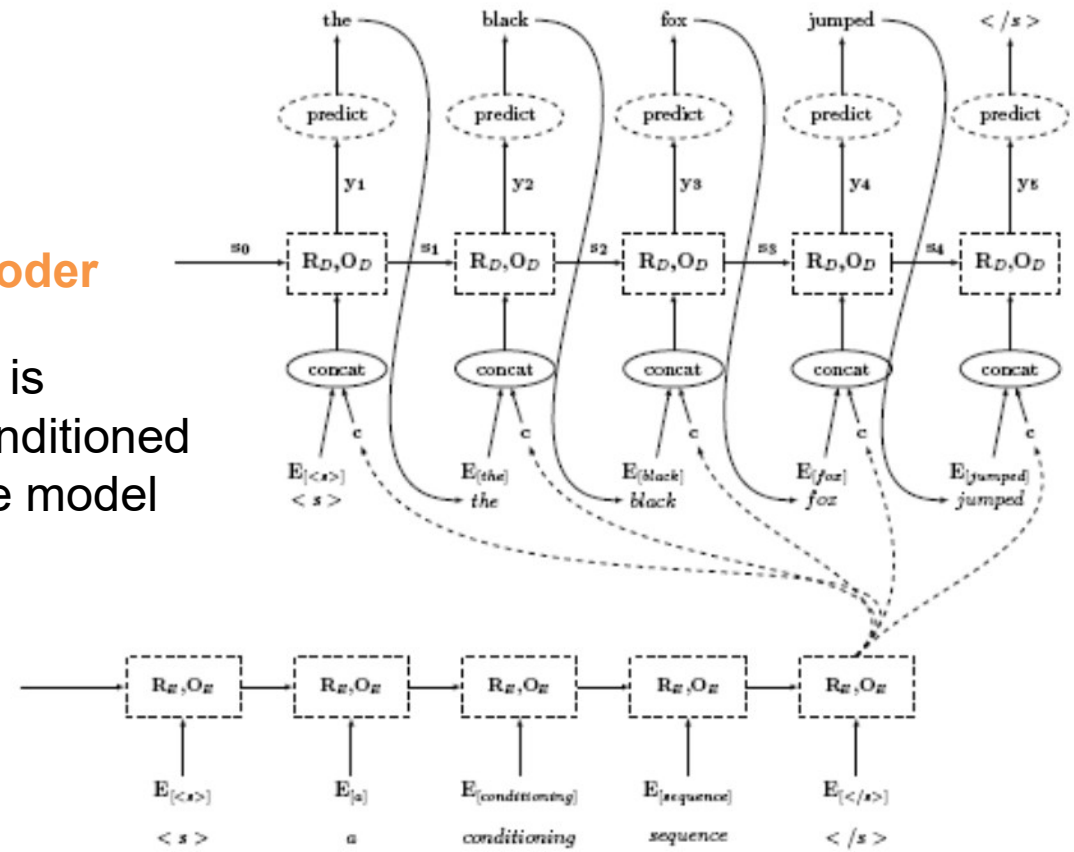
Sequence to Sequence conditioned generation

This is also called "Encoder Decoder" architecture.

Decoder

Decoder is just a conditioned language model

Encoder



The Generation Problem

We have a probability model, how do we use it to generate a sentence?

Two methods:

- **Sampling:** Try to generate a *random* sentence according to the probability distribution.
- **Argmax:** Try to generate the sentence with the *highest* probability.

Ancestral Sampling

Randomly generate words one-by-one.

```
while y_{j-1} != "</s>":
  y_j ~ P(y_j | X, y_1, ..., y_{j-1})
```

An **exact method** for sampling from P(X), no further work needed.

Greedy Search

One by one, pick the single highest-probability word

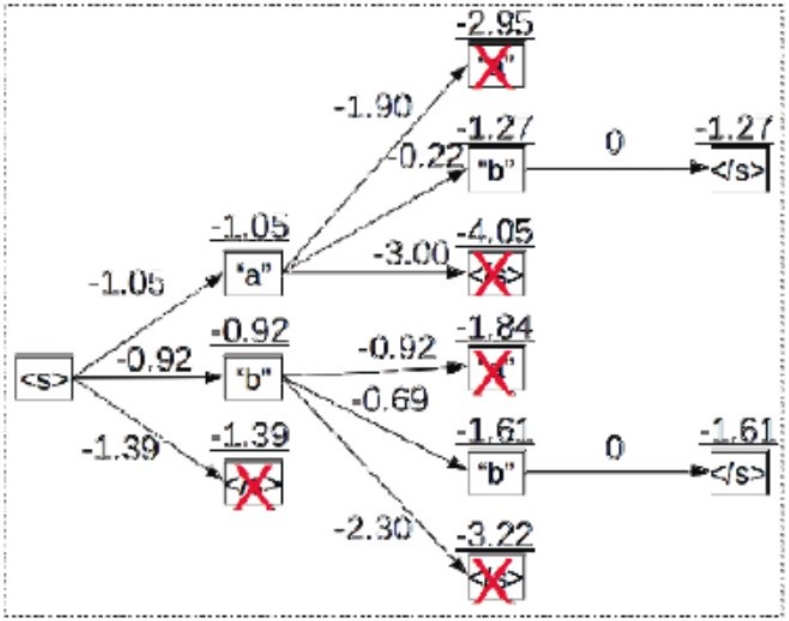
```
while y_{j-1} != "</s>":
  y_j = argmax P(y_j | X, y_1, ..., y_{j-1})
```

Not exact, real problems:

- Will often generate the "easy" words first
- Will prefer multiple common words to one rare word

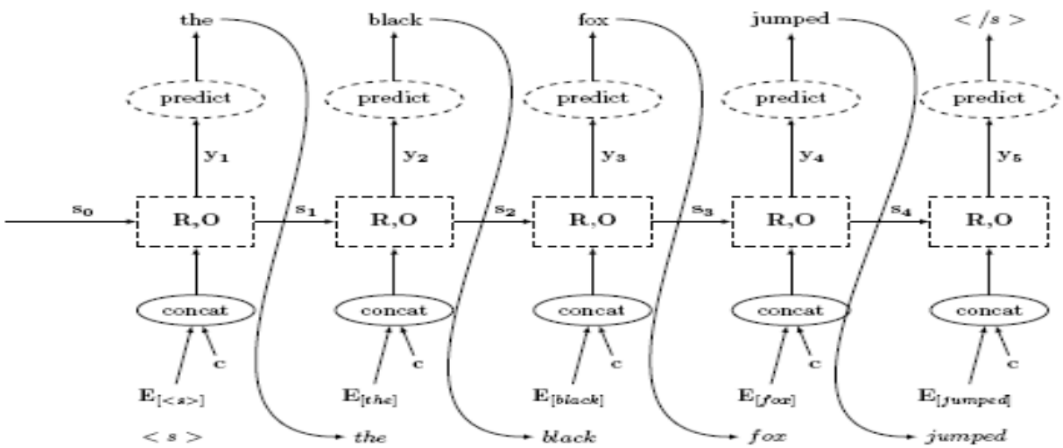
Beam Search

Instead of picking one high-probability word, maintain several paths



How to Train this Model?

- Issues with vanilla training
  - Slow convergence. Model instability. Poor skill.
- Simple idea: **Teacher Forcing**
  - Just feed in the *correct* previous word during training
- Drawback: **Exposure bias**
  - Not exposed to mistakes during training



Solutions to Exposure Bias

- Start with no mistakes, and then
  - gradually introduce them using annealing
- Dropout inputs
  - Helps ensure that the model doesn't rely too heavily on predictions, while still using them
- Corrupt training data

Evaluation Idea 1

- Steps
  - Use parallel test set
  - Use system to generate translations
  - Compare target translations w/ reference

- We train parameters of our model on a **training set**.
- We test the model's performance on data we haven't seen.
  - A **test set** is an unseen dataset that is different from our training set, totally unused.
  - An **evaluation metric** tells us how well our model does on the test set.

Metric: Human Evaluation

太郎が花子を訪れた

	Taro visited Hanako	the Taro visited the Hanako	Hanako visited Taro
Adequate?	Yes	Yes	No
Fluent?	Yes	No	Yes
Better?	1	2	3

- Final goal, but slow, expensive, and sometimes inconsistent

Metric: BLEU

- Works by comparing n-gram overlap w/ reference

Reference: Taro visited Hanako

System: the Taro visited the Hanako

1-gram: 3/5
2-gram: 1/4

Brevity: min(1, |System|/|Reference|) = min(1, 5/3)

brevity penalty = 1.0

BLEU-2 = $(3/5*1/4)^{1/2} * 1.0$
= 0.387

- **Pros:** Easy to use, good for measuring system improvement
- **Cons:** Often doesn't match human eval, bad for comparing very different systems

Metric: METEOR

- Like BLEU in overall principle, with many other tricks: consider paraphrases, reordering, and function word/content word difference
- **Pros:** Generally significantly better than BLEU, esp. for high-resource languages
- **Cons:** Requires extra resources for new languages (although these can be made automatically), and more complicated

Evaluation Idea 2

- Don’t generate a translation. Instead ask:
- Does our language model prefer good sentences to bad ones?
  - Assign a high probability to “real” sentence

Intuition of Perplexity

- The Shannon Game:
  - How well can we predict the next word?

I always order pizza with cheese and ____

The 33rd President of the US was ____

I saw a ____

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

....

fried rice 0.0001

....

and 1e-100

  - Unigrams are terrible at this game. (Why?)
- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

Perplexity is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}}$$

Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

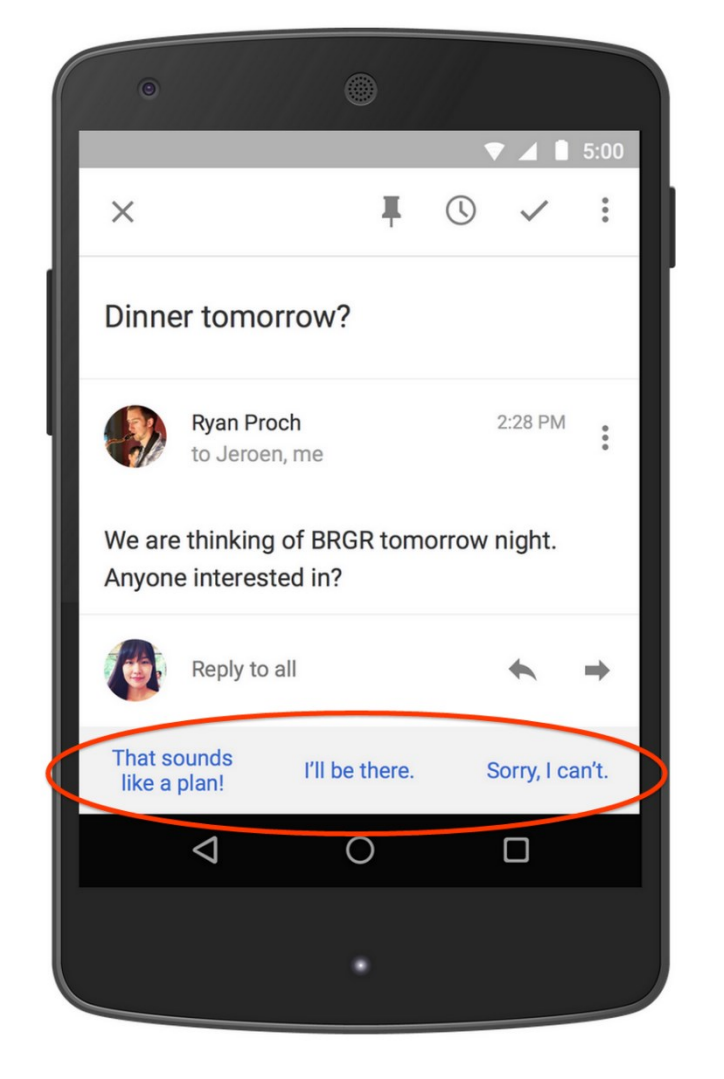
The Shannon Game intuition for perplexity

- How hard is the task of recognizing digits ‘0,1,2,3,4,5,6,7,8,9’
  - Perplexity 10
- How hard is recognizing (30,000) names at Microsoft.
  - Perplexity = 30,000
- If a system has to recognize
  - Operator (1 in 4)
  - Sales (1 in 4)
  - Technical Support (1 in 4)
  - 30,000 names (1 in 120,000 each)
  - Perplexity is 53
- Perplexity is weighted equivalent branching factor

Metric: Perplexity

- Calculate the perplexity of the words in the held-out set *without* doing generation
- **Pros:** Naturally solves multiple-reference problem!
- **Cons:** Doesn't consider decoding or actually generating output.

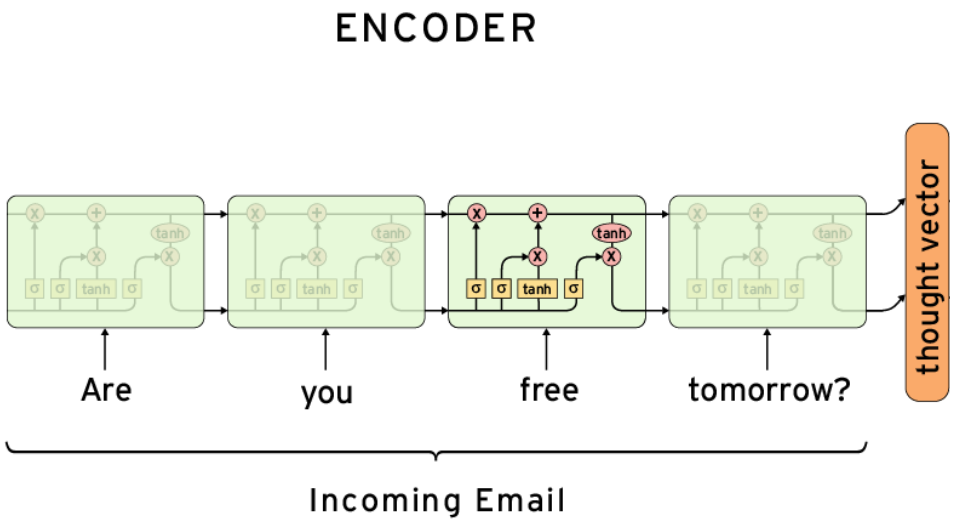
Case Study



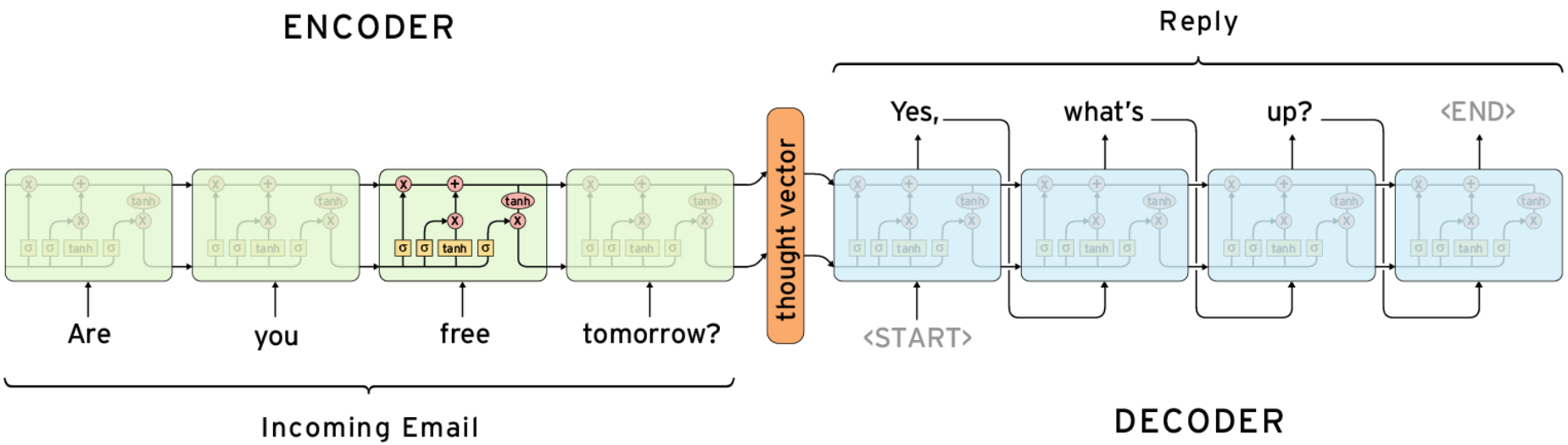
Case Study: Smart Reply in Gmail

Preprocessing an incoming email

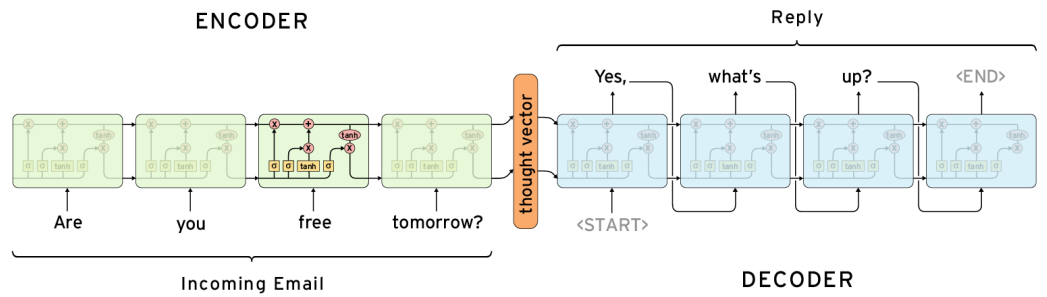
- Language detection
  - Currently handle English, Portuguese, Spanish … a few more languages are in preparation
- Tokenization of subject and message body
- Sentence segmentation
- Normalization of infrequent words and entities – replaced by special tokens
- Removal of quoted and forward email portions
- Removal of greeting and closing phrases (“Hi John”,… “Regards, Mary”)


ENCODER
thought vector
Are
you
free
tomorrow?
Incoming Email

LSTM translation



Vinyals & Le, 2015



Pick the best suggestions
(LSTM)

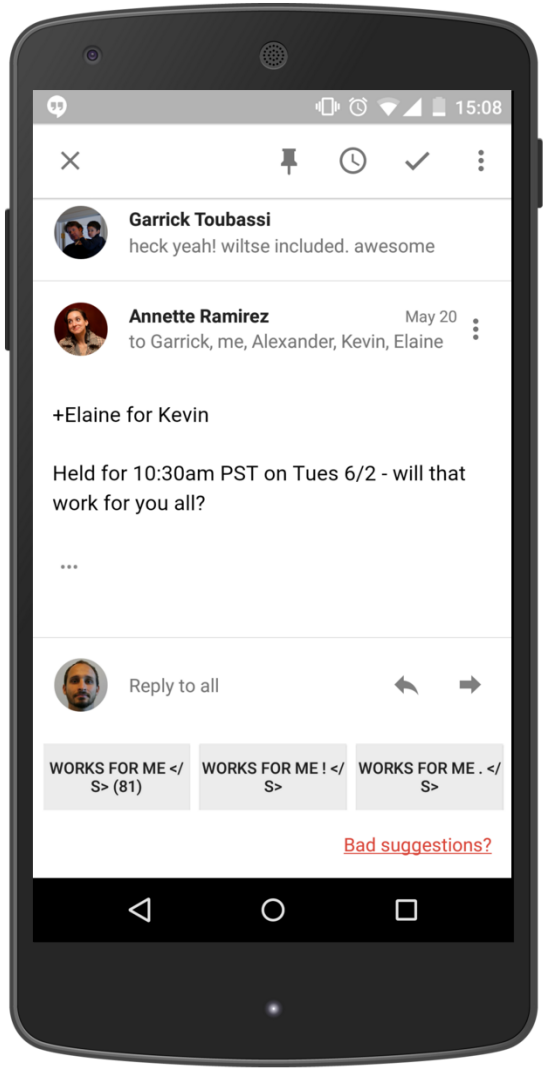
Is it worth it?

- Precision/accuracy - how well can we guess good replies?
- Coverage - do most emails have simple, predictable responses?

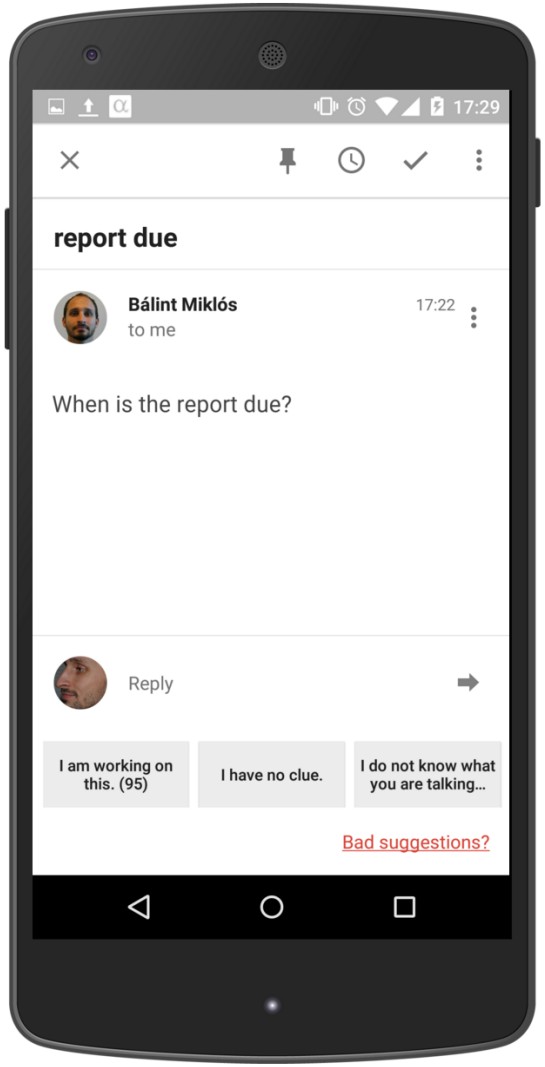
Metric

- What fraction of the time do users select a suggested reply?
  - How many replies do we suggest? 3
  - Constraint based on user interface, but also users' ability to quickly process choices
- We get a boost from allowing users to edit responses before sending
  - In early studies, users were nervous that choosing a response would instantly send
  - Careful tuning of this UI gave us bigger gains than a lot of ML tuning

Some early observations



Some early observations



A scoring algorithm != product

- Semantic variation: all 3 suggestions say the same thing …
  - Can’t simply take the 3 highest scoring suggestions

- The “I love you” problem
  - A human can say them, but not a computer …*
  - A lot of responses in the training corpus have “I love you”
  - In many cases this isn’t appropriate
  - “Family friendliness”

- Sensitivity
  - There are many incoming emails where you don’t want the computer to guess replies - Bad news, etc

- * our expectations of “working” AI are higher than of humans

Michael Gadberry @michaelgadberry · 13h

Google Inbox's automated suggested replies are mind-blowingly awesome and accurate. #CheckOutDatStuff #GoogleInbox @inboxbygmail

Simon Dingle @SimonDingle · Nov 12

It's like @inboxbygmail has telepathy with its automated responses.

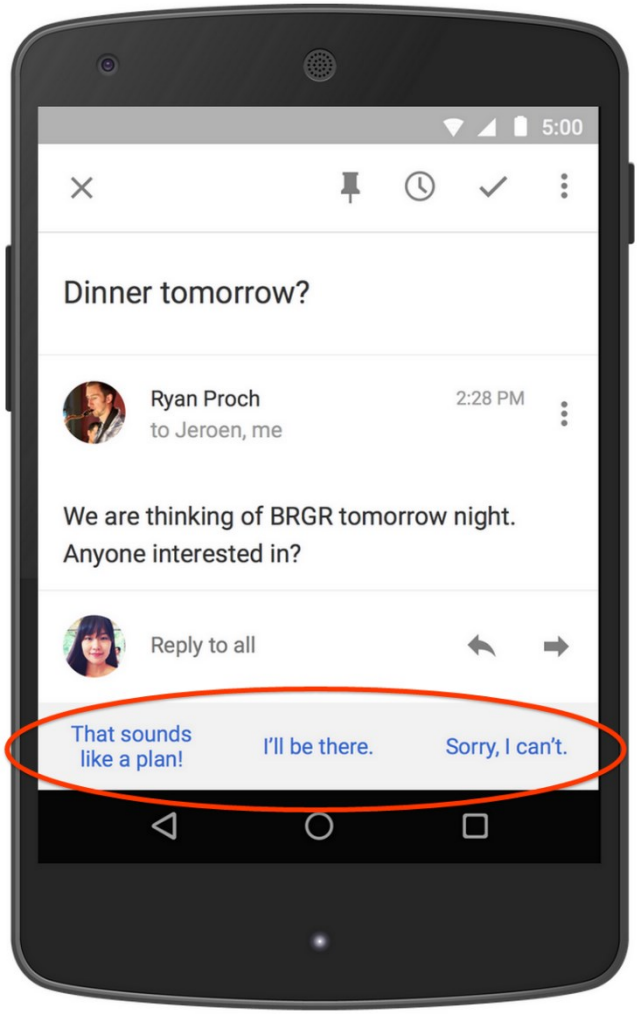
Tatiana King Jones @TatianaKing · Nov 12

The new @inboxbygmail auto response choices have been pretty good so far. Have been using them maybe 50% of the time.

>10%

of Gmail responses are Smart Replies.

(Users accept computer-generated replies.)



Encoder-Decoder with different modalities

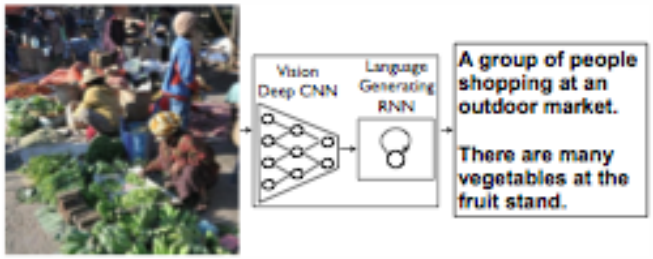
The encoded conditioning context need not be text, or even a sequence.

Encoder-Decoder with different modalities

Show and Tell: A Neural Image Caption Generator

Oriol Vinyals, Google, vinyals@google.com
Alexander Toshev, Google, toshev@google.com
Samy Bengio, Google, bengio@google.com
Dumitru Erhan, Google, dumitru@google.com

- Encode: **image** to vector.
  Decode: a sentence describing the image.



This sort-of works.
In my opinion, looks more impressive than really is.

I think it's a man in a business suit standing on a bench.

I am not really confident, but I think it's a man standing on a beach near the water.

I think it's a group of people sitting in front of a crowd.

I am not really confident, but I think it's a close up of a sheep.

Sequence 2 Sequence

Part II: with attention

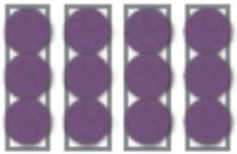
Sentence Representation

You can't cram the meaning of a whole %&!$# sentence into a single $&!#* vector!

But what if we could use multiple vectors, based on the length of the sentence.

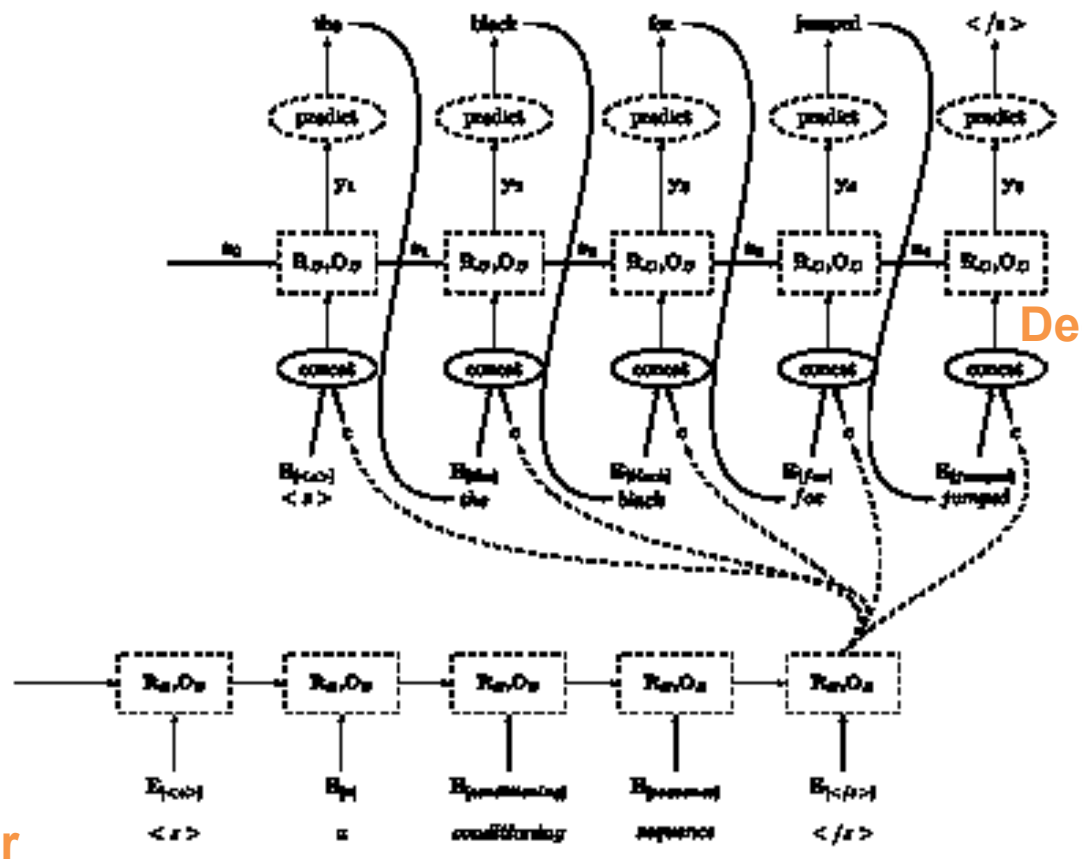
this is an example →

this is an example →



Sequence to Sequence conditioned generation

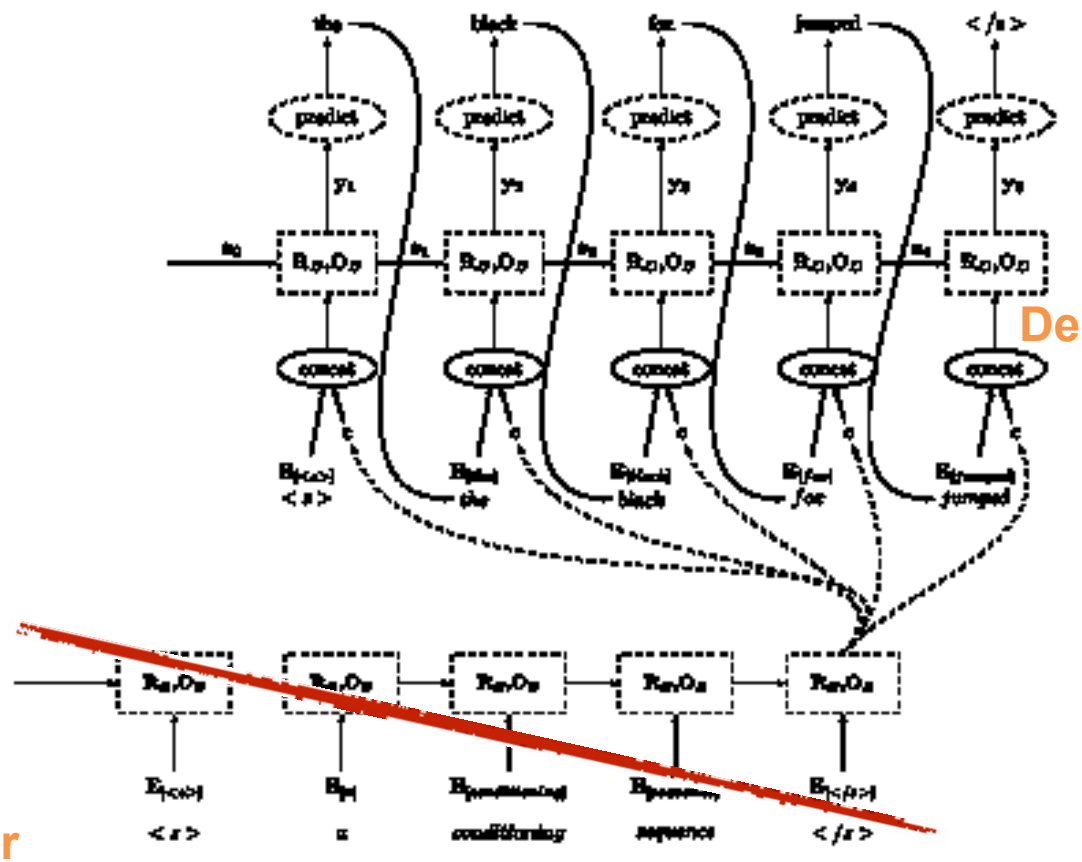
main idea:
encoding
a **single vector** is
too restrictive.



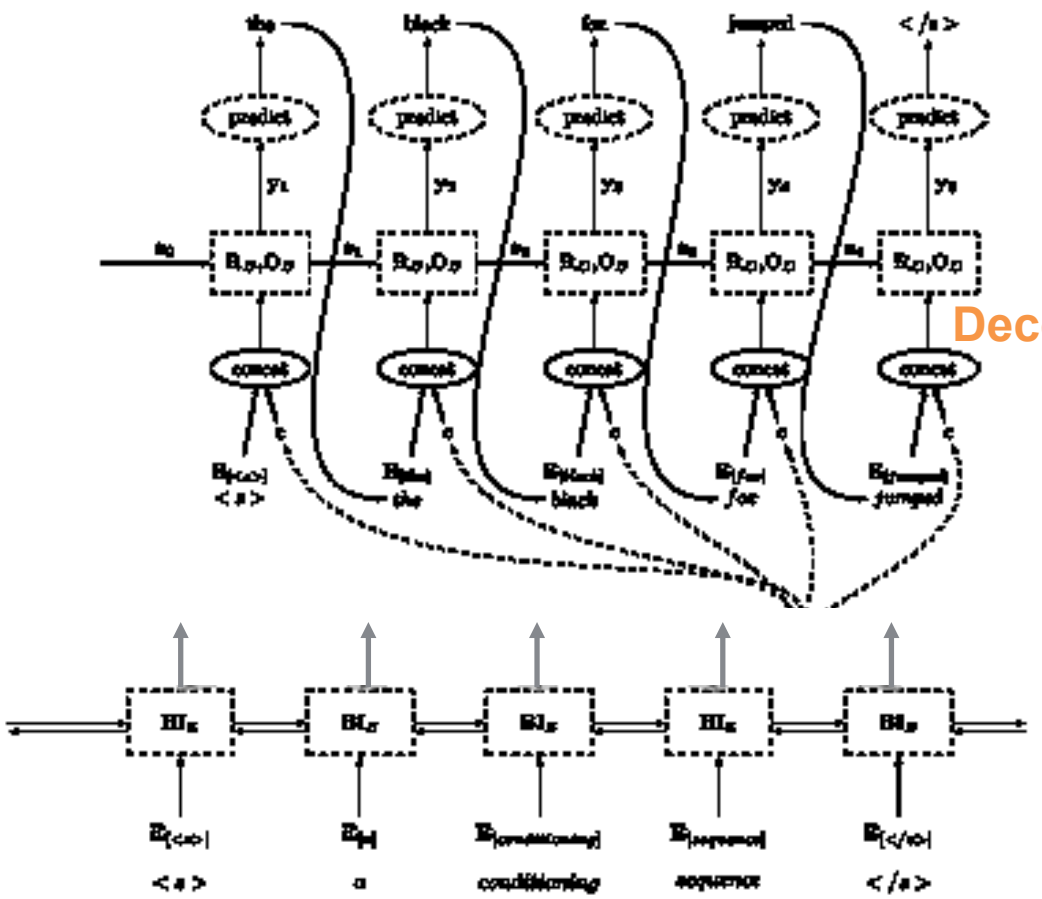
Attention

- 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

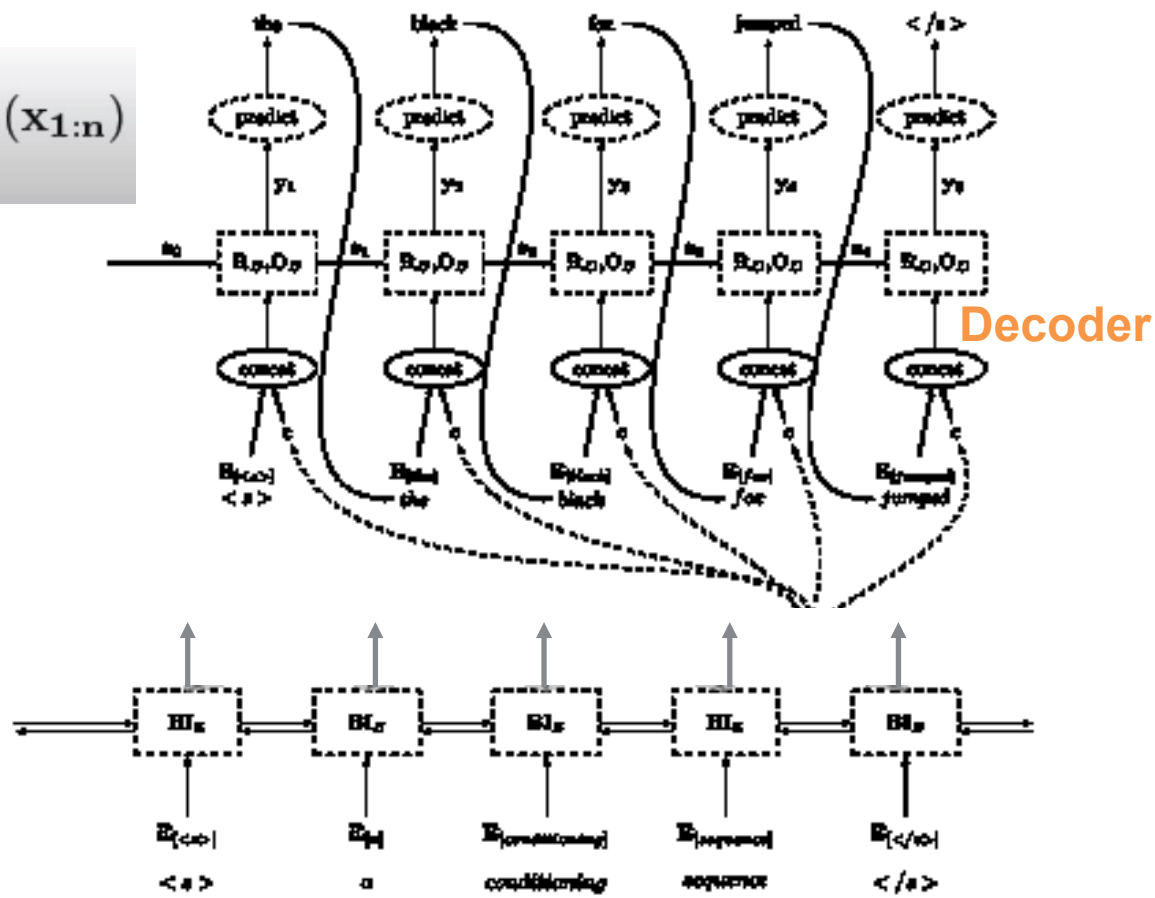


Sequence to Sequence conditioned generation



Sequence to Sequence conditioned generation

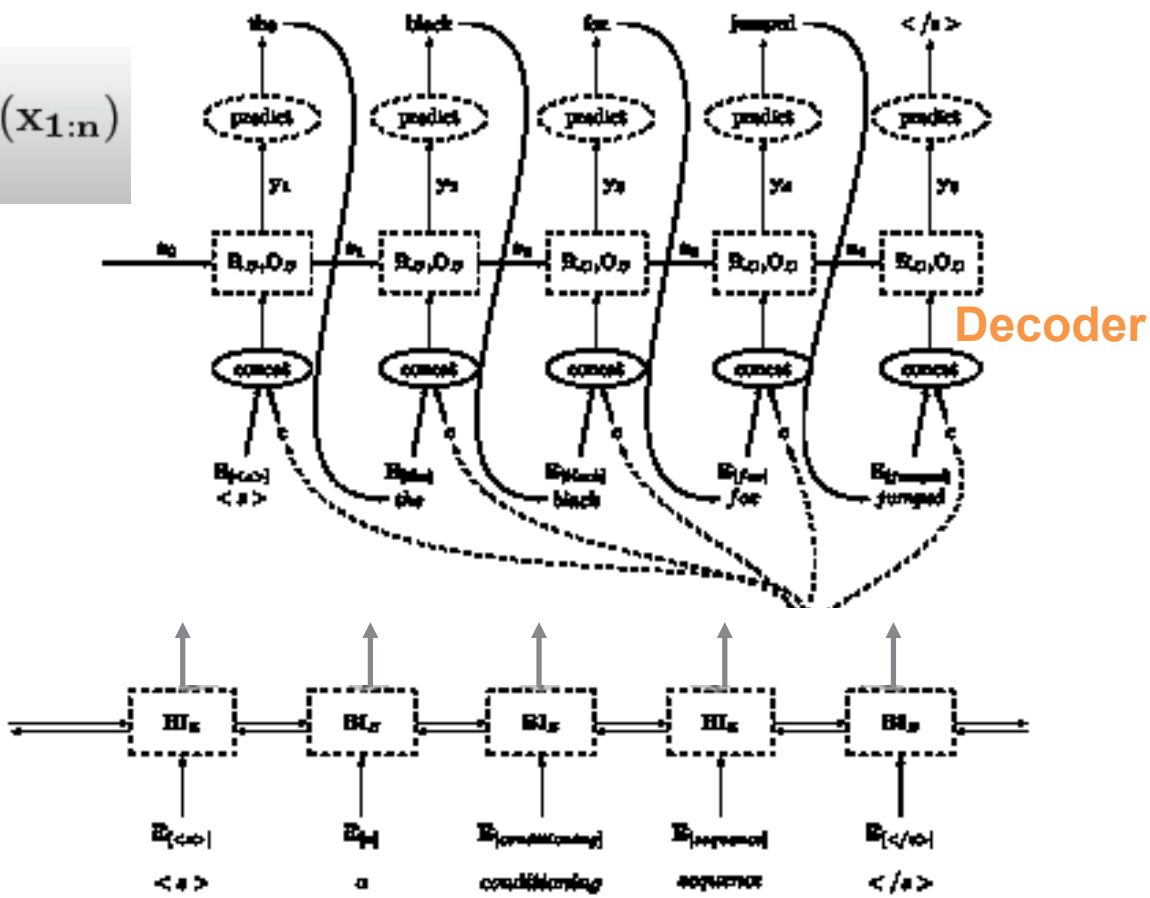
$$\mathbf{c}_{1:n} = \text{ENC}(\mathbf{x}_{1:n}) = \text{biRNN}^{\star}(\mathbf{x}_{1:n})$$



Sequence to Sequence conditioned generation

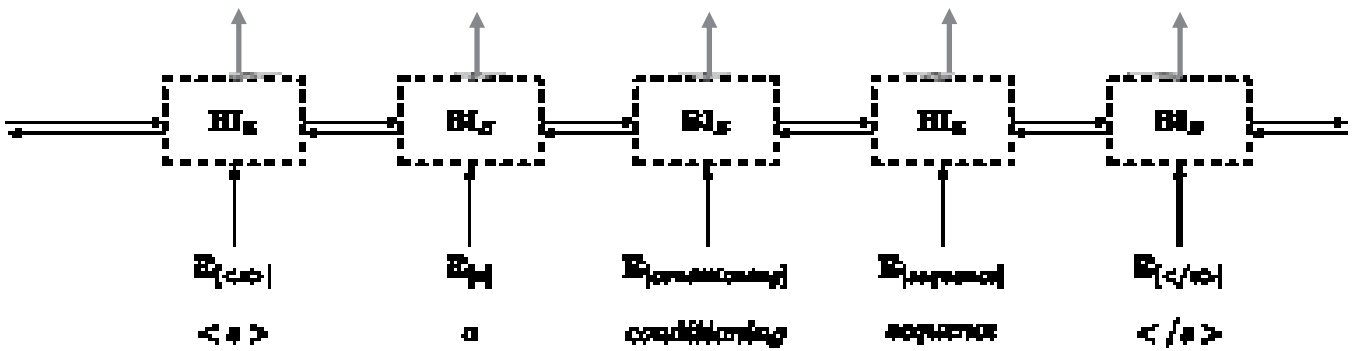
$$\mathbf{c}_{1:n} = \text{ENC}(\mathbf{x}_{1:n}) = \text{biRNN}^{\star}(\mathbf{x}_{1:n})$$

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 (attended summary)

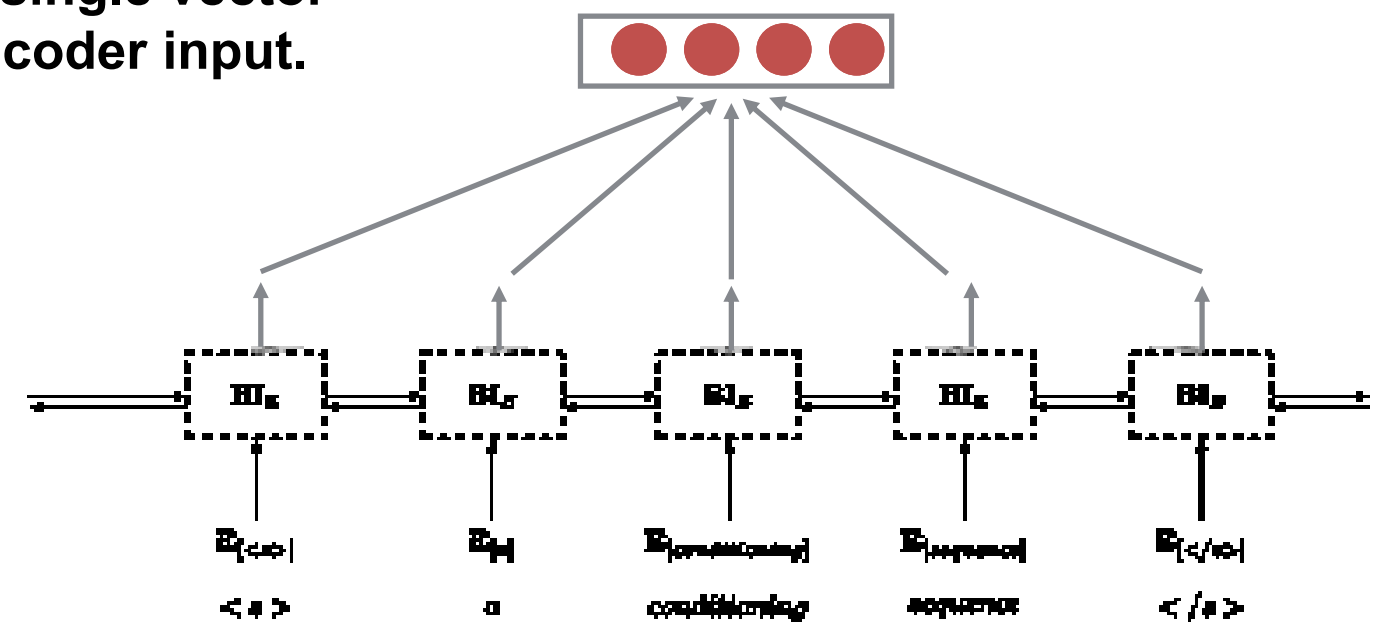


Encoder

Sequence to Sequence conditioned generation

we can combine the different outputs
into a single vector (attended summary)

a different single vector at each encoder input.



Encoder

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

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

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

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

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

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

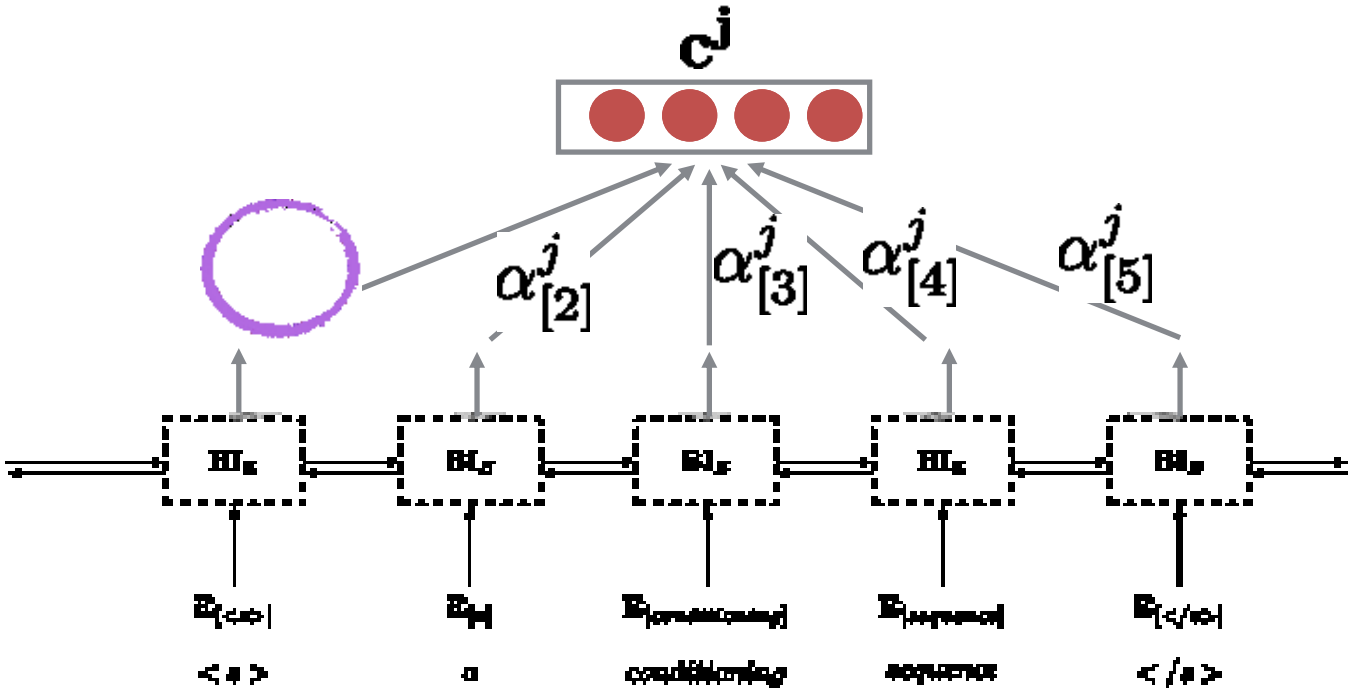
$$\hat{t}_j \sim p(t \mid \hat{t}_{1:j-1}, \mathbf{x_{1:n}})$$

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

Sequence to Sequence conditioned generation

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

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



Encoder

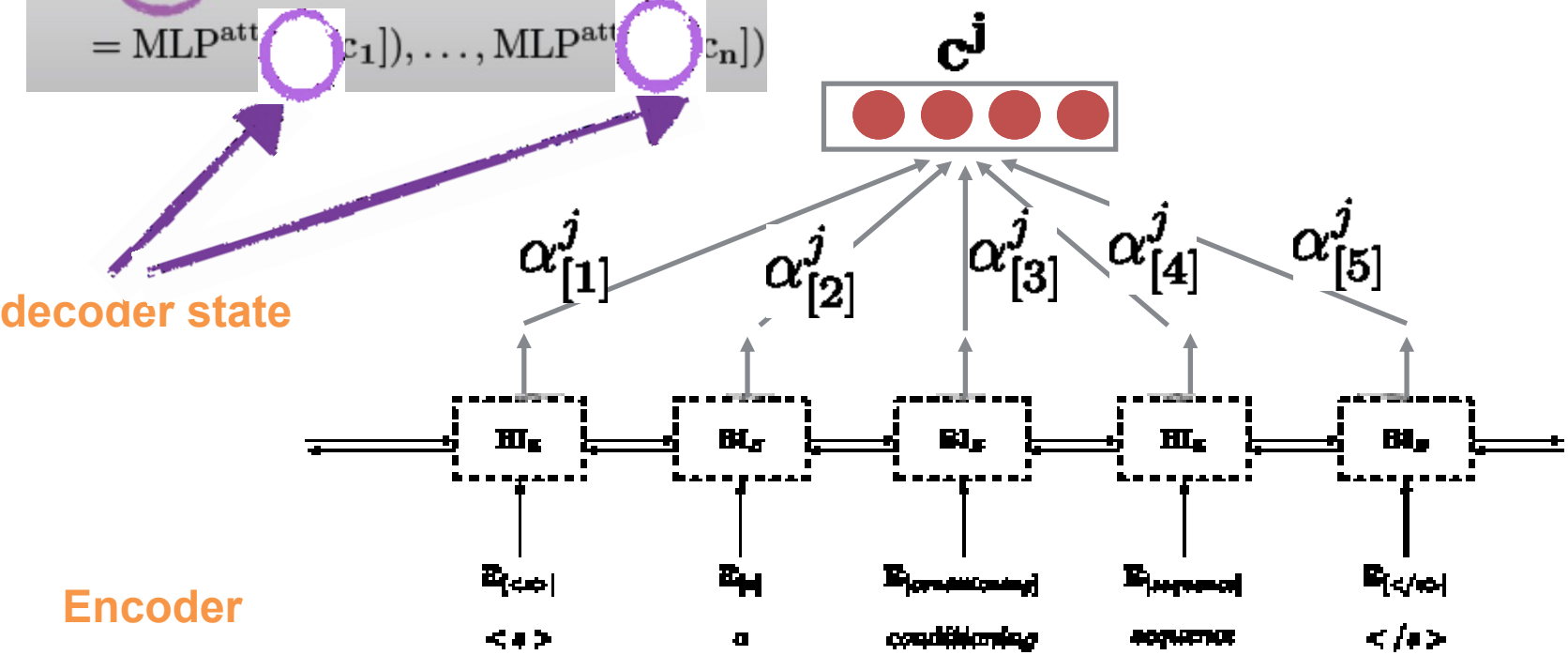
Sequence to Sequence conditioned generation

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

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

$$= \mathrm{MLP}^{\mathrm{att}}([\quad \mathbf{c}_1]), \ldots, \mathrm{MLP}^{\mathrm{att}}([\quad \mathbf{c}_{\mathbf{n}}])$$

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



encoder-decoder with attention

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

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

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

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

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

$$\bar{\alpha}^{\mathbf{j}}_{[\mathbf{i}]} = \text{MLP}^{\text{att}}([\mathbf{s_j}; \mathbf{c_i}])$$

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

$$f(\mathbf{z}) = \text{softmax}(\text{MLP}^{\text{out}}(\mathbf{z}))$$

$$\text{MLP}^{\text{att}}([\mathbf{s_j}; \mathbf{c_i}]) =$$

encoder-decoder with attention

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

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

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

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

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

$$\bar{\alpha}^{\mathbf{j}}_{[\mathbf{i}]} = \text{MLP}^{\text{att}}([\mathbf{s_j}; \mathbf{c_i}])$$

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

$$f(\mathbf{z}) = \text{softmax}(\text{MLP}^{\text{out}}(\mathbf{z}))$$

$$\text{MLP}^{\text{att}}([\mathbf{s_j}; \mathbf{c_i}]) =$$

encoder-decoder with attention

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

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

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

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

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

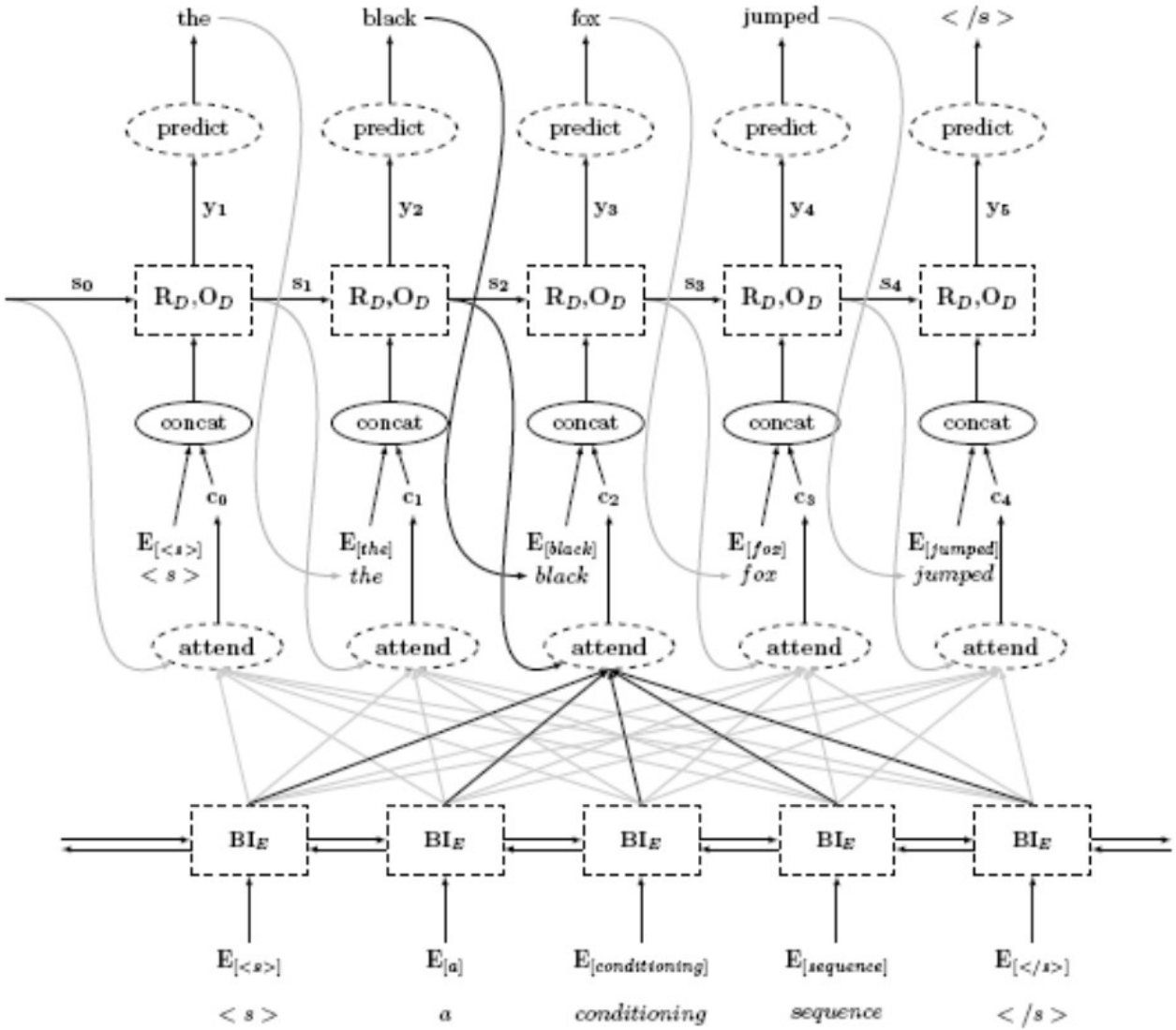
$$\bar{\alpha}^{\mathbf{j}}_{[\mathbf{i}]} = \text{MLP}^{\text{att}}([\mathbf{s_j}; \mathbf{c_i}])$$

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

$$f(\mathbf{z}) = \text{softmax}(\text{MLP}^{\text{out}}(\mathbf{z}))$$

$$\text{MLP}^{\text{att}}([\mathbf{s_j}; \mathbf{c_i}]) =$$

encoder-decoder with attention



encoder-decoder with attention

- Encoder encodes a sequence of vectors, $c_1,\ldots,c_n$
- 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.

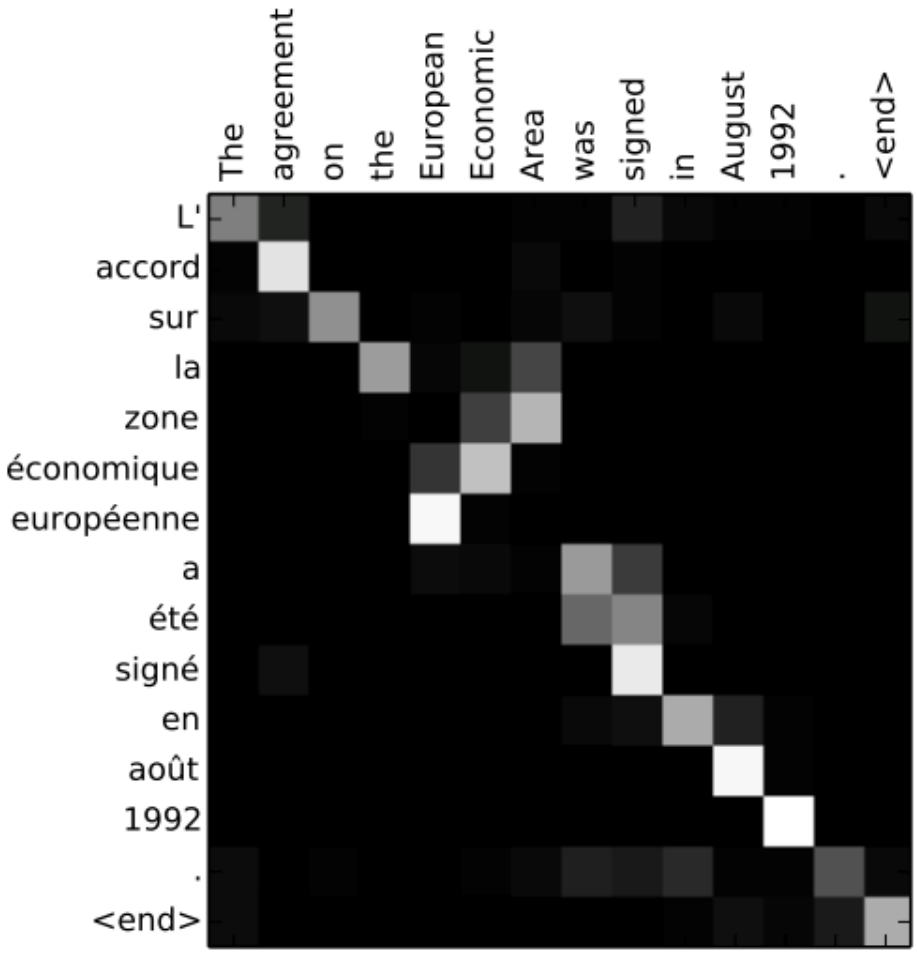
- The encoder acts as a read-only memory for the decoder
- The decoder chooses what to read at each stage

Attention

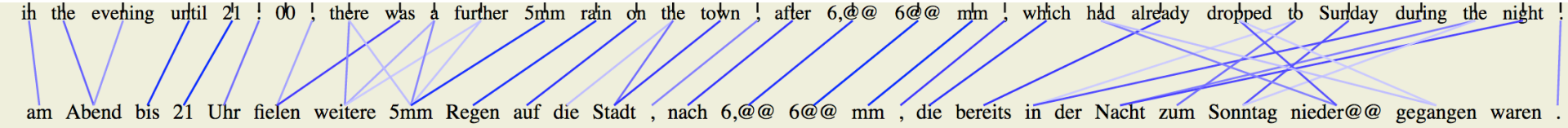
- Attention is very effective for sequence-to-sequence tasks.
- Current state-of-the-art systems all use attention.
  (this is basically how Machine Translation works)

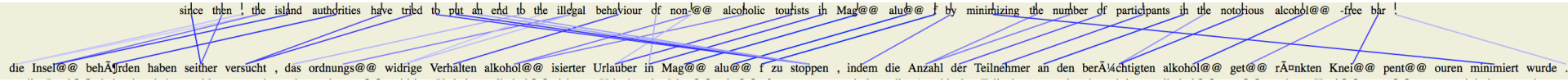
- Attention makes models somewhat more ~interpretable.
- (we can see where the model is "looking" at each stage of the prediction process)

Attention


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

Attention





Attention is not Explanation

Sarthak Jain
Northeastern University
jain.sar@husky.neu.edu

Byron C. Wallace
Northeastern University
b.wallace@northeastern.edu

Complexity

- Encoder decoder:

- Encoder-decoder with attention:

Complexity

- Encoder decoder: O(n+m)
- Encoder-decoder with attention: O(nm)

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 Functions

v: attended vec, q: query vec
$MLP^{att}(q;v)=$

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

- Dot Product: $\mathbf{v} \cdot \mathbf{q}$

- Bilinear attention: $\mathbf{v}^\top \mathbf{W}\mathbf{q}$

Additive vs Multiplicative

While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code.

While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k [3]. 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 $\frac{1}{\sqrt{d_k}}$.

$$\frac{\mathbf{v} \cdot \mathbf{q}}{\sqrt{d_k}}$$

d_k is the dimensionality of q and v

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

Key-Value Attention

- Split v into two vectors $v=[v_k;v_v]$
  - v_k: key vector
  - v_v: value vector

- Use key vector for computing attention

  $MLP^{att}(q;v)= u\,g(\mathbf{W^1}v_k + \mathbf{W^2}q)$ //additive

- Use value vector for computing attended summary

$$\mathbf{v}^{\mathbf{j}} = \sum_{i=1}^{n} \alpha^{\mathbf{j}}_{[\mathbf{i}]} \cdot (v_v)_i$$

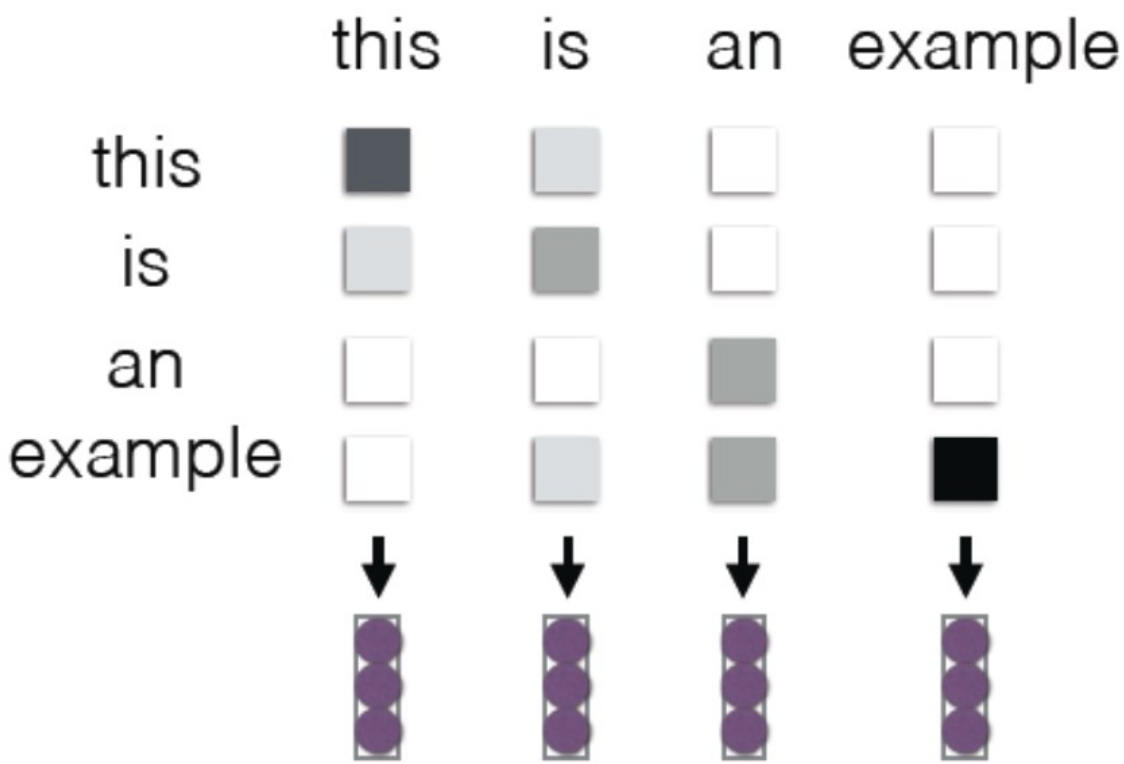
Multi-head Key-Value Attention

- For each head
  - Learn different projection matrices $\mathbf{W_q}$, $\mathbf{W_k}$, $\mathbf{W_v}$
- $MLP^{att}(q;v) = [(v_k\mathbf{W_k}).(q\mathbf{W_q})]/sqrt(d_k)$
- For summary use $v_v\mathbf{W_v}$ (instead of v_v)

- Train many such heads and
  - use aggr(all such attended summaries)

Self-attention/Intra-attention

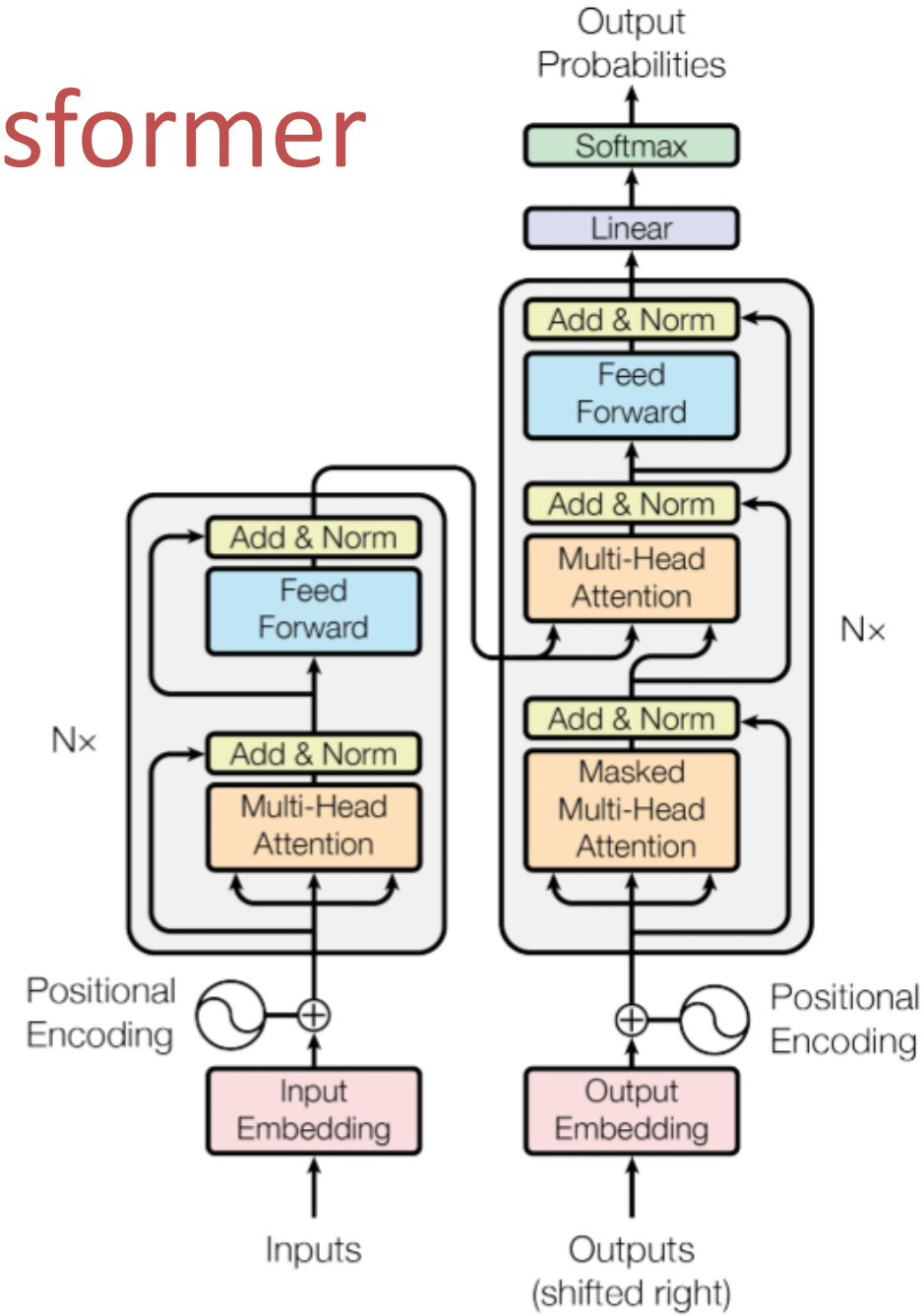
Each element in the sentence attends to other elements → context sensitive encodings!



Do we “need” an LSTM?

- They are slow
  - Sequential nature of computation makes it tough to optimize operations on GPUs
  - Contrast to CNNs: convolutions completely parallelizable
- They are not deep
  - Vanishing gradient: aggravated for deeper networks
  - Less depth → low compositionality power of the network
  - Deepest LSTM networks are 8 layered
    - in-contrast to 50-layered Resnets
- They don’t transfer well
  - Networks trained on one task, do not generalize well to even othe datasets in the same task, not to speak about other tasks
  - ImageNet-trained ResNet fine-tuned on many other datasets

Transformer



BERT: Transformer + PreTraining

- In NLP, we are interested in solving a variety of end tasks - Question Answering, Search, etc.

- One approach - train neural models from scratch

- Issue - this involves two things
  - Modelling of Syntax and Semantics of the language
  - Modelling of the end-task

- Pretraining - Learns the modelling of syntax and semantics - through another task

- So the current model can focus exclusively on modelling of end-task

Pretraining - Masked Language Modelling

- How to pretrain?
- Which base task to choose:
  - Must have abundant data available
  - Must require learning of syntax and semantics
- Language Modelling (Self-supervision)
  - Does not require human annotated labels - abundance of sentences
  - Requires understanding of both syntax and semantics to predict the next word in sentence

Summary

- RNNs are very capable learners of sequential data.
  - n -> 1: (bi)RNN acceptor
  - n -> n : biRNN (transducer)
  - 1 -> m : conditioned generation (conditioned LM)
  - n -> m : conditioned generation (encoder-decoder)
  - n -> m : encoder-decoder with attention
- Transformer
  - More scalable than RNN
  - May slowly replace RNNs