

# An Intro to Deep Learning for NLP

Mausam

(several slides by Yoav Goldberg, Graham Neubig)

# NLP before DL #1

## Assumptions

- doc: bag/sequence/tree of words
- model: bag of features (linear)
- feature: symbolic (diff wt for each)



Model  
(NB, SVM, CRF)



Features



Supervised  
Training  
Data

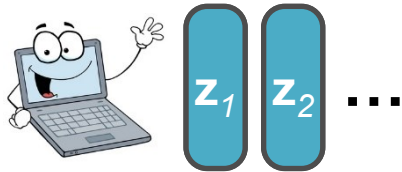


Optimize function  
(LL, sqd error, margin...)



Learn feature weights

# NLP before DL #2



## Assumptions

- doc/query/word is a vector of numbers
- dot product can compute similarity
  - via distributional hypothesis



Model  
(MF, LSA, IR)

Unsupervised  
Co-occurrence  
Data

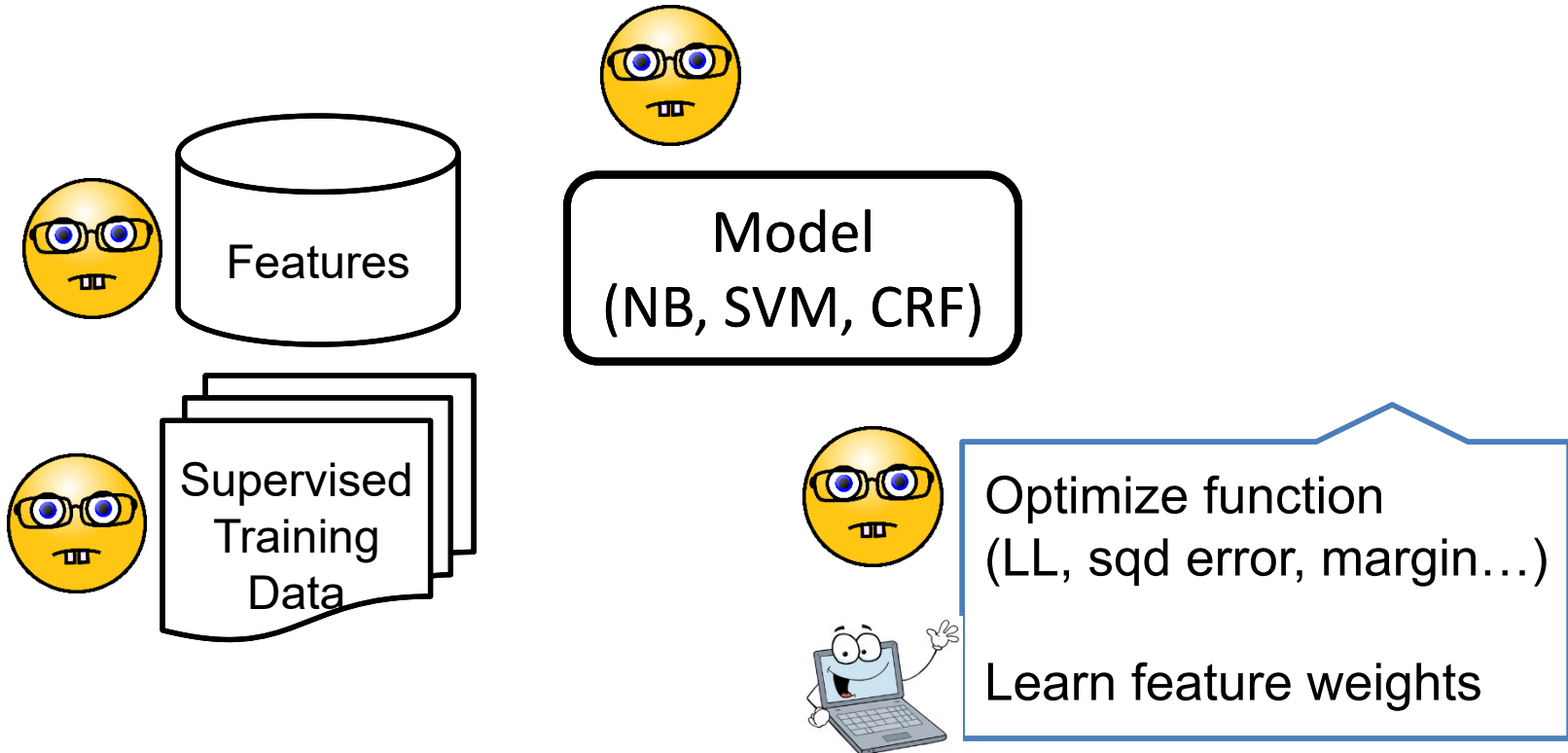


Optimize function  
(LL, sqd error, margin...)

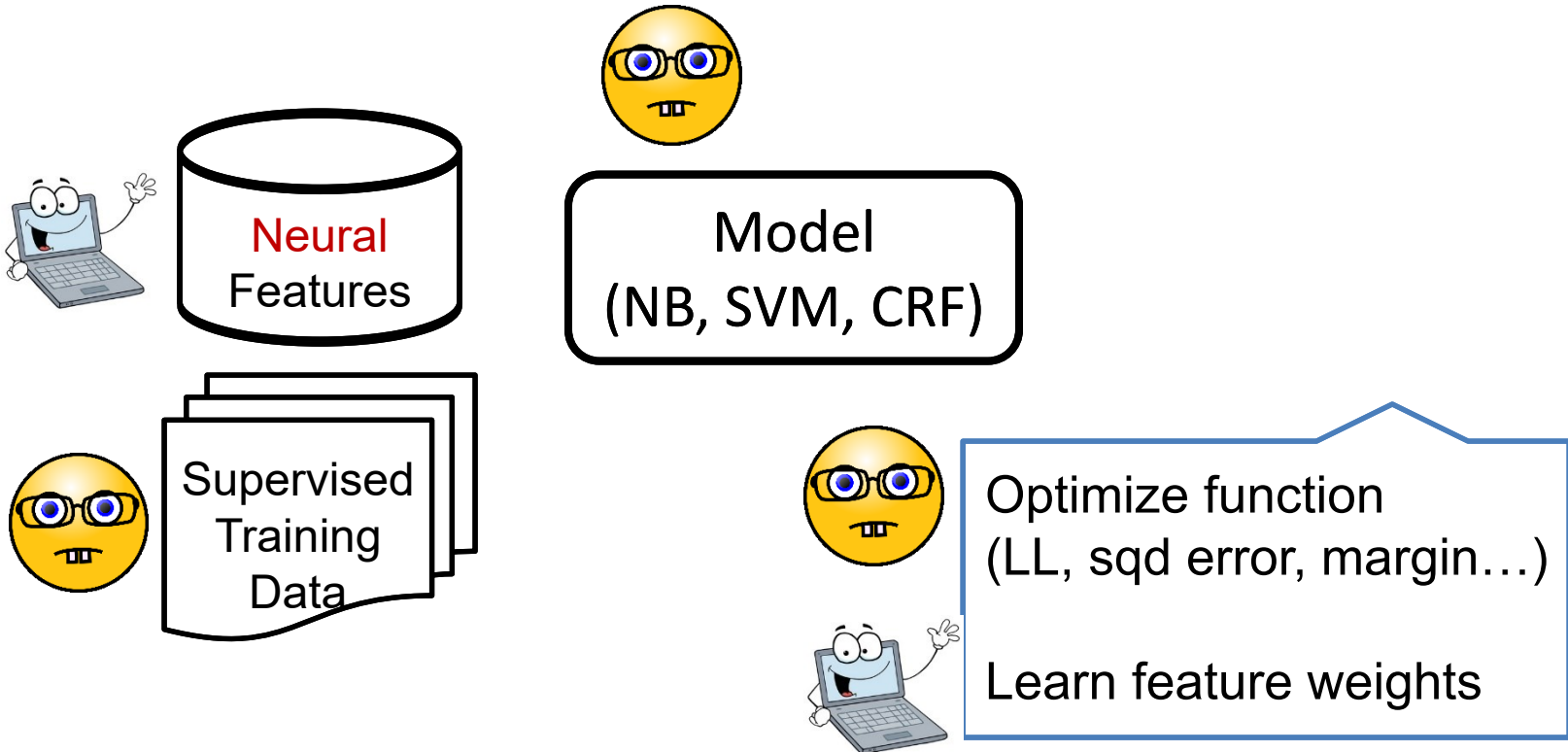
Learn vectors



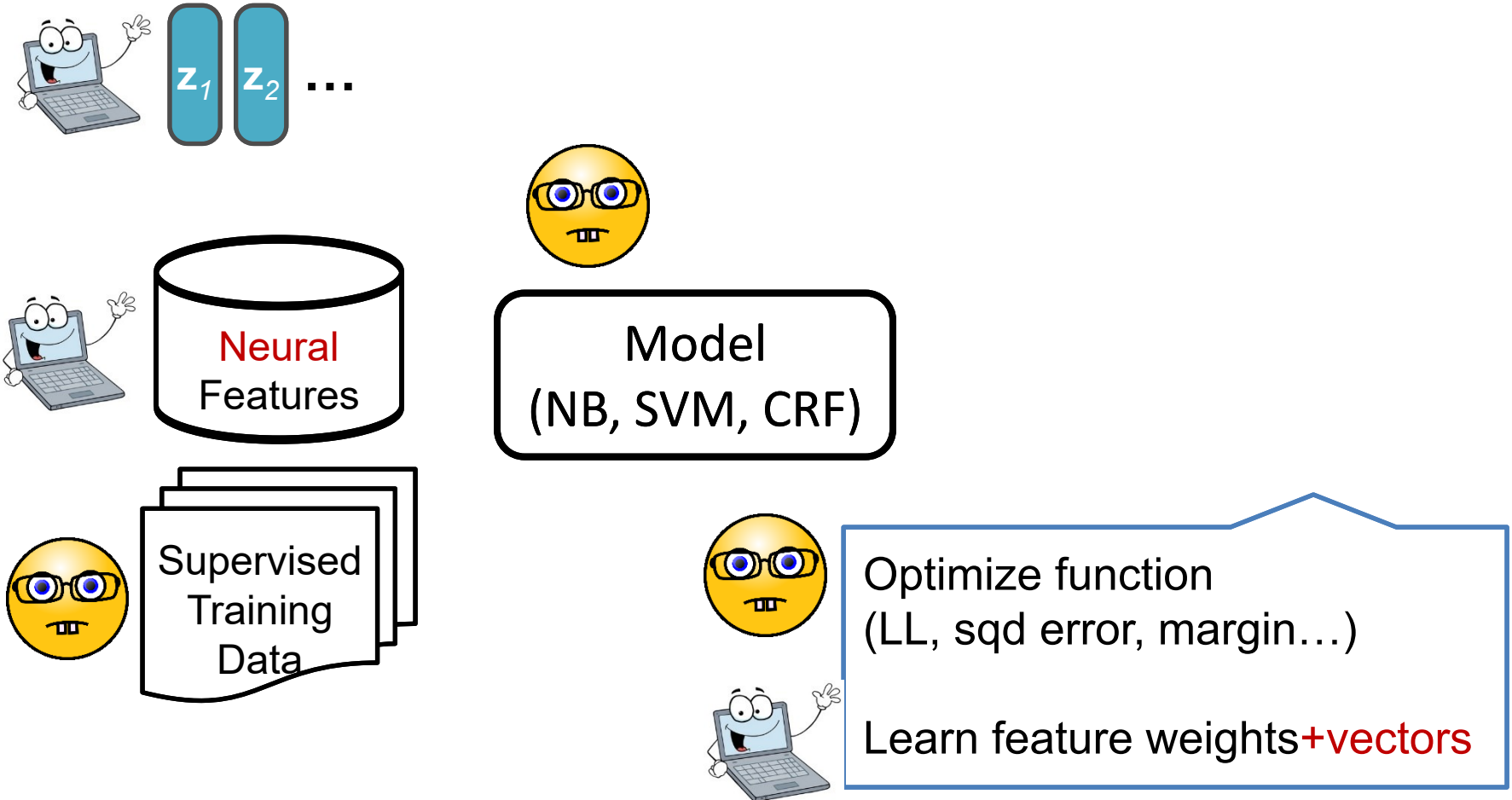
# NLP with DL



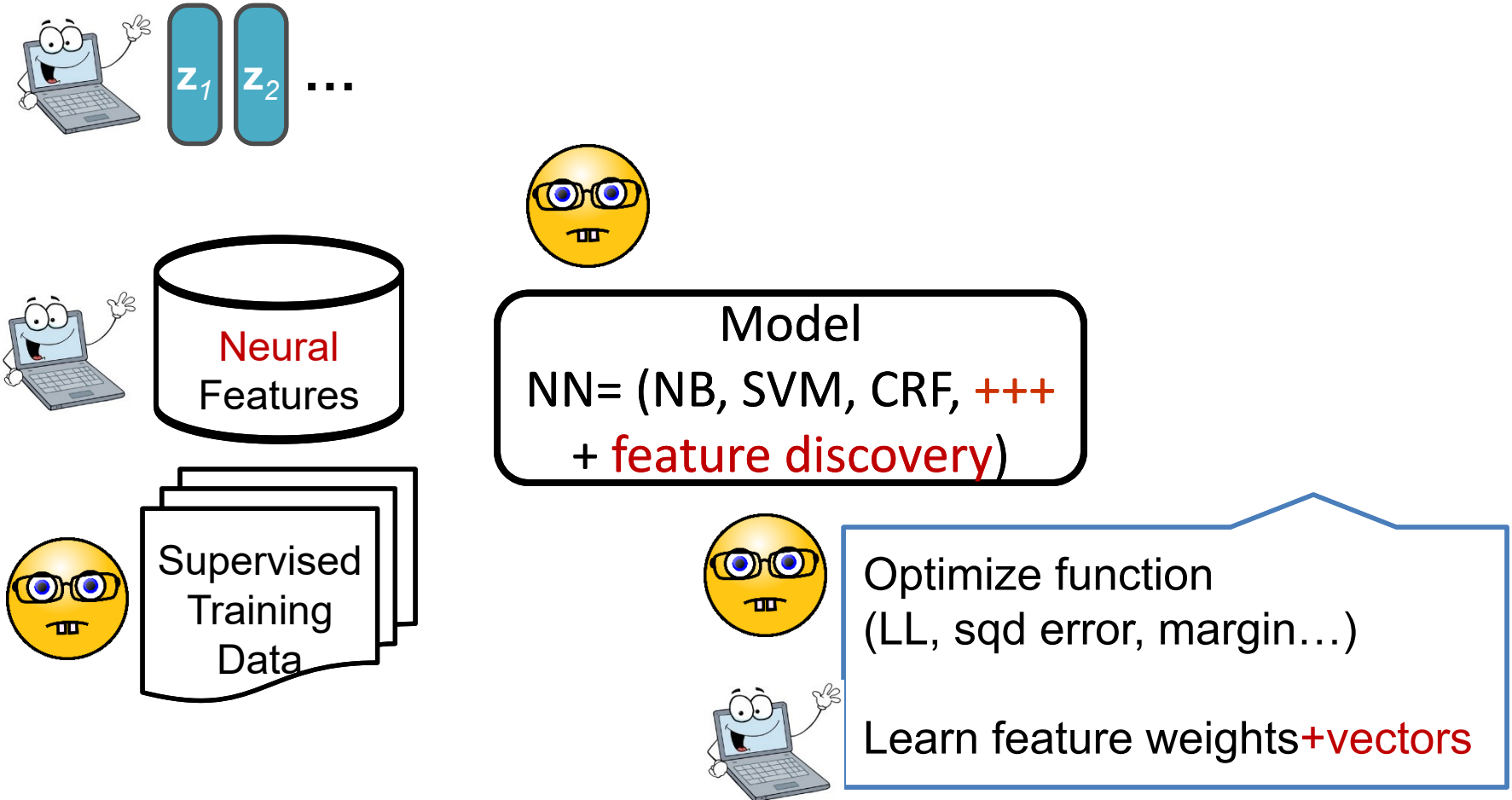
# NLP with DL



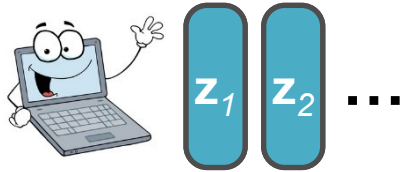
# NLP with DL



# NLP with DL

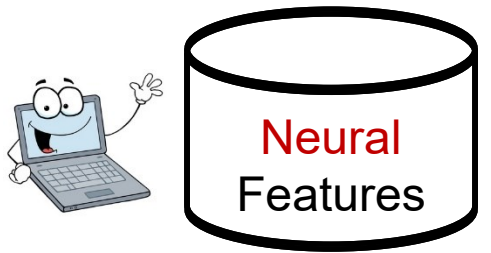


# NLP with DL



## Assumptions

- doc/query/word is a vector of numbers
- doc: bag/sequence/tree of words
- feature: **neural (weights are shared)**
- model: bag/**seq** of features (**non-linear**)



Model  
NN= (NB, SVM, CRF, +++  
+ **feature discovery**)



Optimize function  
(LL, sqd error, margin...)



Learn feature weights+**vectors**



# Meta-thoughts

# Features

- Learned
- in a task specific end2end way
- not limited by human creativity

# Everything is a “Point”

- Word embedding
- Phrase embedding
- Sentence embedding
- Word embedding in context of sentence
- Etc

Points are good → reduce sparsity by wt sharing  
a single (complex) model can handle all pts

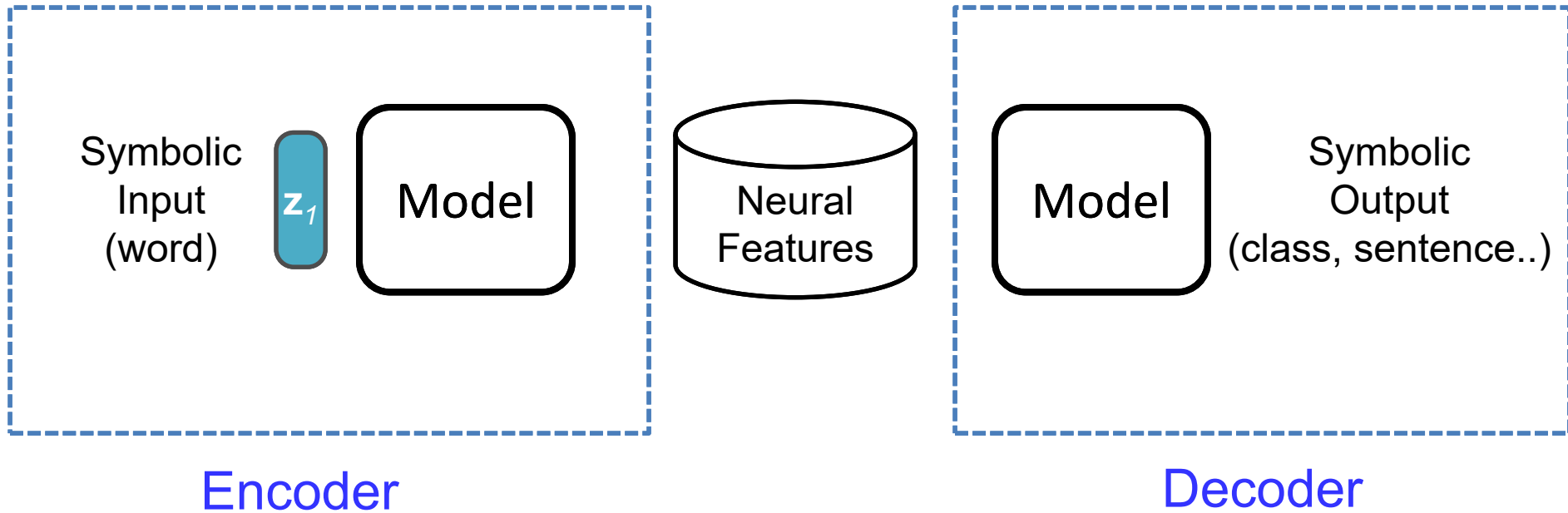
# Universal Representations

- Non-linearities
  - Allow complex functions
- Put anything computable in the loss function
  - Any additional insight about data/external knowledge

# Make symbolic operations continuous

- Symbolic  $\rightarrow$  continuous
  - Yes/No  $\rightarrow$  (number between 0 and 1)
  - Good/bad  $\rightarrow$  (number between -1 and 1)
  
  - Either remember or forget  $\rightarrow$  partially remember
  - Select from n things  $\rightarrow$  weighted avg over n things

# Encoder-Decoder



Different assumptions on data create different architectures

# 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?

# A Primer on D.L. Building Blocks

- A single vector for an ordered pair of vectors? •  $x;y$
- A single vector for a variable-sized bag of vectors? •  $\sum_i x_i$
- Project a vector to a new space? •  $Wx$
- Are two vectors (from same space) similar? •  $x.y$
- Are two vectors (from different space) similar? •  $xWy$
- A new vector that depends on some vector input? •  $g(Wx+b)$



# A Primer on D.L. Building Blocks

- Output a probability
- Output one of two classes
- Output one of many classes
- A feature w/ positive & negative influence
- A feature w/ positive influence for “deep” nets

# A Primer on D.L. Building Blocks

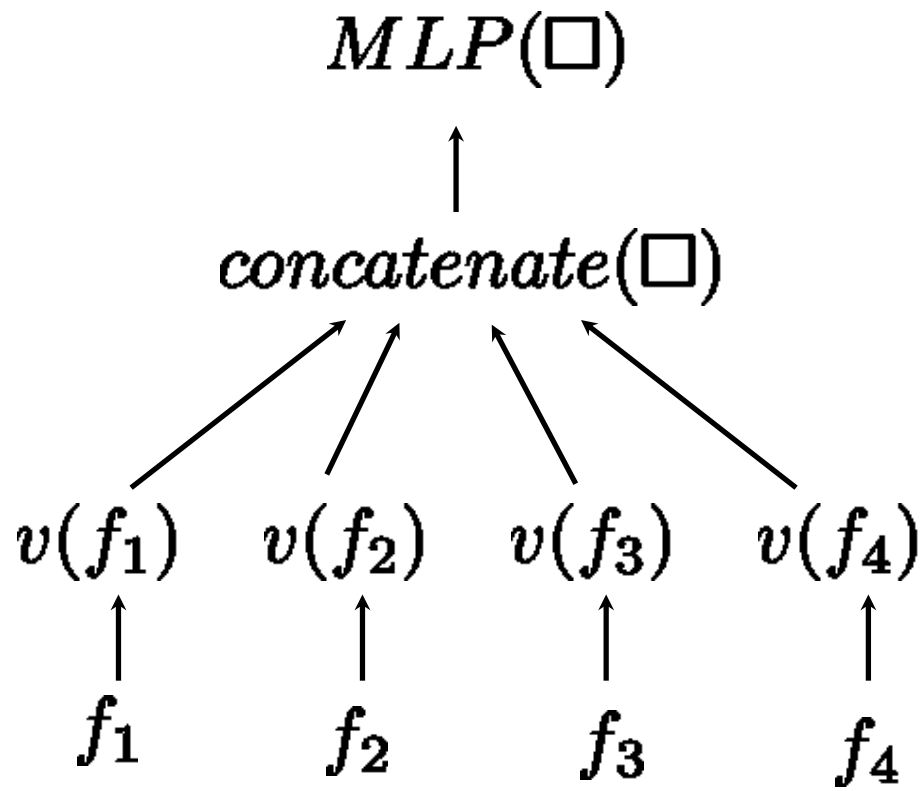
- Output a probability •  $\sigma$
- Output one of two classes •  $\sigma$
- Output one of many classes • softmax
- A feature w/ positive & negative influence • tanh
- A feature w/ positive influence for “deep” nets • ReLu

# Building Blocks

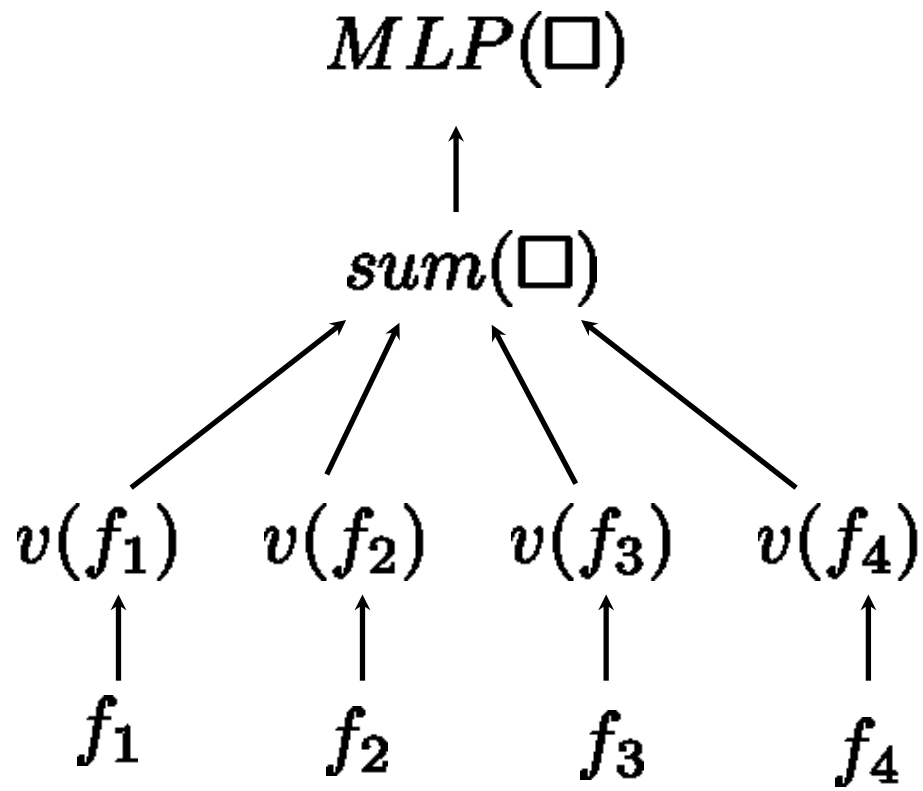
+ ; .

Matrix-mult gate non-linearity

$x; y$



$x+y$



# Concat vs. Sum

- **Concatenating** feature vectors: the "roles" of each vector is retained.

*concat (v("the"), v("thirsty"), v("dog"))*

prev  
word

current  
word

next  
word

- Different features can have vectors of different dim.
- Fixed number of features in each example (need to feed into a fixed dim layer).

# Concat vs. Sum

- **Summing** feature vectors: "bag of features"

$$\text{sum}(v(\text{"the"}), v(\text{"thirsty"}), v(\text{"dog"}))$$

word                  word                  word

- Different feature vectors should have same dim.
- **Can encode a bag of arbitrary number of features.**

x.y

- degree of closeness
- alignment
- Uses
  - question aligns with answer //QA
  - sentence aligns with sentence //paraphrase
  - word aligns with (~important for) sentence //attention



$$g(Ax+b)$$

- 1-layer MLP
- Take  $x$ 
  - project it into a different space //relevant to task
  - add some scalar bias (only increases/decreases it)
  - convert into a required output
- 2-layer MLP
  - Common way to convert input to output

# Encoding Architectures

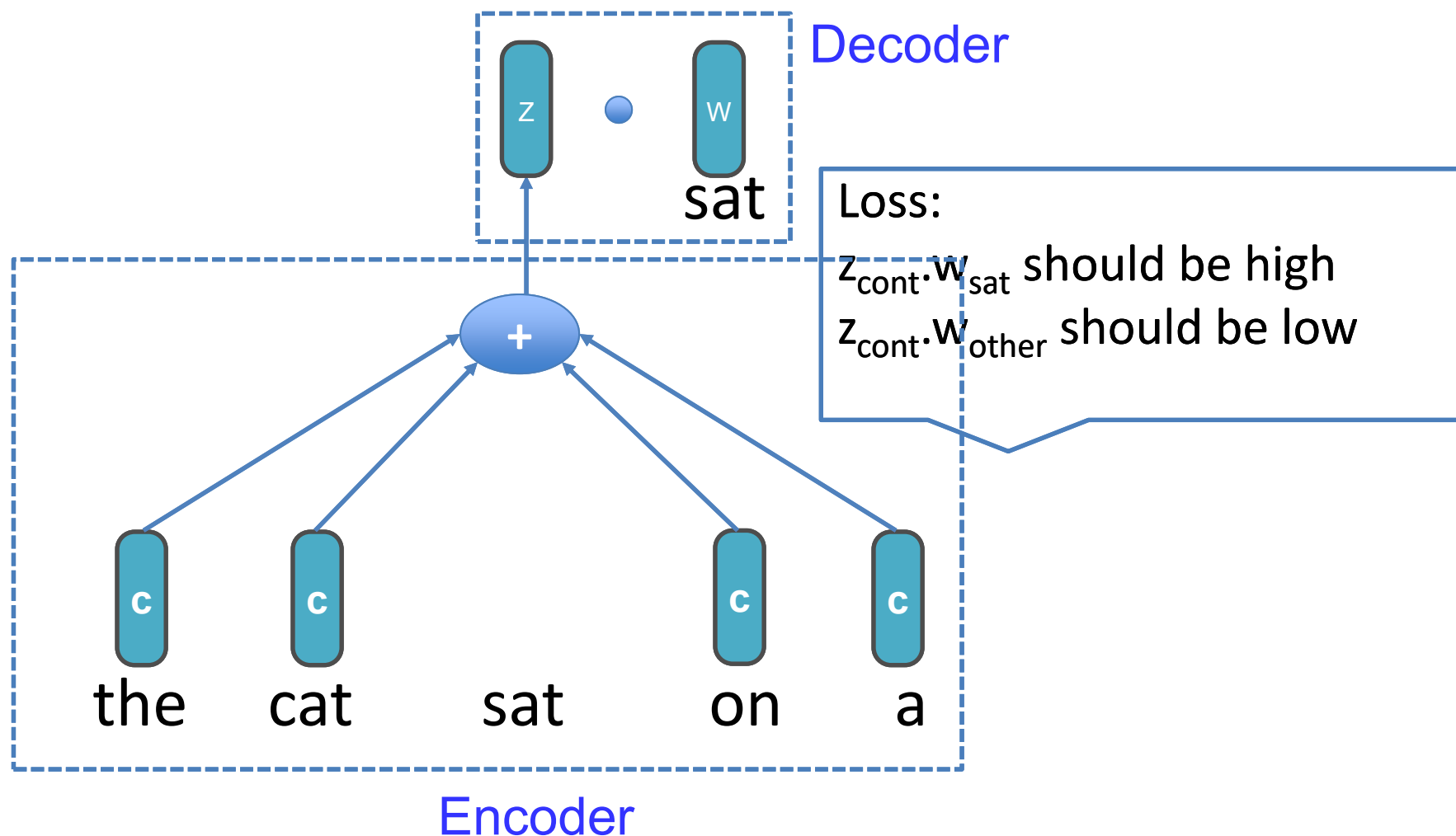
BoW

Bag(N-grams)

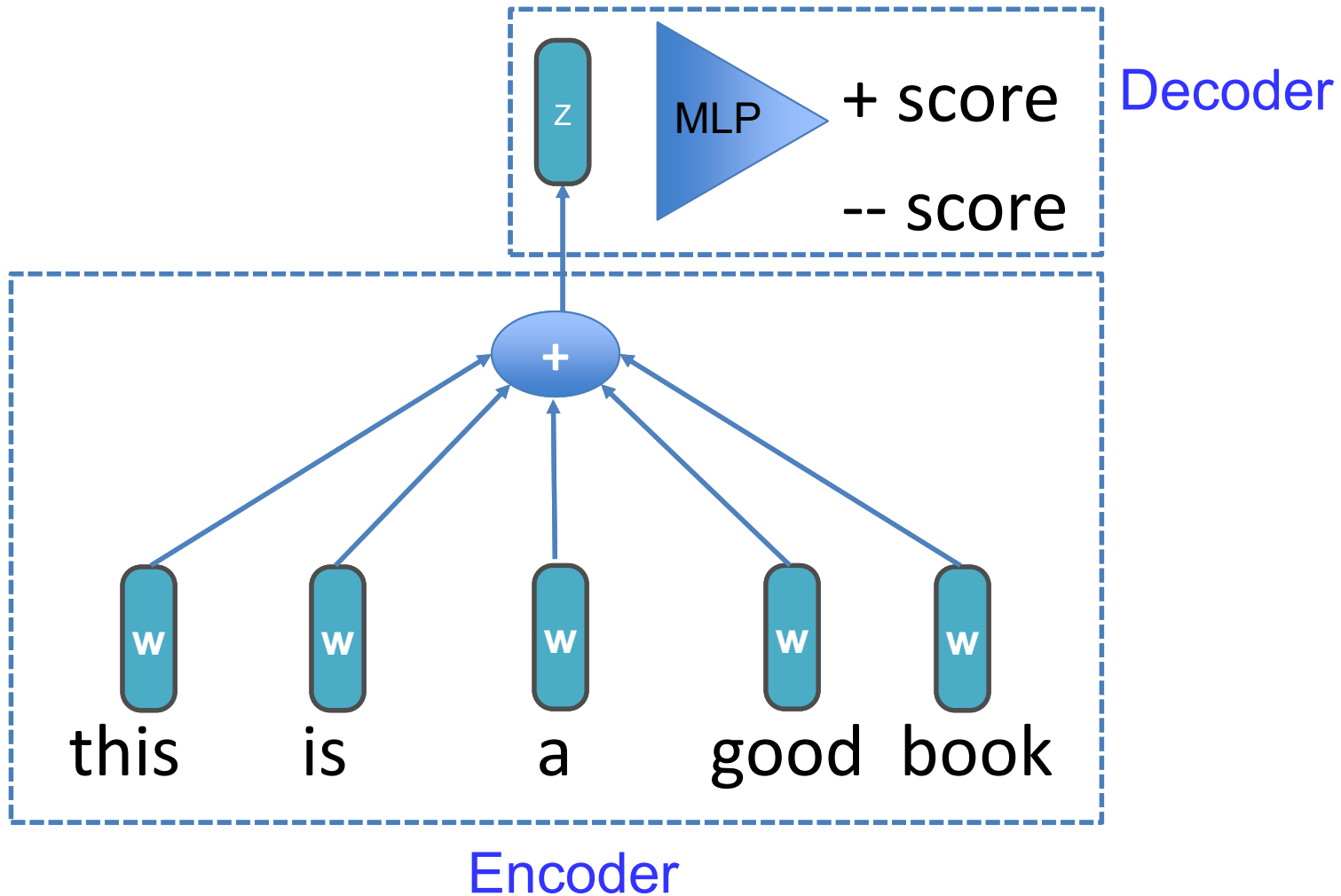
Complete History

Complete History & Future

# Word2Vec: Bag of (Context) Words



# Very Simple Text Classification



# Importance of Ngrams

- We did '+' bec sentences are variable length
  - Ignored order
- While we can ignore global order in many cases...
- ... local ordering is still often very important.
- Local sub-sequences encode useful structures.

**(so why not just assign a vector to each ngram?)**

# CNN: Convolutional Neural Nets

- bag of n-grams encoding
- feature extractor
  - finds whether/how much feature is present
- Instead of sum uses max
  - Indicates presence instead of strength
  - Also called “Max Pooling”



the

actual

service

was

not

very

good



dot



the

actual

service

was

not

very

good

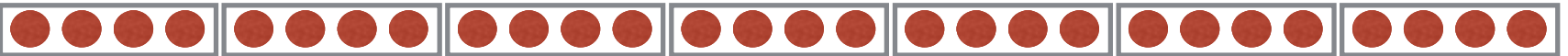




||



dot



the

actual

service

was

not

very

good

the actual



||



dot



the

actual

service

was

not

very

good

the actual

actual service



||



dot



the

actual

service

was

not

very

good

the actual

actual service

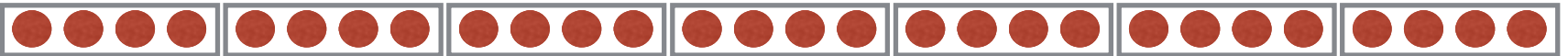
service was



||



dot



the

actual

service

was

not

very

good

the actual

actual service

service was

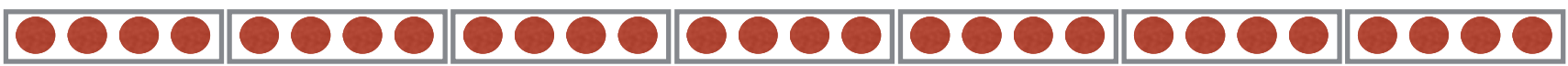
was not



||



dot



the

actual

service

was

not

very

good

the actual

actual service

service was

was not

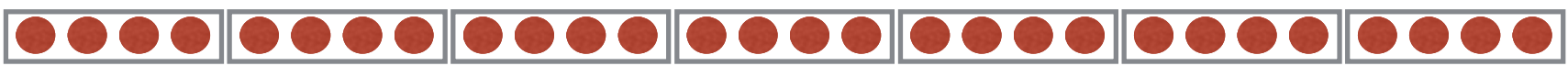
not very



||



dot



the

actual

service

was

not

very

good

the actual

actual service

service was

was not

not very

very good



||



dot



the

actual

service

was

not

very

good

the actual



||



dot



the

actual

service

was

not

very

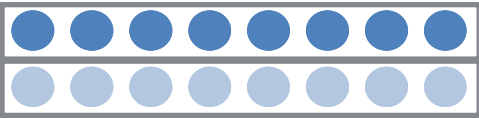
good



the actual



||



dot



the

actual

service

was

not

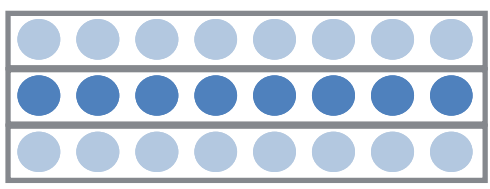
very

good

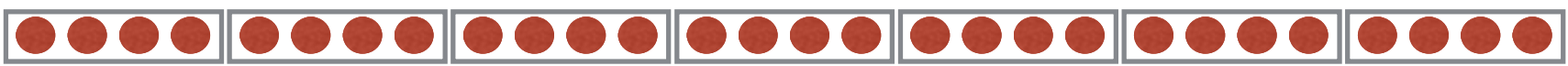
the actual



||



dot



the

actual

service

was

not

very

good

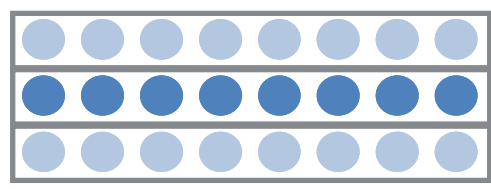
the actual



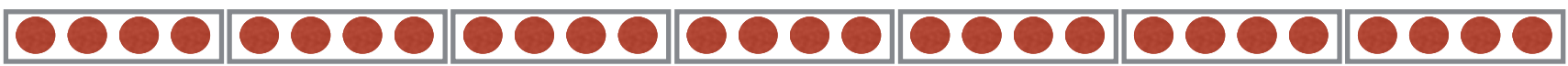
actual service



||



dot



the

actual

service

was

not

very

good

the actual



actual service



service was



was not



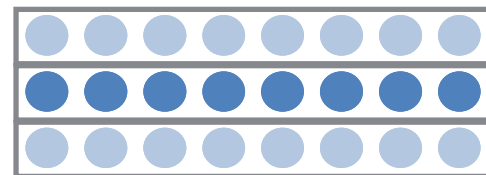
not very



very good



||



dot



the

actual

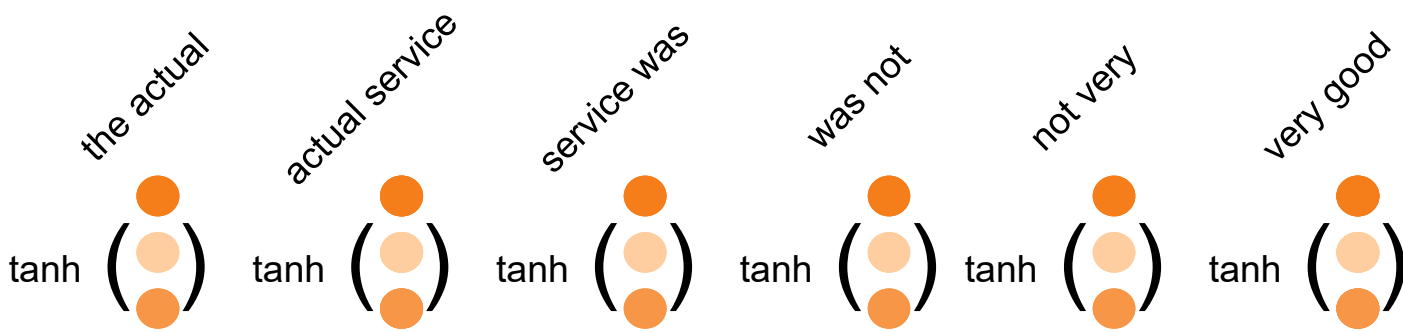
service

was

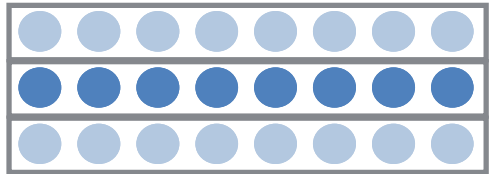
not

very

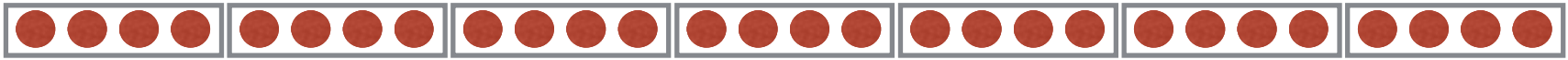
good



||



dot



the      actual      service      was      not      very      good

**(usually also add non linearity)**

the actual



+

actual service



+

service was



+

was not



+

not very

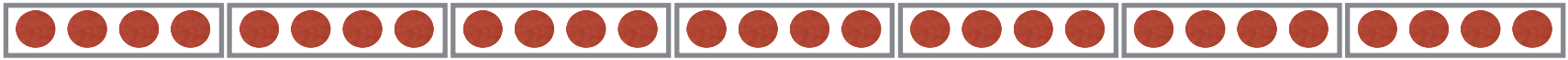


+

very good



=



the

actual

service

was

not

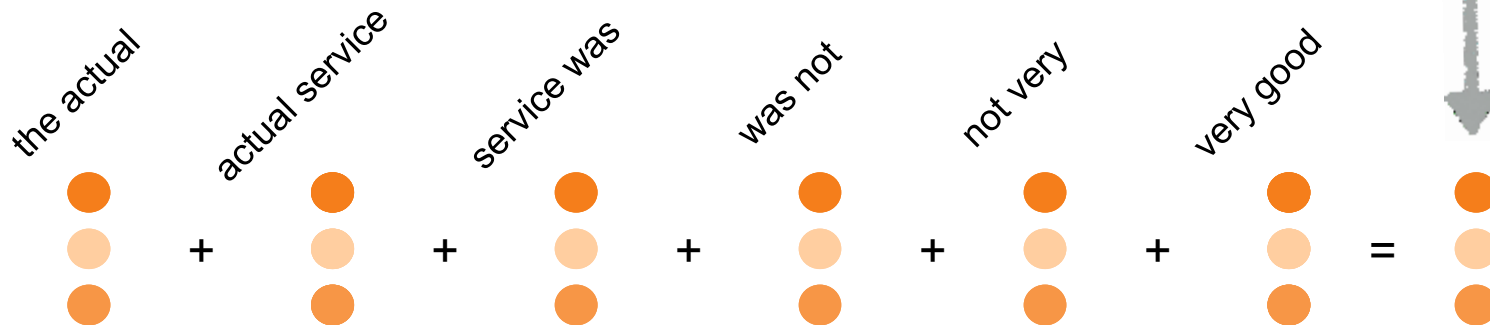
very

good

can do "pooling"

# average pooling

# average vector



the

actual

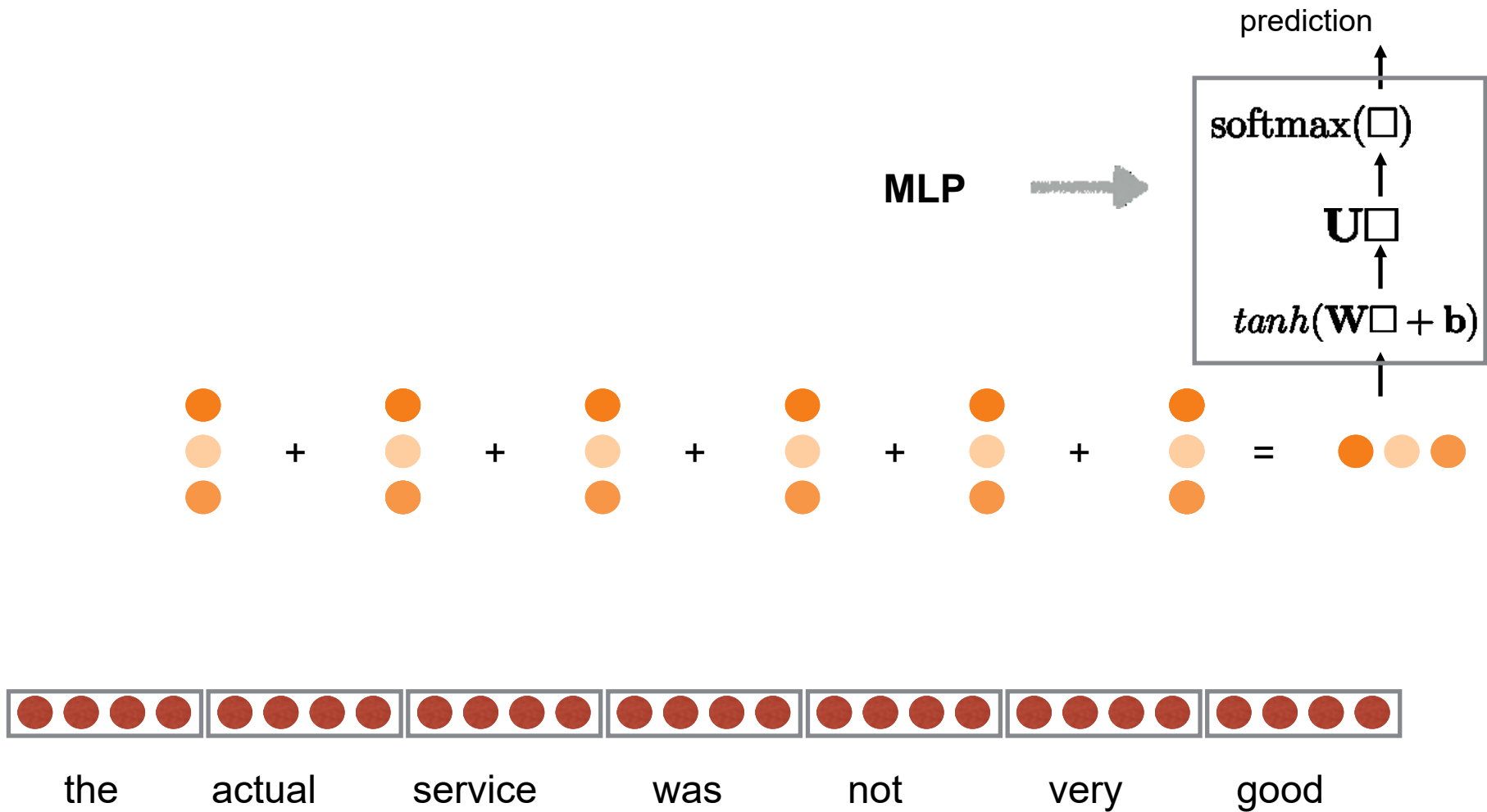
service

was

not

very

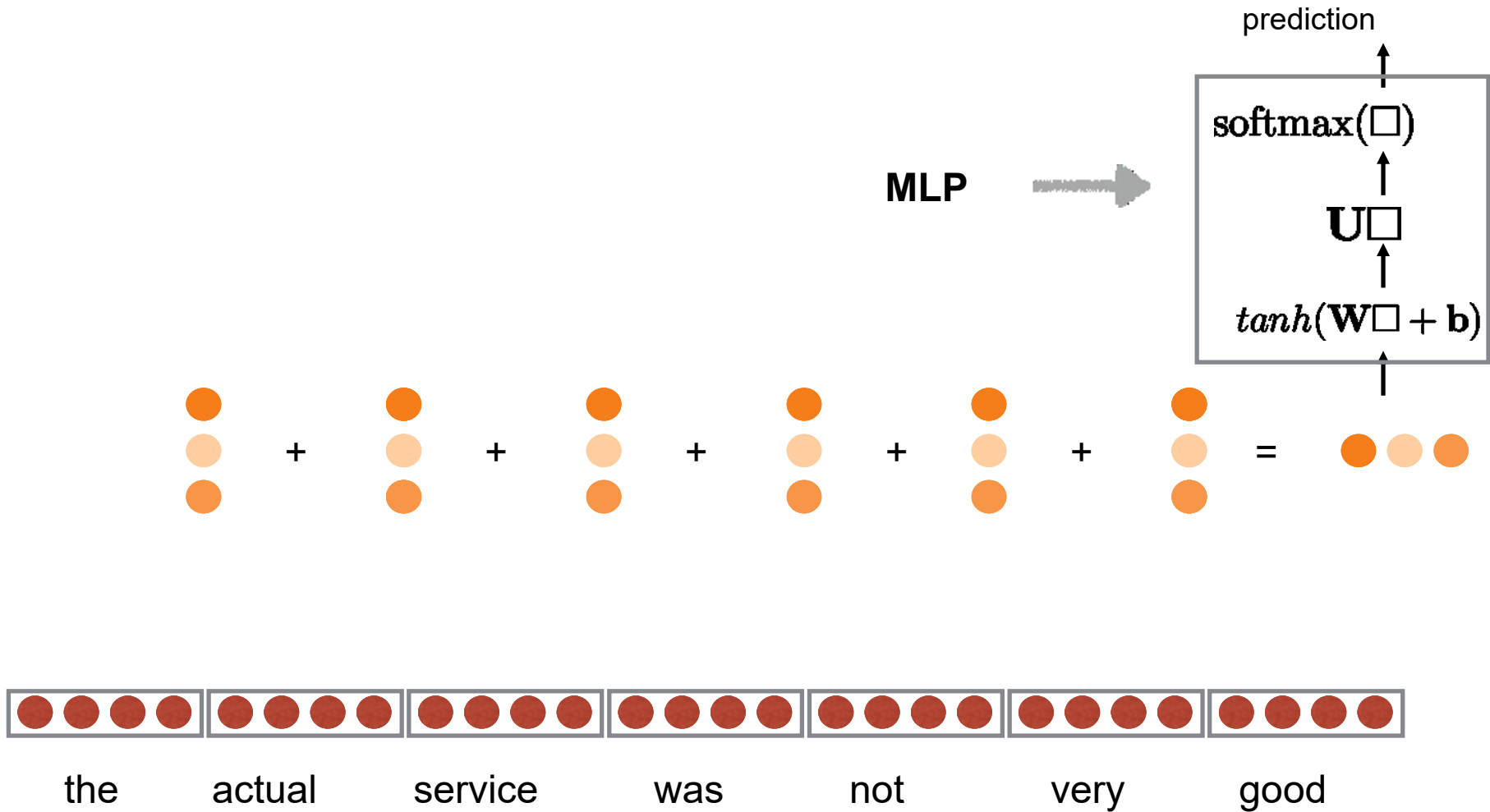
good



**train end-to-end for some task**

(train the MLP, the filter matrix, and the embeddings together)





**train end-to-end for some task**

(train the MLP, the filter matrix, and the embeddings together)

**the vectors learn to capture what's important**

# max pooling

# max vector

the actual

actual service

service was

was not

not very

very good



the

actual

service

was

not

very

good

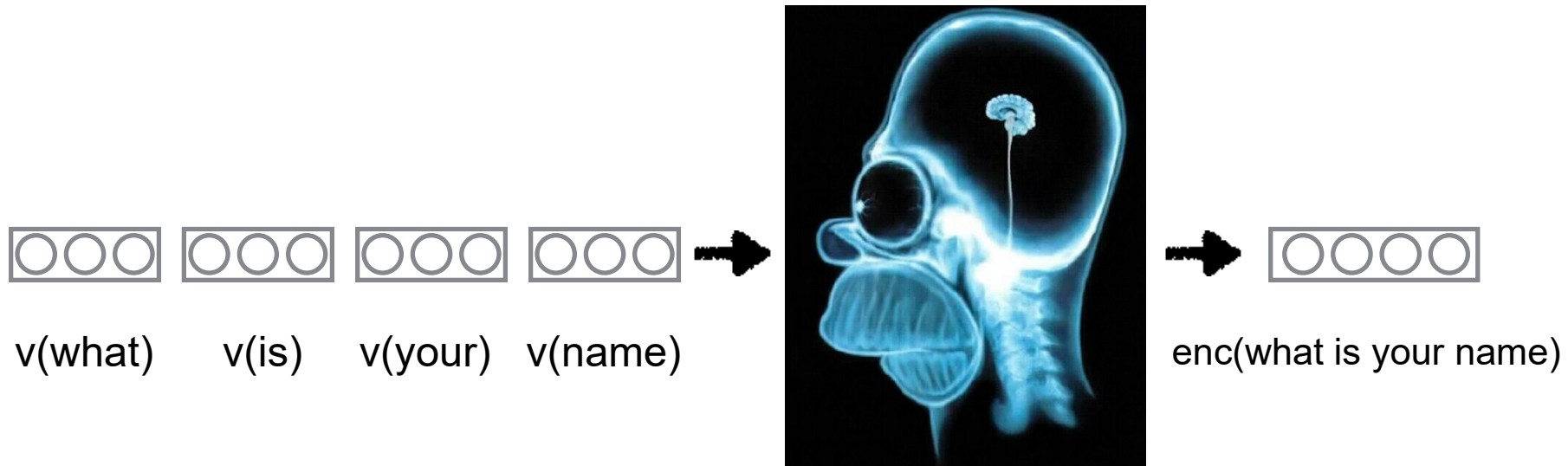
(max in each coordinate)

CNNs → RNNs

CNNs consider **local** word order

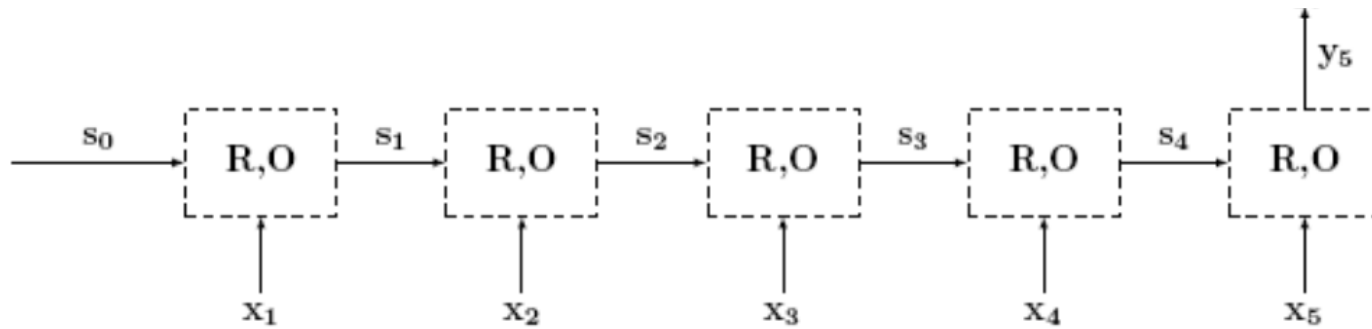
How can we consider **global** word order?

# Recurrent Neural Networks



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

# Recurrent Neural Networks



# Simple RNN (Elman RNN)

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

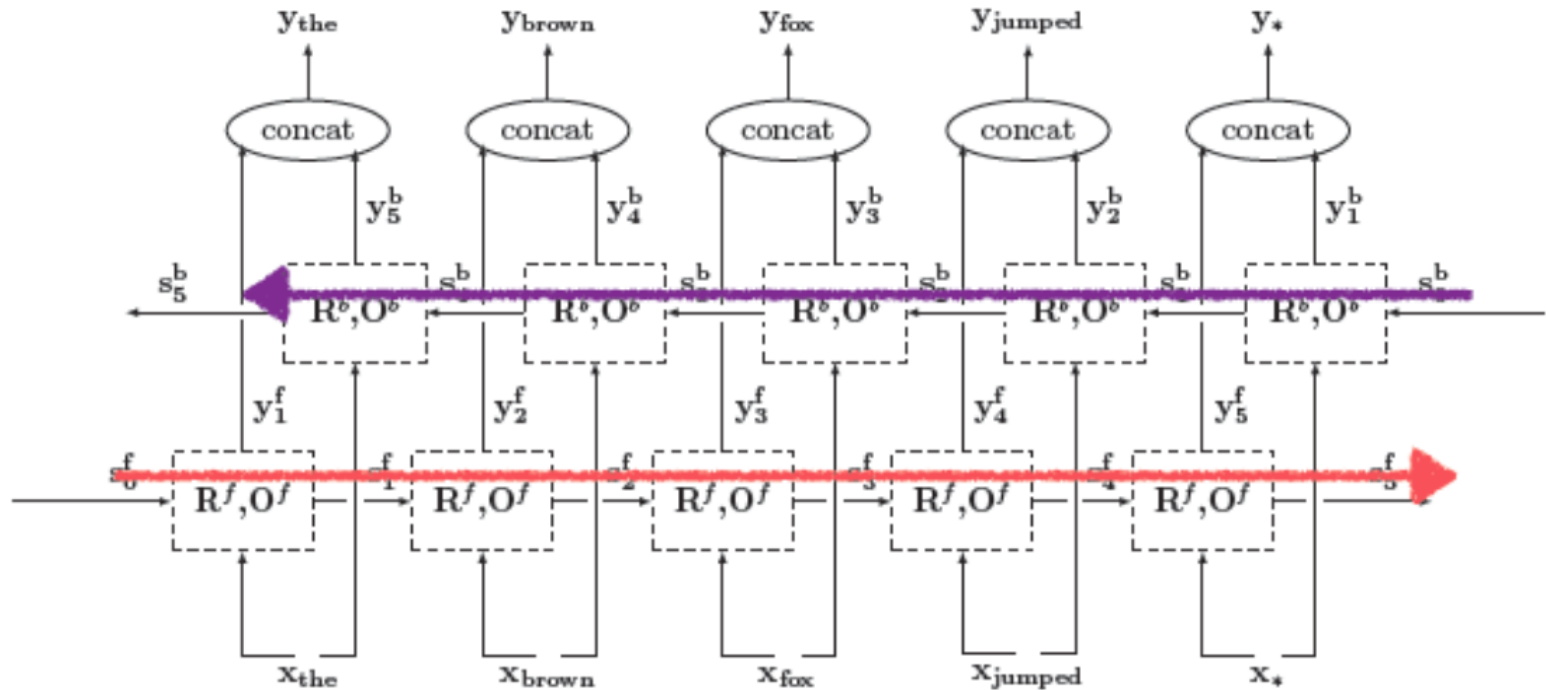
In principle: capture infinite history upto this point

In practice: have issues with long sequences

## RNN → LSTM

Good for backpropagating through long chain sequences

# Bi-directional RNN

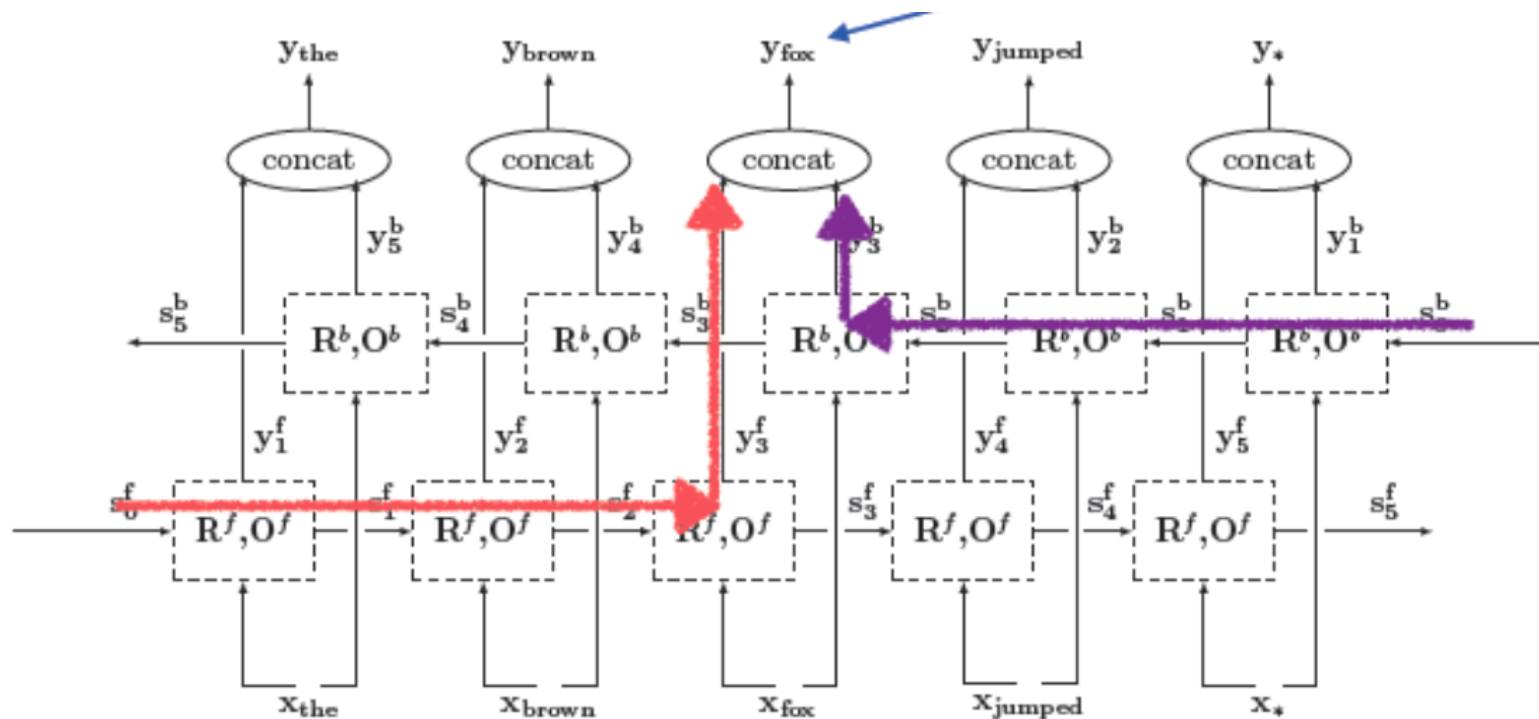


One RNN runs left to right.

Another runs right to left.

Encode **both future and history** of a word.

## Infinite window around the word



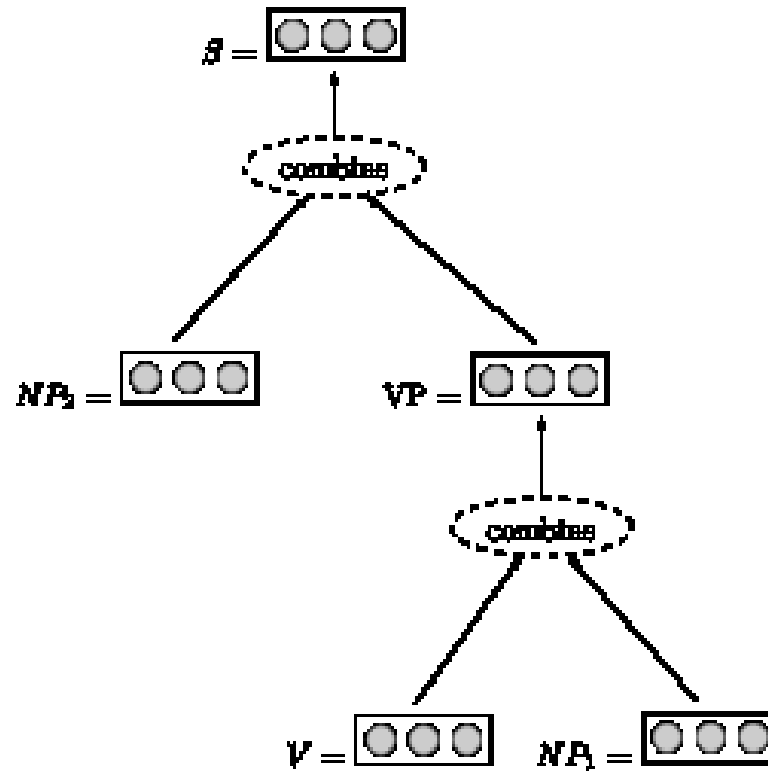
One RNN runs left to right.

Another runs right to left.

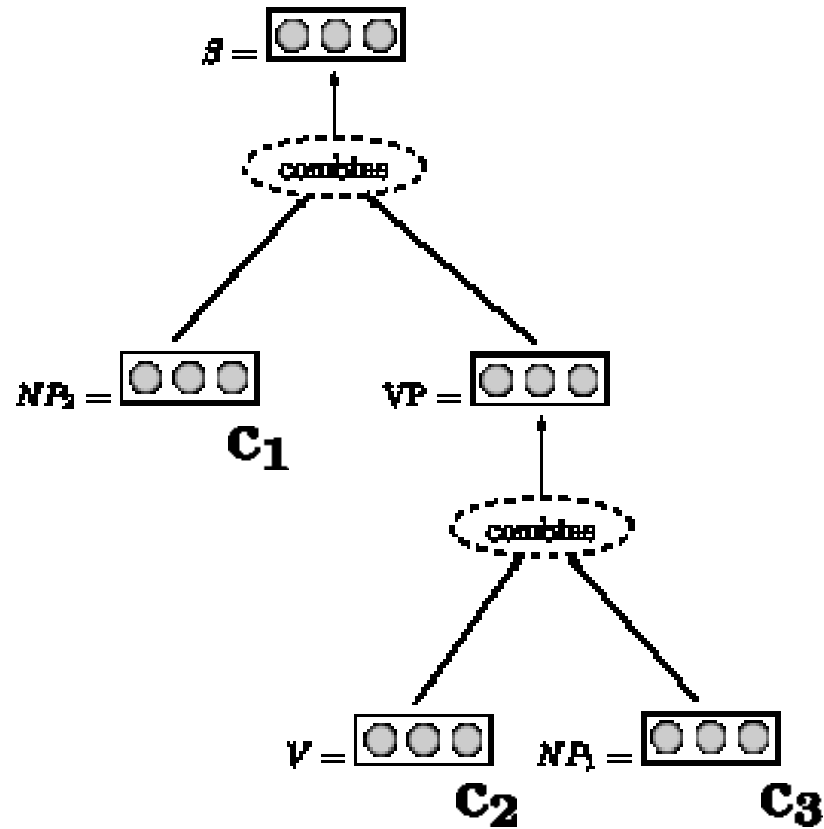
Encode **both future and history** of a word.



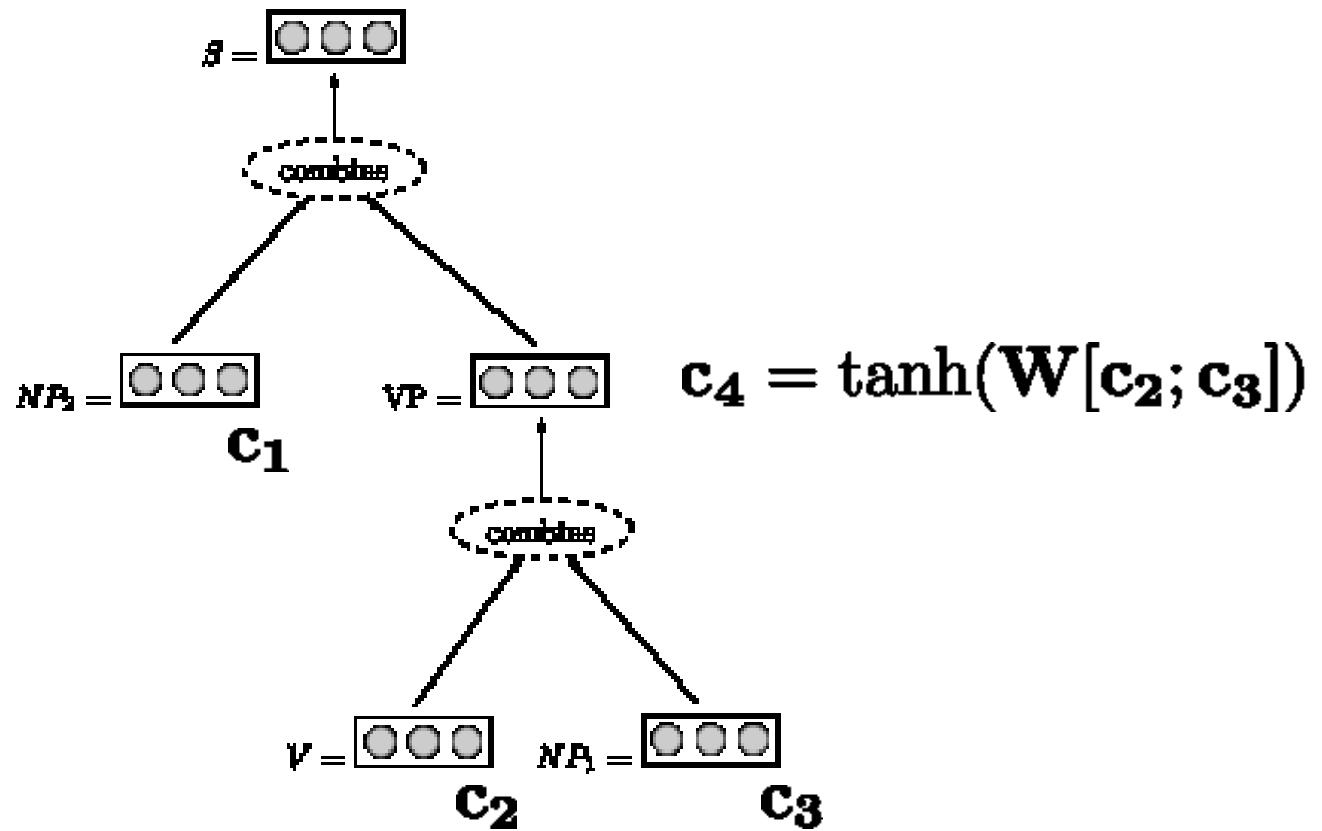
# Recursive Neural Nets



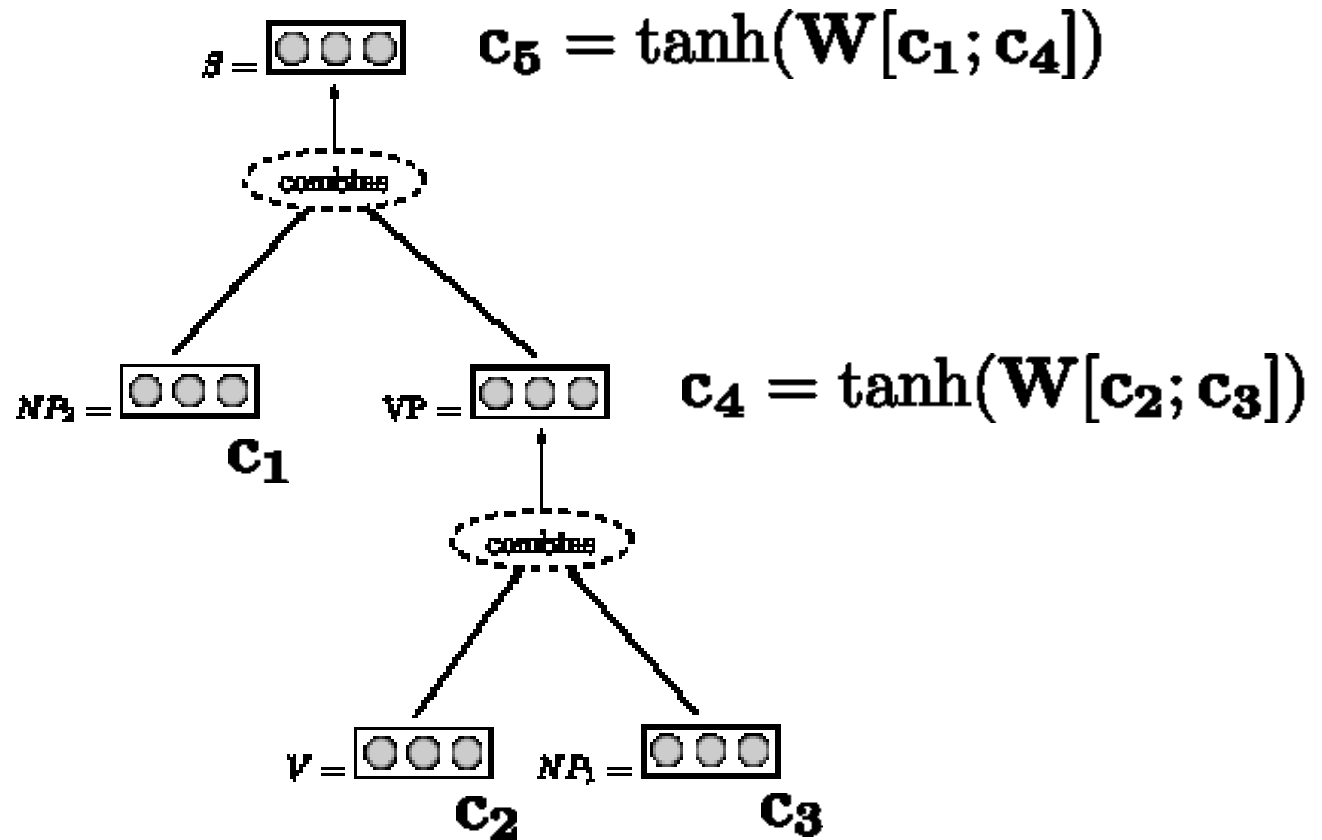
# Recursive Neural Nets



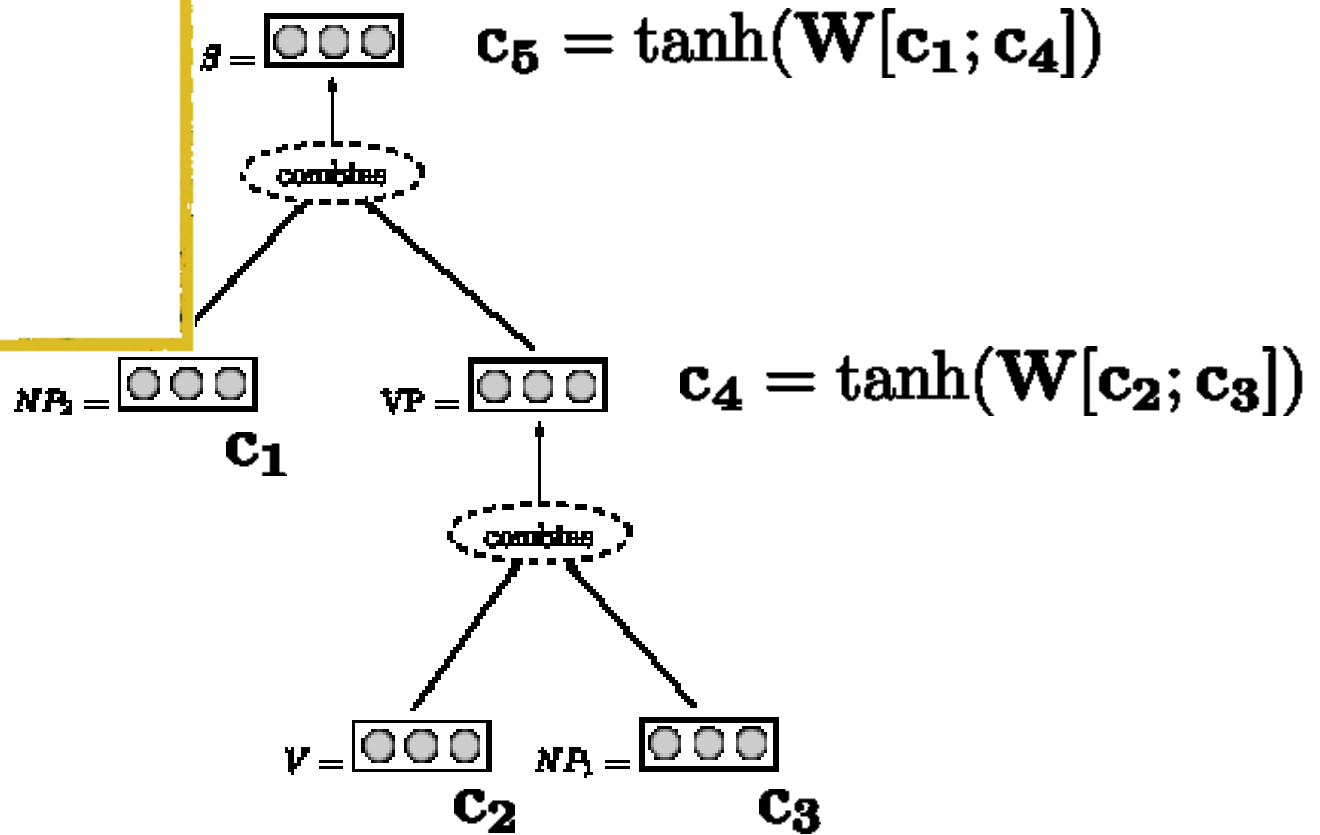
# Recursive Neural Nets



# Recursive Neural Nets



# Recursive Neural Nets



# Decoding Architectures

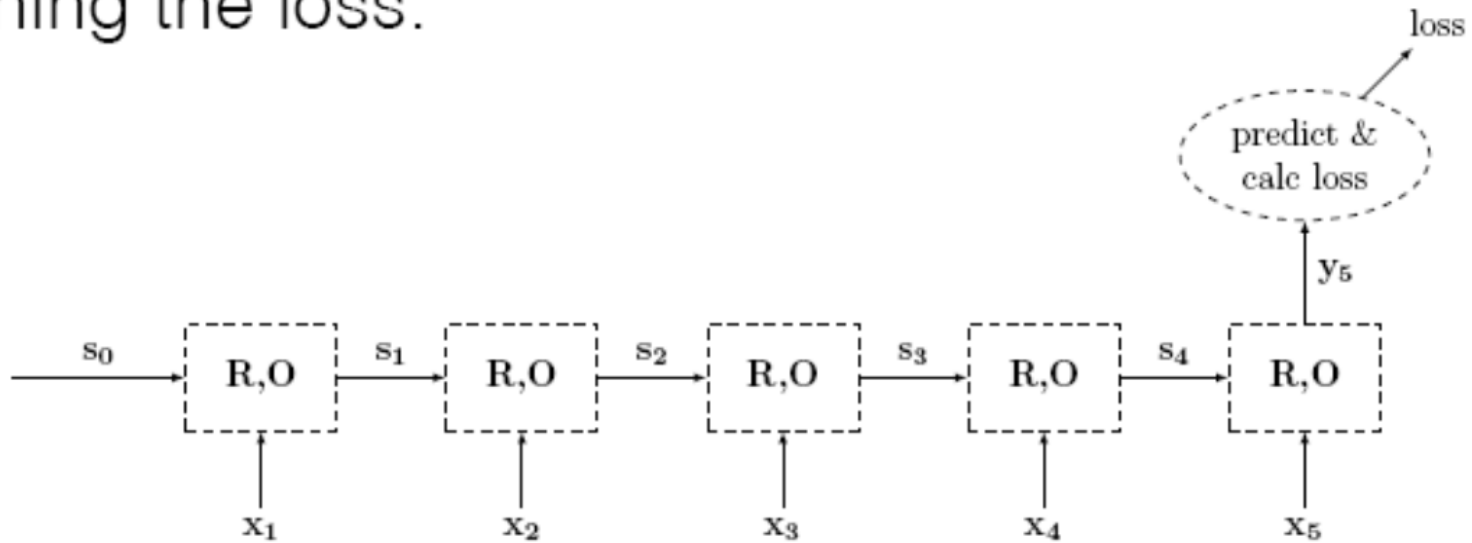
Acceptor

Transducer

Language Model

# RNN Acceptor

Defining the loss.

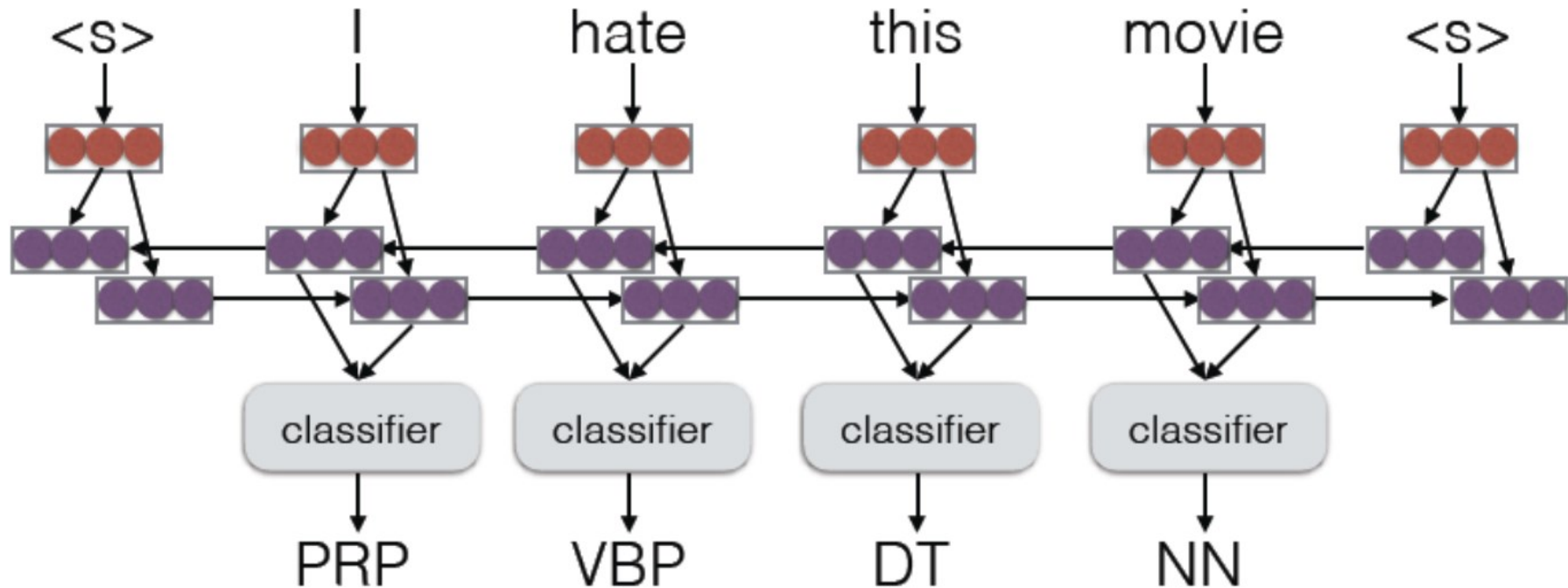


**Acceptor:** predict something from end state.

Backprop the error all the way back.

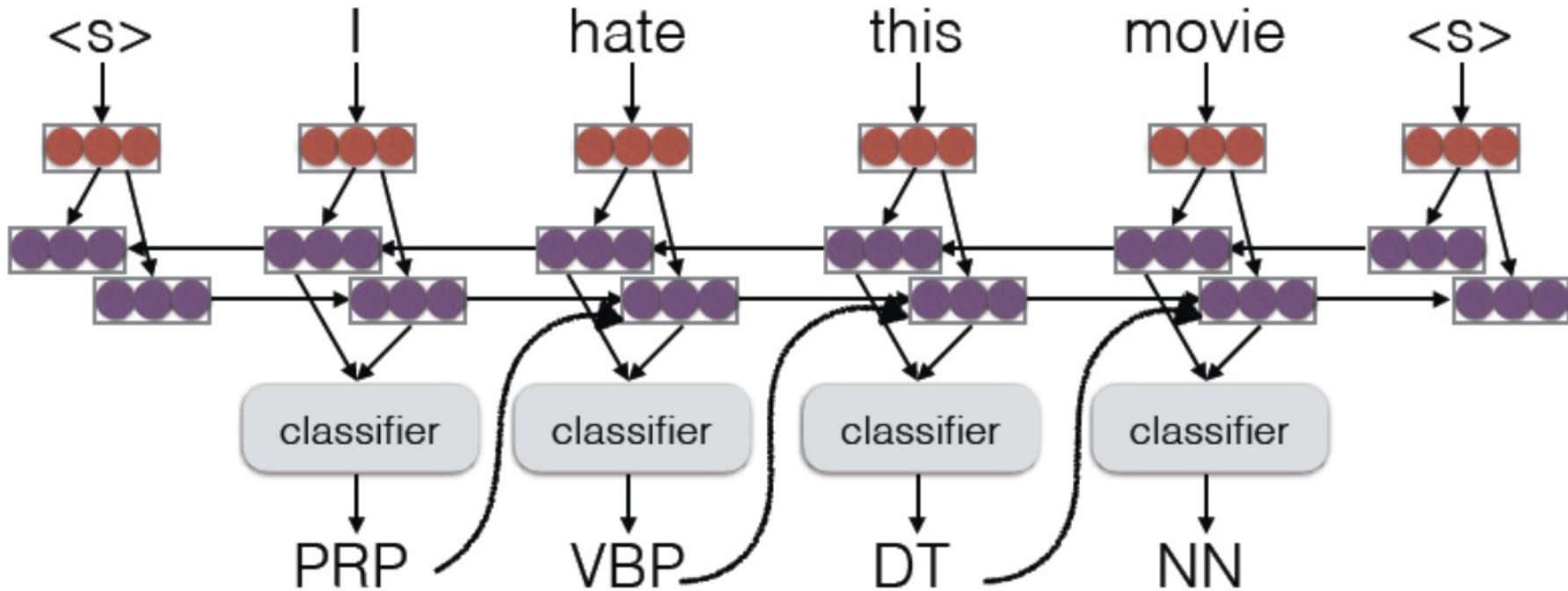
Train the network to capture meaningful information

# Sequence Labeling with Transducer BiLSTM





# A Tagger Considering Output Structure



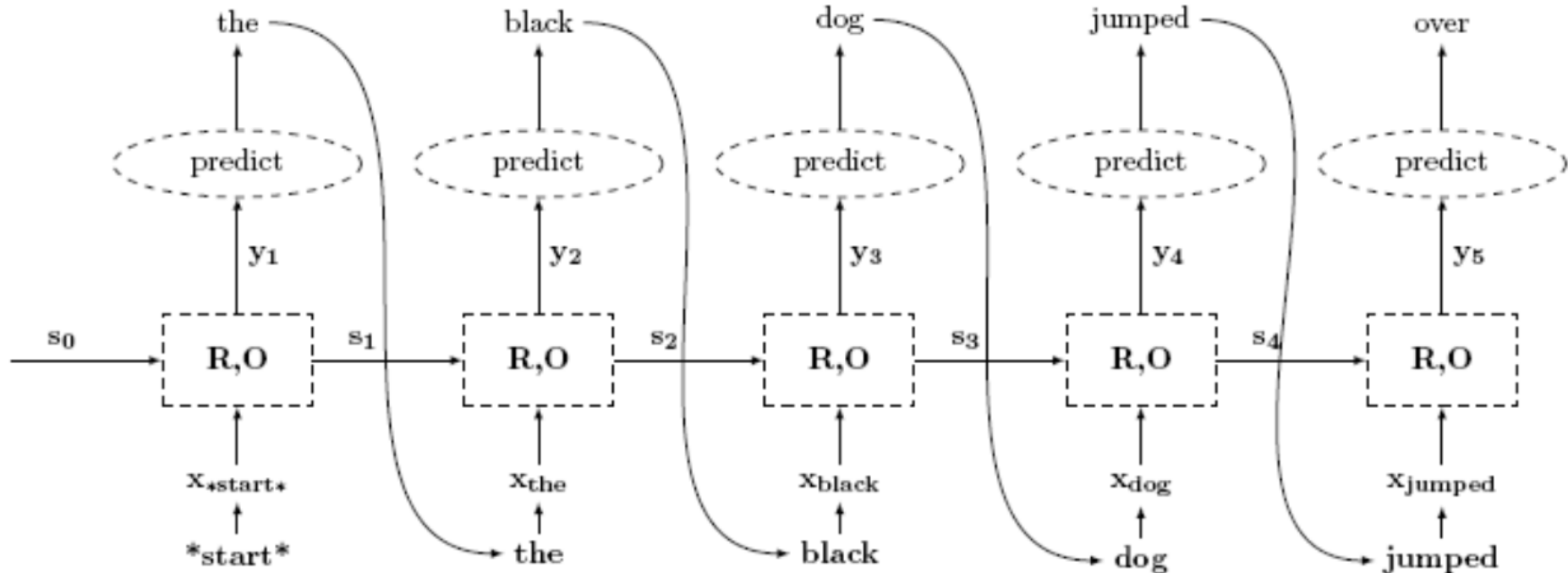
# Aside

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  $i+1$ .



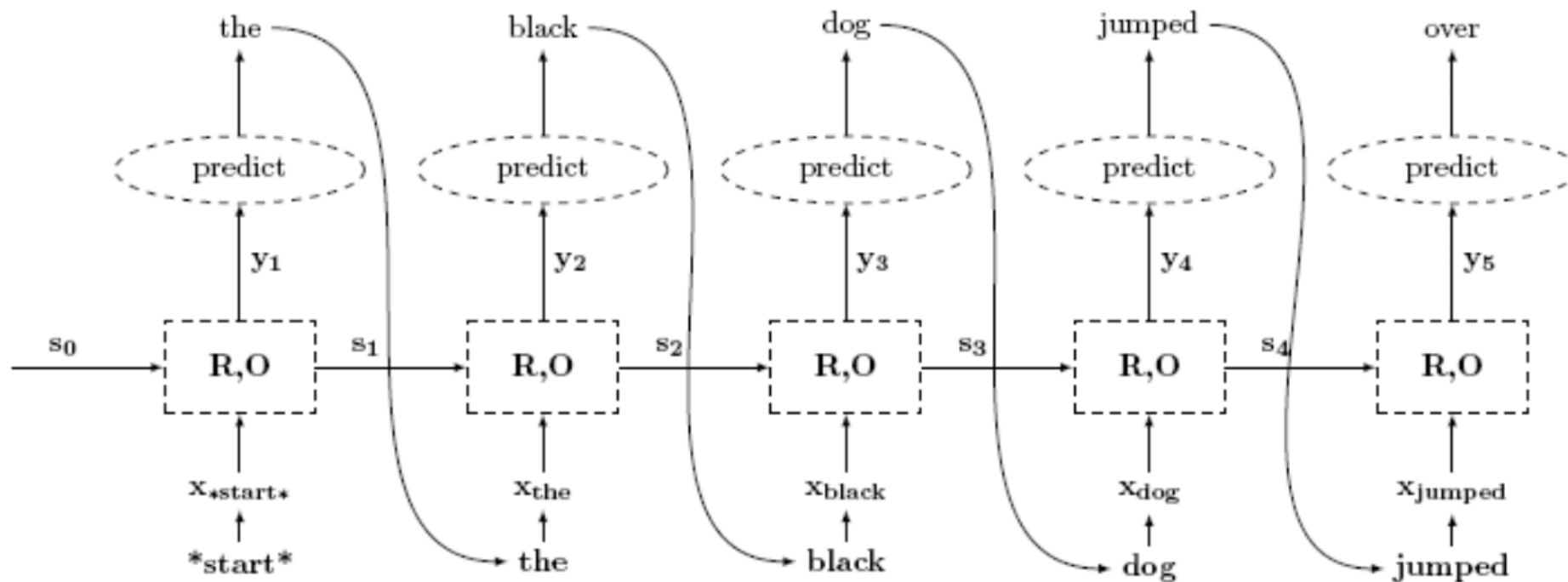
# RNN Language Model for generation

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

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

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

# RNN Language Models



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

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

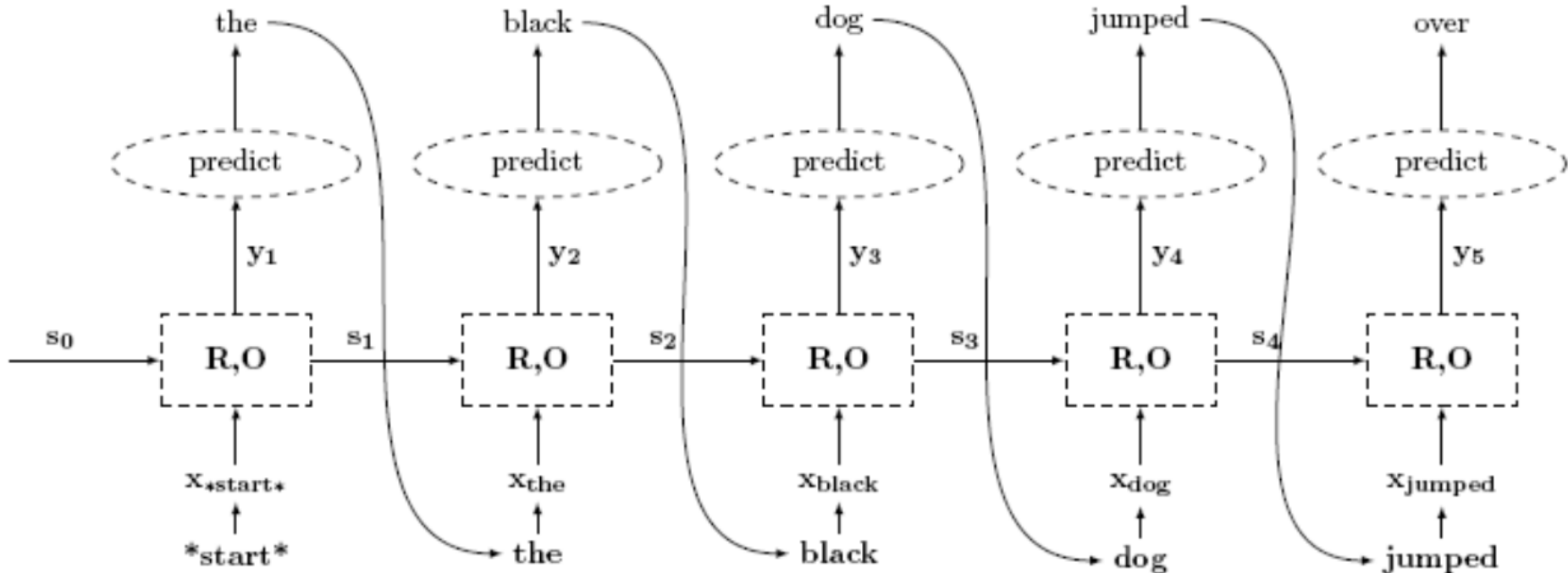
$$p(t_{j+1} = k \mid \hat{t}_{1:j}) = f(O(s_{j+1}))$$

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

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

# RNN Language Models

- *Generation*: the output of step  $i$  is input to step  $i+1$ .



# 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

<u>Input X</u>	<u>Output Y (<b>Text</b>)</u>	<u>Task</u>
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.

$$P(\tau) = \prod_{i=1}^I P(t_i \mid t_1, \dots, t_{i-1})$$

Next Word      Context

$$P(\tau \mid C) =$$

# How to Pass Context

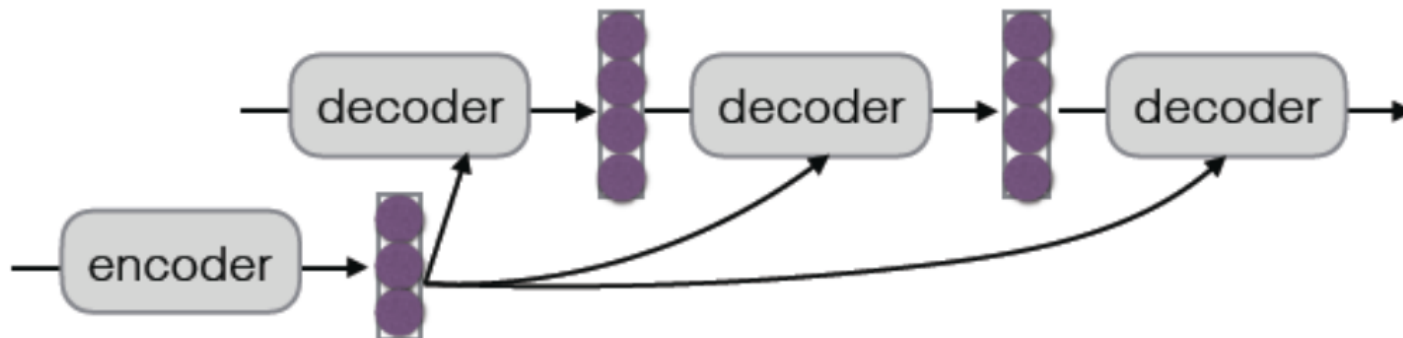
- Initialize decoder w/ encoder (Sutskever et al. 2014)



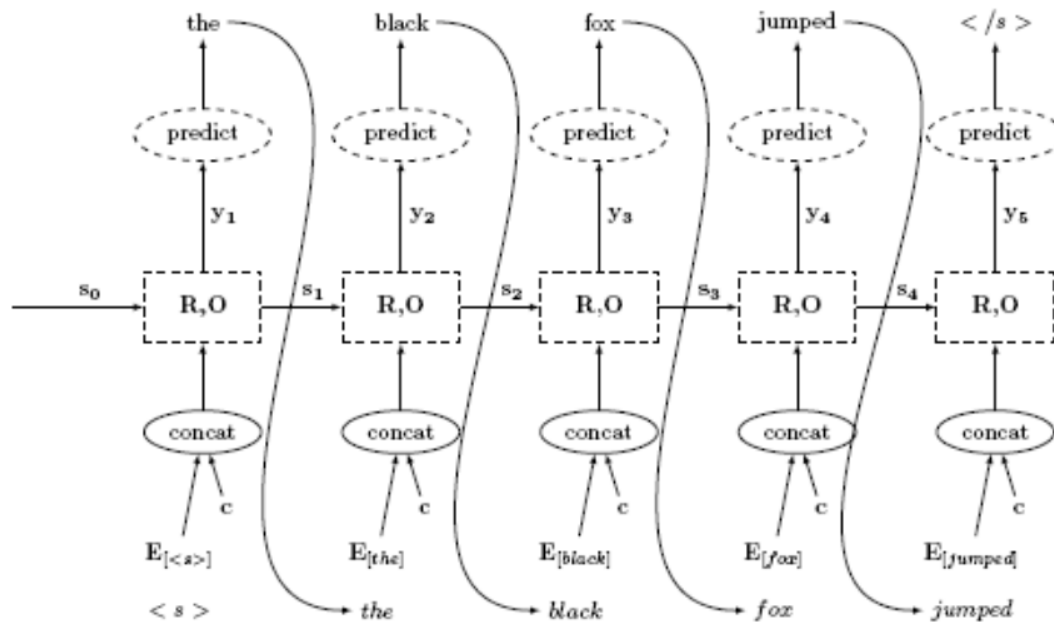
- Transform (can be different dimensions)



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



# RNN Language Model for Conditioned generation



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

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

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

# RNN Language Model for **Conditioned generation**

what if we want to condition on an entire sentence?

just encode it as a vector...

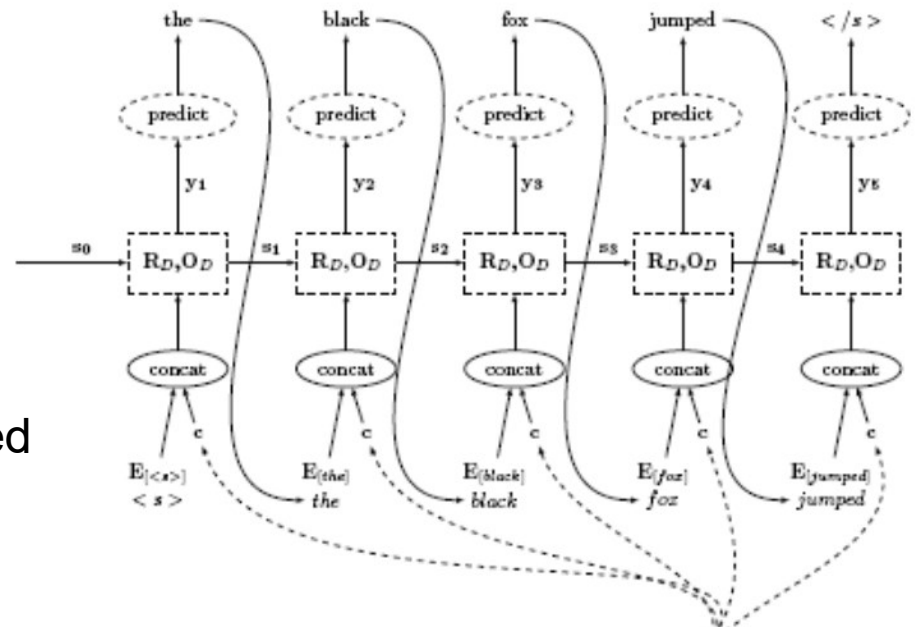
$$\mathbf{c} = \text{RNN}^{\text{enc}}(\mathbf{x}_{1:n})$$

# Sequence to Sequence conditioned generation

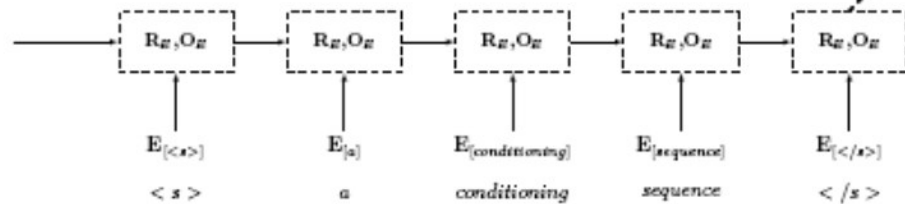
This is also called  
"Encoder Decoder"  
architecture.

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  $y_{j-1} \neq \text{"</s>"}$ :  
   $y_j \sim P(y_j \mid X, y_1, \dots, y_{j-1})$ 
```

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

# Greedy Search

One by one, pick the single highest-probability word

```
while  $y_{j-1} \neq \text{"</s>"}$ :  
   $y_j = \operatorname{argmax} P(y_j \mid X, y_1, \dots, y_{j-1})$ 
```

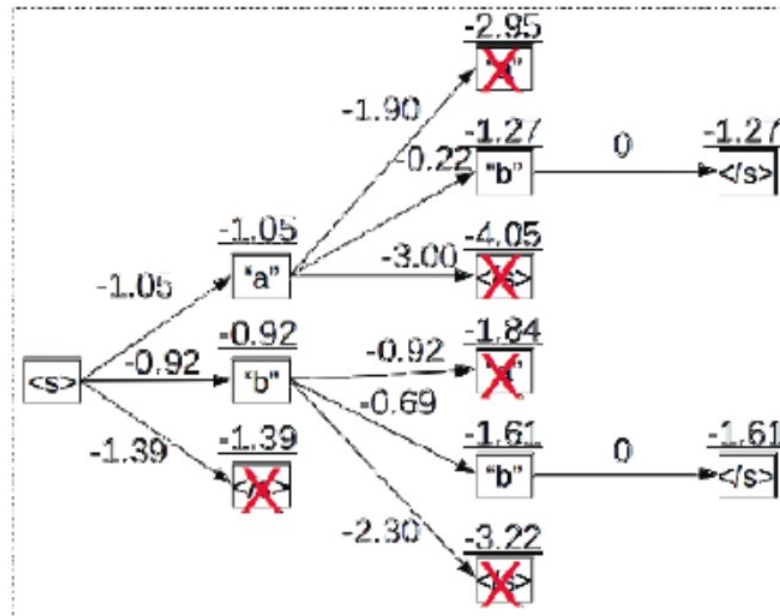
## **Not exact, real problems:**

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



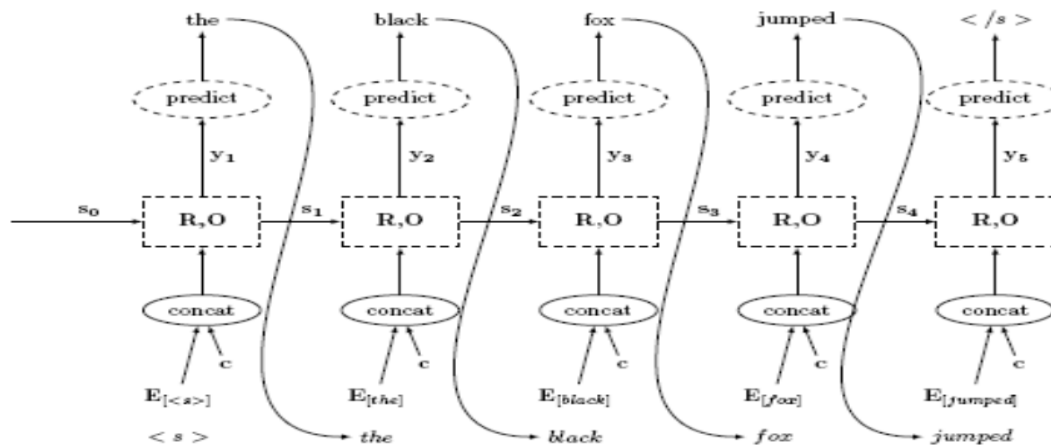
# Beam Search

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

Sequence 2 Sequence


Part II: with attention

# Sentence Representation

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



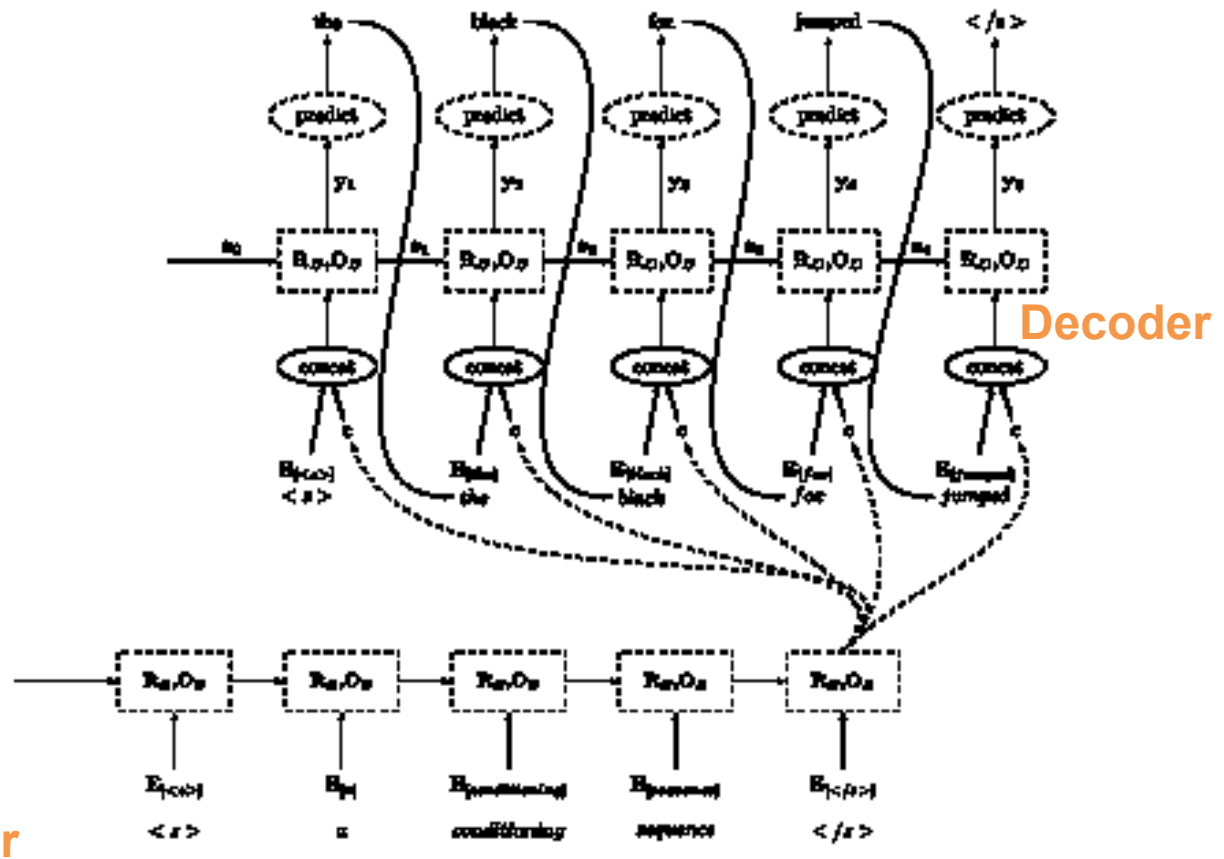
But what if we could use multiple vectors, based on the length of the sentence.

this is an example → 

this is an example → 

# Sequence to Sequence conditioned generation

main idea:  
encoding  
a single vector is  
too restrictive.



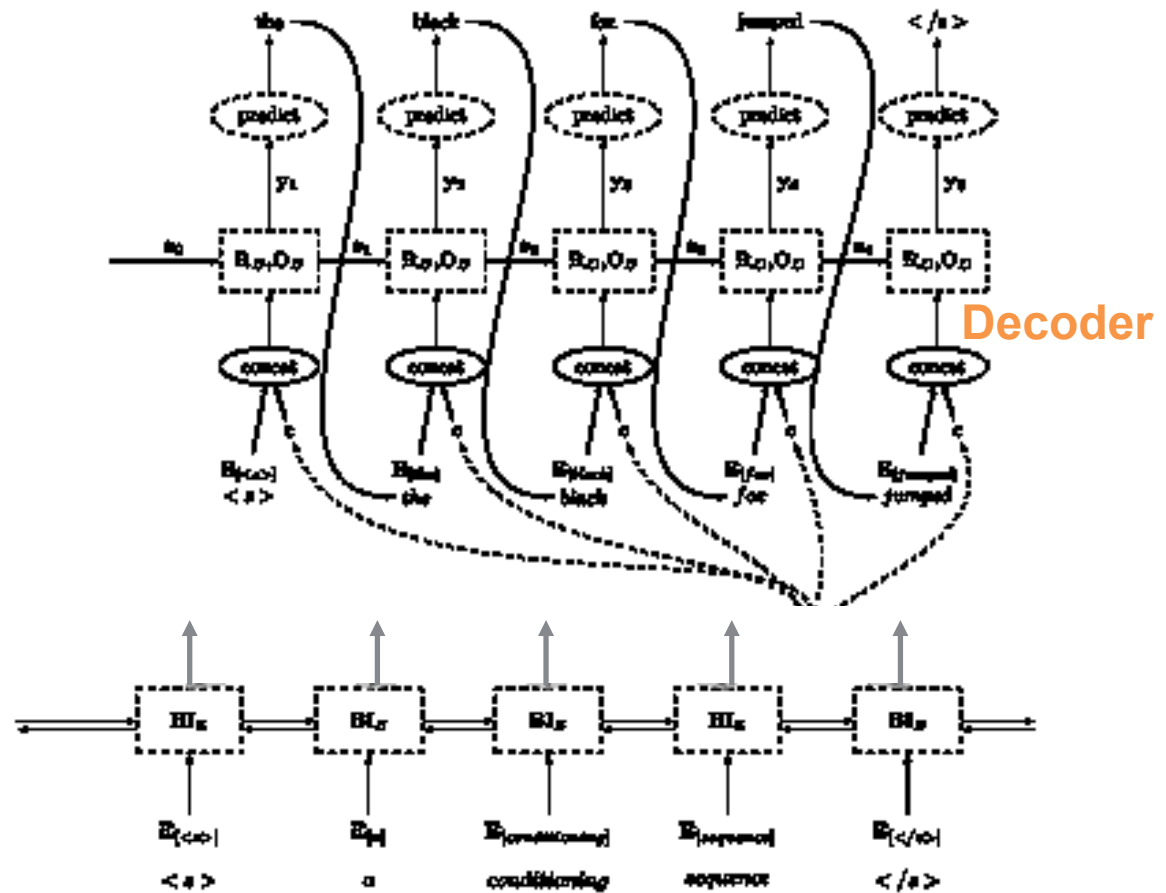
# Attention

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



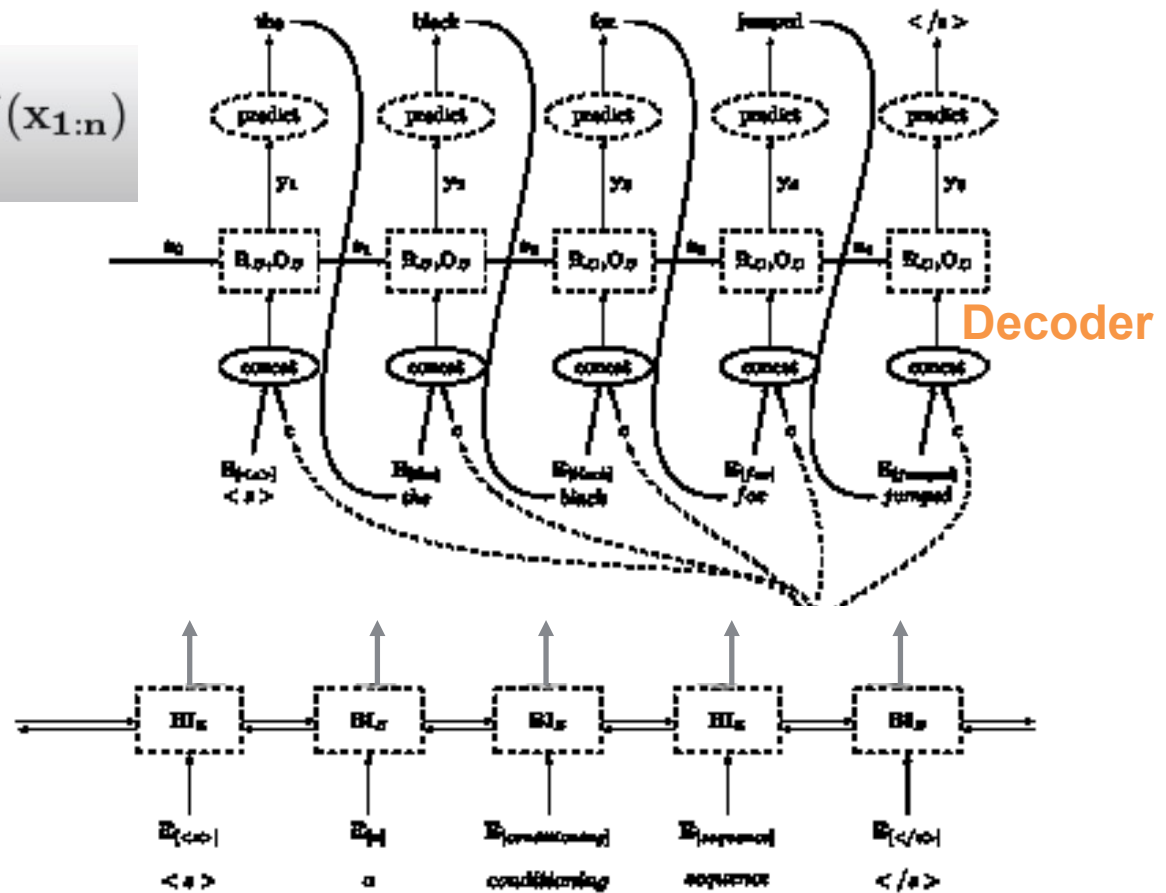


# Sequence to Sequence conditioned generation



# Sequence to Sequence conditioned generation

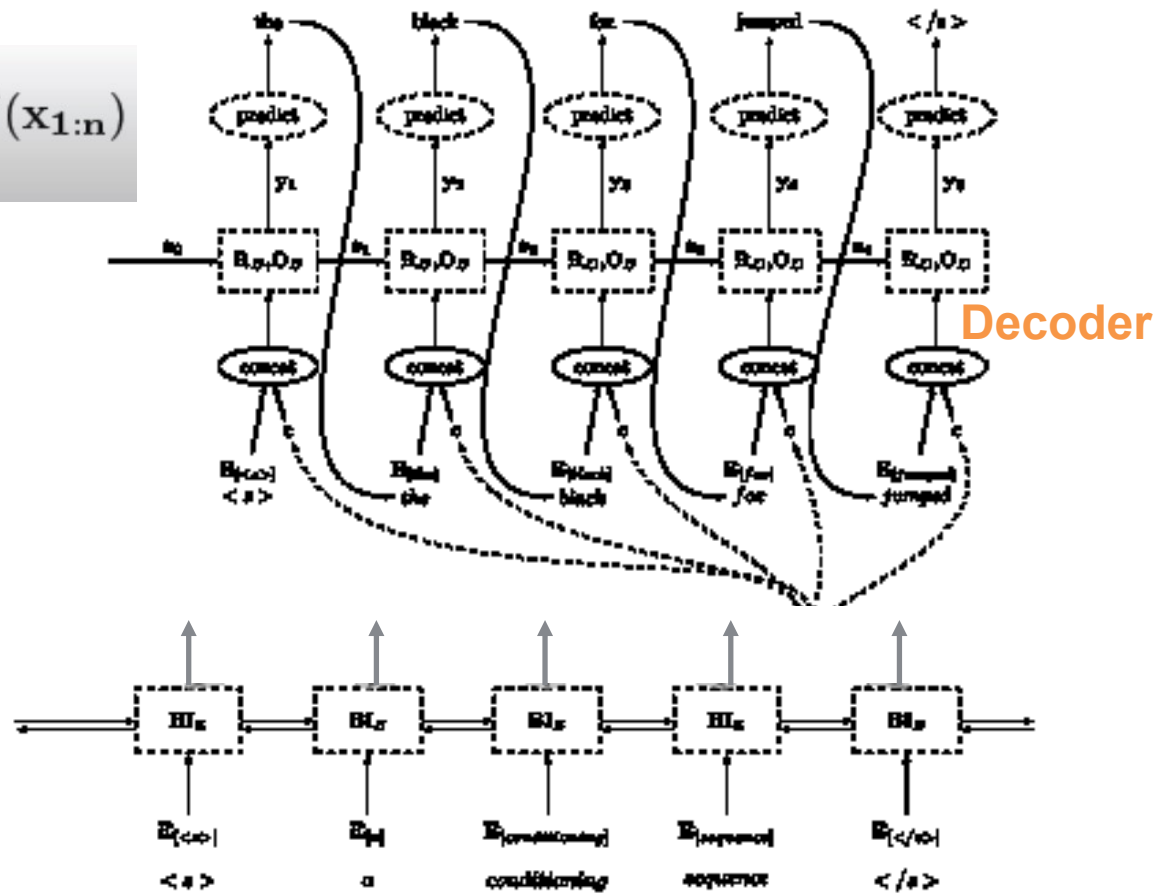
$$c_{1:n} = \text{ENC}(x_{1:n}) = \text{biRNN}^*(x_{1:n})$$



# Sequence to Sequence conditioned generation

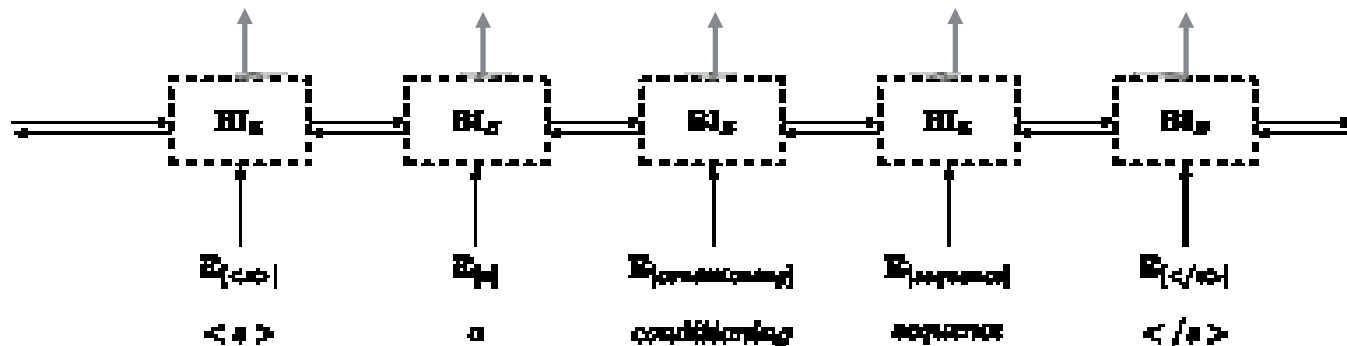
$$c_{1:n} = \text{ENC}(x_{1:n}) = \text{biRNN}^*(x_{1:n})$$

but how do we feed this sequence to the decoder?



# Sequence to Sequence conditioned generation

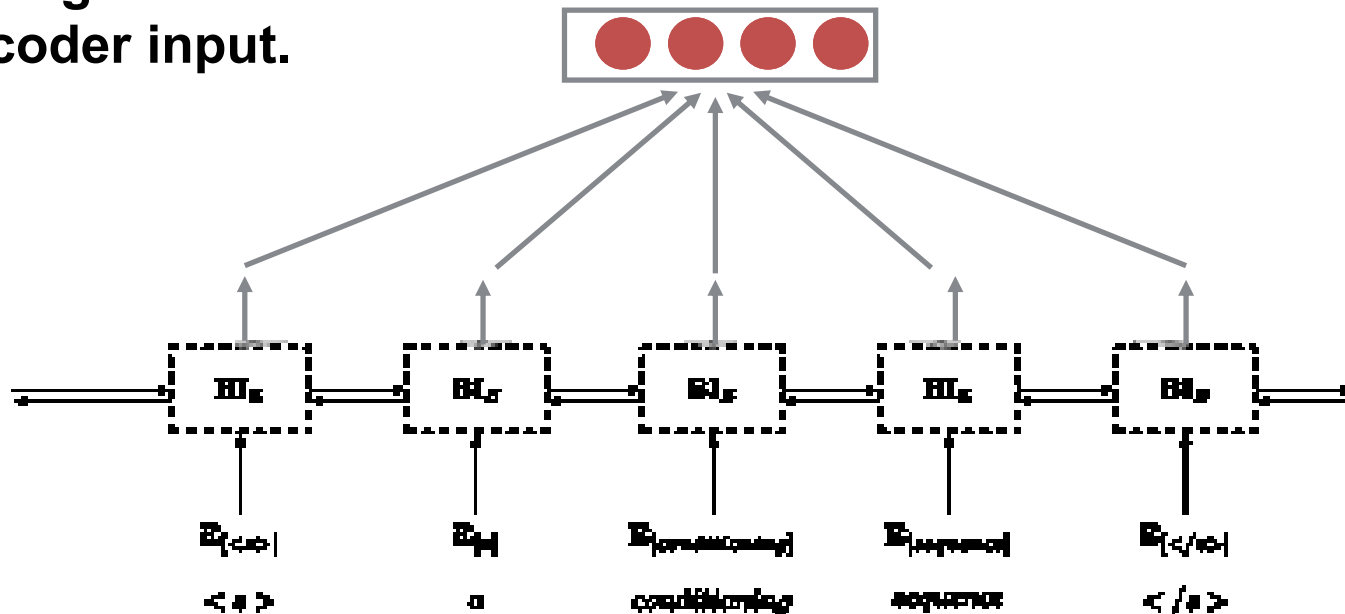
we can combine the different outputs  
into a single vector (attended summary)



# 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.



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

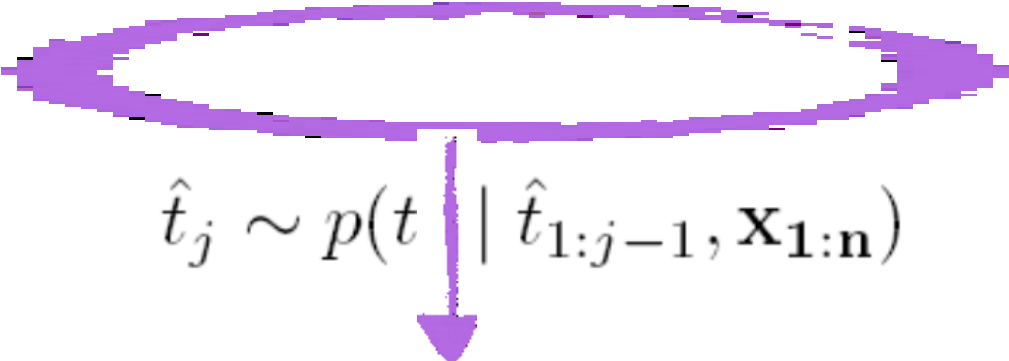
$$\mathbf{s}_{j+1} = R(\mathbf{s}_j, [\hat{\mathbf{t}}_j \circ])$$

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

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

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

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

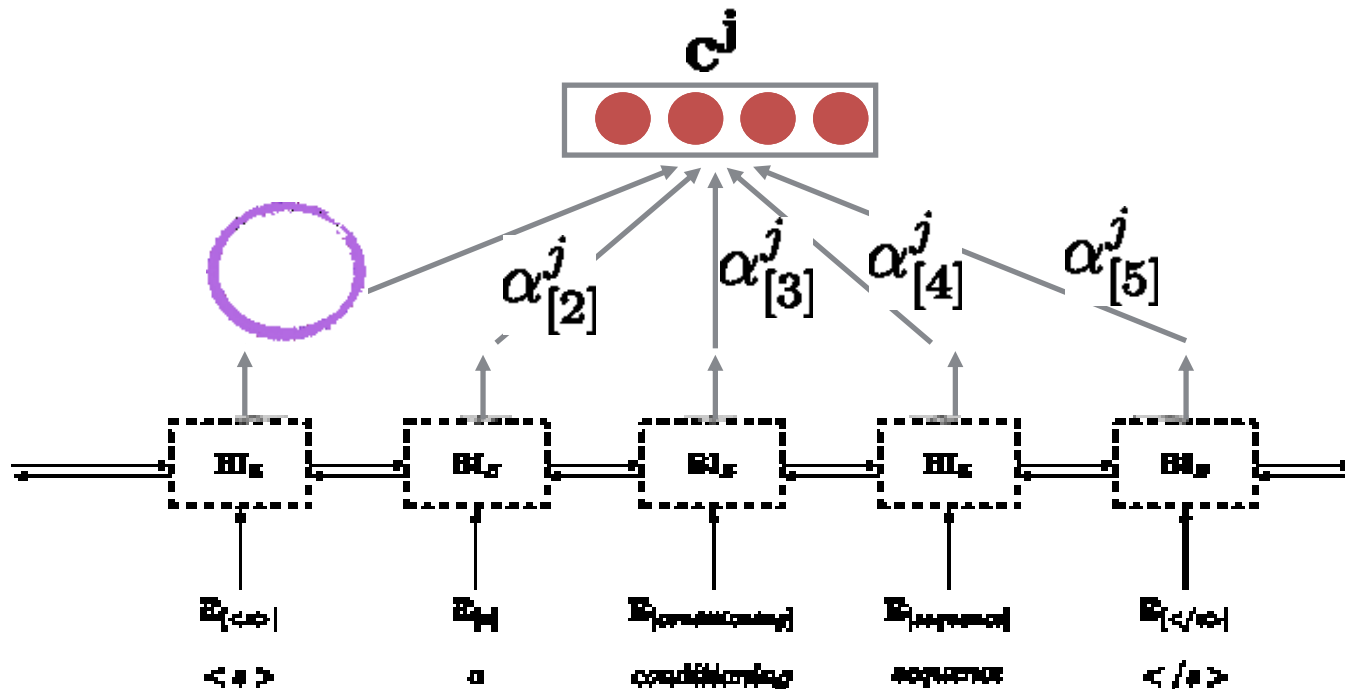

$$\hat{t}_j \sim p(t \mid \hat{t}_{1:j-1}, \mathbf{X}_{1:n})$$

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

# Sequence to Sequence conditioned generation

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

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



Encoder

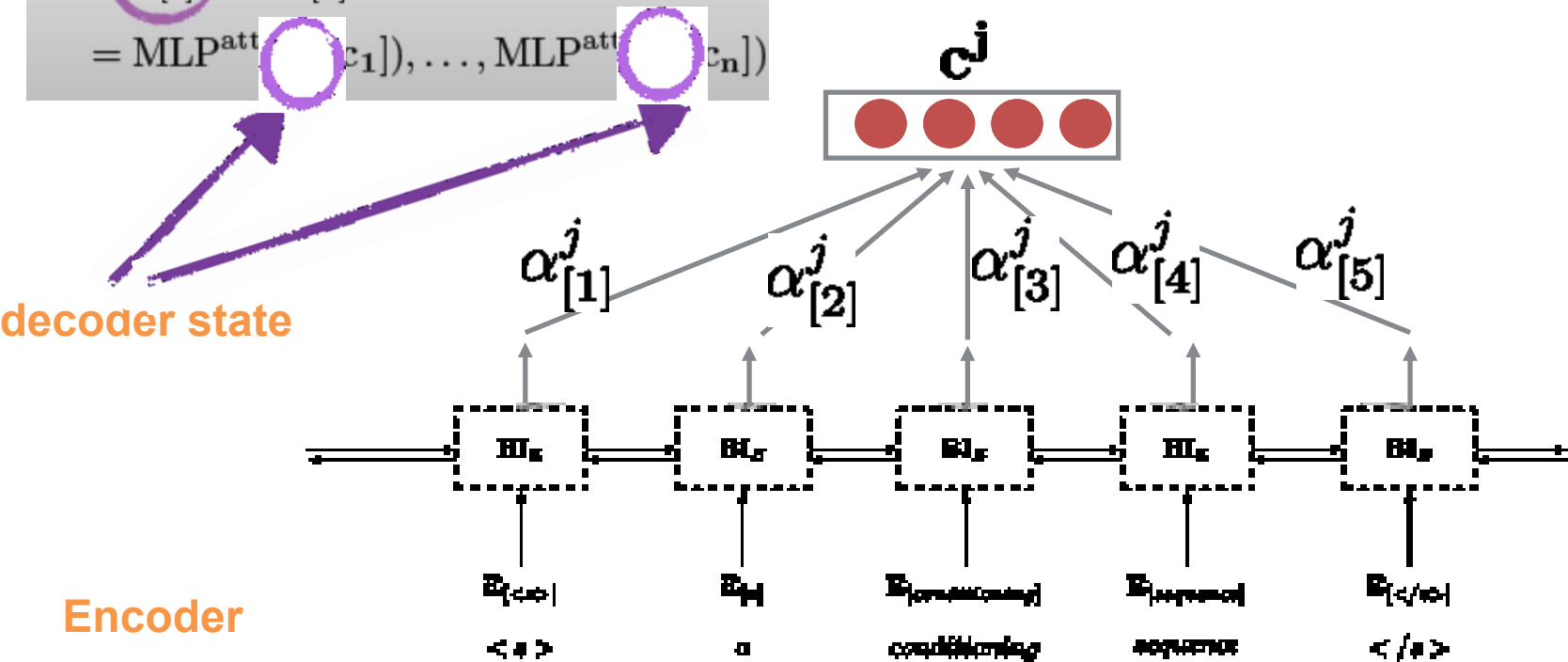


# Sequence to Sequence conditioned generation

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

$$\bar{\alpha}^j = (\bar{\alpha}_{[1]}^j, \dots, \bar{\alpha}_{[n]}^j) = \text{MLP}^{\text{att}}(c_1), \dots, \text{MLP}^{\text{att}}(c_n)$$

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



# encoder-decoder with attention

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

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

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

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

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

$$\bar{\alpha}_{[i]}^j = \text{MLP}^{\text{att}}([\mathbf{s}_j; \mathbf{c}_i])$$

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

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

$$\text{MLP}^{\text{att}}([\mathbf{s}_j; \mathbf{c}_i]) =$$

# encoder-decoder with attention

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

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

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

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

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

$$\bar{\alpha}_{[i]}^j = \text{MLP}^{\text{att}}([\mathbf{s}_j; \mathbf{c}_i])$$

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

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

$$\text{MLP}^{\text{att}}([\mathbf{s}_j; \mathbf{c}_i]) =$$

# encoder-decoder with attention

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

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

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

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

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

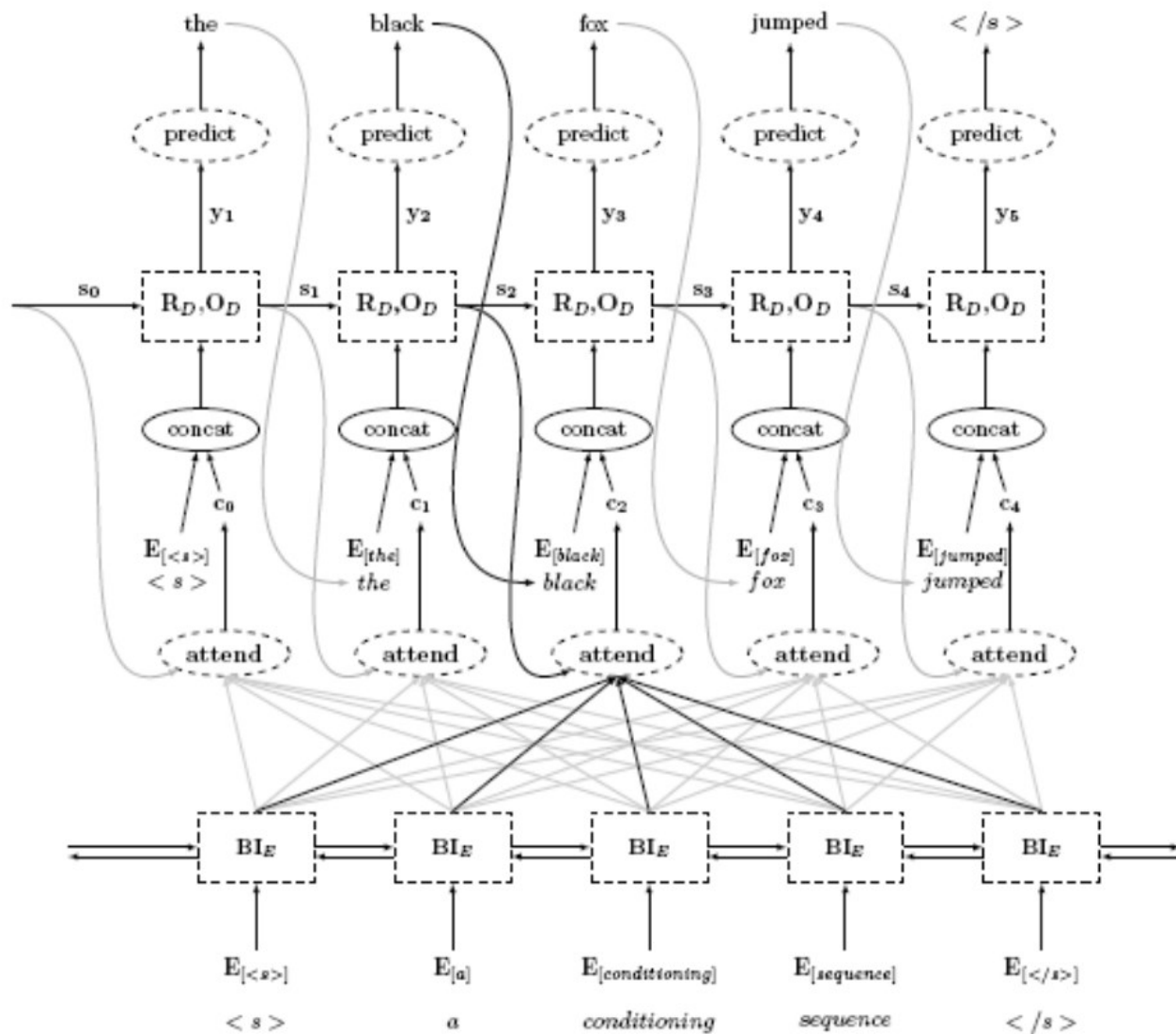
$$\bar{\alpha}_{[i]}^j = \text{MLP}^{\text{att}}([\mathbf{s}_j; \mathbf{c}_i])$$

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

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

$$\text{MLP}^{\text{att}}([\mathbf{s}_j; \mathbf{c}_i]) =$$

# encoder-decoder with attention



# encoder-decoder with attention

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

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

since then, the island authorities have tried to put an end to the illegal behaviour of non-alcoholic tourists in Magaluf by minimizing the number of participants in the notorious alcohol-free bar.  
die Inselbehörden haben seither versucht, das ordnungswidrige Verhalten alkoholisierter Urlauber in Magaluf zu stoppen, indem die Anzahl der Teilnehmer an den berüchtigten alkoholgetränkten Kneipen minimiert wurde.

## 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

$\text{MLP}^{\text{att}}(\mathbf{q};\mathbf{v})=$

- Additive Attention:  $\text{ug}(\mathbf{W}^1\mathbf{v} + \mathbf{W}^2\mathbf{q})$
- 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<sup>4</sup>. 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  $v$

## Paper's Justification:

To illustrate why the dot products get large, assume that the components of  $q$  and  $k$  are independent random variables with mean 0 and variance  $\rightarrow$  Then their dot product,  $q \cdot k$  has mean 0 and variance  $d_k$

# 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  
 $\text{MLP}^{\text{att}}(q;v)= \text{ug}(\mathbf{W}^1v_k + \mathbf{W}^2q)$  //additive
- Use value vector for computing attended summary

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

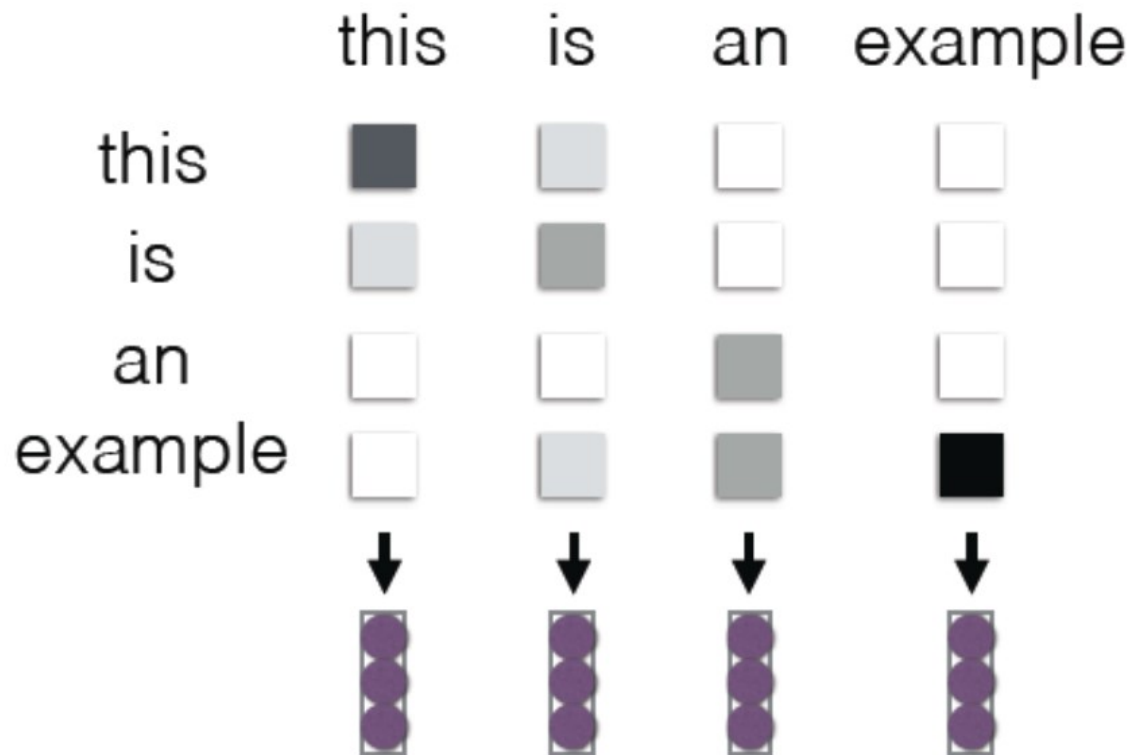


# Multi-head Key-Value Attention

- For each head
  - Learn different projection matrices  $\mathbf{W}_q$ ,  $\mathbf{W}_k$ ,  $\mathbf{W}_v$
- $\text{MLP}^{\text{att}}(q;v) = [(v_k \mathbf{W}_k) \cdot (q \mathbf{W}_q)] / \text{sqrt}(d_k)$
- For summary use  $v_v \mathbf{W}_v$  (instead of  $v_v$ )
- Train many such heads and
  - use  $\text{aggr}(\text{all such attended summaries})$

# 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

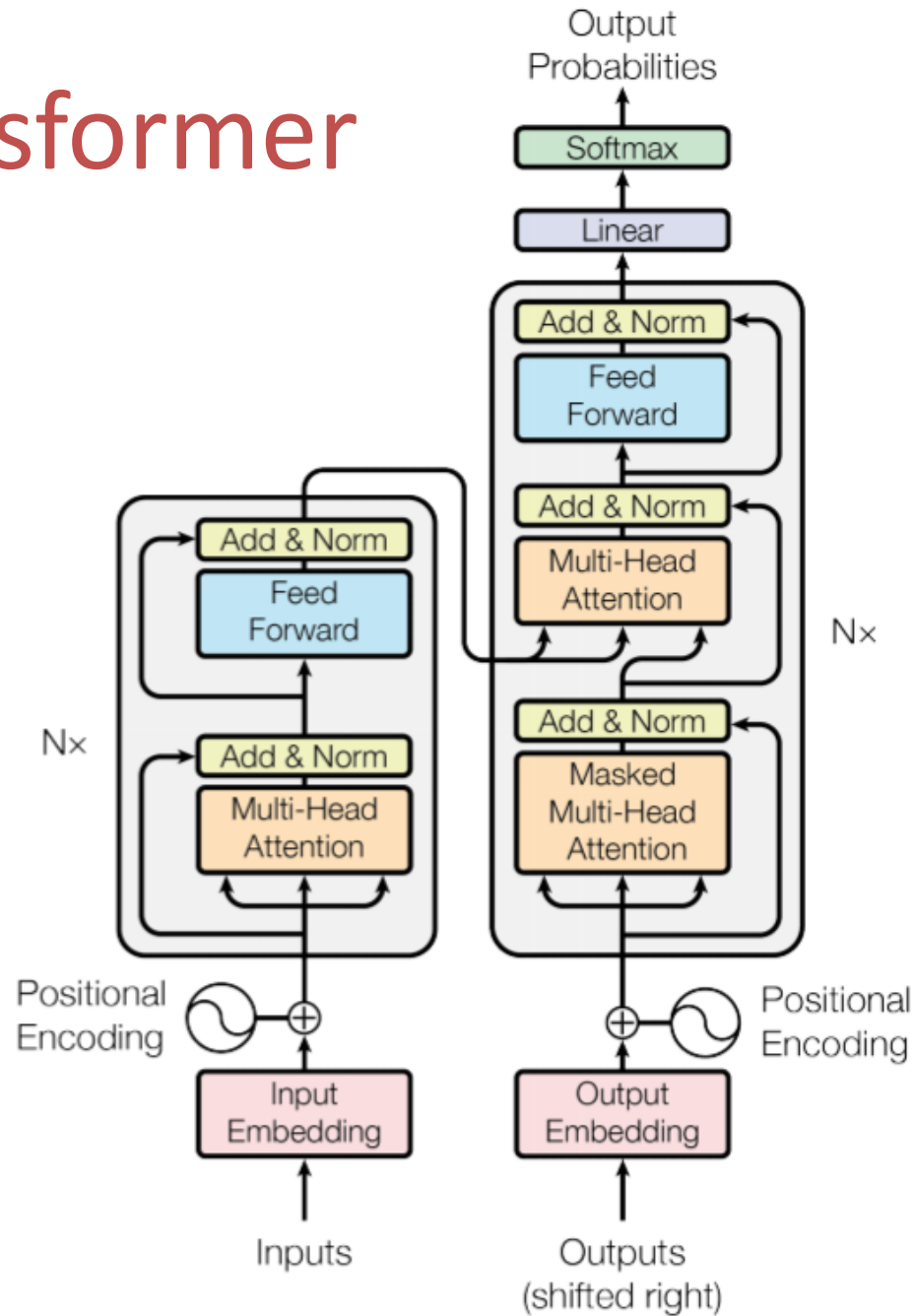
- **They are not deep**

- 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 other datasets in the same task, not to speak about other tasks
- ImageNet-trained ResNet fine-tuned on many other datasets

# Transformer



# 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

# Encoder Summary

- Shallow NNs
  - Bag(words)
- Convolutional NNs
  - Handle bag (fixed length n-grams)
- Recurrent NNs
  - Handle small variable length histories
- LSTMs/GRUs
  - Handle larger variable length histories
- Bi-LSTMs
  - Handle larger variable length histories and futures
- Recursive NNs
  - Handle variable length partially ordered histories

# Summary (contd)

- Hierarchical Recurrent NNs
  - RNN over RNNs (e.g., HRED)
- Neural language models
- Conditioned language models
  - Encoder-Decoder Models
- Attention models
  - attach non-uniform importance to histories based on evidence (question)



# Prior Knowledge about Task

- Add as penalty in loss term
- Add as a hard constraint
- Add via architectural choice
- Add as a symbolic feature
- Supply through data augmentation

# Deep Learning Strengths

- universal representation
- compositional representation
- feature discovery in task-driven way
- weight sharing in features
- seamlessly combine unsupervised w supervised
- natural fit for multi-modal and multi-task settings

# Deep Learning Weaknesses

- high variance
- difficult to debug
- uninterpretable
- data hungry