Deep Natural Language Processing

Reyyan Yeniterzi
Sabancı University
reyyan@sabanciuniv.edu

3. Veri Bilimi Yaz Okulu (VBYO 2019)

Outline

- Language Modeling
  - Problems with ngrams
  - Neural Network Language Models (NNLM)
  - Word Embeddings
  - CNN Applied to Text
  - Recurrent Neural Networks (RNNs)
  - Recurrent Neural Network Language Models (RNNLM)
- Sequence-to-Sequence Models
- Attention Models

- If not stated otherwise, images are from “Neural Network Methods for Natural Language Processing” book by Yoav Goldberg

Language Modeling

- Probability of a sequence of words
  - *P(the, quick, brown, fox, jumps, over)*
- Probability of seeing a particular word after a sequence of words
  - *P(the | the, quick, brown, fox, jumps, over)*

$$p(w_i | w_1, w_2, \ldots, w_{i-1}) = \frac{p(w_1, w_2, \ldots, w_{i-1}, w_i)}{p(w_1, w_2, \ldots, w_{i-1})}$$

- The Chain Rule

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

- The Markov Assumption

$$P(w_{i+1} \mid w_{1:i}) \approx P(w_{i+1} \mid w_{i-k:i})$$

$$\Downarrow$$

$$P(w_{1:n}) \approx \prod_{i=1}^{n} P(w_i \mid w_{i-k:i-1})$$

- The Chain Rule

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

- The Markov Assumption

$$P(w_{i+1} \mid w_{1:i}) \approx P(w_{i+1} \mid w_{i-k:i})$$

$$\Downarrow$$

$$P(w_{1:n}) \approx \prod_{i=1}^{n} \boxed{P(w_i \mid w_{i-k:i-1})}$$

Approaches to Language Modeling

- Count-based (MLE-based) Language Modeling (ngrams)

Count-based LM

- Apply Maximum Likelihood Estimate over a large corpus

$$\hat{p}_{\mathrm{MLE}}(w_{i+1} = m | w_{i-k:i}) = \frac{\#(w_{i-k:i+1})}{\#(w_{i-k:i})}$$

Count-based LM

- Apply Maximum Likelihood Estimate over a large corpus

$$\hat{p}_{\text{MLE}}(w_{i+1} = m | w_{i-k:i}) = \frac{\#(w_{i-k:i+1})}{\#(w_{i-k:i})}$$

- Effective but ...

$$\#(w_{i-k:i+1}) = 0$$

Count-based LM

- **Smoothing:** Steal a bit of probability mass from more frequent events and give it to the unseen events

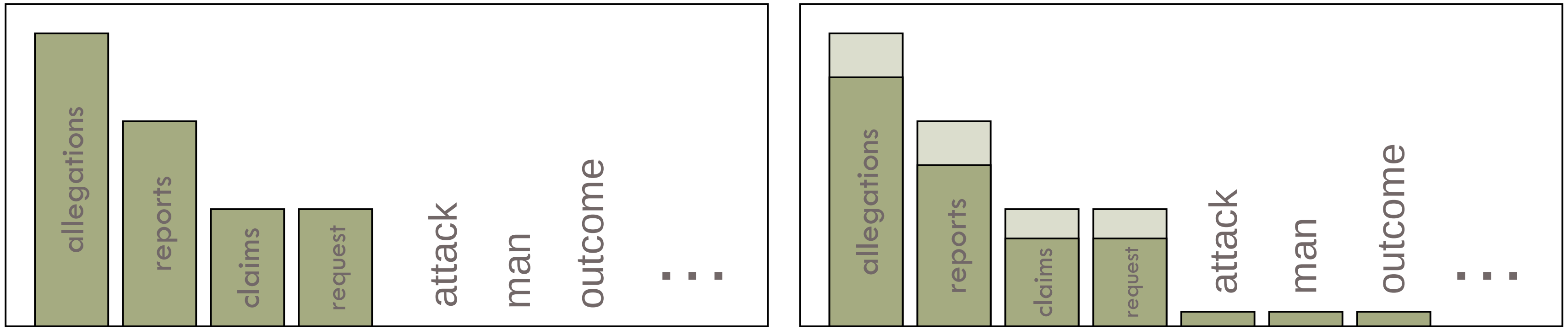


Image Credit: Dan Klein

Count-based LM

- Pros
  - Very scalable, can be trained on very large data easily
  - Constant time evaluation during testing
  - Sophisticated smoothing techniques help
- Cons
  - Scaling to larger ngrams is hard
    - Larger ngrams are sparse
    - Very expensive in terms of memory

Problems with Count-based LM

- Train: “Ms. Jane Brown”
- Test: “Ms. Açılay Brown”

Problems with Count-based LM

- Problem 1: Intervening words cause limited context modeling
  - Train: "Ms. Jane Brown"
  - Test: "Ms. Açılay Brown"
  - Seeing 'Ms.' should be useful to guess 'Brown'
  - Count-based LMs do backoff and lose context
- Proposed Solution: Skip-gram LMs
  - Instead of conditioning on previous word, condition on two words ago

Problems with Count-based LM

- Train: “The cat is walking in the bedroom”
- Test: “A dog was running in a room”

Example Credit: A Neural Probabilistic Language Model by Bengio et al.

Problems with Count-based LM

- Problem 2: Ignoring the "similarity" between words
  - Train: "The cat is walking in the bedroom"
  - Test: "A dog was running in a room"
  - Seeing the first sentence in the training corpus should be useful for the second sentence during testing
- Proposed Solution: Class-based LMs

Example Credit: A Neural Probabilistic Language Model by Bengio et al.

Problems with Count-based LM

- Train: “For tennis class, he wanted to buy his own …”
- Test: “For programming class, he wanted to buy his own …”

Example Credit: Graham Neubig

Problems with Count-based LM

- Problem 3: Cannot handle long-distance dependencies
  - Train: “For tennis class, he wanted to buy his own racquet”
  - Test: “For programming class, he wanted to buy his own computer”

Example Credit: Graham Neubig

Approaches to Language Modeling

- Count-based (MLE-based) Language Modeling (ngrams)
- Neural Network Language Models (NNLM)
  - Bengio, Yoshua, et al. "A Neural Probabilistic Language Model." *Journal of Machine Learning Research* 3, 2003.

Approaches to Language Modeling

- Count-based (MLE-based) Language Modeling (ngrams)
  - # possible ngrams over vocabulary V is $|V|^n$
    - Increase n by 1 will result in a $|V|$ fold increase
- Neural Network Language Models (NNLM)
  - Bengio, Yoshua, et al. "A Neural Probabilistic Language Model." *Journal of Machine Learning Research* 3, 2003.
  - The fight with the curse of dimensionality!

Curse of Dimensionality

- As dimensions grow, dimensions space increases exponentially

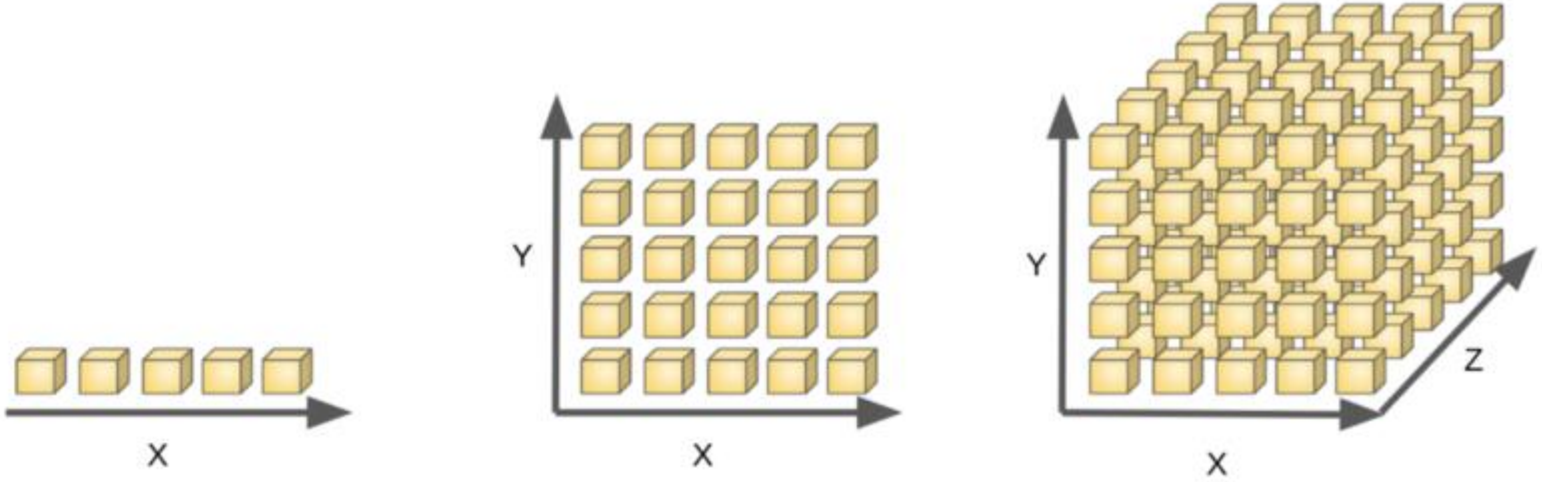


Image Source

Curse of Dimensionality

- As dimensions grow, dimensions space increases exponentially
  - High data sparsity
  - More storage space
  - More processing time

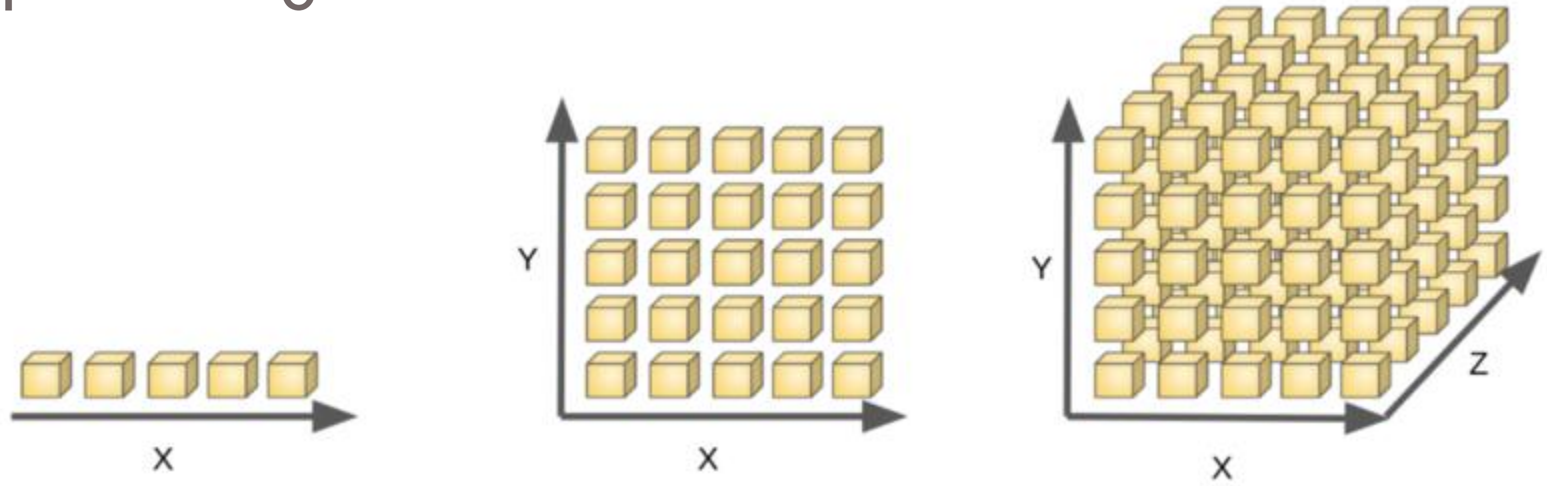


 Image Source

Curse of Dimensionality

- As dimensions grow, dimensions space increases exponentially
  - High data sparsity
  - More storage space
  - More processing time

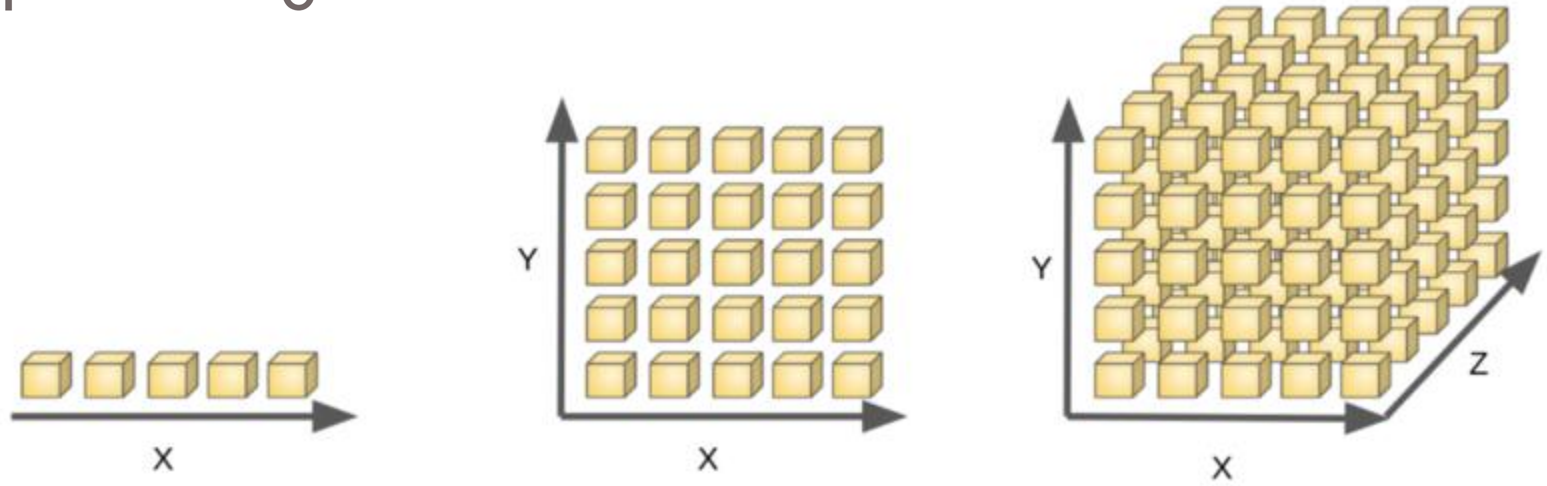


Image Source

Fighting the Curse of Dimensionality

- Generalization btw semantically & syntactically similar words
- Similar words are expected to have similar feature vectors
  - Training data:
    - The cat is walking in the bedroom
  - Test data:
    - A dog was running in a room
    - The cat is running in a room
    - A dog is walking in a bedroom
    - The dog was walking in the room

Neural Probabilistic Language Model

- Distributed Representations
  - A distributed word feature vector
    - Real-valued vector
    - # features <<< vocabulary size
  - Learn the LM and word feature vectors simultaneously

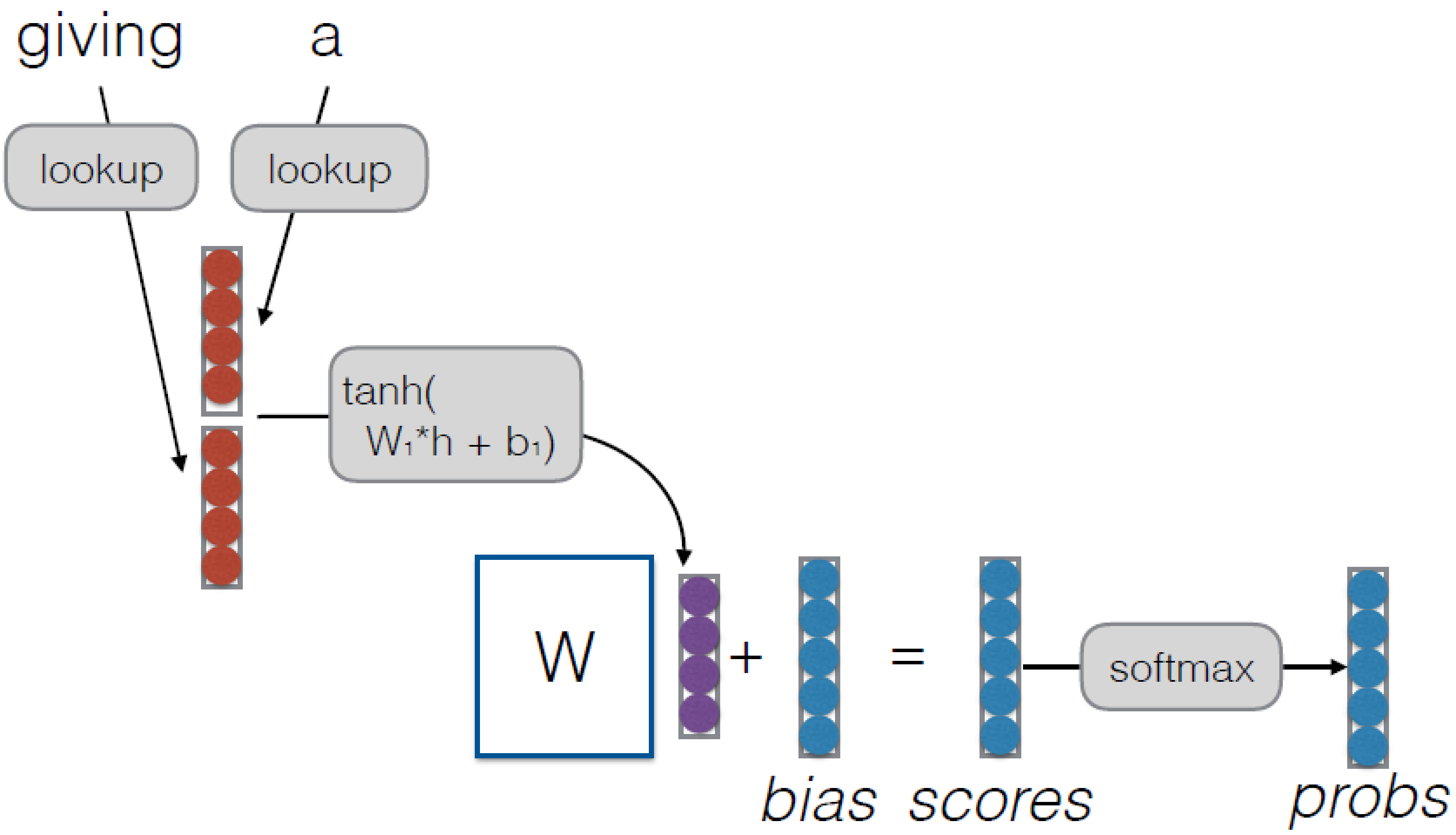


Image Credit: Graham Neubig

Approaches to Language Modeling

- Count-based (MLE-based) Language Modeling (ngrams)
  - # possible ngrams over vocabulary V is $|V|^n$
    - Increase n by 1 will result in a $|V|$ fold increase
- Neural Network Language Models (NNLM)
  - # parameters:
    - linearly with the size of the input window n
    - linearly with the size of the vocabulary
  - Amount of computation required >> n-gram models

Approaches to Language Modeling

- Count-based (MLE-based) Language Modeling (ngrams)
  - # possible ngrams over vocabulary V is $|V|^n$
    - Increase n by 1 will result in a $|V|$ fold increase
- Neural Network Language Models (NNLM)
  - # parameters:
    - linearly with the size of the input window n
    - linearly with the size of the vocabulary
  - Amount of computation required >> n-gram models

Fight with the curse of dimensionality ✓

Approaches to Language Modeling

- Brown corpus: 1.2M words
  - Count-based (MLE-based) Language Modeling (ngrams)
    - Perplexity: 312 (SOA: Class-based back-off)
  - Neural Network Language Models (NNLM)
    - Perplexity: 270

Approaches to Language Modeling

- Brown corpus: 1.2M words
  - Count-based (MLE-based) Language Modeling (ngrams)
    - Perplexity: 312 (SOA: Class-based back-off)
  - Neural Network Language Models (NNLM)
    - Perplexity: 270

Effectiveness

By Product: Distributed Word Representations

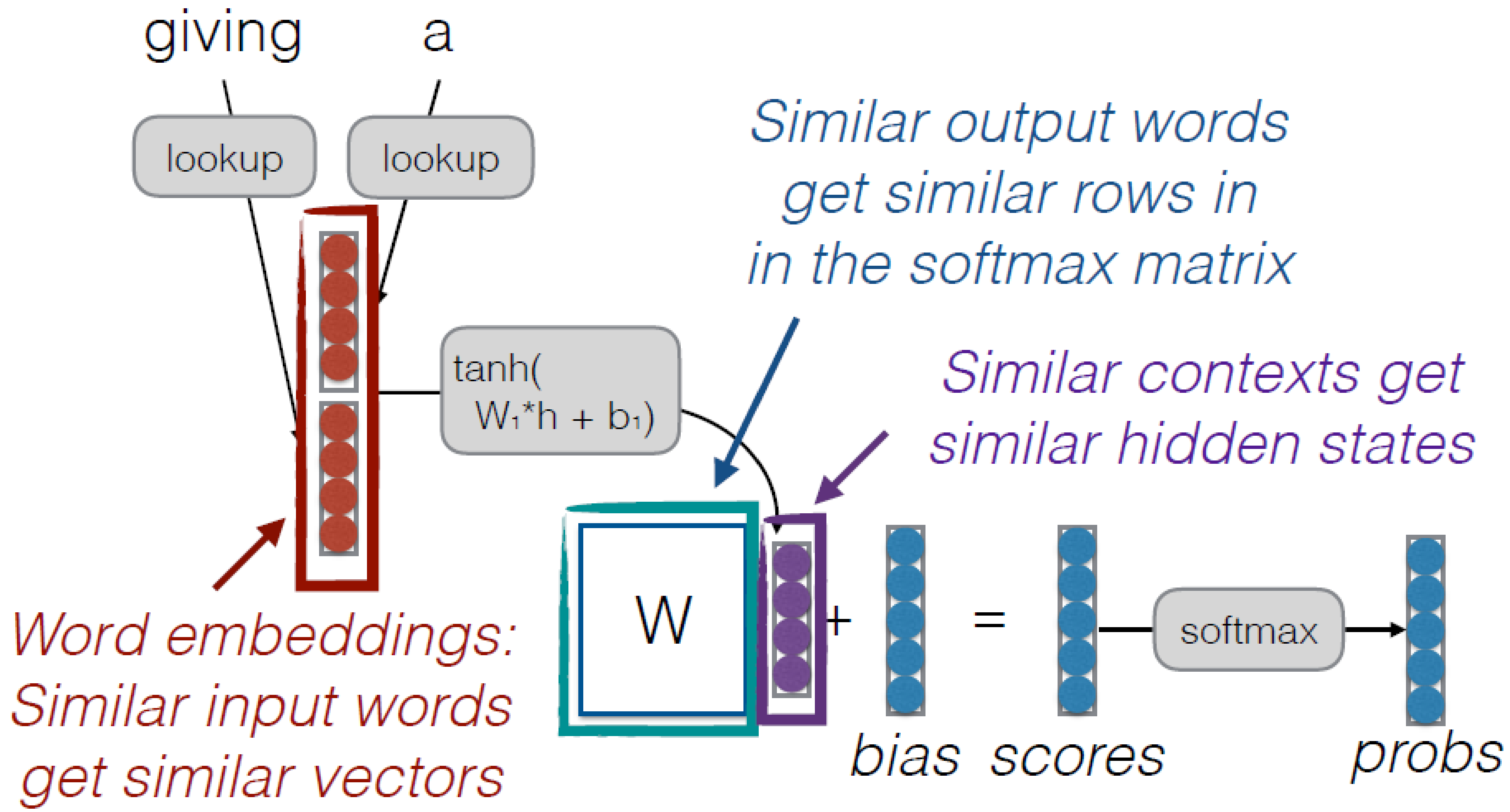


Image Credit: Graham Neubig

Outline

- Language Modeling
  - Problems with ngrams
  - Neural Network Language Models (NNLM)
  - Word Embeddings
  - CNN Applied to Text
  - Recurrent Neural Networks (RNNs)
  - Recurrent Neural Network Language Models (RNNLM)

Distributed Word Representations

- Yoshua Bengio, 2003, Neural Probabilistic Language Model
  - Jointly learning word representation and statistical LM
- Tomas Mikolov, 2007, LM for Speech Recognition in Czech
  - First word vectors are learned using NN with one hidden layer
  - Then vectors are used to train NNLM
- Non-linear hidden layer is the computational hot point ☹
- A simpler and efficient model?

Word2Vec

- Continuous Bag-of-Words: Predict the current word given context

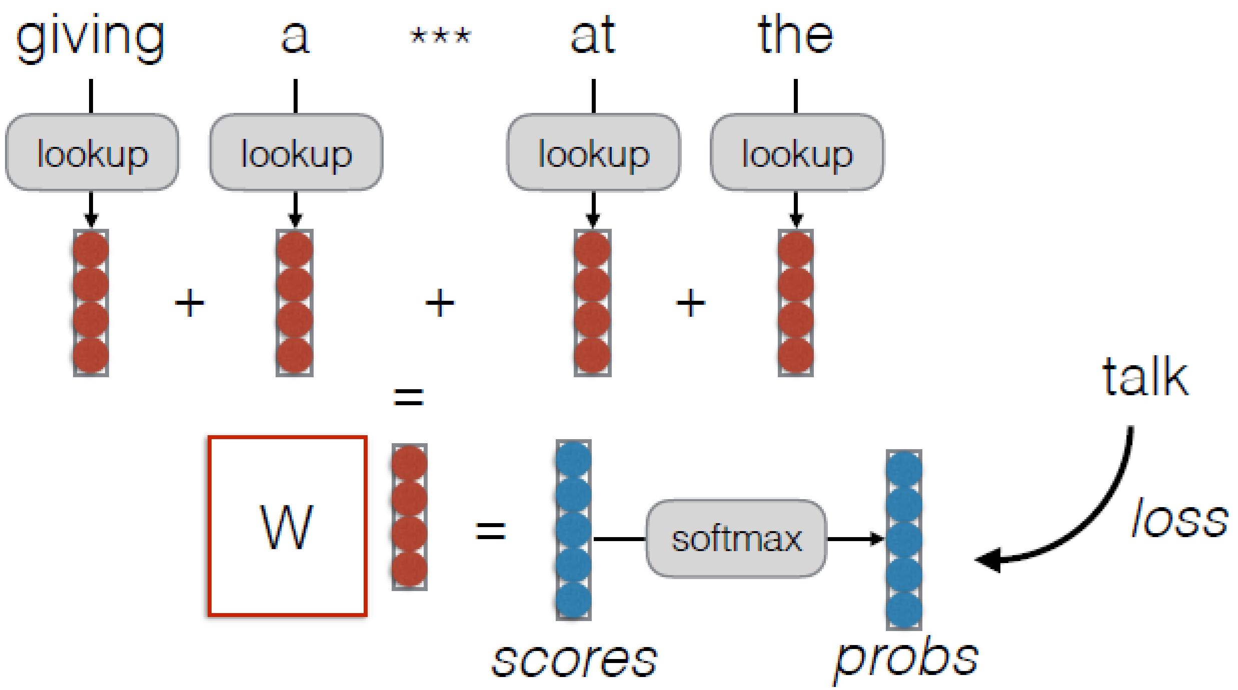


Image Credit: Graham Neubig

Word2Vec (Mikolov et. al, 2013)

- Skip-gram: Given a word, predict the context words

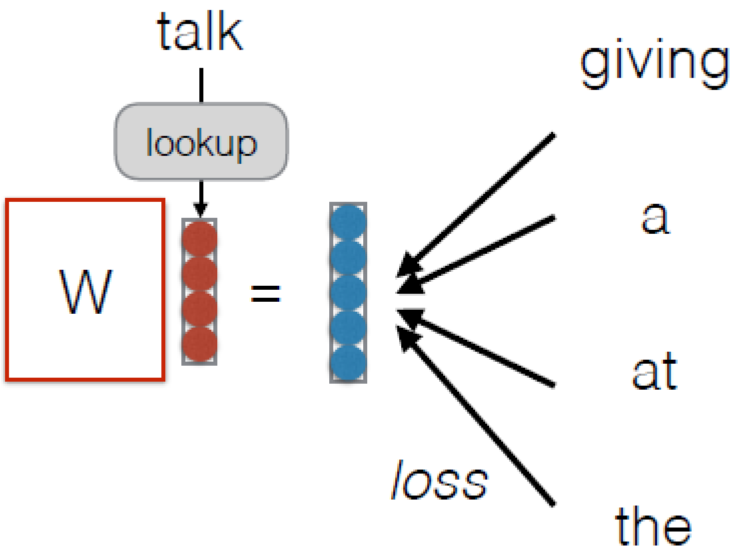


Image Credit: Graham Neubig

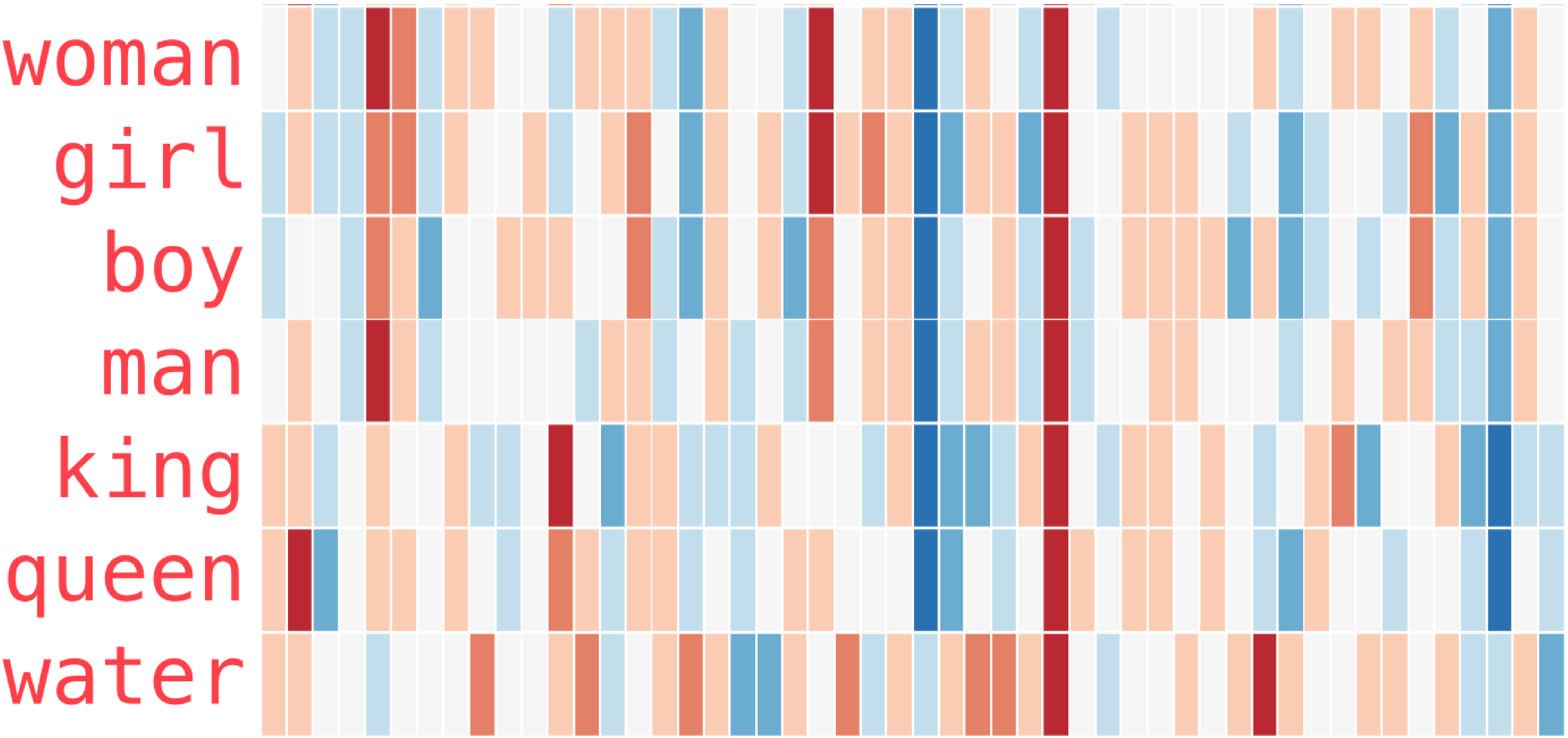


Image Source: http://jalammar.github.io/illustrated-word2vec/

king − man + woman ~= queen

```
model.most_similar(positive=["king","woman"], negative=["man"])
```

```
[('queen', 0.8523603677749634),
 ('throne', 0.7664333581924438),
 ('prince', 0.7592144012451172),
 ('daughter', 0.7473883032798767),
 ('elizabeth', 0.7460219860076904),
 ('princess', 0.7424570322036743),
 ('kingdom', 0.7337411642074585),
 ('monarch', 0.721449077129364),
 ('eldest', 0.7184862494468689),
 ('widow', 0.7099430561065674)]
```


Image Source: http://jalammar.github.io/illustrated-word2vec/

CNN Applied to Text

ngram detectors

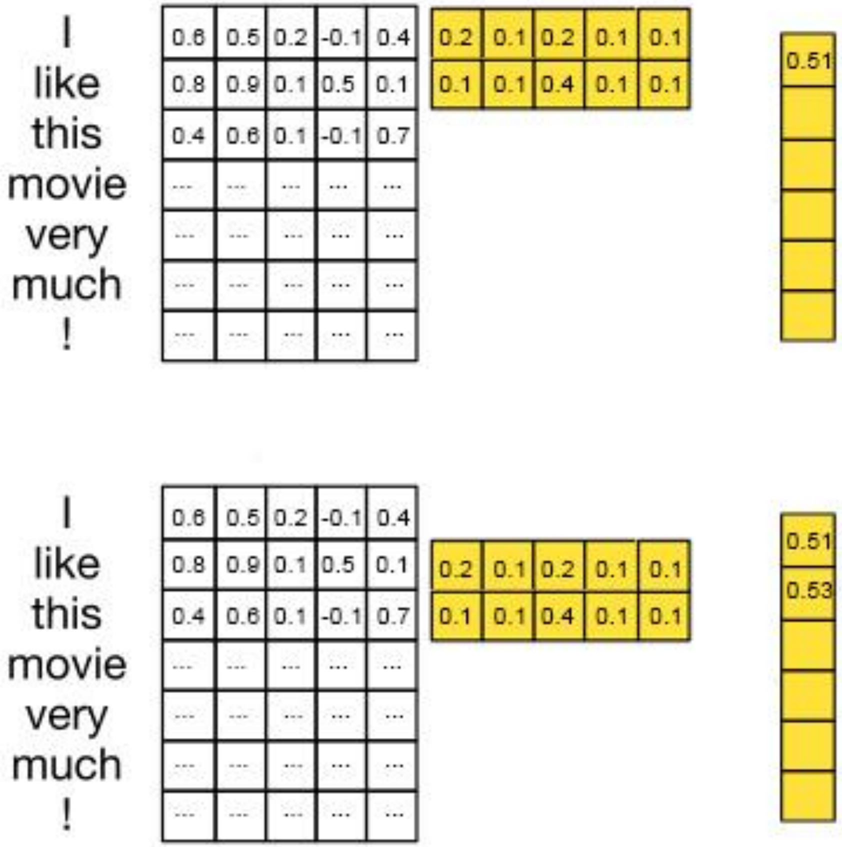


Image Source

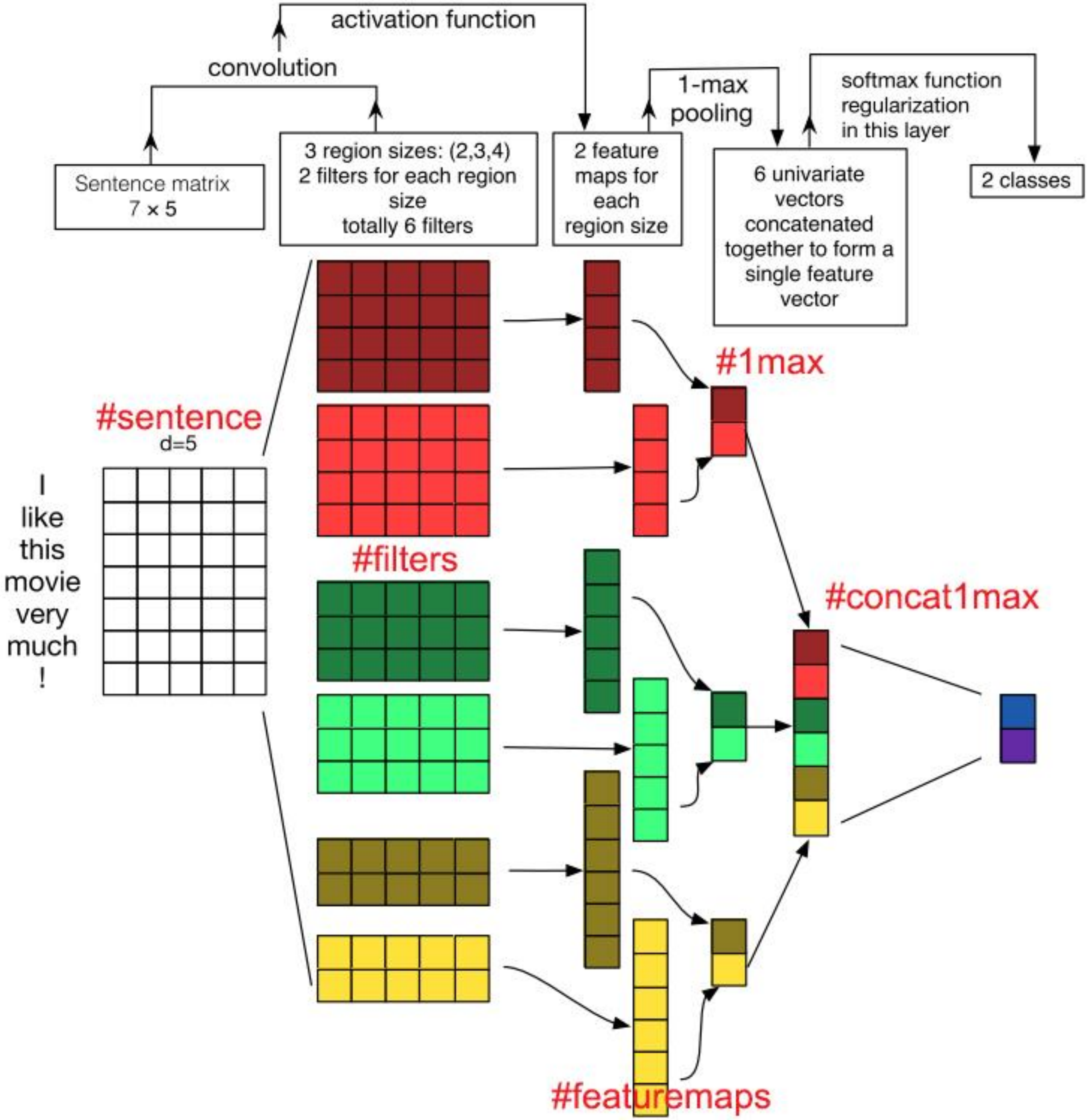


Image Source

Convolutional Neural Networks

- CNN Architecture
  - Identify indicative local predictors in a large structure
  - Combine them to produce a fixed size vector representation
  - Capture local aspects that are most informative for prediction task

CNN for Sentence Classification by Yoon Kim

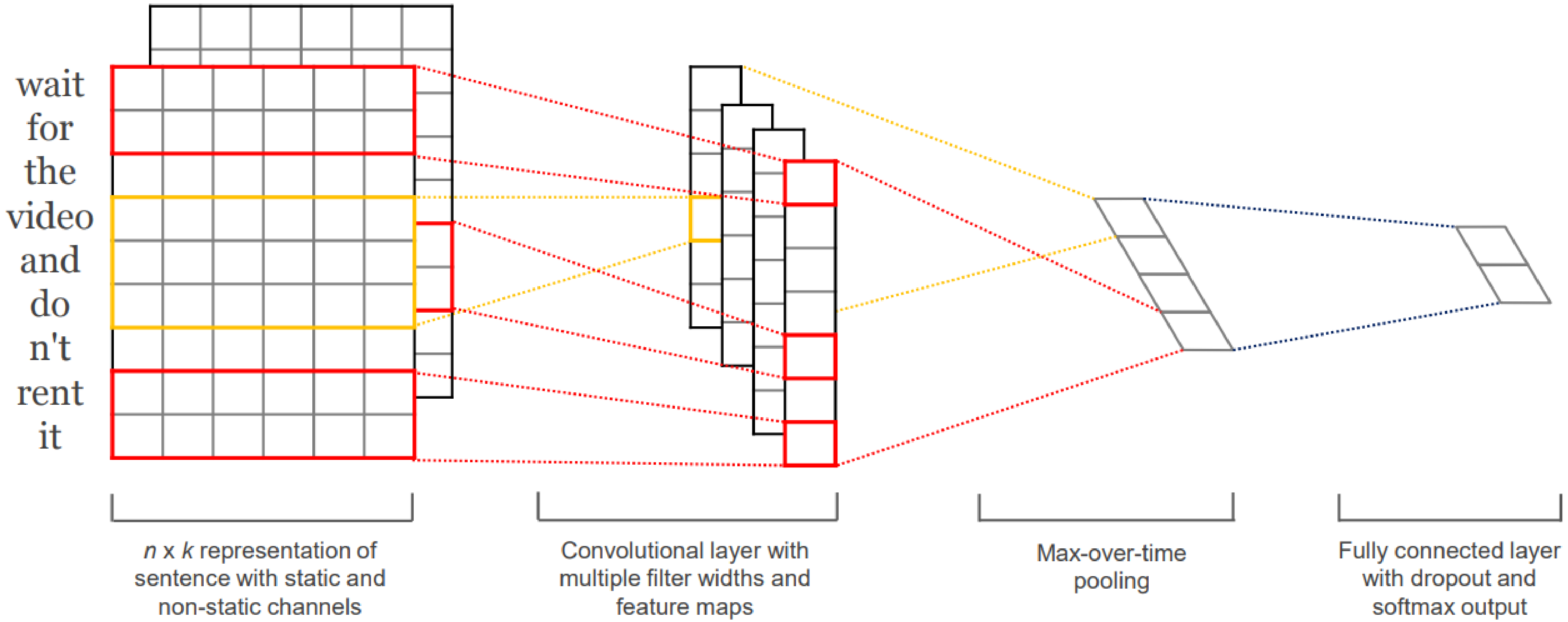
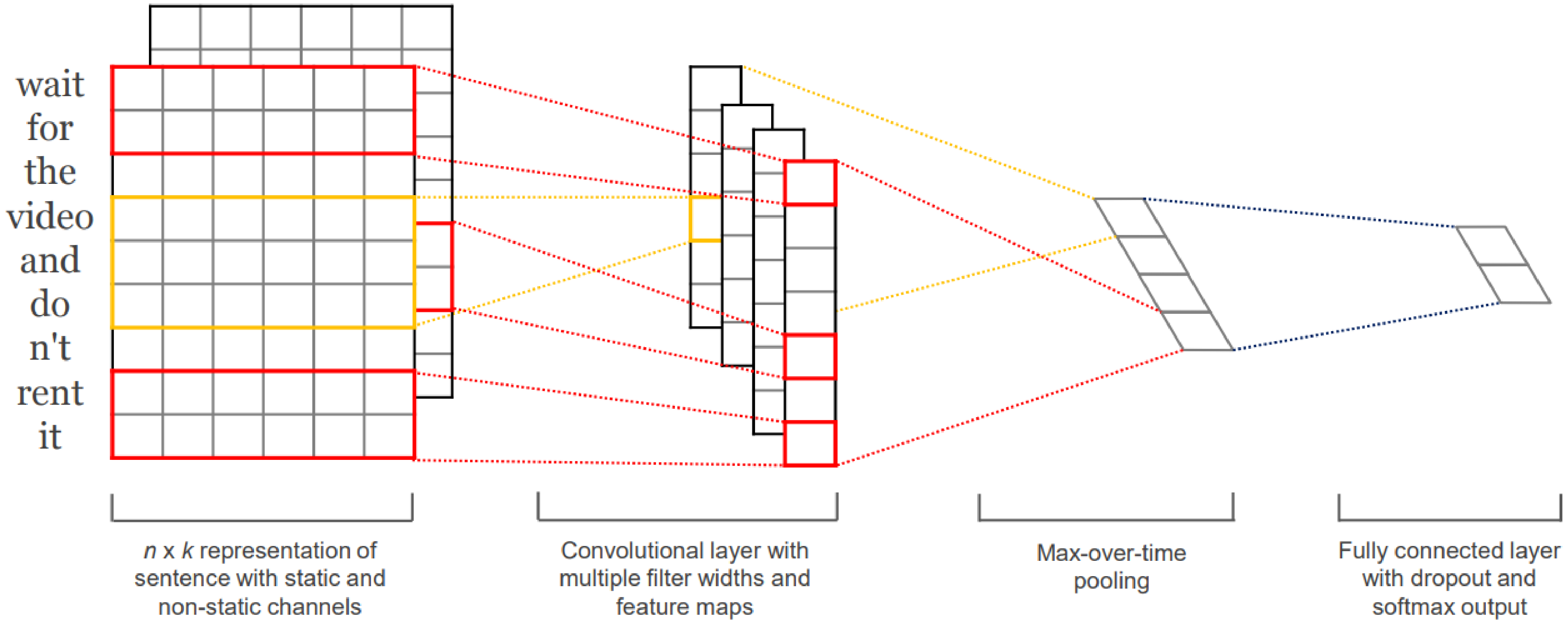


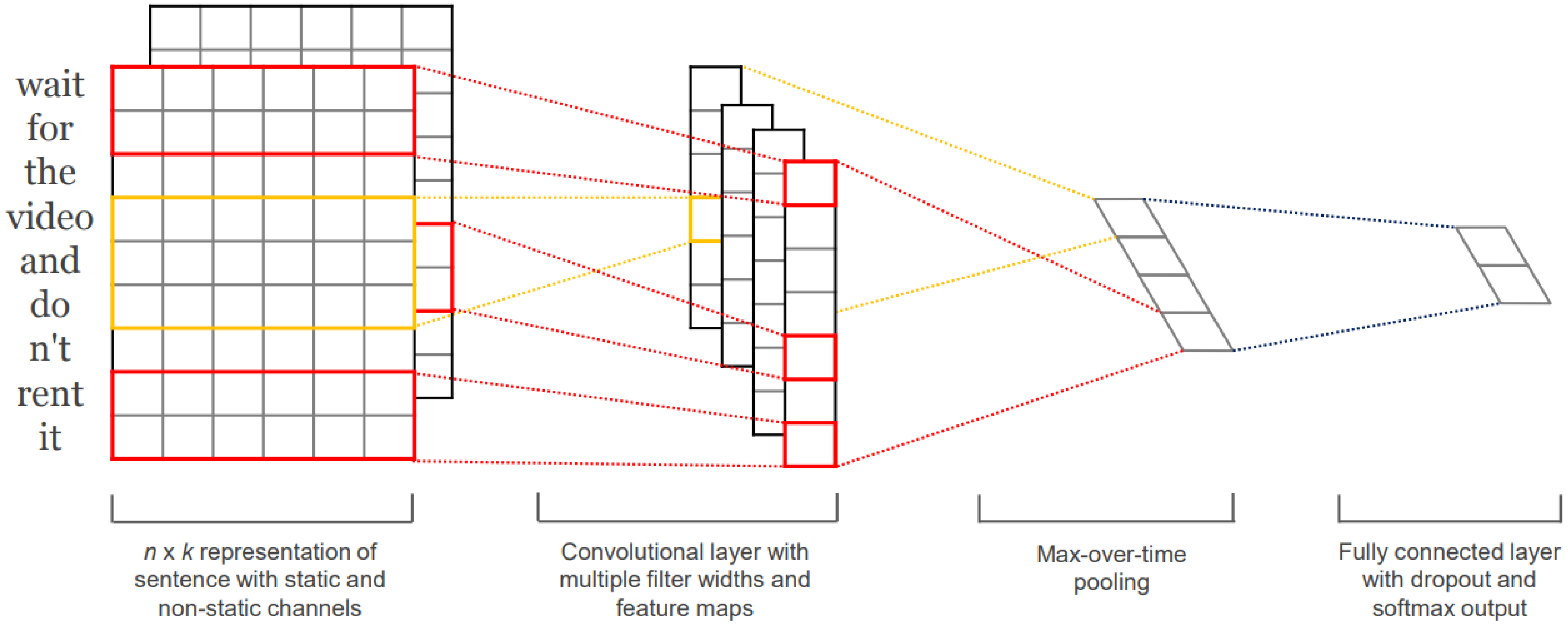
Figure 1: Model architecture with two channels for an example sentence.

CNN for Sentence Classification by Yoon Kim



Two channels of word vectors: (1) static and (2) non-static (fine tuned via backpropagation)

CNN for Sentence Classification by Yoon Kim



Multiple filters (with varying window sizes) are used to obtain multiple features.

CNN for Sentence Classification by Yoon Kim

- Competitive results compared to more sophisticated models
- Pretrained word vectors are important

CNN for Sentence Classification by Yoon Kim

- Competitive results compared to more sophisticated models
- Pretrained word vectors are important
- But retraining is also important

	Most Similar Words for	
	Static Channel	Non-static Channel
bad	*good* *terrible* *horrible* *lousy*	*terrible* *horrible* *lousy* *stupid*
good	*great* *bad* *terrific* *decent*	*nice* *decent* *solid* *terrific*
n't	*os* *ca* *ireland* *wo*	*not* *never* *nothing* *neither*

Outline

- Language Modeling
  - Problems with ngrams
  - Neural Network Language Models (NNLM)
  - Word Embeddings
  - CNN Applied to Text
  - Recurrent Neural Networks (RNNs)
  - Recurrent Neural Network Language Models (RNNLM)

Neural Network Language Models

- Pros
  - Can generalize unseen contexts
  - Can scale to much larger n's
- Cons
  - More expensive than count-based LMs

Neural Network Language Models

- Pros
  - Can generalize unseen contexts
  - Can scale to much larger n's
- Cons
  - More expensive than count-based LMs
- NNLMs can get good perplexities even with relatively small training sets, but in MT, they may not beat count-based LMs

Neural Network Language Models

- Pros
  - Can generalize unseen contexts
  - Can scale to much larger n's
- Cons
  - More expensive than count-based LMs
- NNLMs can get good perplexities even with relatively small training sets, but in MT, they may not beat count-based LMs
  - black horse, brown horse, white horse but not red horse

A Neural Probabilistic Language Model

- Do we still have these problems?
  - Problem 1: Intervening words cause limited context modeling
  - Problem 2: Ignoring the "similarity" between words
  - Problem 3: Cannot handle long-distance dependencies

A Neural Probabilistic Language Model

- Do we still have these problems?
  - Problem 1: Intervening words cause limited context modeling
  - Problem 2: Ignoring the “similarity” between words
  - Problem 3: Cannot handle long-distance dependencies

A Neural Probabilistic Language Model

- Do we still have these problems?
  - Problem 1: Intervening words cause limited context modeling
  - Problem 2: Ignoring the "similarity" between words
  - Problem 3: Cannot handle long-distance dependencies

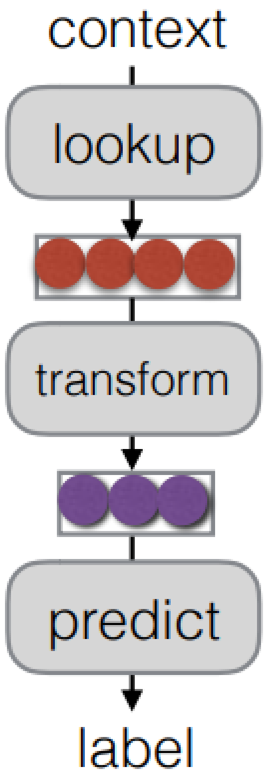
Recurrent Neural Network Language Models (RNNLM)

Outline

- Language Modeling
  - Problems with ngrams
  - Neural Network Language Models (NNLM)
  - Word Embeddings
  - CNN Applied to Text
  - Recurrent Neural Networks (RNNs)
    - RNN Abstraction
    - Concrete RNN Architecture (LSTM)
  - Recurrent Neural Network Language Models (RNNLM)

Recurrent Neural Networks (RNN)

Feed-forward NN



Recurrent NN

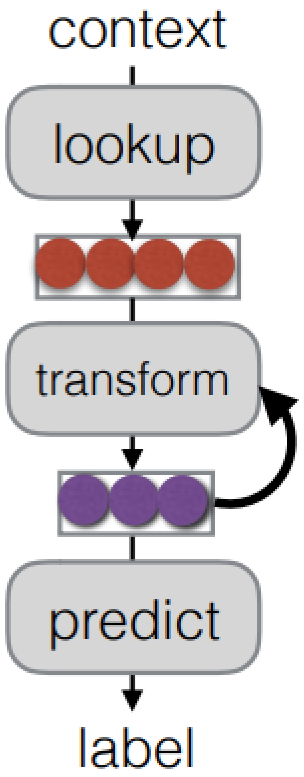


Image Credit: Graham Neubig

RNN Unrolled in Time

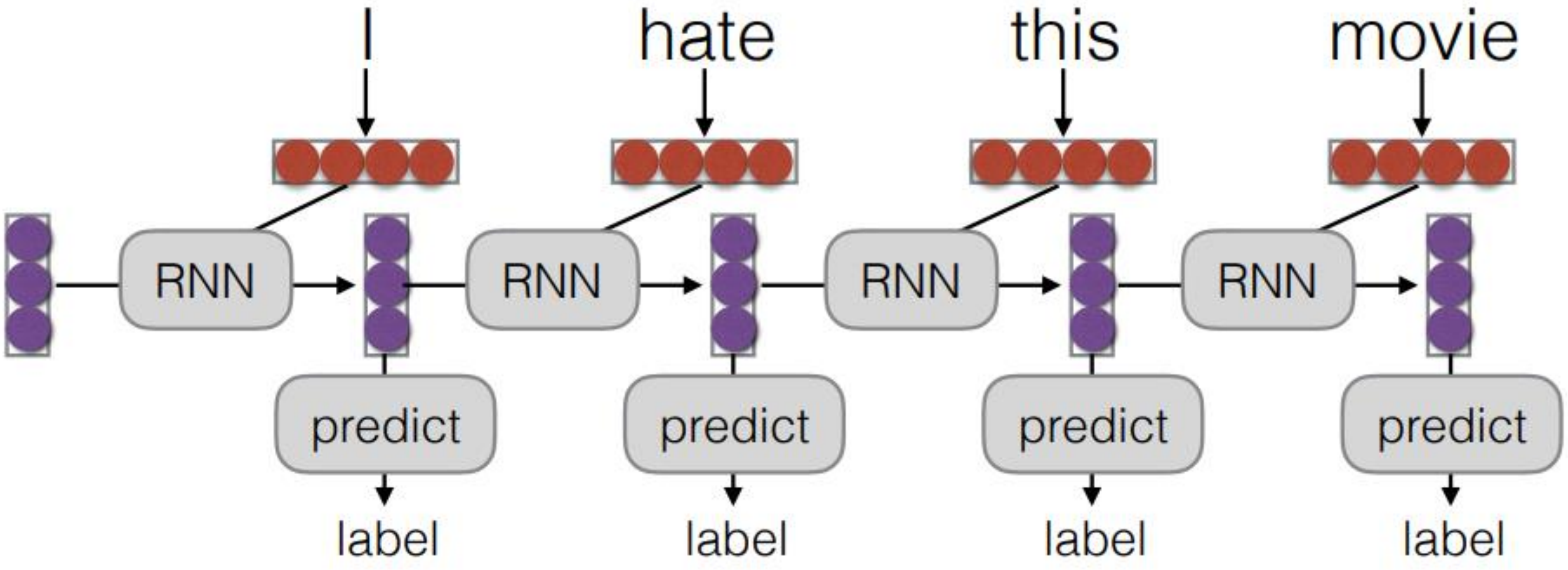


Image Credit: Graham Neubig

Recurrent Neural Networks (RNN)

- The unrolled RNN is like a deep NN in which
  - The parameters are shared across layers
  - Additional input is added at various layers

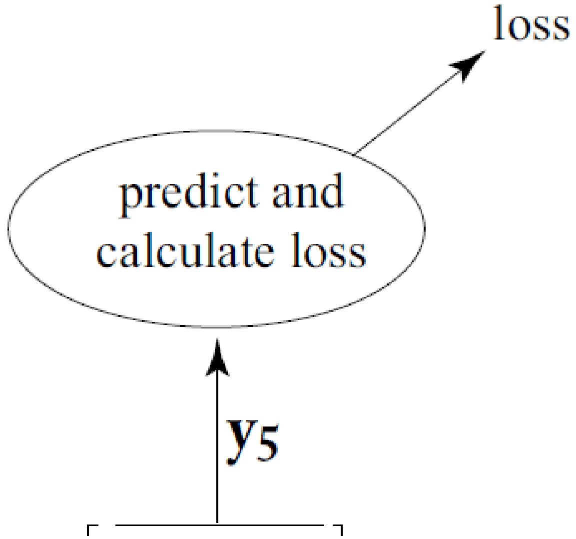
- RNN is just a component to encode the input sequence
  - Through training RNN learns to represent the input more suited for the final prediction task

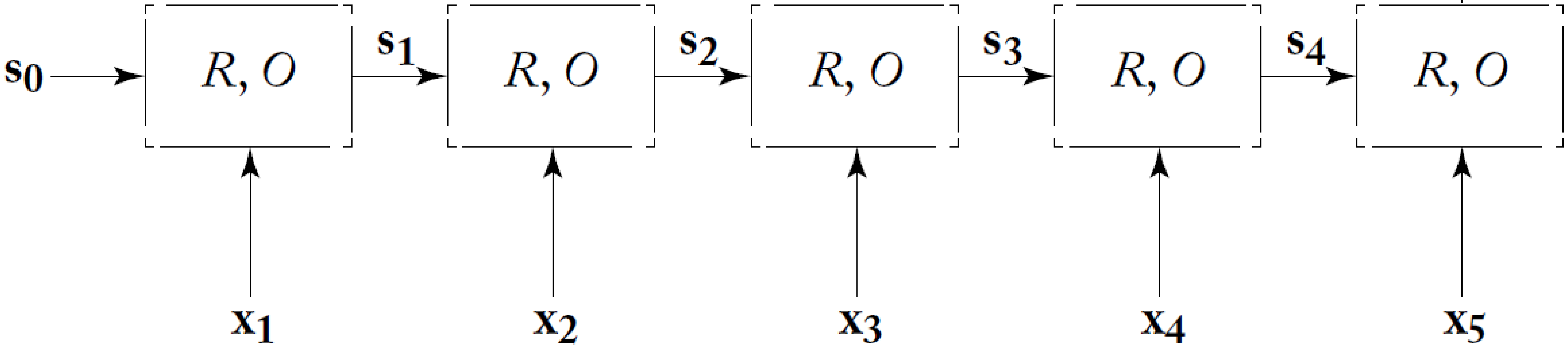
Recurrent Neural Networks (RNN)

- There are several RNN usage patterns:
  - Acceptor
  - Encoder
  - Transducer

RNN as Acceptor

- Supervision signal is based on only the final output vector
- Decide on an outcome after observing the final state



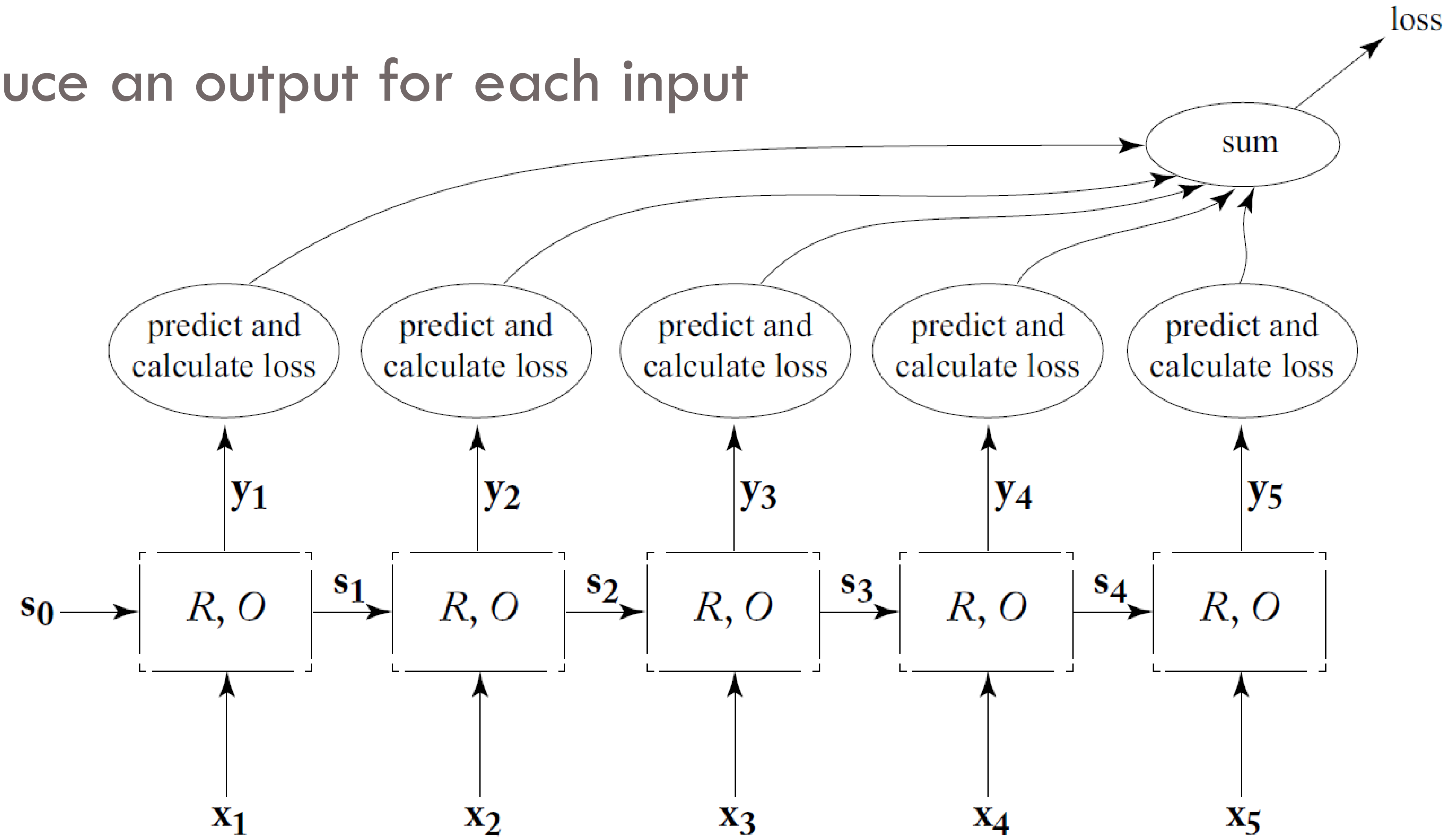


RNN as Encoder

- Like the acceptor, it only uses the output vector
- Unlike the acceptor, prediction is not based on the output vector only
- Output vector is an encoding of the input sequence which is used together with other useful signals for prediction

RNN as Transducer

- Produce an output for each input



RNN as Transducer

- Example:
  - Task: Language Modeling
  - Input: Current word (previous context)
  - Output: Next word (a tag is anything in the vocabulary)

RNN as Transducer

- Example:
  - Task: Language Modeling

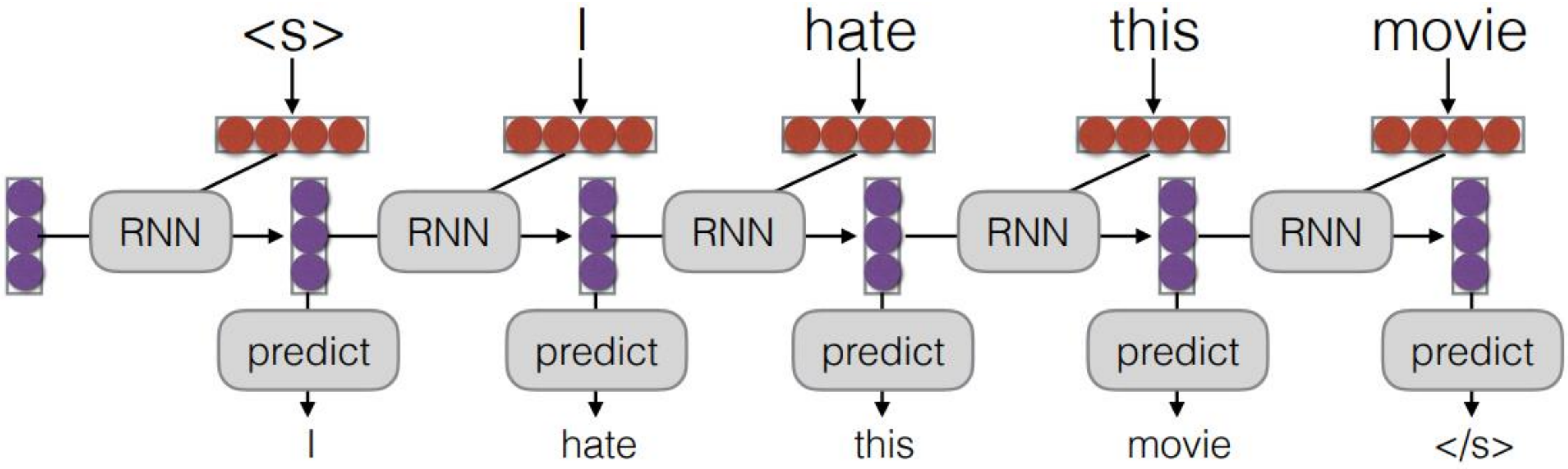


Image Credit: Graham Neubig

BiDirectional RNN (BiRNN)

- Use both previous and following words

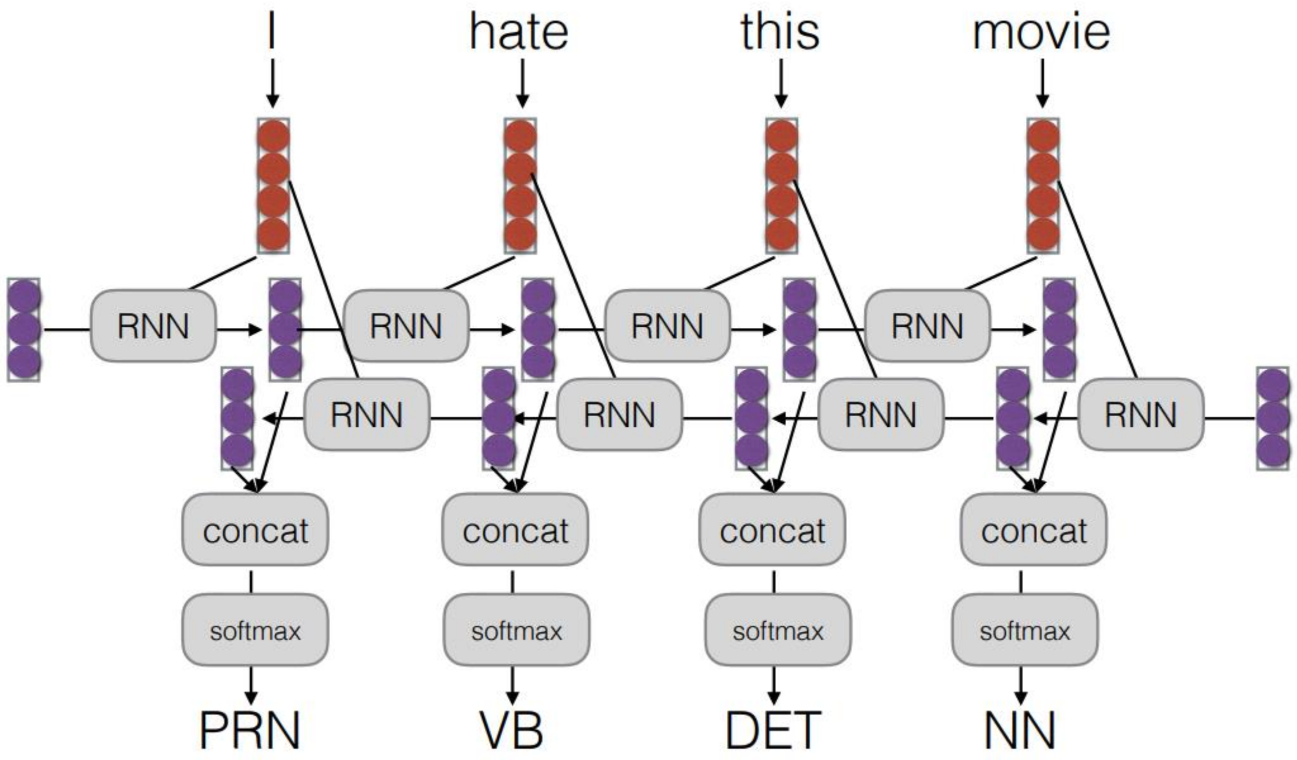


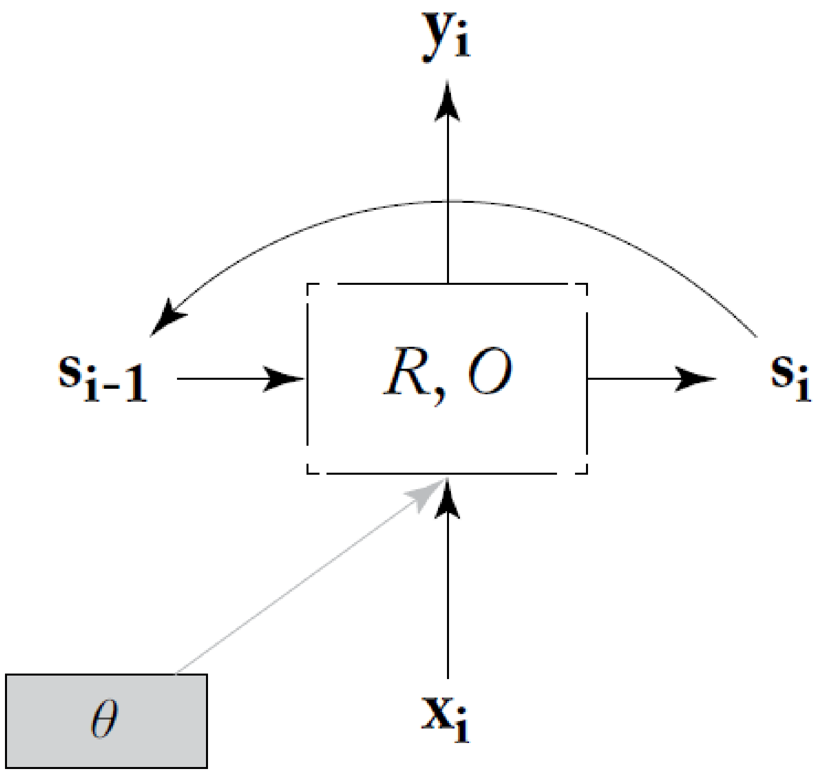
Image Credit: Graham Neubig

RNN Architectures

- The RNN abstraction

$$y_i = O(s_i)$$

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



- We need concrete definitions of R and O functions

Simple RNN (SRNN)

$$\boldsymbol{s_i} = R_{\text{SRNN}}(\boldsymbol{x_i}, \boldsymbol{s_{i-1}}) = g(\boldsymbol{s_{i-1}} \boldsymbol{W^s} + \boldsymbol{x_i} \boldsymbol{W^x} + \boldsymbol{b})$$

$$\boldsymbol{y_i} = O_{\text{SRNN}}(\boldsymbol{s_i}) = \boldsymbol{s_i}$$

$$\boldsymbol{s_i}, \boldsymbol{y_i} \in \mathbb{R}^{d_s}, \quad \boldsymbol{x_i} \in \mathbb{R}^{d_x}, \quad \boldsymbol{W^x} \in \mathbb{R}^{d_x \times d_s}, \quad \boldsymbol{W^s} \in \mathbb{R}^{d_s \times d_s}, \quad \boldsymbol{b} \in \mathbb{R}^{d_s}$$

The Problem of S-RNN

- Hard to train due to vanishing gradient problem
  - Error signals (gradients) diminish quickly and do not reach to earlier input signals (cannot capture long-range dependencies)

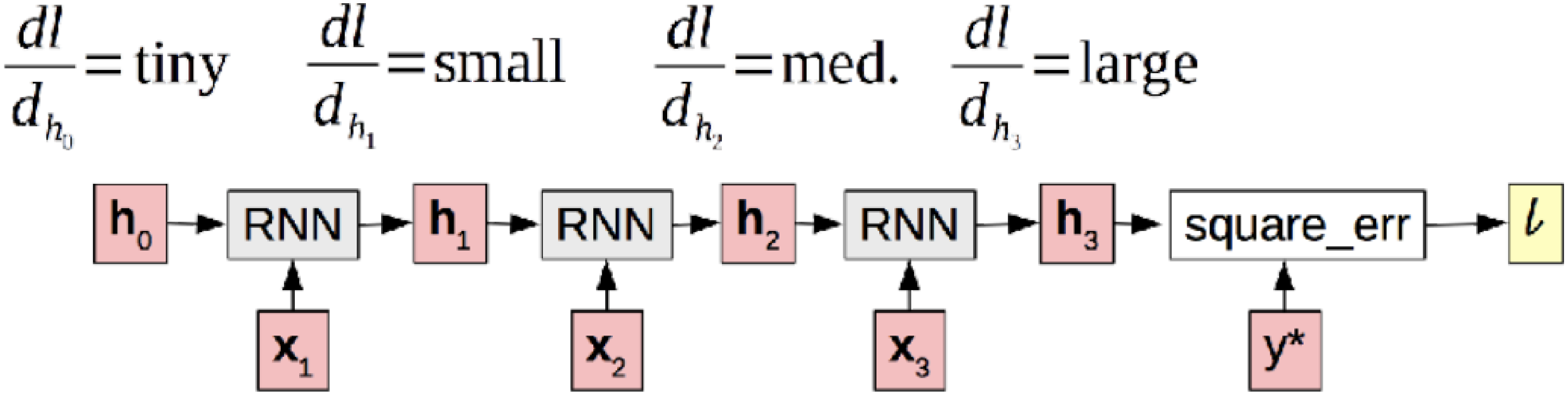


Image Credit: Graham Neubig

The Problem of S-RNN

- Vanishing Gradient Problem:
  - Aren't the parameters tied within the whole network? So why do we have a learning problem at the beginning of the sentence?
    - Yes, but
    - Word embeddings won't get updated for words at the beginning of the sentence
    - RNN matrices won't be updated in the ideal way for the beginning of the sentence

The Problem of S-RNN

- Consider the RNN architecture
  - Reads the current memory state s_i
  - Reads the input x_{i+1}
  - Operates on these
  - Write the new memory state s_{i+1}
- The problem is that the memory access is not controlled
  - At each step entire memory is read and written
- Gated architectures solve this problem

Gated Architectures

$g \in \{0,1\}^n$

$$\boxed{\boldsymbol{s}' \leftarrow \boldsymbol{g} \odot \boldsymbol{x} + (\mathbf{1} - \boldsymbol{g}) \odot (\boldsymbol{s})}$$

$\boldsymbol{g} \in 0, 1^d$ $\quad \boldsymbol{x} \in \mathbb{R}^d$ $\quad \boldsymbol{s} \in \mathbb{R}^d$

$$\underset{\mathbf{s}'}{\begin{bmatrix} 8 \\ 11 \\ 3 \\ 7 \\ 5 \\ 15 \end{bmatrix}} \leftarrow \underset{\mathbf{g}}{\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}} \odot \underset{\mathbf{x}}{\begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{bmatrix}} + \underset{(\mathbf{1}-\mathbf{g})}{\begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}} \odot \underset{\mathbf{s}}{\begin{bmatrix} 8 \\ 9 \\ 3 \\ 7 \\ 5 \\ 8 \end{bmatrix}}$$

Long Short Term Memory (LSTM)

- Designed to avoid long-term dependency problem
  - State vector s_i has two parts:
    - Cell state (c_i): Long term memory
      - Memory cells where long term dependencies are preserved
    - Hidden state (h_i): Short term memory (working memory)
      - h controls gates and decide what will go to long term memory
- There are 3 gates: input, forget, output
  - Linear combination of the input x and previous state h
  - Passed through sigmoid function

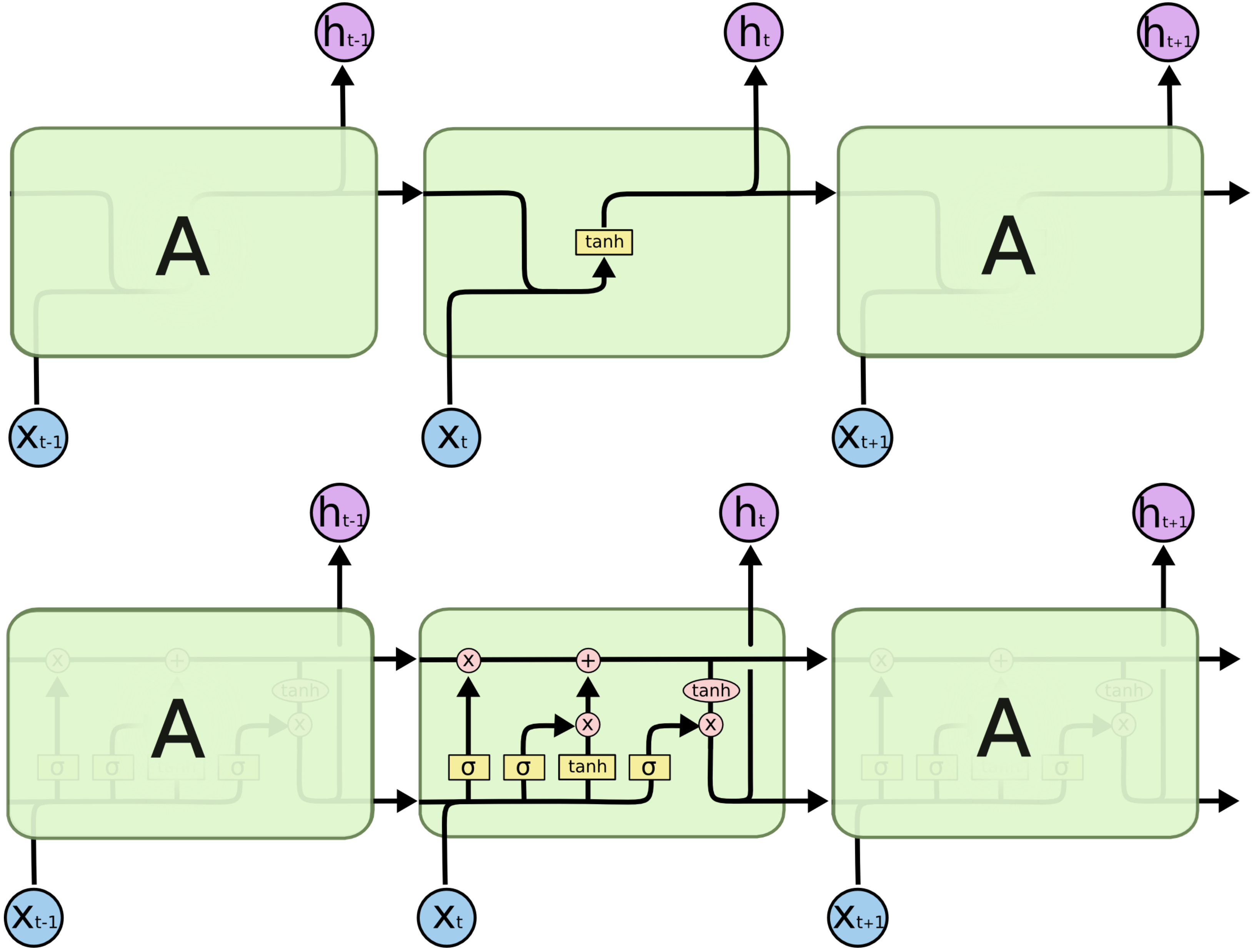


Image Credit: Chris Olah's blog post on Understanding LSTM Networks

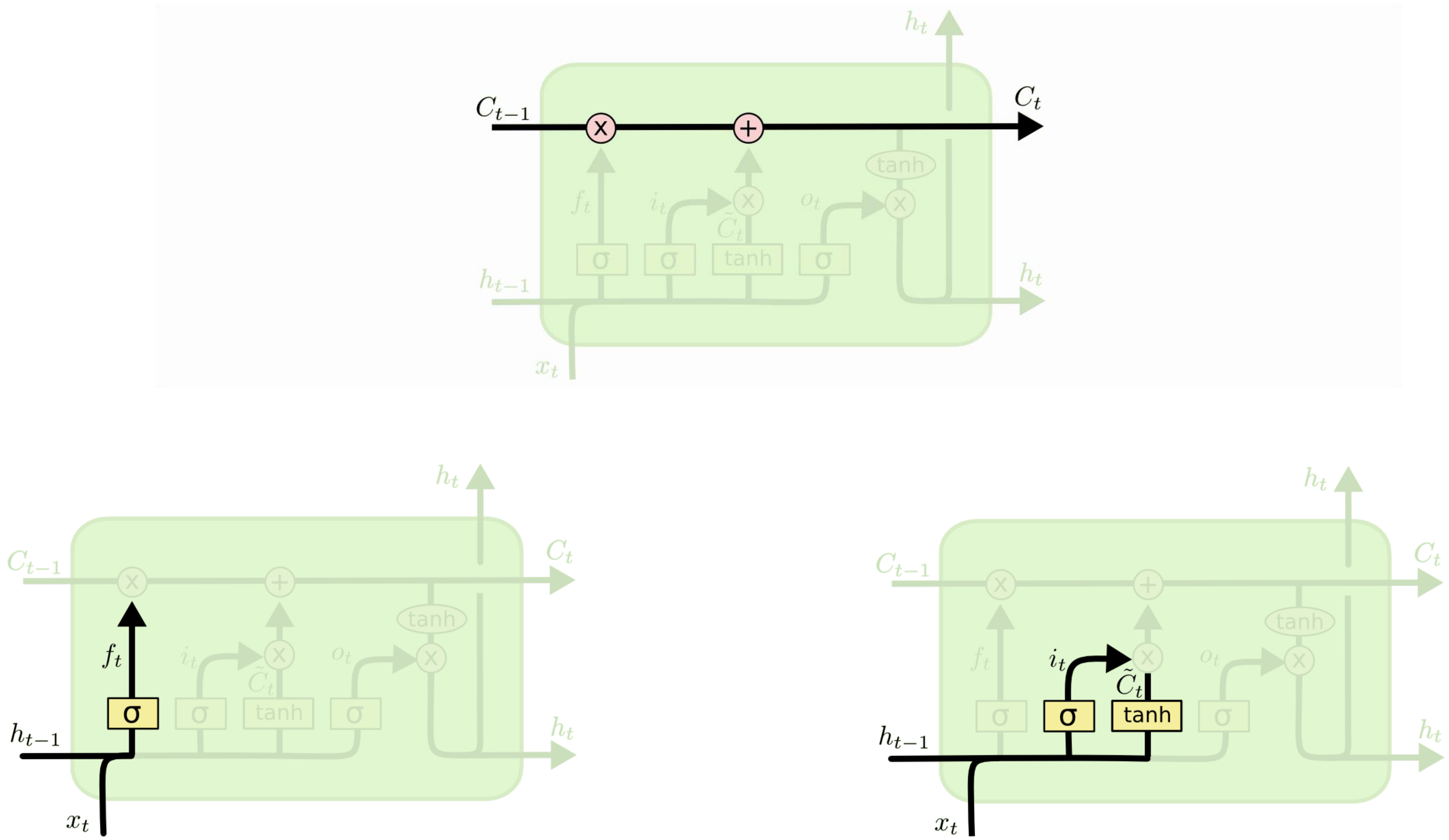
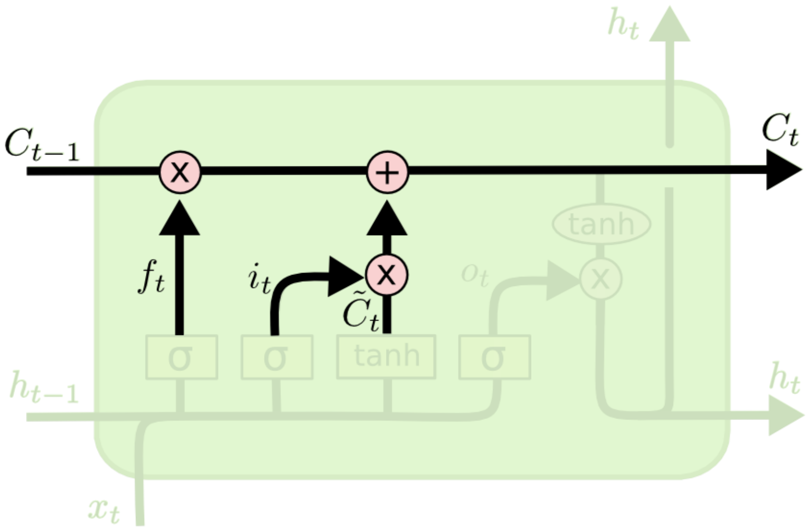


Image Credit: Chris Olah’s blog post on Understanding LSTM Networks



$$C_t = \underbrace{f_t * C_{t-1}} + \underbrace{i_t * \tilde{C}_t}$$

- Update the old cell state into new cell state
  - Multiply the old state with forget gate
  - Multiply the state update with input gate

Image Credit: Chris Olah's blog post on Understanding LSTM Networks

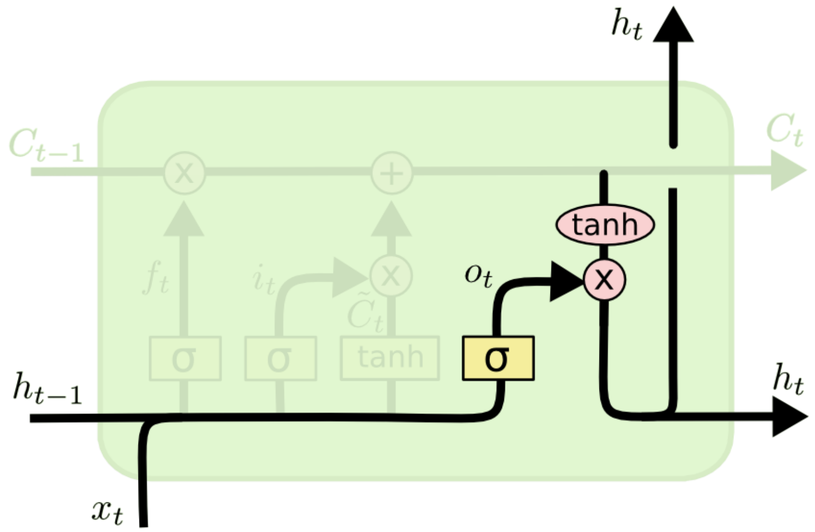


Image Credit: Chris Olah's blog post on Understanding LSTM Networks

Outline

- Language Modeling
  - Problems with ngrams
  - Neural Network Language Models (NNLM)
  - Word Embeddings
  - CNN Applied to Text
  - Recurrent Neural Networks (RNNs)
  - Recurrent Neural Network Language Models (RNNLM)

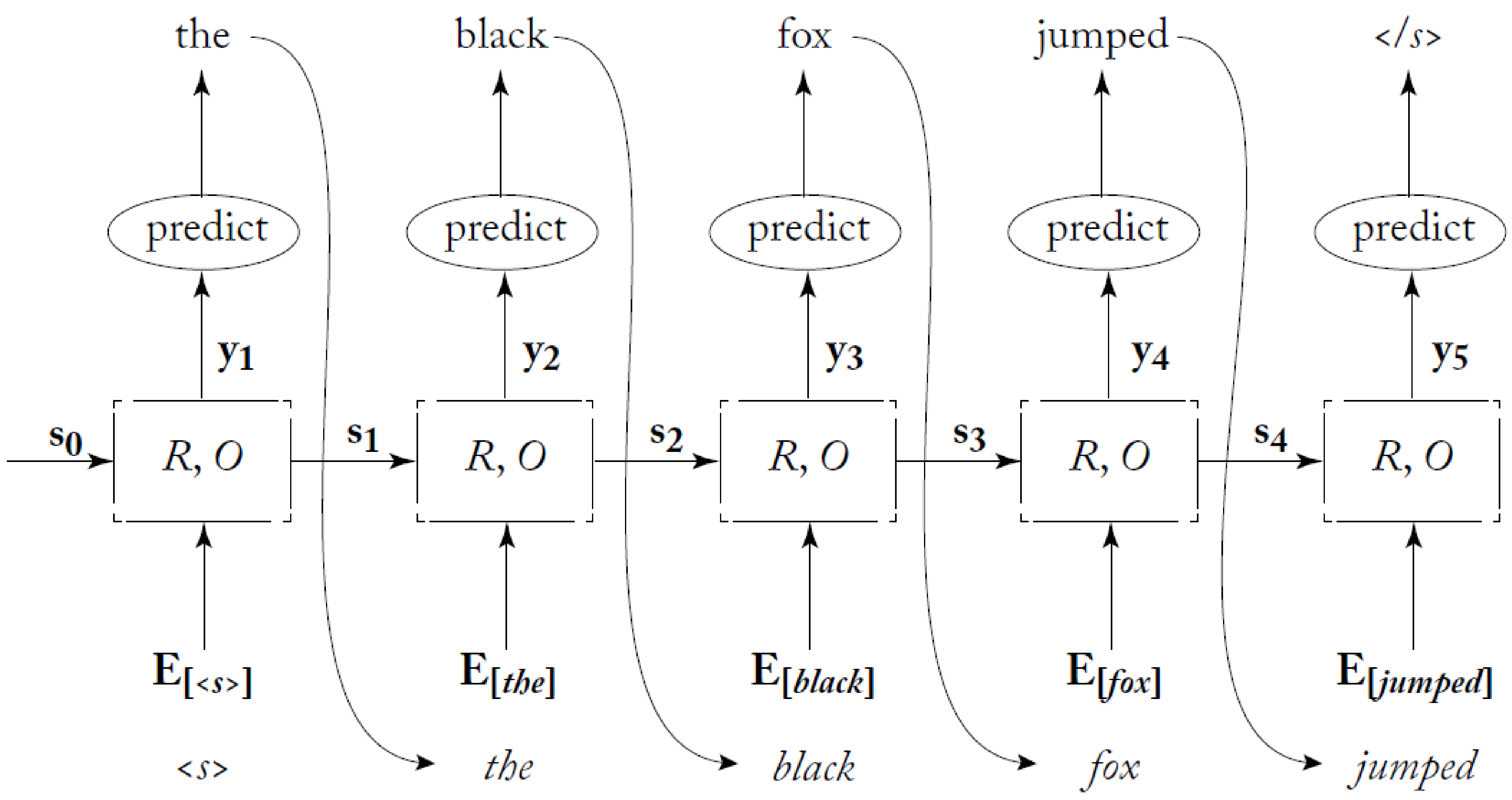
RNN for Language Modeling

- RNNs as non-markovian LMs
  - Relax the Markov assumption which is used in traditional LM
  - Can condition on the entire history

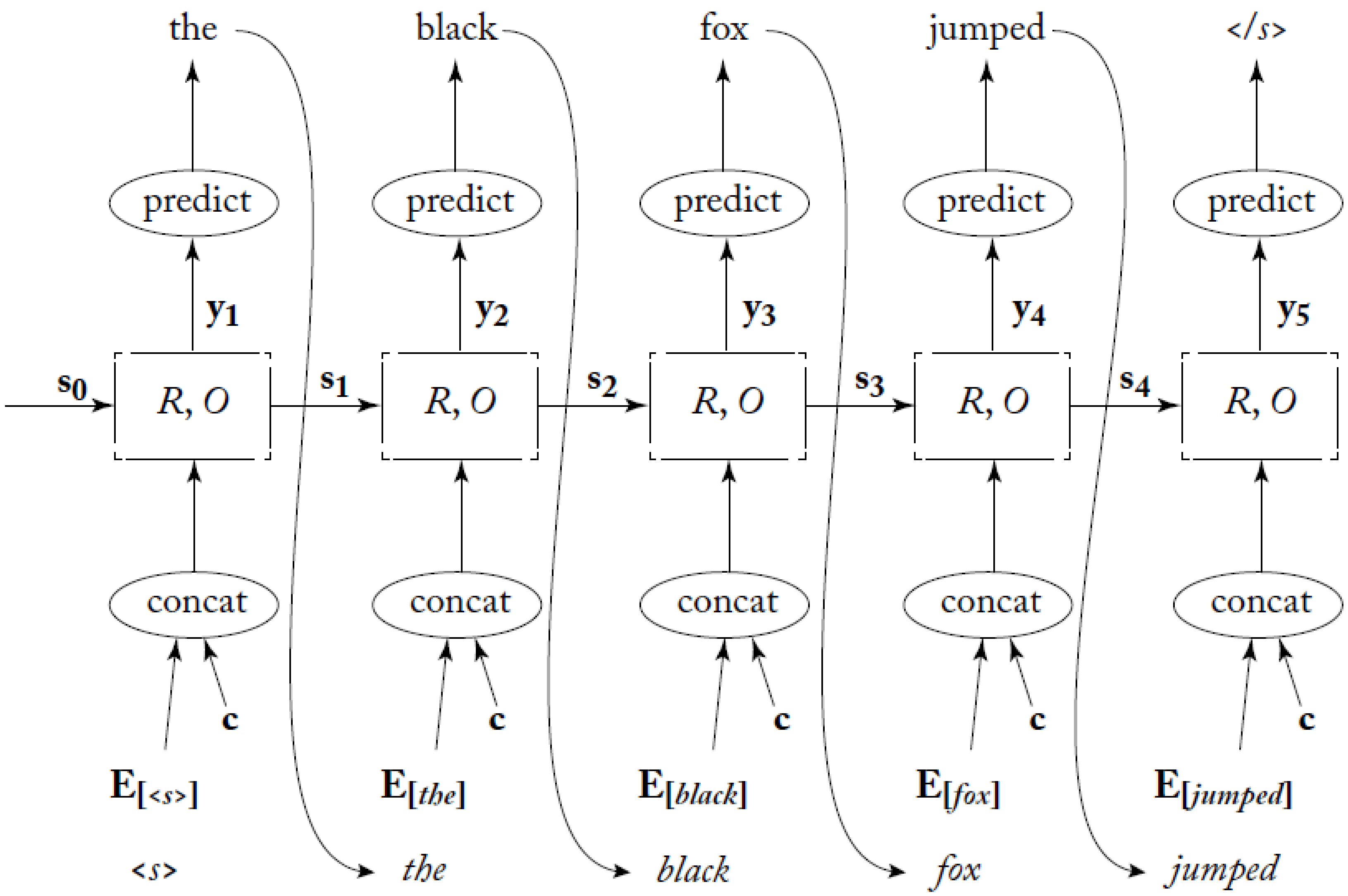
RNN Generators

- Sequence generation
  - A special case of RNN transducers
  - Generation works by tying the output of the transducer at time i with its input at time $i+1$

RNN Generators



RNN Conditioned Generators

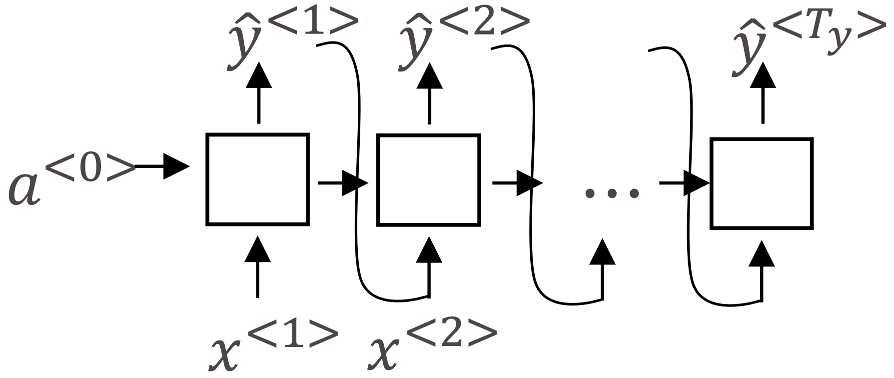


Conditioned Generations

- What is encoded in the context c?
  - Anything useful
    - Predefined topic
    - Inferred property
- The context c can have many forms
  - fixed length or set-like examples
  - sequence
- Sequence to sequence (encoder-decoder)

Sequence to Sequence Models

Language model:



Machine translation:

(Conditional Language model)

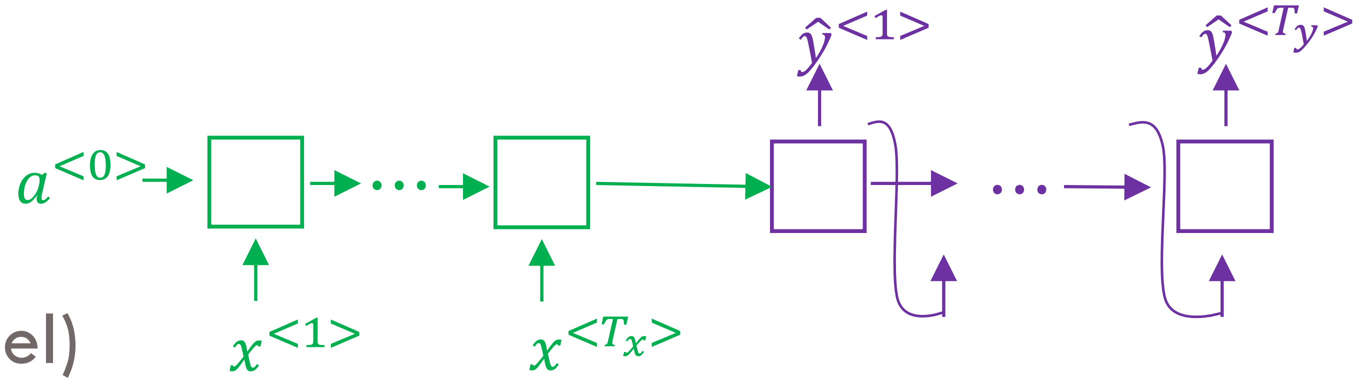
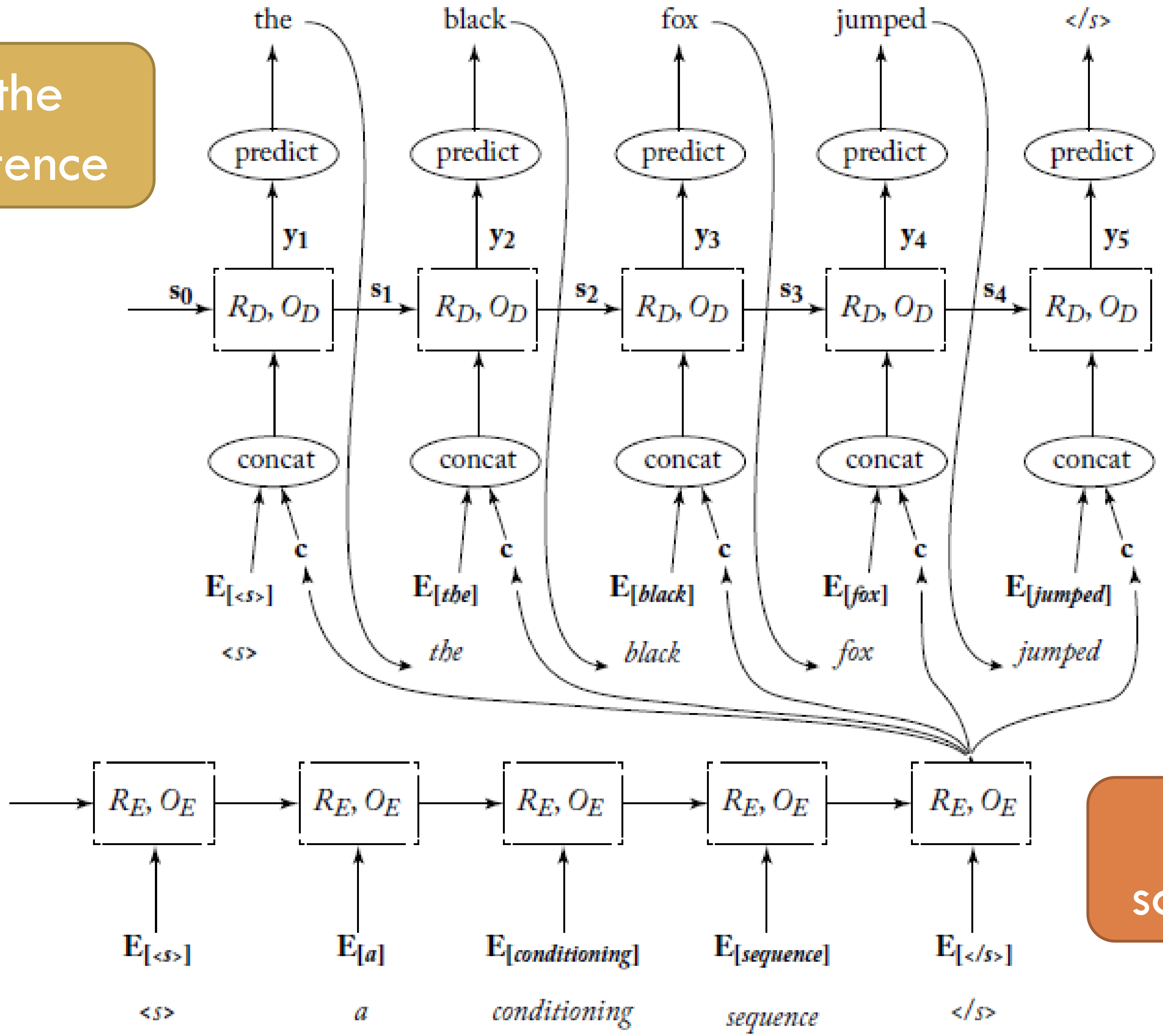


Image Credit: Andrew Ng

Decode the target sentence



Encode the source sentence

Outline

- Language Modeling
  - Problems with ngrams
  - Neural Network Language Models (NNLM)
  - Word Embeddings
  - CNN Applied to Text
  - Recurrent Neural Networks (RNNs)
  - Recurrent Neural Network Language Models (RNNLM)
- Sequence-to-Sequence Models
- Attention Models

Conditioned Generation

- Encode all information into a single fixed-size vector

$$c = \textsc{Enc}(\boldsymbol{x}_{1:n})$$

- Decoder use this vector and generates the output sequence
- This architecture works quite well, most of the time

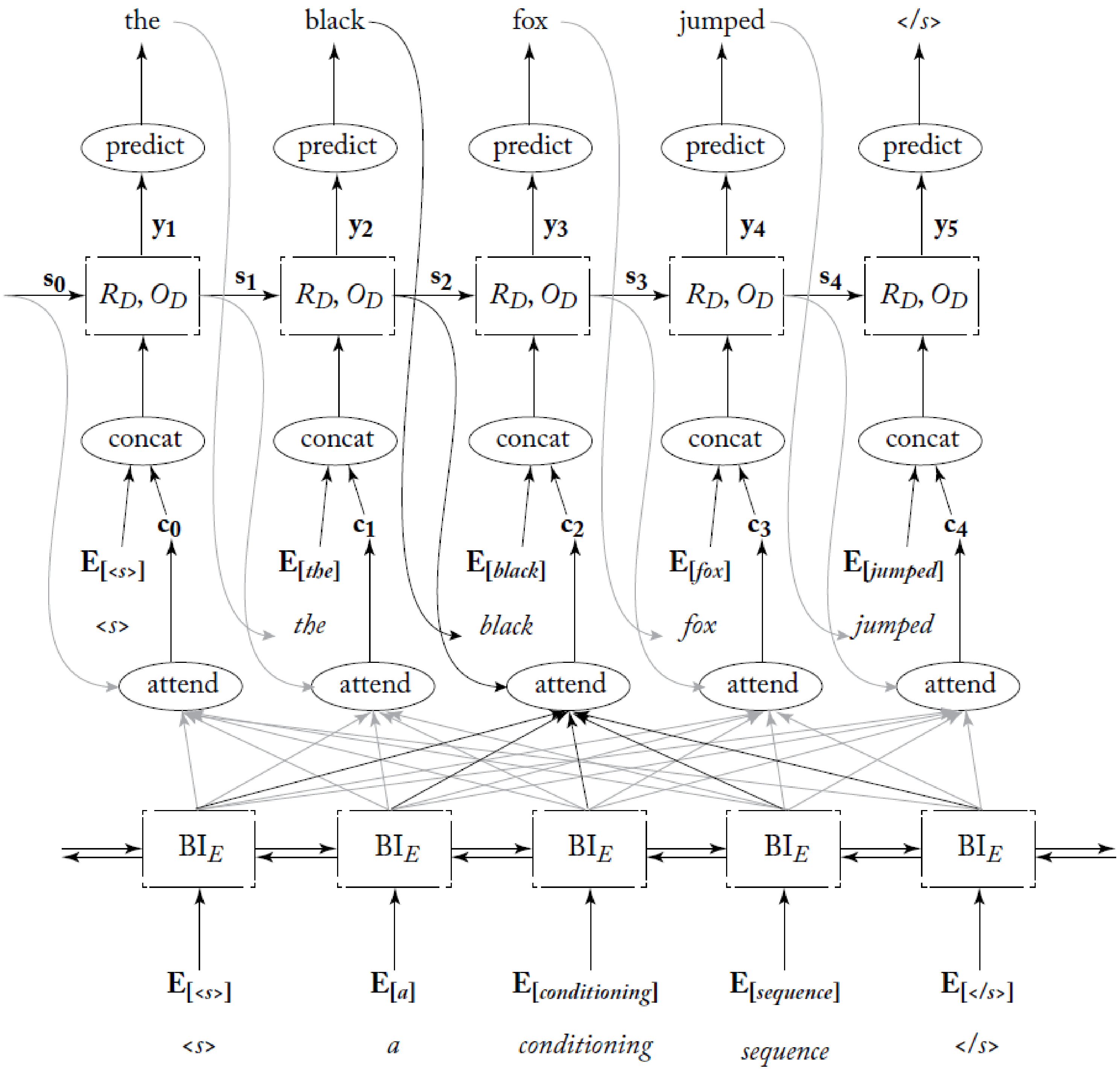
Conditioned Generation with Attention

- Conditioned Generation
  - Input is encoded into a single vector

$$\boldsymbol{c} = \textsc{Enc}(\boldsymbol{x}_{\mathbf{1}:n})$$

- Conditioned Generation with Attention
  - Input is encoded as a sequence of vectors

$$\boldsymbol{c}_{\mathbf{1}:n} = \textsc{Enc}(\boldsymbol{x}_{\mathbf{1}:n})$$



the
black
fox
jumped
</s>
predict
predict
predict
predict
predict
y1
y2
y3
y4
y5
s0
s1
s2
s3
s4
R_D, O_D
R_D, O_D
R_D, O_D
R_D, O_D
R_D, O_D
concat
concat
concat
concat
concat
c0
c1
c2
c3
c4
E[<s>]
E[the]
E[black]
E[fox]
E[jumped]
<s>
the
black
fox
jumped
attend
attend
attend
attend
attend
BI_E
BI_E
BI_E
BI_E
BI_E
E[<s>]
E[a]
E[conditioning]
E[sequence]
E[</s>]
<s>
a
conditioning
sequence
</s>

Attention Function

- Attention mechanism is *soft*
  - at each stage the decoder sees a weighted average of the vectors $c_{1:n}$

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

  - weights are produced by a feed-forward network which uses both the decoder state at time j and each of the vectors c_i

RNN without Attention

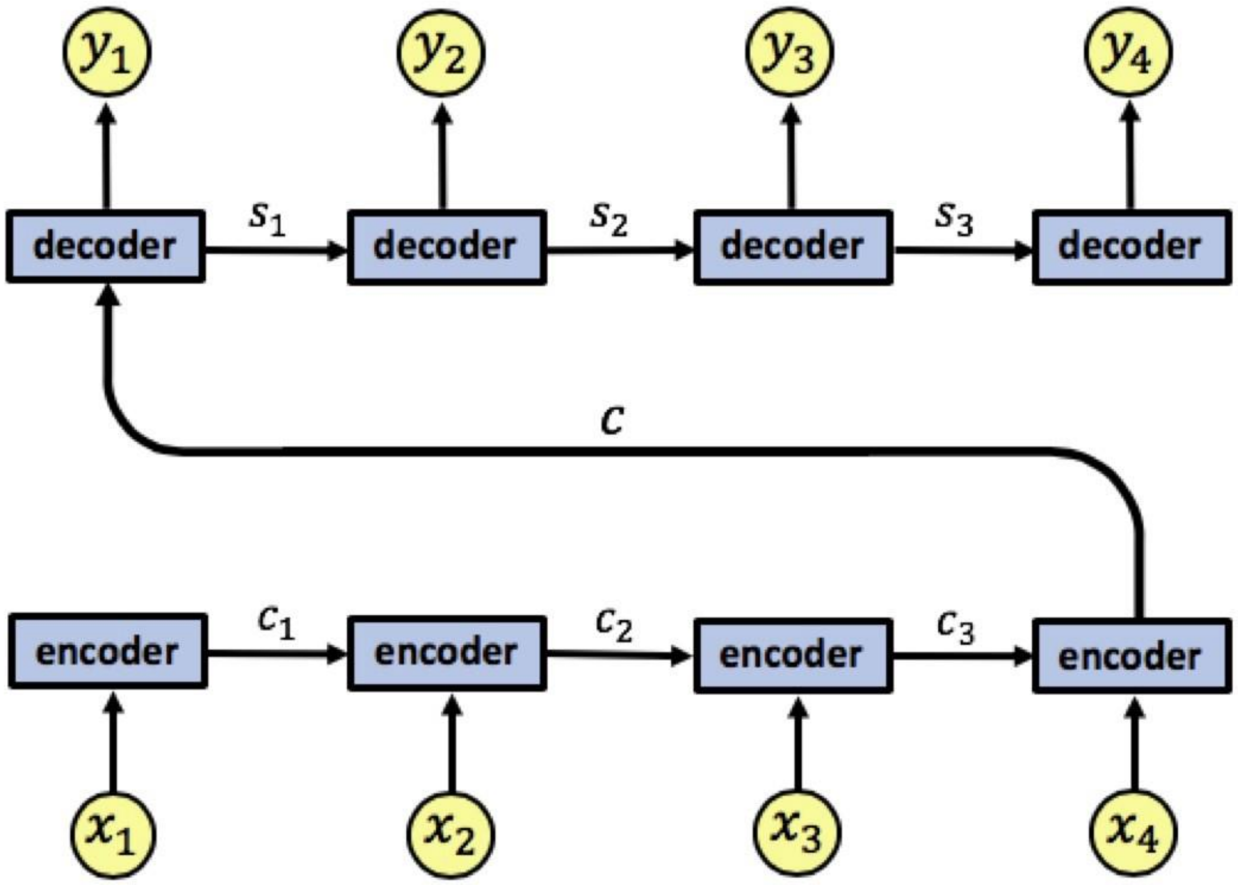


Image Source: https://medium.com/datadriveninvestor/attention-in-rnns-321fbcd64f05

RNN with Attention

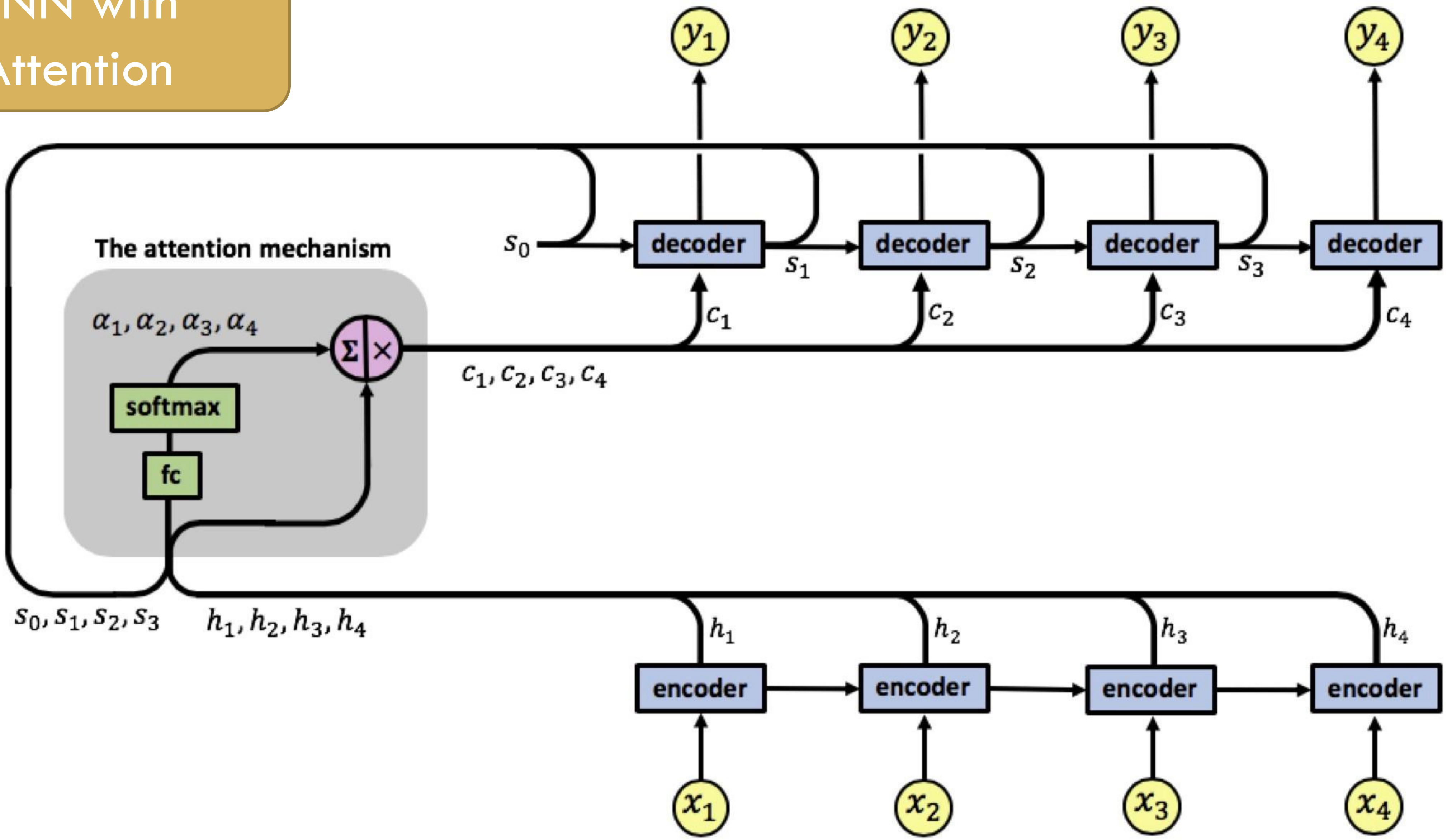


Image Source: https://medium.com/datadriveninvestor/attention-in-rnns-321fbcd64f05

RNN with Attention

$$e_{ij} = \mathrm{fc}\left(s_{i-1}, h_j\right)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{4} exp(e_{ik})}$$

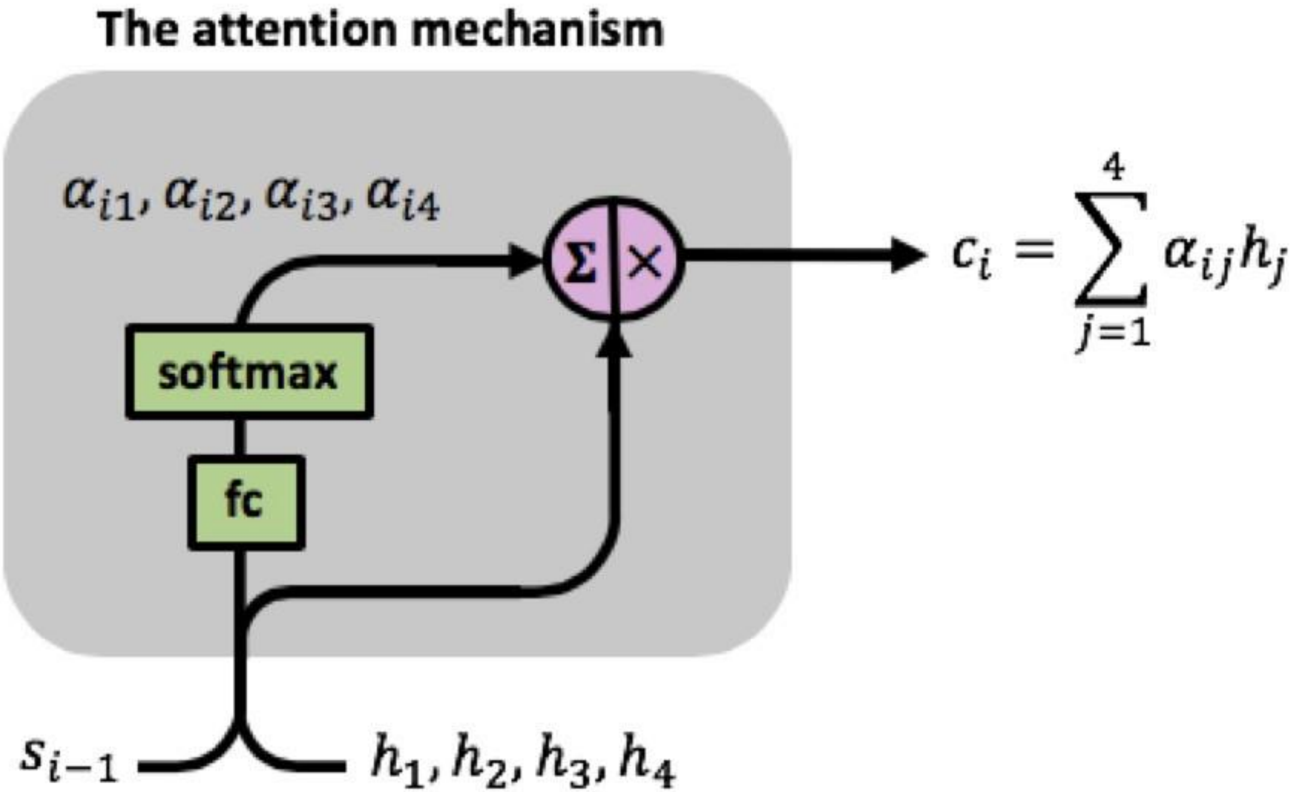


Image Source: https://medium.com/datadriveninvestor/attention-in-rnns-321fbcd64f05

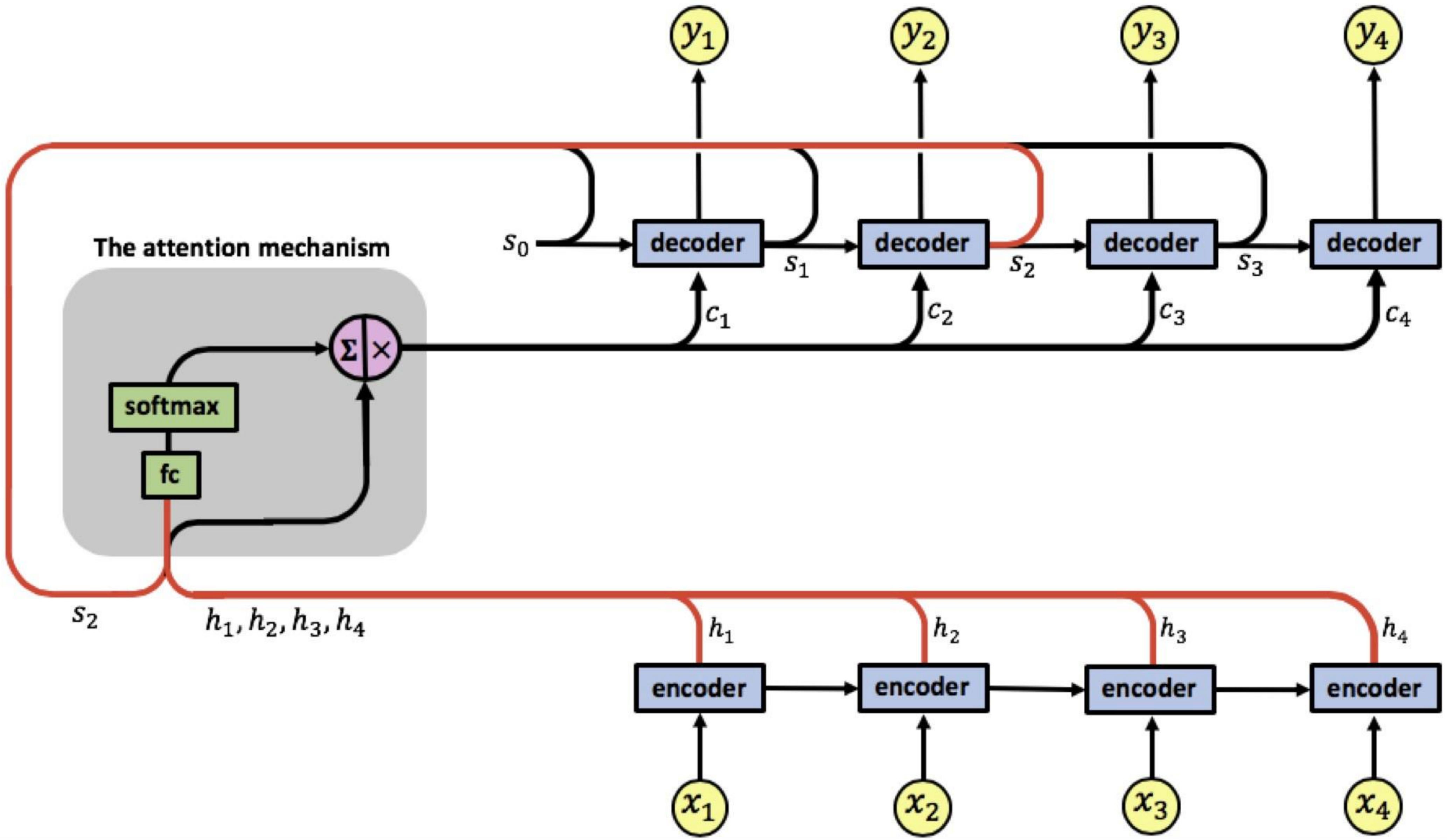


Image Source: https://medium.com/datadriveninvestor/attention-in-rnns-321fbcd64f05

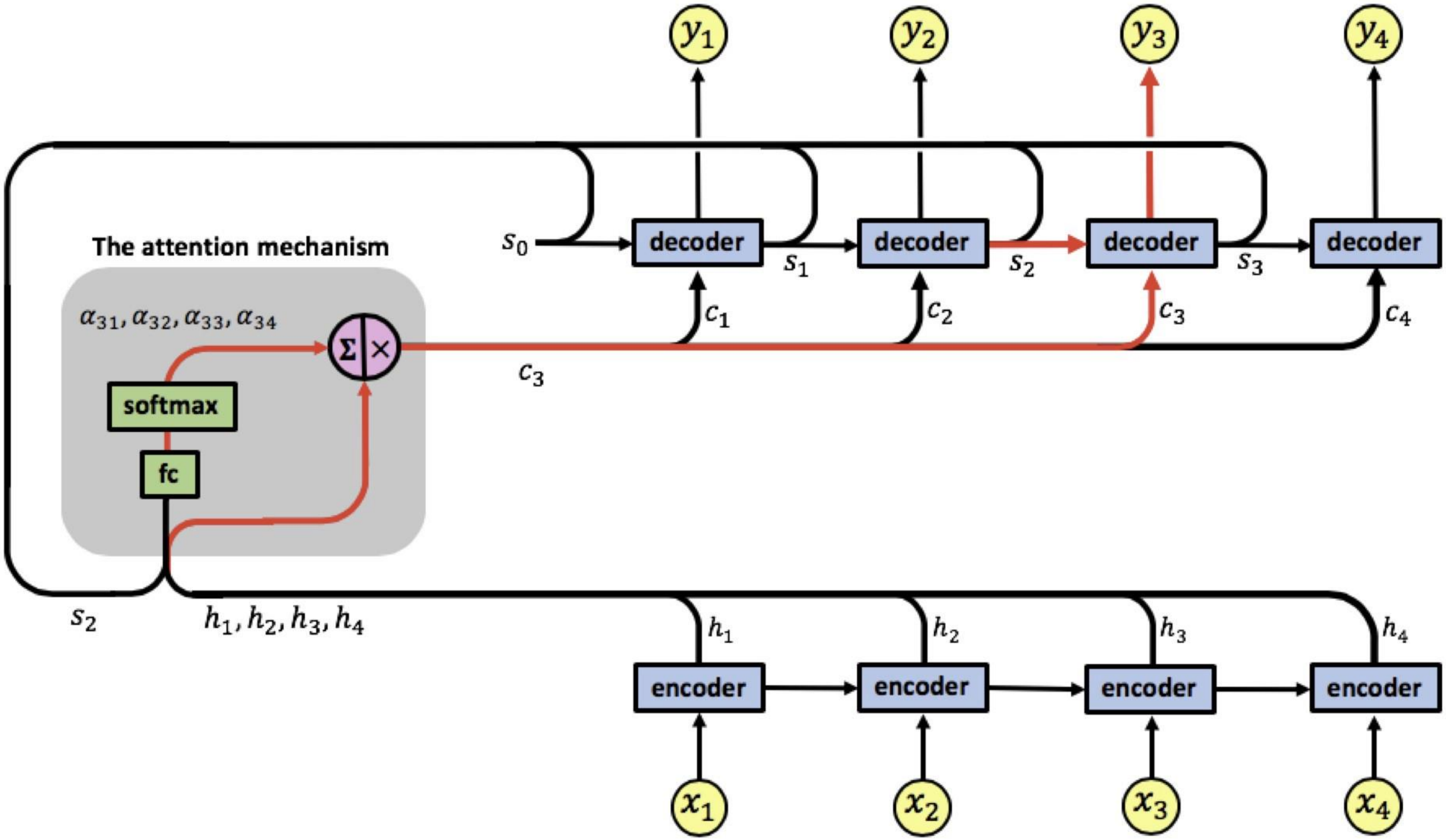


Image Source: https://medium.com/datadriveninvestor/attention-in-rnns-321fbcd64f05

$$e_{ij} = \mathrm{fc}(s_{i-1}, h_j)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{4} exp(e_{ik})}$$

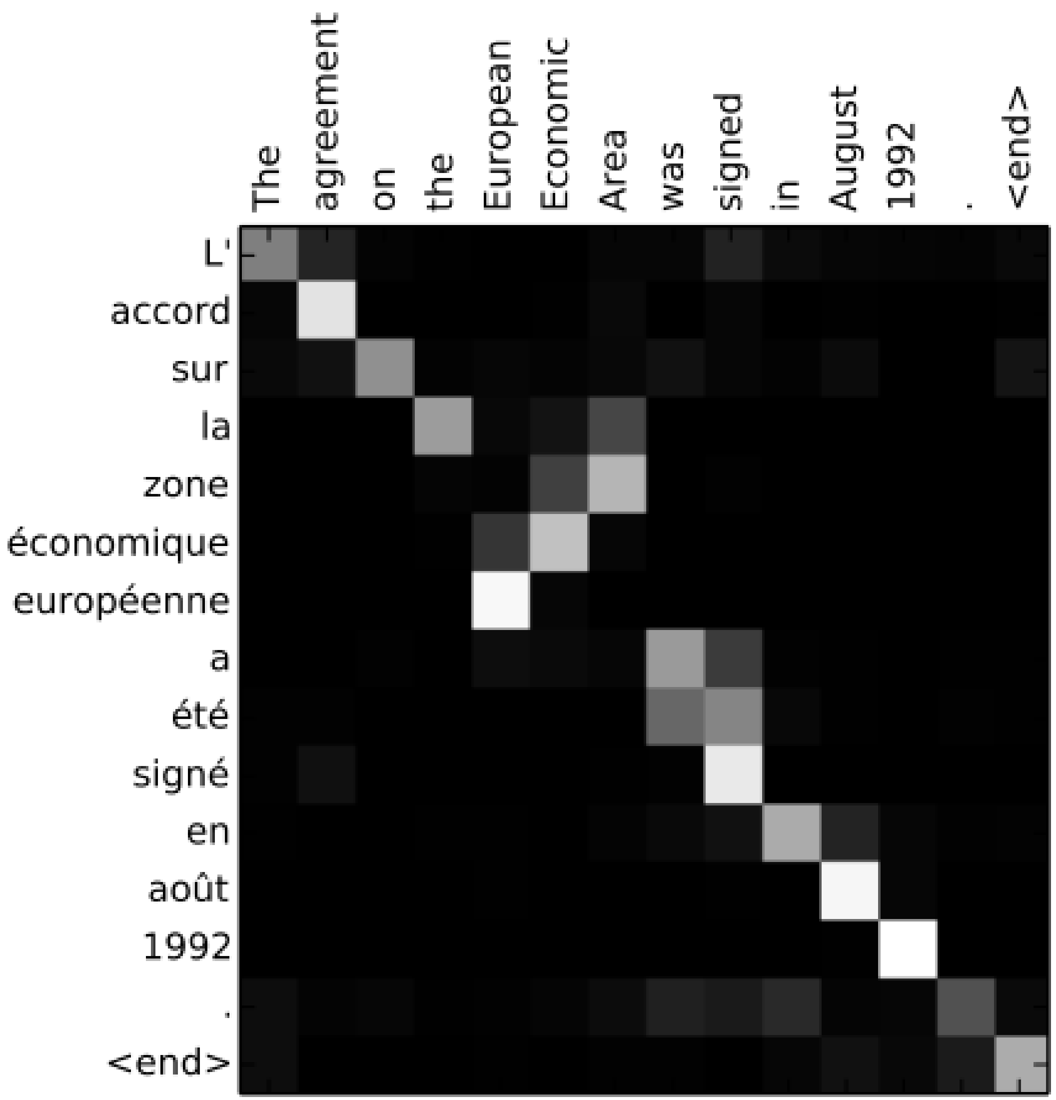
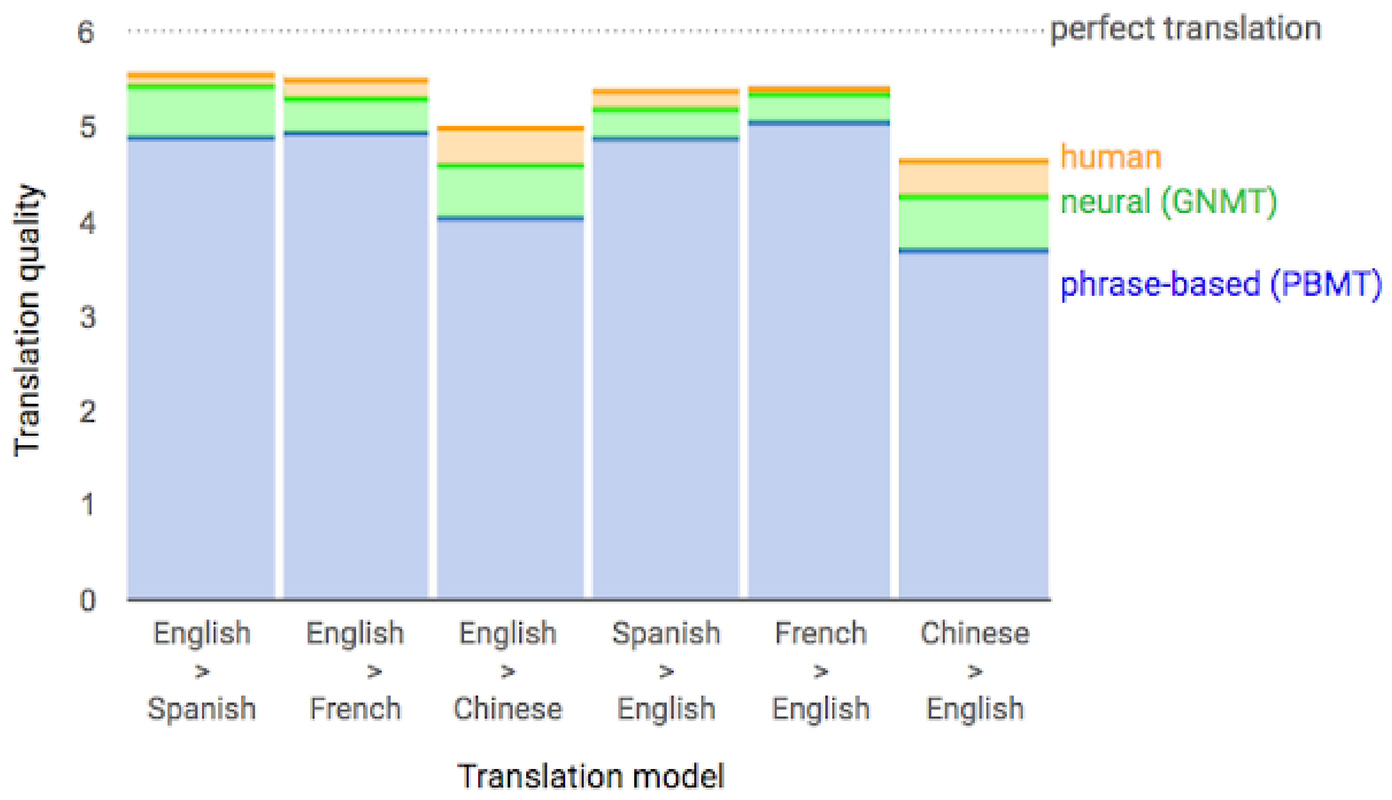


Image Source: Neural MT by Jointly Learning to Align and Translate



https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html

Any Questions?