Language Modeling
&
Sequence-to-Sequence Modeling

Anoop Kunchukuttan

Microsoft AI Core, Hyderabad

anoop.kunchukuttan@gmail.com

For CS772 at IIT Bombay, Sep 2025

Course Instructor: Prof. Pushpak Bhattacharyya

Language Modeling

Fundamental Task in NLP

The capital of Maharashtra is _____
The capital of ______ is Mumbai

The ability to predict words is a sign of language skill

In statistical NLP, such a capability is at the core of many NLP applications

N-gram Language Models (textbook chapter)

Predicting the next token well means that you understand the underlying reality that led to the creation of that token.

People have thoughts and they have feelings and they have ideas and they do things in certain ways, all of those could be deduced from next token prediction.

I challenge the claim that next token prediction cannot surpass human performance

Prediction is compression … to predict the data well, to compress it well, you need to understand more and more about the world that produced the data.

Ilya Sutskever on Language Modeling

Language Modeling Task

Given a sequence of words ➔ $(x_1 \ x_2 \ x_3 \ x_4 \ldots\ldots x_i)$

Compute the probability distribution of the next word ➔ $P(x_{i+1} | x_1 \ x_2 \ x_3 \ x_4 \ldots\ldots x_i)$

The capital of Maharashtra is _____

$P(Mumbai|X)$

$P(Bihar|X)$

$P(Chennai|X)$

$P(market|X)$

LM can assign probability to a sentence

N-gram based Language Modeling

$$P(w \mid The\ capital\ of)$$

$$=$$

$$\frac{count(The\ capital\ of\ w)}{count(The\ capital\ of)}$$

$$P(x_1\ x_2\ x_3\ x_4 \ldots\ldots x_i, x_{i+1})$$
$$= P(x_{i+1} \mid x_1\ x_2\ x_3\ x_4 \ldots\ldots x_i) \times P(x_1\ x_2\ x_3\ x_4 \ldots\ldots x_i, x_{i+1})$$

$$P(x_{i+1} \mid x_1\ x_2\ x_3\ x_4 \ldots\ldots x_i) = P(x_{i+1} \mid x_i)$$ (bigram LM)

$$P(x_{i+1} \mid x_1\ x_2\ x_3\ x_4 \ldots\ldots x_i) = P(x_{i+1} \mid x_i, x_{i-1})$$ (trigram LM)

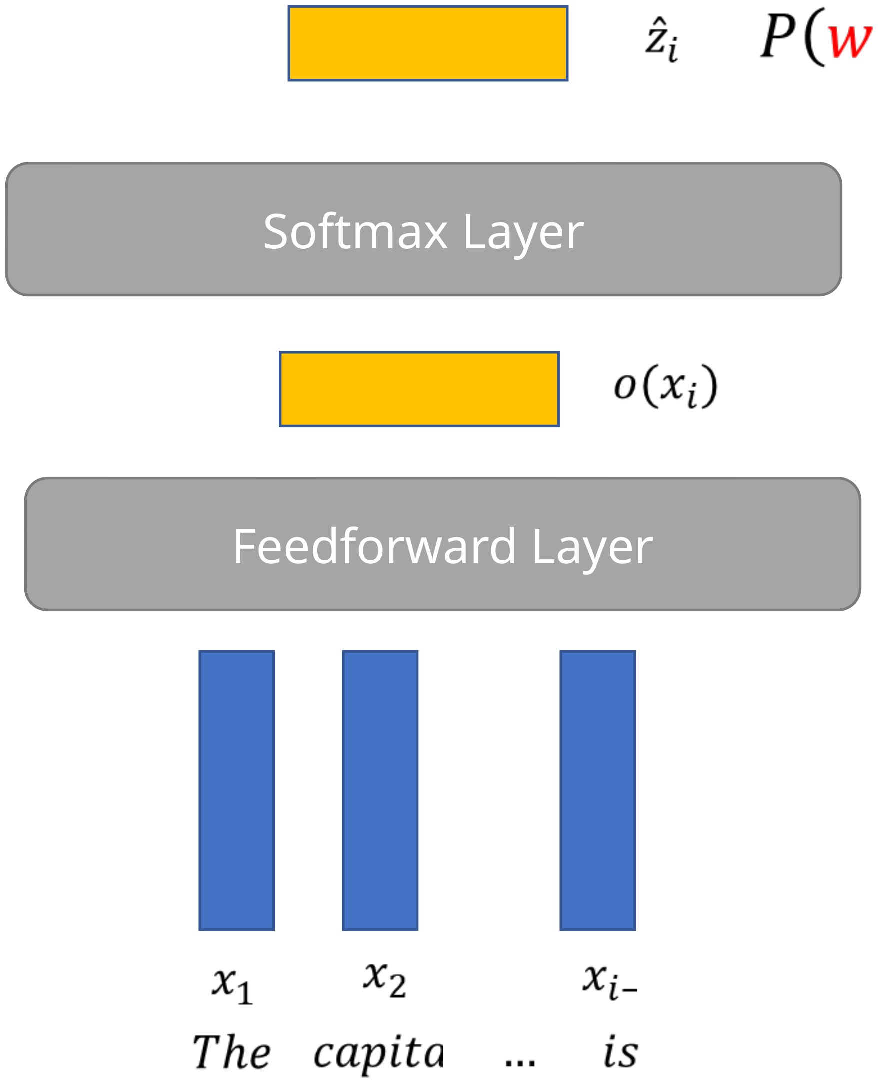
Sparsity issues
Storage issues
Limited context

- *Unbounded context not possible* ➔ *Markov assumption & Chain Rule*
- *Estimate based on counting* ➔ *model is a large lookup table of probabilities*
- *Large number of n-grams*
  - *Smoothing, interpolation, etc. to approximate probabilities for rare events*

Smoothing techniques: http://nrs.harvard.edu/urn-3:HUL.InstRepos:25104739

Feedforward NN based LM

Fixed limited context



$$P(w|\,The\ capital\ of\ Maharashtra\ is)$$

$$h(x_i) = \sigma(W^x x_i + b_2)$$

$$o(x_i) = W^o h(x_i) + b_1)$$

$$\hat{z}_i = softmax(o(x_i))$$

$\hat{z}_i$ is the probability distribution of the next word

Word embeddings address sparsity issues

Prob distribution implemented as NN
→ makes the model size reasonably compact

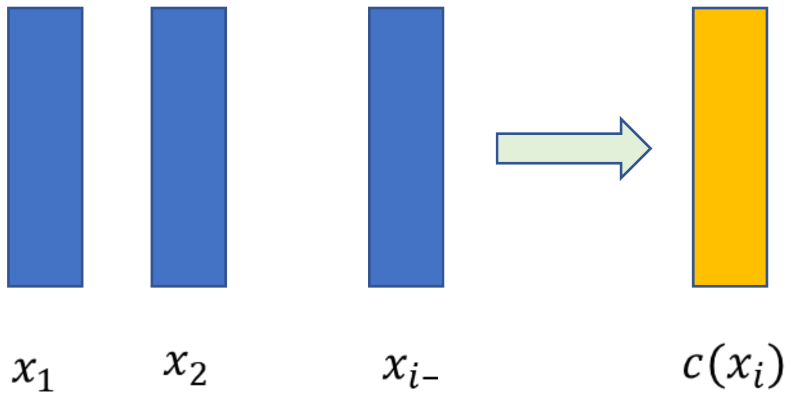
https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

What do we want?

- LM learning should not run into sparsity issues as context window increases
  - Distributed Representations can help address the problem
  - Low-dimensional representation of context
- Ability to capture long-distance dependency effects
  - Unbounded context

Recurrent Neural Networks: 2 Key Ideas

1. Summarize context information into a single vector



$$c(x_i) = F(x_1, x_2, \dots, x_{i-1})$$

$$P(x_i | c(x_i))$$

Nature of $P(.)$

n-gram LM: look-up table

FF LM: $c(x_i) = G(x_{i-1}, x_{i-2})$ (trigram LM)

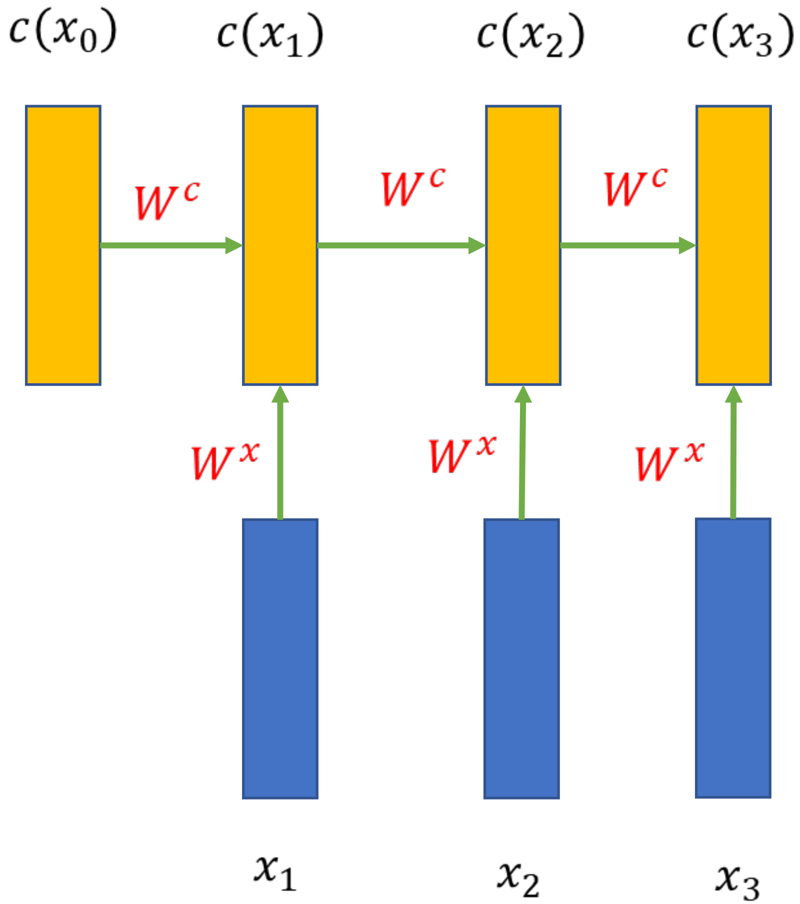
RNN LM: $c(x_i) = F(x_1, x_2, \dots, x_{i-1})$ (unbounded context)

Function G requires all context inputs at once

How does RNN address this problem?

2 Key Ideas

2. Recursively construct the context



$$c(x_i) = F(c(x_{i-1}), x_i)$$

We just need two inputs to construct the context vector:

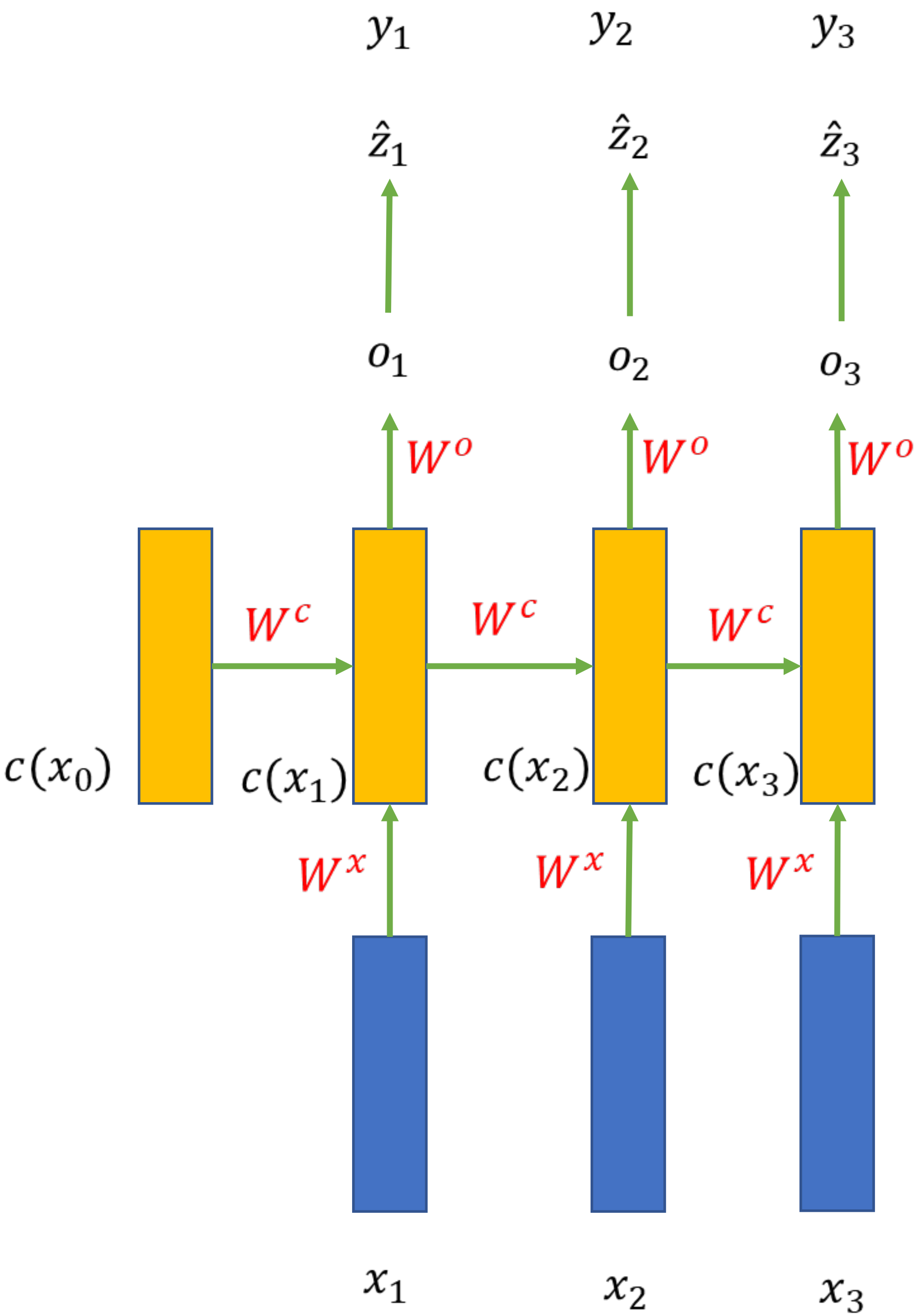
- Context vector of previous timestep
- Current input

The context vector ➔ state/hidden state/contextual representation

$F(.)$ can be implemented as

$$c(x_i) = \sigma(W^c c(x_{i-1}) + W^x x_i + b_1)$$

Like a feed-forward network



Generate output give the current input and state/context

$$o(x_i) = W^o c(x_i) + b_2$$

We are generally interested in categorical outputs

$$\hat{z}_i = softmax(o(x_i)) = P(y_i | ctx(x_i))$$

$$\widehat{z_i^w} = P(y_i = w | ctx(x_i))$$

The same parameters are used at each time-step

Model size does not depend on sequence length

Sequence Labelling Task

Input Sequence: $(x_1\ \ x_2\ \ x_3\ \ x_4 \ldots\ldots x_i \ldots\ldots x_N)$

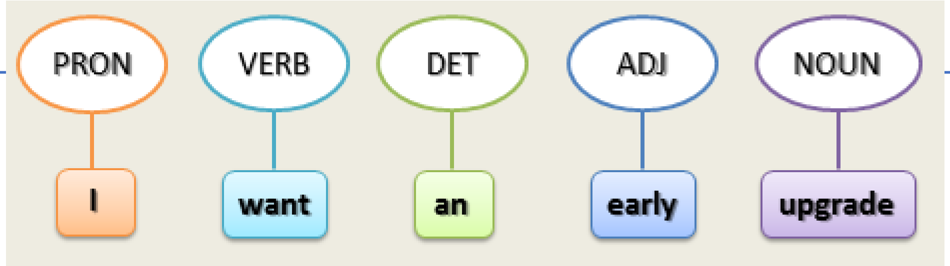
Output Sequence: $(y_1\ \ y_2\ \ y_3\ \ y_4 \ldots\ldots y_i \ldots\ldots y_N)$

Input and output sequences have the same length

Variable length input

Output contains categorical labels

Output at any time-step typically depends on neighbouring output labels and input elements



Part-of-speech tagging

Recurrent Neural Network is a powerful model to learn sequence labelling tasks

How do we model language modeling as a sequence labeling task?

The	capital	of	Maharashtra	is	Mumbai	<EOS>
<BOS>	The	capital	of	Maharashtra	is	Mumbai

The output sequence is one-time step ahead of the input sequence

Training Language Models

Input: large monolingual corpus

- Each example is a tokenized sentence (sequence of words)
- At each time step, predict the distribution of the next word given all previous words
- Loss Function:
  - Minimize cross-entropy between actual distribution and predicted distribution
  - Equivalently, maximize the likelihood

At a single time-step: $J_i(\theta) = CE(z_i, \hat{z}_i) = -\sum_{w \in V} z_i^w \log \widehat{z_i^w} = -\log \widehat{z_i^L}$

where $y_i = L$

Average over time steps for example n: $J^n(\theta) = \frac{1}{T}\sum_{i=1}^{T} J_i(\theta)$

Average over entire corpus: $J(\theta) = \frac{1}{N}\sum_{k=1}^{N} J^n(\theta)$

How do we learn model parameters? More on that later!

How do we evaluate quality of language models?

Evaluate the ability to predict the next word given a context

Evaluate the probability of a testset of sentences

Standard testsets exist for evaluating language models: Penn Treebank, Billion Word Corpus, WikiText

Language Model Perplexity

Perplexity: $\exp\left(J(\theta)\right)$

$J(\theta)$ is cross-entropy on the test set

Cross-entropy is measure of difference between actual and predicted distribution

Lower perplexity and cross-entropy is better

Training objective matches evaluation metric *n-gram*

RNN variants

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

RNN models outperform n-gram models

A special kind of RNN network – LSTM does even later ➔ we will see that soon

Generating text from a language model

We know the probability distribution → $P(x_{i+1}|x_1\ \ x_2\ \ x_3\ \ x_4 \ldots\ldots x_i)$

At each step, sample a word as per the distribution

$P(Mumbai|X) = 0.8$

The capital of Maharashtra is _____

$P(Bihar|X) = 0.1$

$P(Chennai|X) = 0.075$

$P(market|X) = 0.025$

Examples of generated text

```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

By a character level RNN-LM trained on Shakespeare’s works

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

It is a curious fact that the last remaining form of social life in which the people of London are still interested is Twitter. I was struck with this curious fact when I went on one of my periodical holidays to the sea-side, and found the whole place twittering like a starling-cage. I called it an anomaly, and it is.

I spoke to the sexton, whose cottage, like all sexton's cottages, is full of antiquities and interesting relics of former centuries. I said to him, "My dear sexton, what does all this twittering mean?" And he replied, "Why, sir, of course it means Twitter."
"Ah!" I said, "I know about that. But what is Twitter?"

"It is a system of short and pithy sentences strung together in groups, for the purpose of conveying useful information to the initiated, and entertainment and the exercise of wits to the initiated, and entertainment and the exercise of wits to the rest of us."

"Very interesting," I said. "Has it a name?"
"It has," he said; "it is called Twitter.
"Yes," I said, "I know that, but what is it?"
"It is a system of information," he said.
"Oh, yes," I replied; "but what is it?"

Generated by GPT-3 after being prompted by the first word

Note: GPT-3 is not an RNN-LM

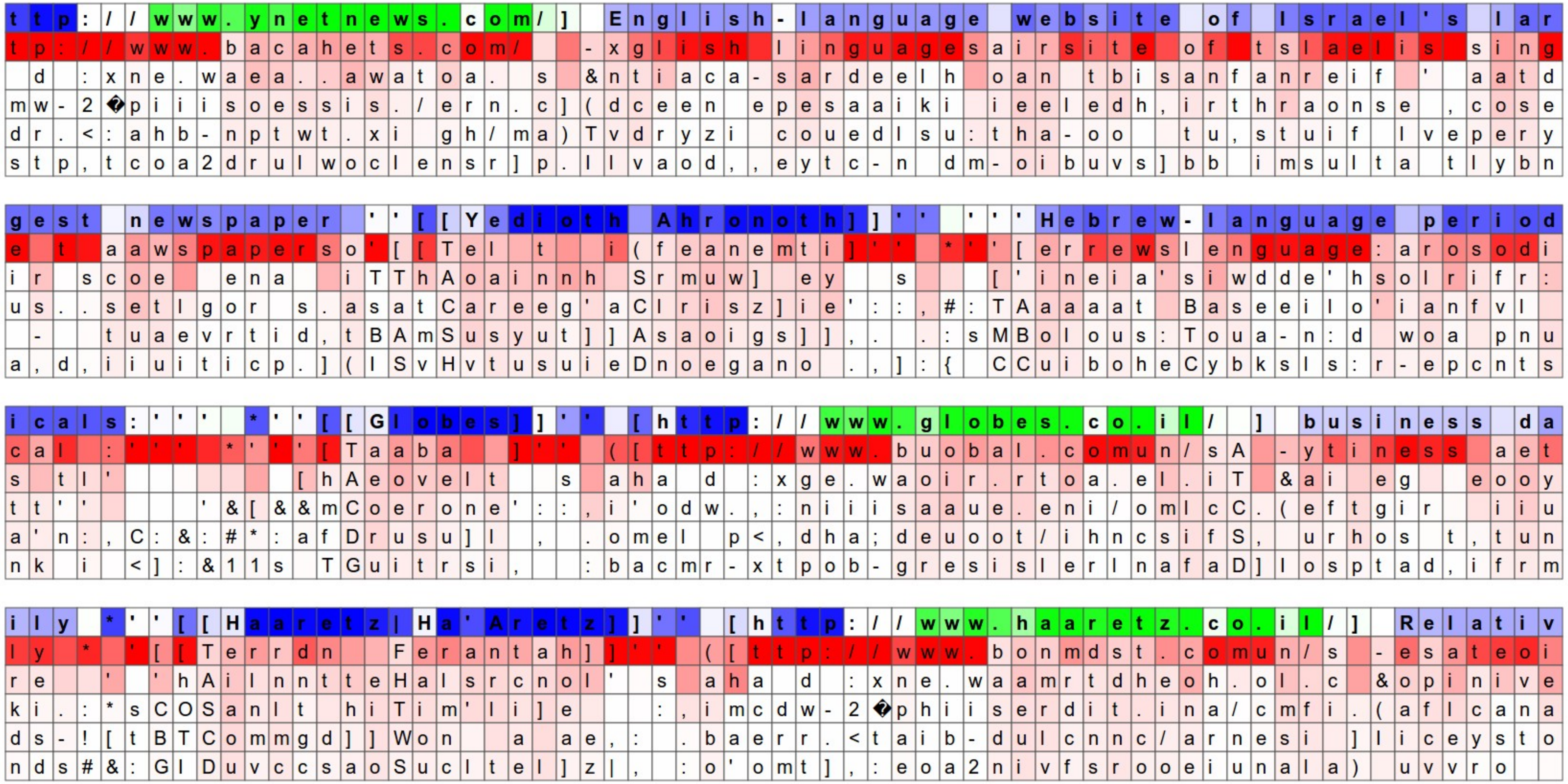
Neural networks have enabled generation of very fluent text

Led to a revolution in natural language generation

Enabled huge advances in natural language understanding

https://www.technologyreview.com/2020/07/20/1005454/openai-machine-learning-language-generator-gpt-3-nlp/

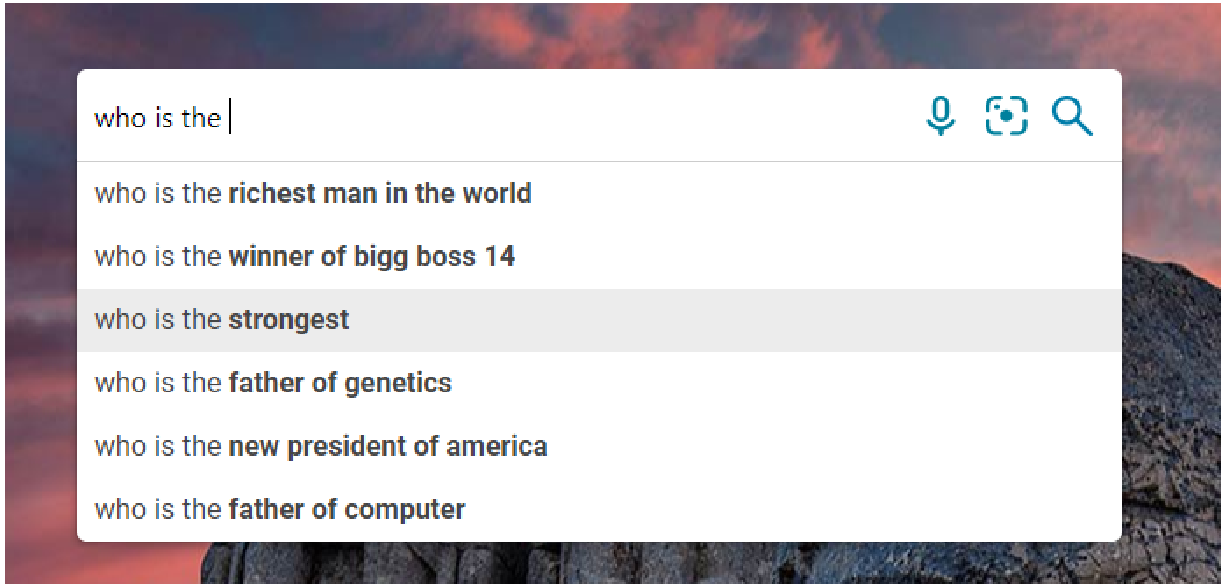
Visualizations

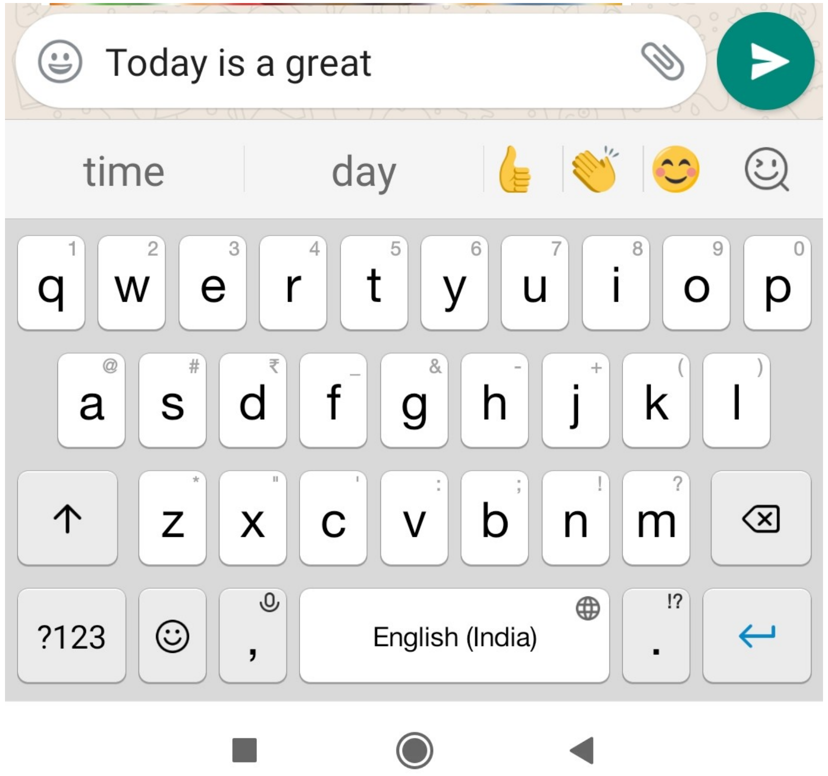


Activation of a neuron in the LSTM hidden layer indicate it fires for URLs

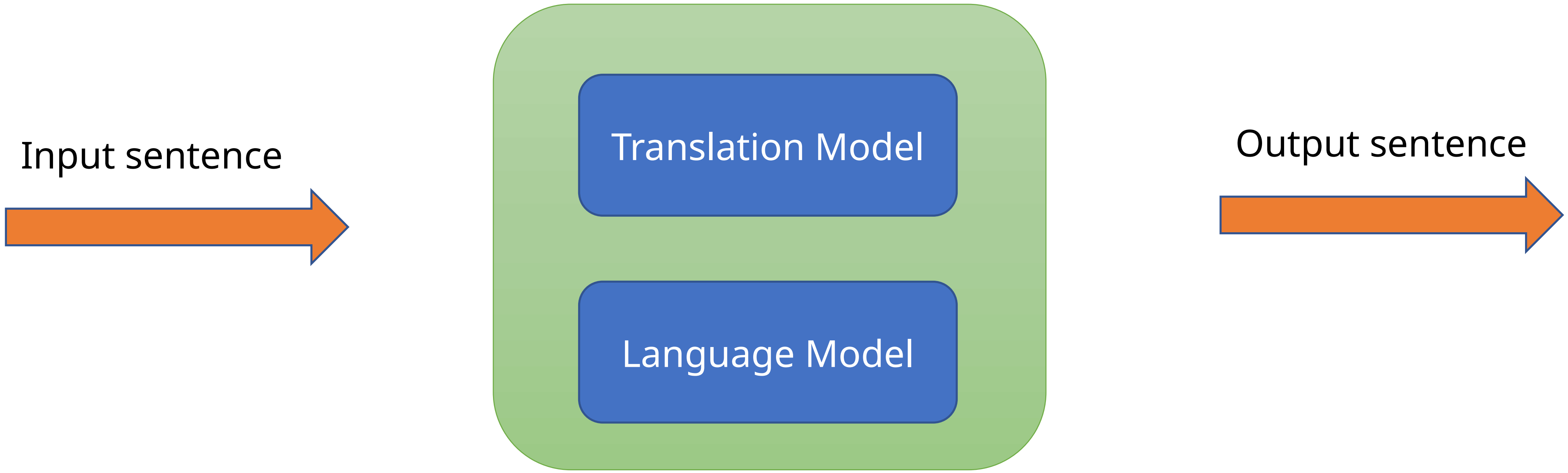
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Language models for auto-suggest





Language models to score outputs from generation systems



Automatic Speech recognition systems combine language models with acoustic models

Transliteration models combine character mapping models with character level language model

Natural Language Understanding

Classification

Sequence Labelling tasks

Sequence Labelling Tasks

Named Entity Recognition

```
[PERS Pierre Vinken] , 61 years old , will join
[ORG IBM] 's board as a nonexecutive director
[DATE Nov. 2] .
```

Shallow Parsing

```
[NP Pierre Vinken] , [NP 61 years] old , [VP will join]
[NP IBM] 's [NP board] [PP as] [NP a nonexecutive
director] [NP Nov. 2] .
```

NP Chunking

```
[NP Pierre Vinken] , [NP 61 years] old , will join
[NP IBM] 's [NP board] as [NP a nonexecutive director]
[NP Nov. 2] .
```

https://courses.engr.illinois.edu/cs447/fa2018/Slides/Lecture07.pdf

The BIO encoding

We define three new tags:

- **B-NP**: beginning of a noun phrase chunk
- **I-NP**: inside of a noun phrase chunk
- **O**: outside of a noun phrase chunk

```
[NP Pierre Vinken] , [NP 61 years] old , will join
[NP IBM] ‘s [NP board] as [NP a nonexecutive director]
[NP Nov. 2] .
```

```
Pierre_B-NP Vinken_I-NP ,_O 61_B-NP years_I-NP
old_O ,_O will_O join_O IBM_B-NP ‘s_O board_B-NP as_O
a_B-NP nonexecutive_I-NP director_I-NP Nov._B-NP
29_I-NP ._O
```

https://courses.engr.illinois.edu/cs447/fa2018/Slides/Lecture07.pdf

Inference in sequence labelling tasks

We know the probability distribution ➔ $P(y_i|x_1\ x_2\ x_3\ x_4 \ldots\ldots x_i)$

At each step, sample a word as per the distribution

The capital of Maharashtra is Mumbai

$$P(LOC|Mumbai) = 0.65$$

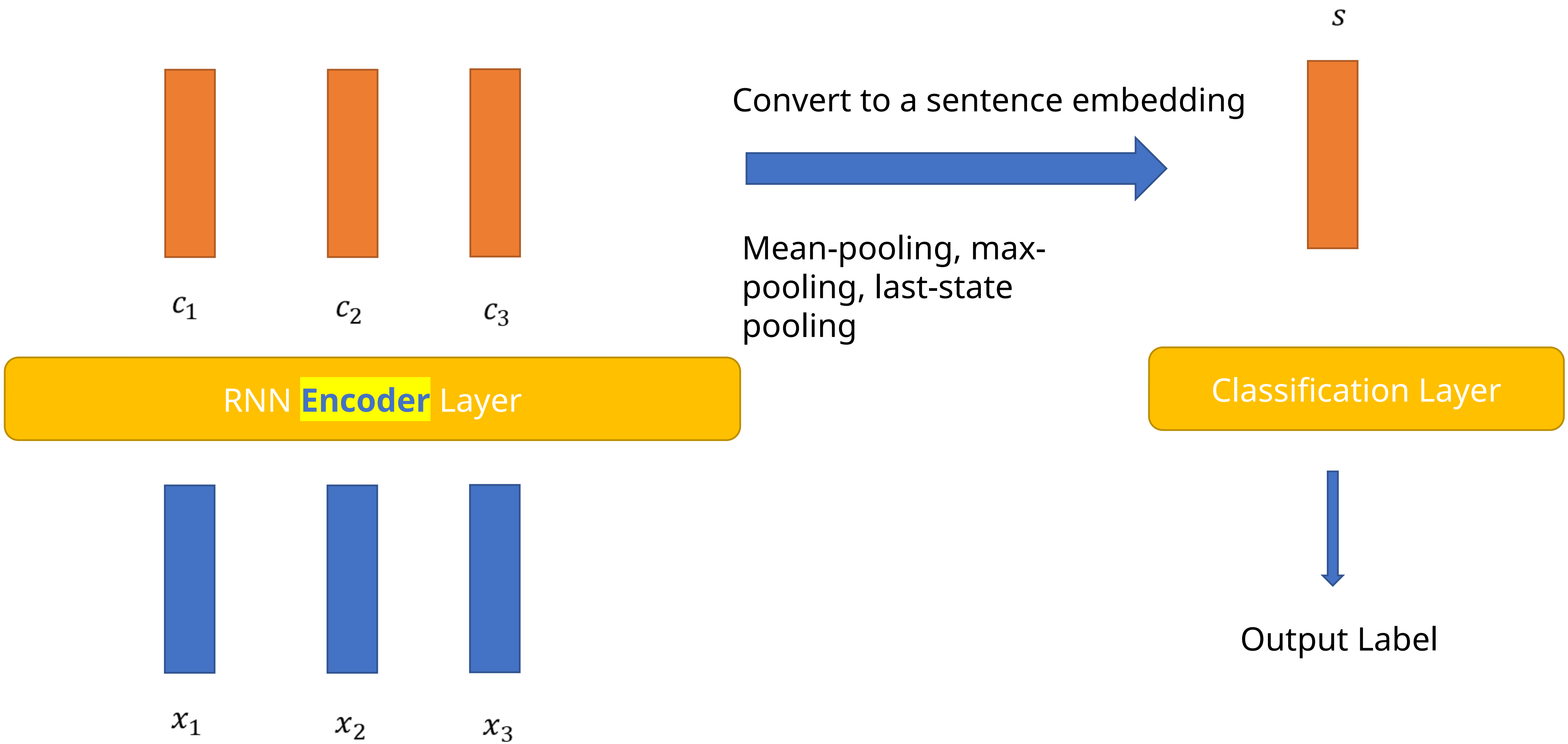
$$P(ORG|Mumbai) = 0.3$$

$$P(PER|Mumbai) = 0.05$$

Greedy decoding or beam search decoding ➔ more on this later

Classification Tasks

Sentiment classification, hate speech detection, etc.



Language Model Pre-training

Do we have to train the RNN parameters for every task like sentiment analysis, POS tagging?

➔ *we may not have enough training data for a task*

➔ *training data for a task will not capture all linguistic knowledge*

Train a large language model on lots of text data ➔ pre-trained encoder module

For a particular task ➔ add classification heads (and optional layers) on top

Finetune the network for the task

Many models like: ELMO, ULMFit

Sequence to Sequence Tasks

More about these later

Input and output sequences of different lengths

Machine Translation, Machine Transliteration, Summarization, etc.

Can be solved with conditional RNNs

Back to Training Recurrent Neural Models

Brief sketch of Backpropagation Through Time (BPTT)

Training Objective revisited

At a single time-step i: $J_i(\theta) =- -\log \widehat{z_i^L}$ where $y_i = L$

Average over time steps for example : $J(\theta) = \sum_{i=1}^{T} J_i(\theta)$

Gradient Computation $$\frac{\partial J(\theta)}{\partial \theta} = \sum_{i=1}^{T} \frac{\partial J_i(\theta)}{\partial \theta}$$

Model Parameters: $\theta = (W^x, W^c, W^o, b_1, b_2)$

$$\frac{\partial J_i(\theta)}{\partial W^o}$$

$J_i(\theta)$ is a direct function of W^o, so it is easy to compute

Let's focus on gradients w.r.t to W^c

$$\frac{\partial J_i(\theta)}{\partial W^c} = \frac{\partial J_i(\theta)}{\partial c_i} \frac{\partial c_i}{\partial W^c}$$

c_i is a function of W^c and c_{i-1}

In turn, c_{i-1} is a function of W^c and c_{i-2} and so on ....

$$\frac{\partial c_i}{\partial W^c} = \frac{\partial^+ c_i}{\partial W^c} + \frac{\partial c_i}{\partial c_{i-1}} \frac{\partial c_{i-1}}{\partial W^c}$$

Multivariate chain rule

Theorem 7.35. Multivariate Chain Rule. *Suppose that $z = f(x, y)$, f is differentiable, $x = g(t)$, and $y = h(t)$. Assuming that the relevant derivatives exist,*

$$\frac{dz}{dt} = \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt}.$$

$c_i = f(W^c, c_{i-1})$ Plug in $z = c_i, x = W^c, y = c_{i-1}$

$$\frac{\partial c_i}{\partial W^c} = \frac{\partial^+ c_i}{\partial W^c} + \frac{\partial c_i}{\partial c_{i-1}} \frac{\partial c_{i-1}}{\partial W^c}$$

$\frac{\partial^+ c_i}{\partial W^c}$ ➔ explicit derivative term computed assuming c_{i-1} to be constant

$$\frac{\partial c_i}{\partial W^c} = \frac{\partial^+ c_i}{\partial W^c} + \frac{\partial c_i}{\partial c_{i-1}} \left[\frac{\partial^+ c_{i-1}}{\partial W^c} + \frac{\partial c_{i-1}}{\partial c_{i-2}} \frac{\partial c_{i-2}}{\partial W^c} \right]$$

Because of the recurrence, the gradient of a hidden state at time i depends on gradients of all hidden states before timestep i

$$\frac{\partial c_i}{\partial W^c} = \sum_{k=1}^{i} \frac{\partial c_i}{\partial c_k} \frac{\partial^+ c_k}{\partial W^c}$$

$$\frac{\partial J_i(\theta)}{\partial W^c} = \frac{\partial J_i(\theta)}{\partial c_i} \sum_{k=1}^{i} \frac{\partial c_i}{\partial c_k} \frac{\partial^+ c_k}{\partial W^c}$$

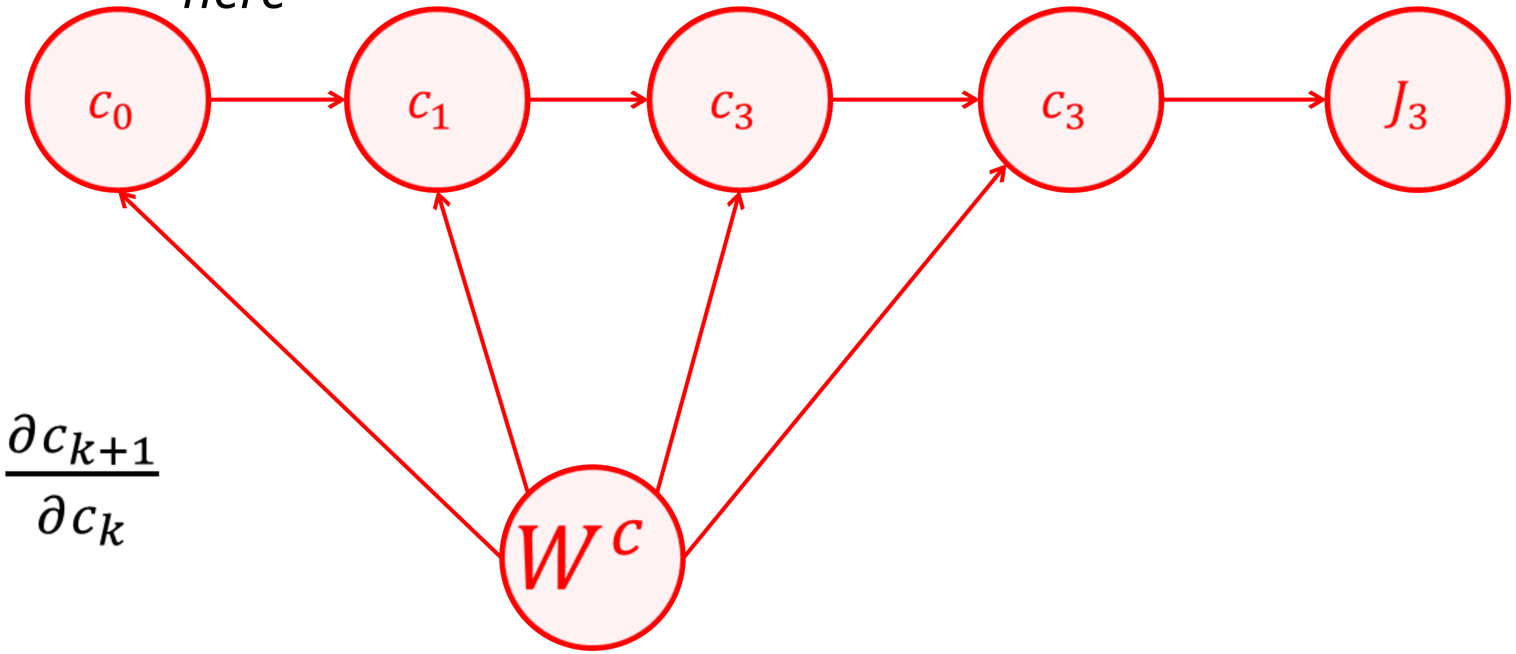
Backpropagation needs to go over all previous time-steps
➔ backpropagation through time (BPTT)

See https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture14.pdf for detailed explanation

Vanishing and exploding gradient

Let's look at this particular component of the gradient computation ➔ there is a problem here

$$\frac{\partial c_i}{\partial W^c} = \sum_{k=1}^{i-1} \frac{\partial c_i}{\partial c_k} \frac{\partial^+ c_k}{\partial W^c}$$



$$\frac{\partial c_i}{\partial c_k} = \frac{\partial c_i}{\partial c_{i-1}} \frac{\partial c_{i-1}}{\partial c_{i-2}} \frac{\partial c_{i-2}}{\partial c_{i-3}} \cdots \frac{\partial c_{k+1}}{\partial c_k}$$

Each Path from W^c to J constitutes a term in the summation,
Each edge a path constitutes one element in the product term

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture14.pdf

$$\frac{\partial c_i}{\partial c_k} = \frac{\partial c_i}{\partial c_{i-1}} \frac{\partial c_{i-1}}{\partial c_{i-2}} \frac{\partial c_{i-2}}{\partial c_{i-3}} \cdots \frac{\partial c_{k+1}}{\partial c_k}$$

We can show some bounds on this quantity

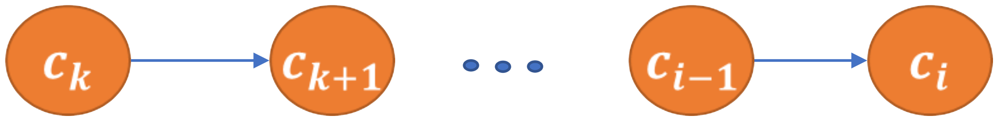
$$\left\| \frac{\partial c_i}{\partial c_{i-1}} \right\| \leq \gamma\lambda \quad \Longrightarrow \quad \left\| \frac{\partial c_i}{\partial c_k} \right\| \leq (\gamma\lambda)^{i-k}$$

$\gamma\lambda < 1$ ➔ vanishing gradient
$\gamma\lambda > 1$ ➔ exploding gradient

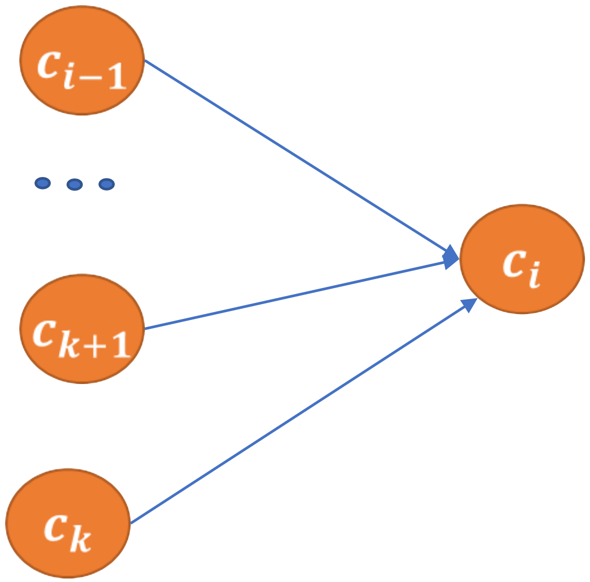
Vanishing Gradient Problem ➔ Intuition

- The product of these quantities can become small if the some of these quantities are small
- Gradient signal can become small for positions that are farther away from a position under consideration ➔ can't learn long term dependency
- Truncated BPTT ➔ restrict the product to fewer terms
- *Exploding Gradient:* Gradient can also become large ➔ clip the gradient before

Another way to address the vanishing gradient problem



c_k can only indirectly impact c_i ➔ what if it could have a direct impact?



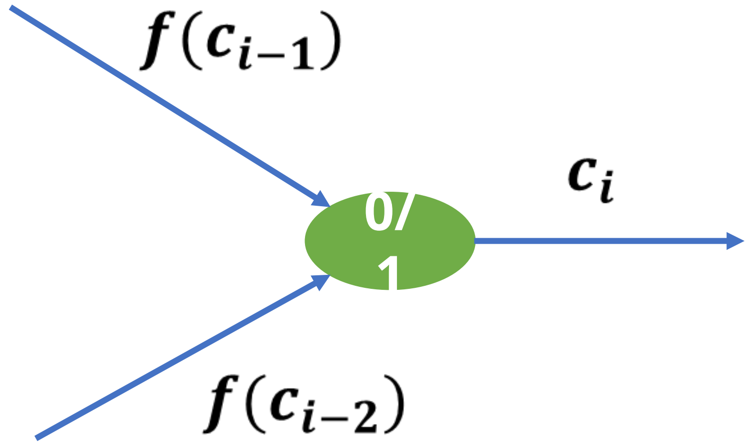
Looks a lot like the feed-forward solution

Cannot handle unbounded context

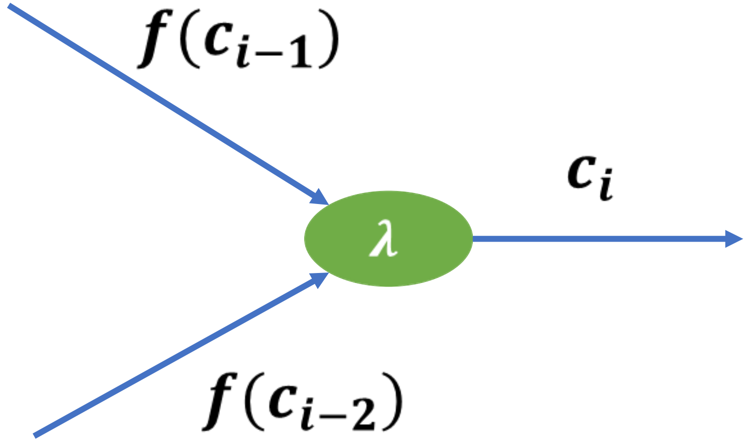
Not a good solution

Selecting hidden states that affect current hidden state

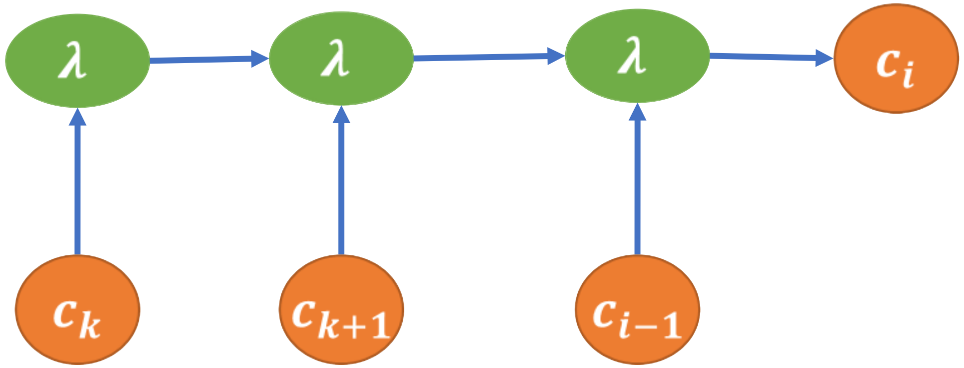
What if there was a switch to select which previous hidden state should impact current hidden state?



Better still, we had a soft-switch



If the switches are set correctly, then distant hidden states can have a major impact on current state



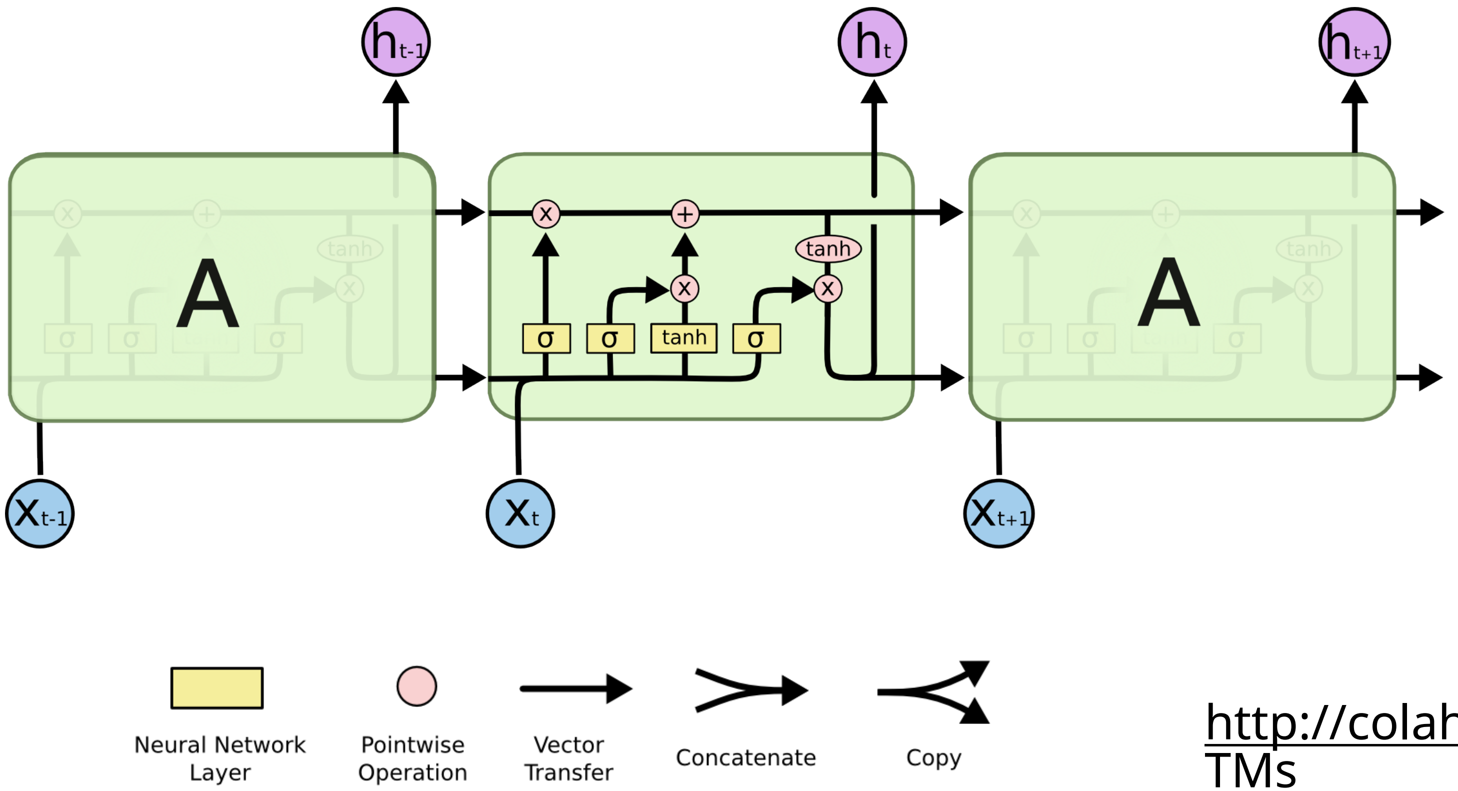
The switches should be set based on the data flowing in.

Switches act a memory control mechanism to decide what affects the output

Long Short-Term Memory (LSTM)

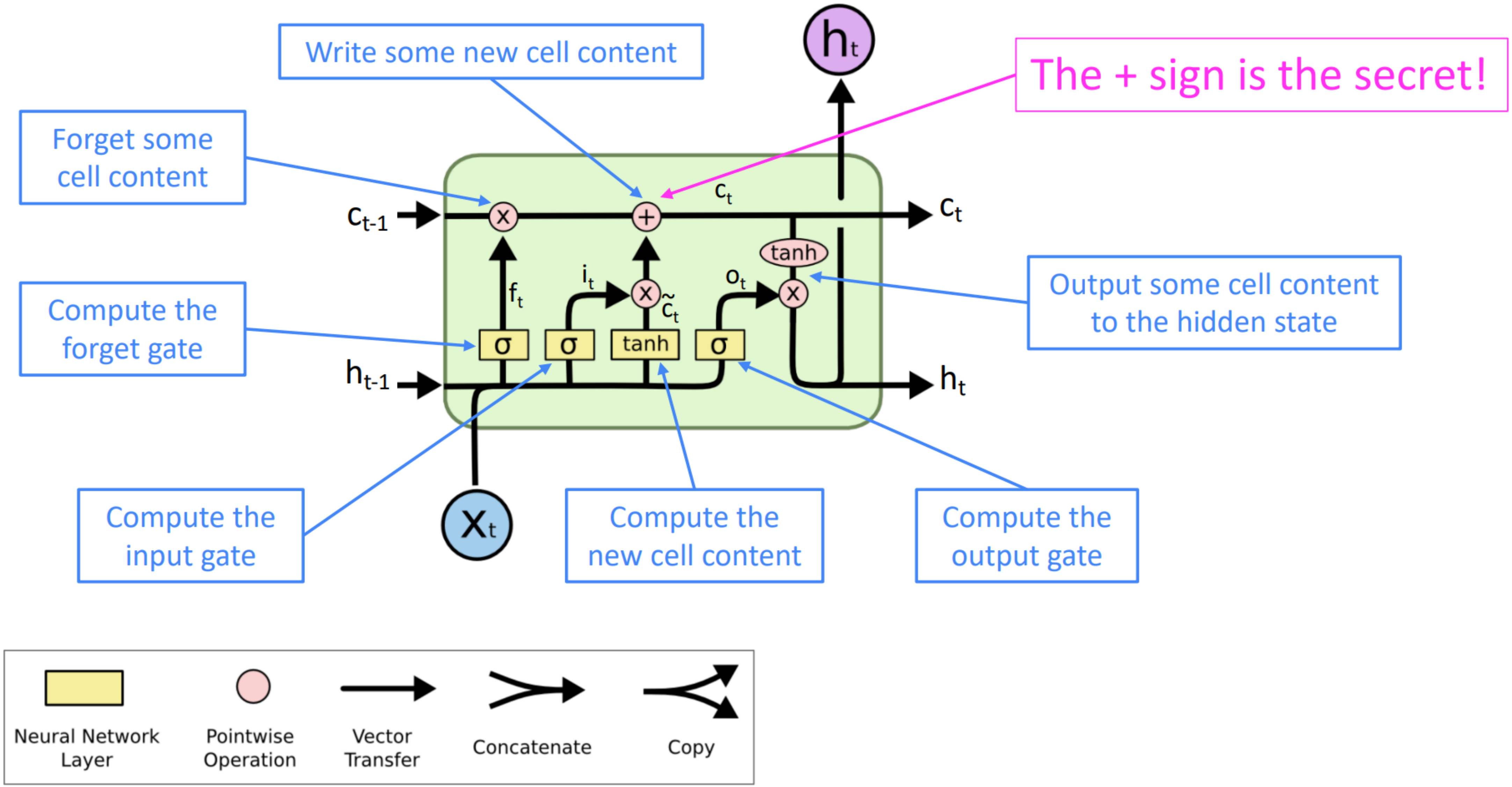
Formalization of the ideas just discussed

A special kind of RNN-cell that enables selective read/write/erasure



Different gates (shown with crosses)control flow of information

http://colah.github.io/posts/2015-08-Understanding-LSTMs



http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture06-fancy-rnn.pd step t:

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \boxed{\sigma}\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \boxed{\sigma}\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \boxed{\sigma}\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

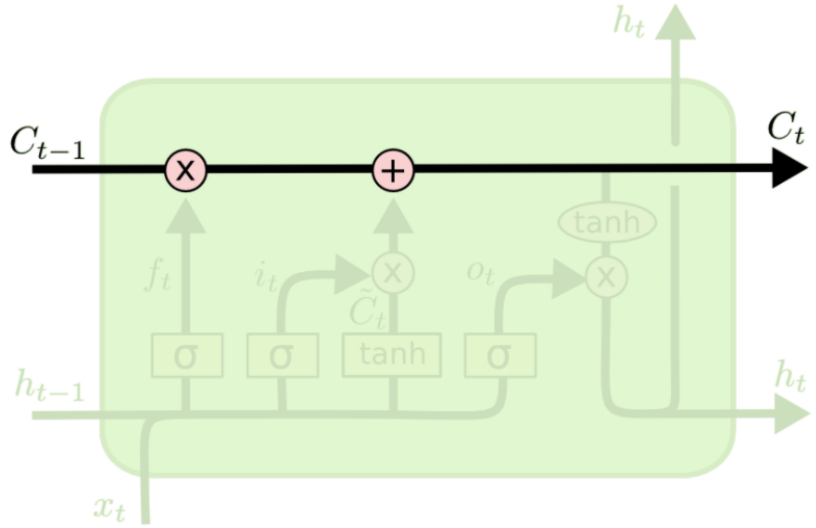
All these are vectors of same length n

Gates are applied using element-wise (or Hadamard) product: $\odot$

http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture06-fancy-rnn.pd

LSTMs solve vanishing gradient problem

Intuition: "The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged."



You can show that the gradient of loss with respect to cell state depends on forget gate values along the path

For a detailed explanation see this: http://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture15.pdf

Other Noteworthy Points

- Bi-directional RNNs
  - All NLU applications typically use this to incorporate context in both directions
  - Run separate RNNs in forward and backward directions
  - Concatenate hidden states at each time step for bidirectional context vector
- Deep RNNs: RNNS layers can be stacked
- Gated Recurrent Units (GRU): a simpler variant of LSTM
- LSTM-CRF network: A LSTM-variant where dependencies between output variables can be modeled

Suggested Reading

- N-gram Language Models (textbook chapter)
- Smoothing techniques: http://nrs.harvard.edu/urn-3:HUL.InstRepos:25104739
- FF-Neural LM: https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf
- The Unreasonable Effectiveness of Recurrent Neural Networks (nice blog post overview)
- Sequence Modeling: Recurrent and Recursive Neural Nets (Sections 10.1 and 10.2)
- On Chomsky and the Two Cultures of Statistical Learning
- Sequence Modeling: Recurrent and Recursive Neural Nets (Sections 10.3, 10.5, 10.7-10.12)
- Learning long-term dependencies with gradient descent is difficult (one of the original vanishing gradient papers)

Sequence to Sequence Modeling

Topics

- *Encoder-Decoder Models*
- *Attention Mechanism*

Sequence to Sequence Task

Input Sequence: $(x_1 \ x_2 \ x_3 \ x_4 \ldots\ldots x_i \ldots\ldots\ldots x_N)$

Output Sequence: $(y_1 \ y_2 \ y_3 \ y_4 \ldots y_k \ldots y_M)$

Input and output sequences have different lengths

Variable length input

Output contains categorical labels

Output at any time-step typically depends on neighbouring output labels and input elements

Machine Translation

Encoder-decoder model is a general framework for sequence to sequence tasks

Many tasks as Sequence to Sequence transformations

- *Summarization: Article ⇒ Summary*
- *Question answering: Question ⇒ Answer*
- *Dialogue: Previous utterance ⇒ next utterance*
- *Transliteration: character sequence ⇒ character sequence*
- *Grammar Correction: Incorrect sentence ⇒ Correct Sentence*
- *Translation Postediting: Incorrect translation ⇒ Correct translation*
- *Image labelling: Image ⇒ Label*

We are interested in modeling $P(\boldsymbol{Y}|\boldsymbol{X})$

Conditional Language Modelling task ➔

Learning Target LM conditioned on the source sentence

$$P(\boldsymbol{Y}|\boldsymbol{X}) = \prod_{j=1}^{M} P(y_j|\underbrace{y_1, y_2, \dots y_{j-1}}_{\textit{Target language RNN}}, \boldsymbol{X})$$

Additional conditioning on the source sentence

How do we model this?

LM for generating the target sequence

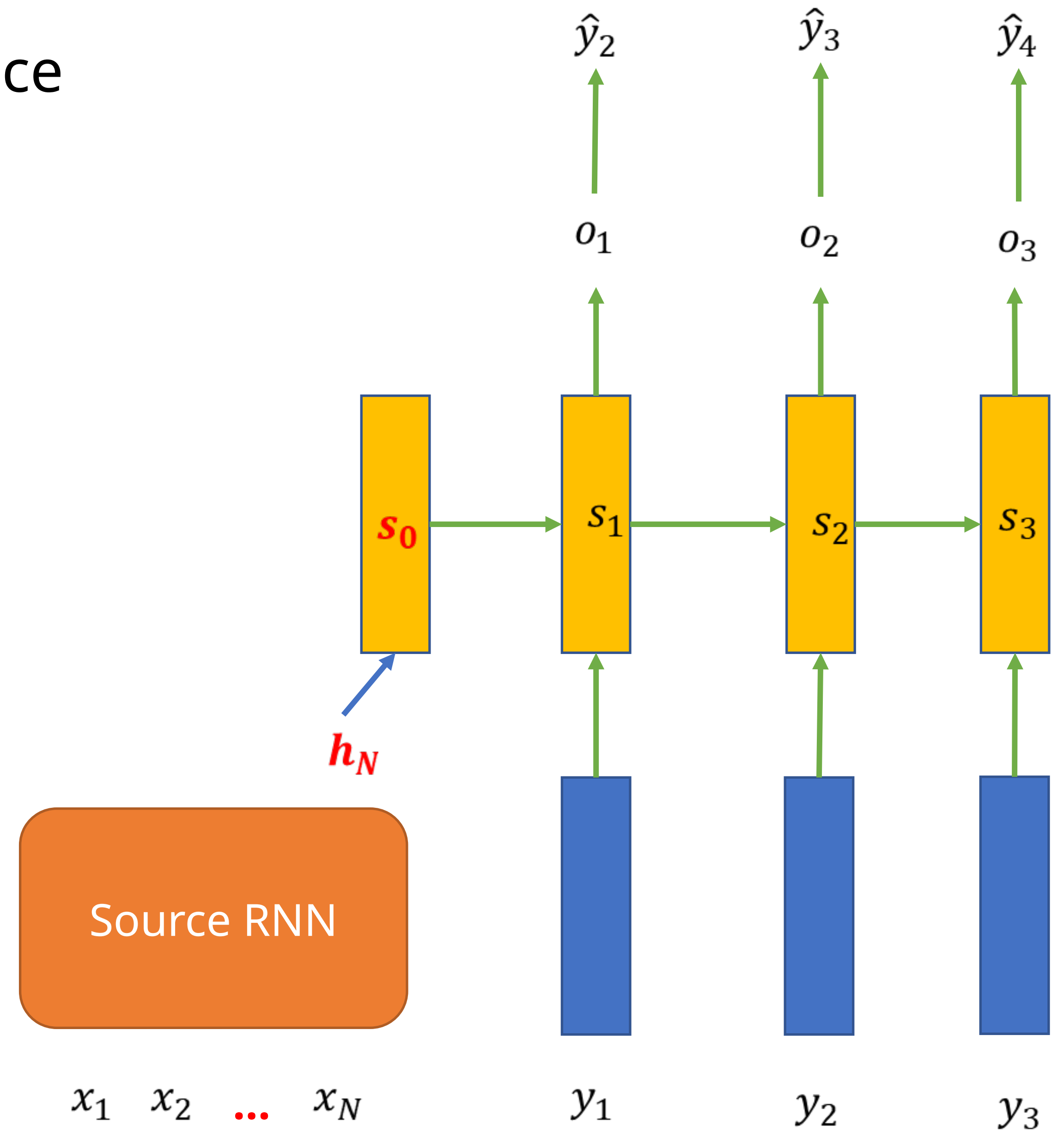
s_0 ➔ *Initial state of target language RNN*

Set s_0 = a vector representation of source

*We have our **conditional** LM*

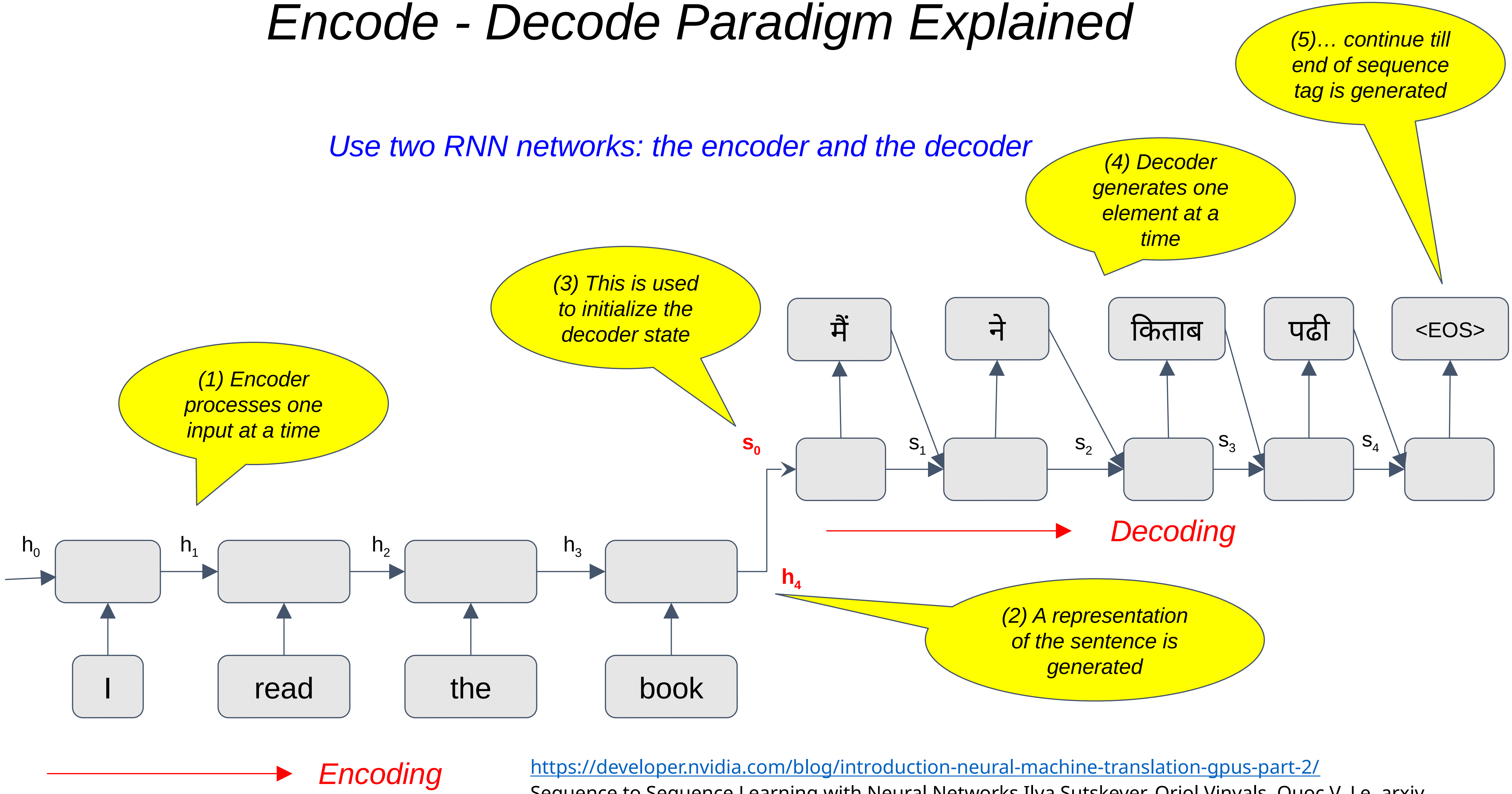
Source Vector Representation ➔
last state of source sentence RNN

$$s_0 = h_N$$



Encode - Decode Paradigm Explained

Use two RNN networks: the encoder and the decoder

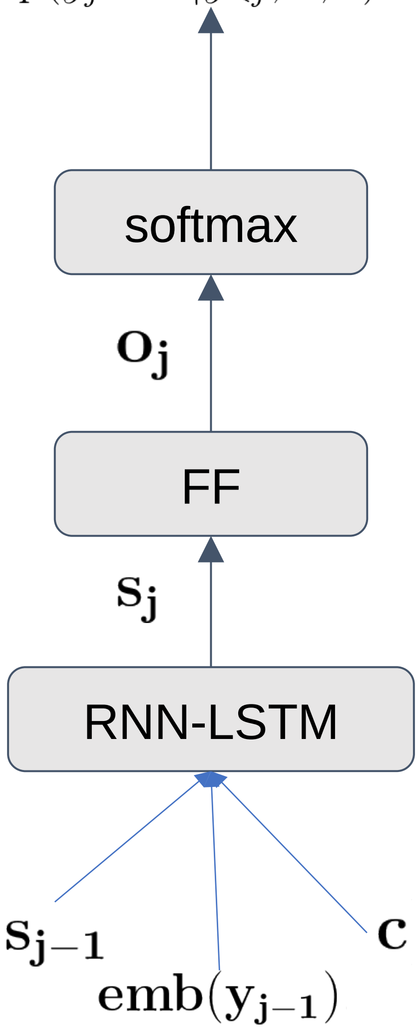


https://developer.nvidia.com/blog/introduction-neural-machine-translation-gpus-part-2/

Sequence to Sequence Learning with Neural Networks Ilya Sutskever, Oriol Vinyals, Quoc V. Le. arxiv pre-print [link]

What is the decoder doing at each time-step?

$$p(y_j = k | y_{<j}, \mathbf{x}; \theta) =$$



This captures $y_{<j}$

$$softmax(o_{jk}) = \frac{\exp(o_{jk})}{\sum_{m=0}^{m=T} \exp(o_{jm})}$$

$$\mathbf{o_j} = FF(s_j)$$

$$\mathbf{s_j} = g(\mathbf{s_{j-1}}, \mathbf{emb(y_{j-1})}, \mathbf{c})$$

This captures x, c=h_4

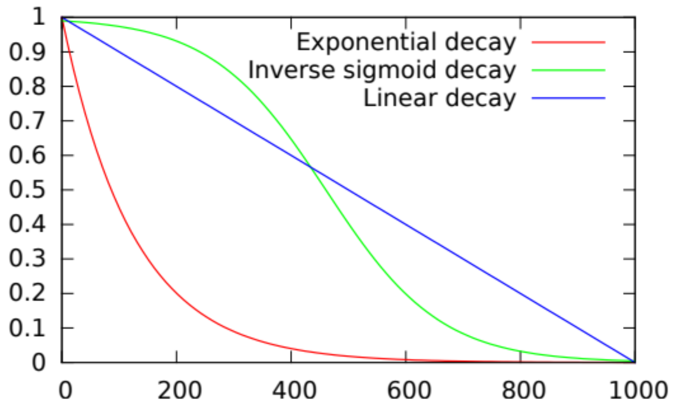
Training an NMT Model

$$p(\mathbf{y}|\mathbf{x};\theta) = \prod_{j=1}^{m} p(y_j|y_{<j},\mathbf{x};\theta) \qquad p(y_j = k|y_{<j},\mathbf{x};\theta) = softmax(o_{jk})$$

$$\mathcal{L}_\theta = \sum_{(\mathbf{x},\mathbf{y})\in\mathbf{C}} \log p(\mathbf{y}|\mathbf{x};\theta)$$

Maximum Likelihood Estimation

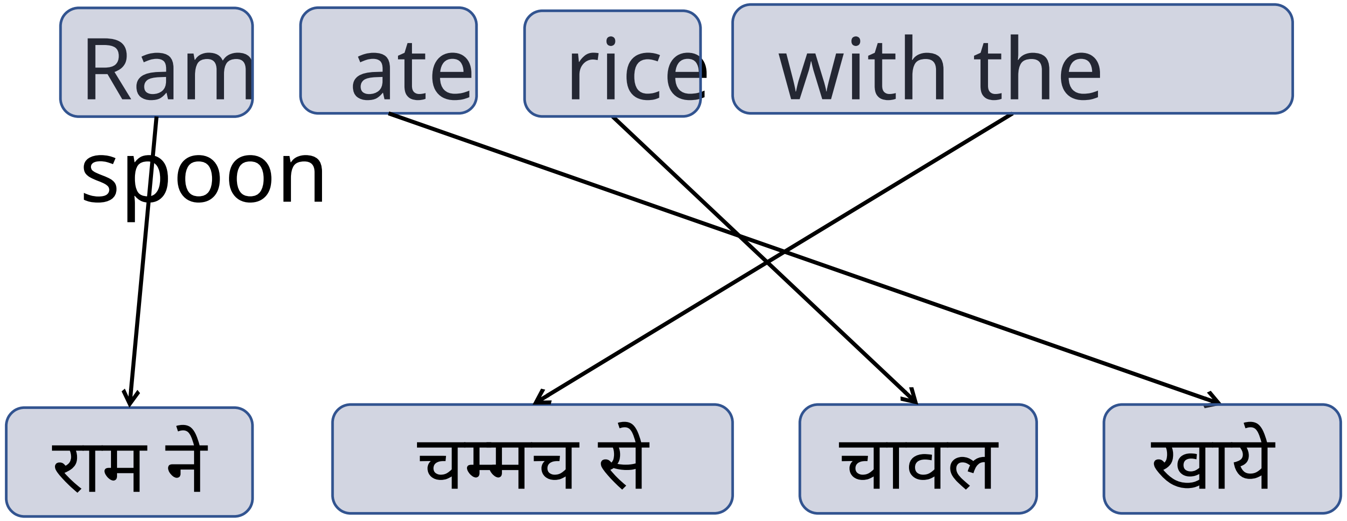
- *At each time decoder step:*
  - Feed model output from previous time step ➔ degrades performance
  - Feed ground-truth output from previous time step ➔ **teacher**
- *Discrepancy in train and test scenarios ➔* ***Exposure bias***
  - Solution → scheduled sampling
  - Sample from ground truth or predicted label
  - Sampling probability is varied: prefer ground truth earlier in training



Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent Neural networks. NeurIPS 2015.

Decoding

Searching for the best translations in the space of all translations



Decoding Strategies

- Exhaustive Search: *Score each and every possible translation – Forget it!* ➔ $O(V^N)$
- Sampling ➔ $O(NV)$
- Greedy ➔ $O(NV)$
- Beam Search ➔ $O(kNV)$

Greedy Decoding

w_1	0.03
w_2	0.7
w_3	0.05
w_3	0.1
w_4	0.08
w_5	0.04

Select best word using the distribution $P(y_j | y_{<j}, \boldsymbol{x})$

Sampling Decoding

w_1	0.03
w_2	0.7
w_3	0.05
w_3	0.1
w_4	0.08
w_5	0.04

Sample next word using the distribution $P(y_j | y_{<j}, \boldsymbol{x})$

Generate one word at a time sequentially

Not used to find best translation, but these methods have their uses ➔ for efficiency reas

Greedy Search is not optimal

	t_1		t_2
w_1	**0.5**	w_1	0.1
w_2	0.4	w_2	0.2
w_3	0.05	w_3	**0.3**
w_3	0.02	w_3	0.1
w_4	0.01	w_4	0.1
w_5	0.02	w_5	0.2

Probability of best sequence w_1w_3 *=0.15*

	t_1		t_2
w_1	0.5	w_1	0.1
w_2	**0.4**	w_2	0.45
w_3	0.05	w_3	0.2
w_3	0.02	w_3	0.15
w_4	0.01	w_4	0.08
w_5	0.02	w_5	0.02

Probability of best sequence w_2w_2 *=0.18*

Beam Search

A compromise solution between greedy decoding and exhaustive search

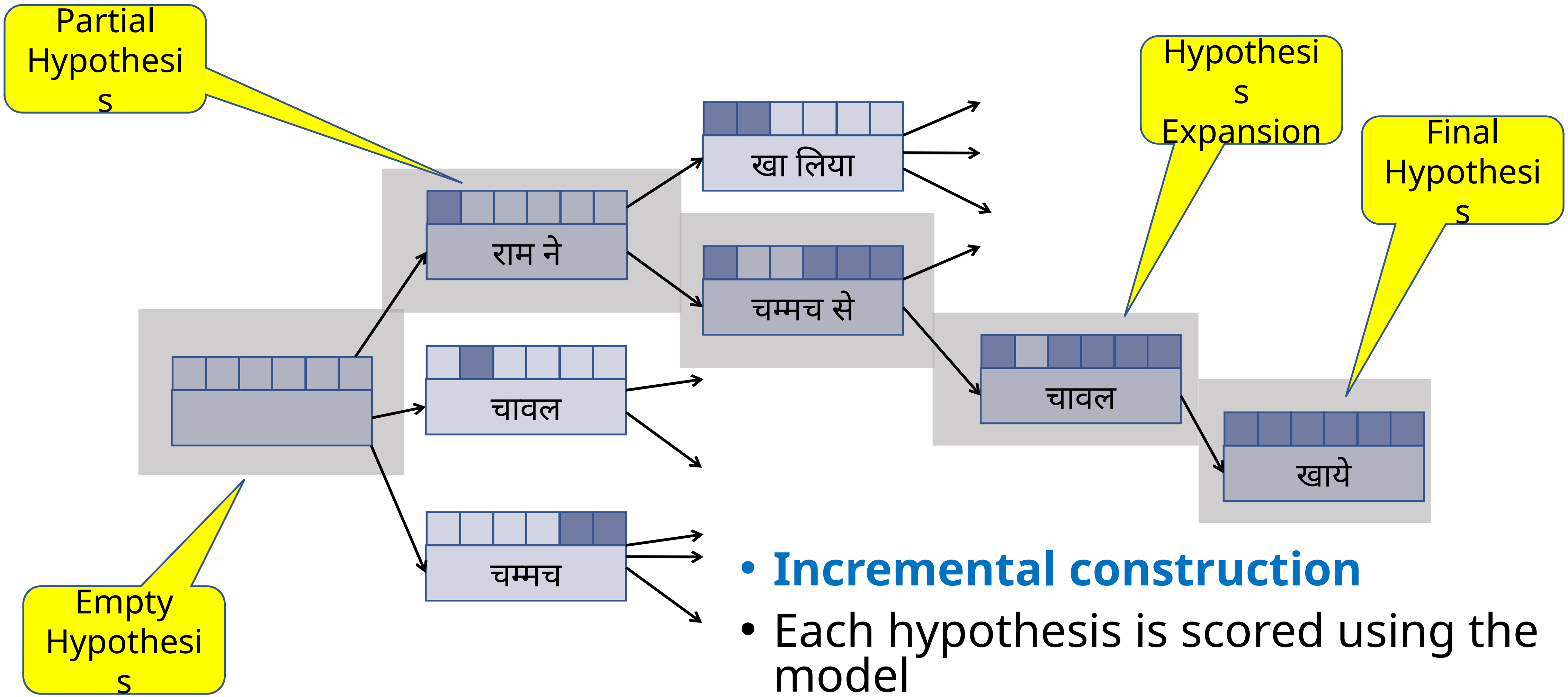
- *Explores more translation candidates than greedy search*
- *More efficient than exhaustive search*

2 Core Ideas:

- ***Incremental*** *construction & scoring of translation candidate (one decoder time step at a time)*
- *At each decoder time step,* ***keep the k-most probable partial translations***
    - *➔ these will be used for candidates expansion*

- ***Not guaranteed to find optimal solution***

http://www.phontron.com/slides/nlp-programming-en-13-search.pdf



- **Incremental construction**
- Each hypothesis is scored using the model
- Hypotheses are maintained in a priority queue

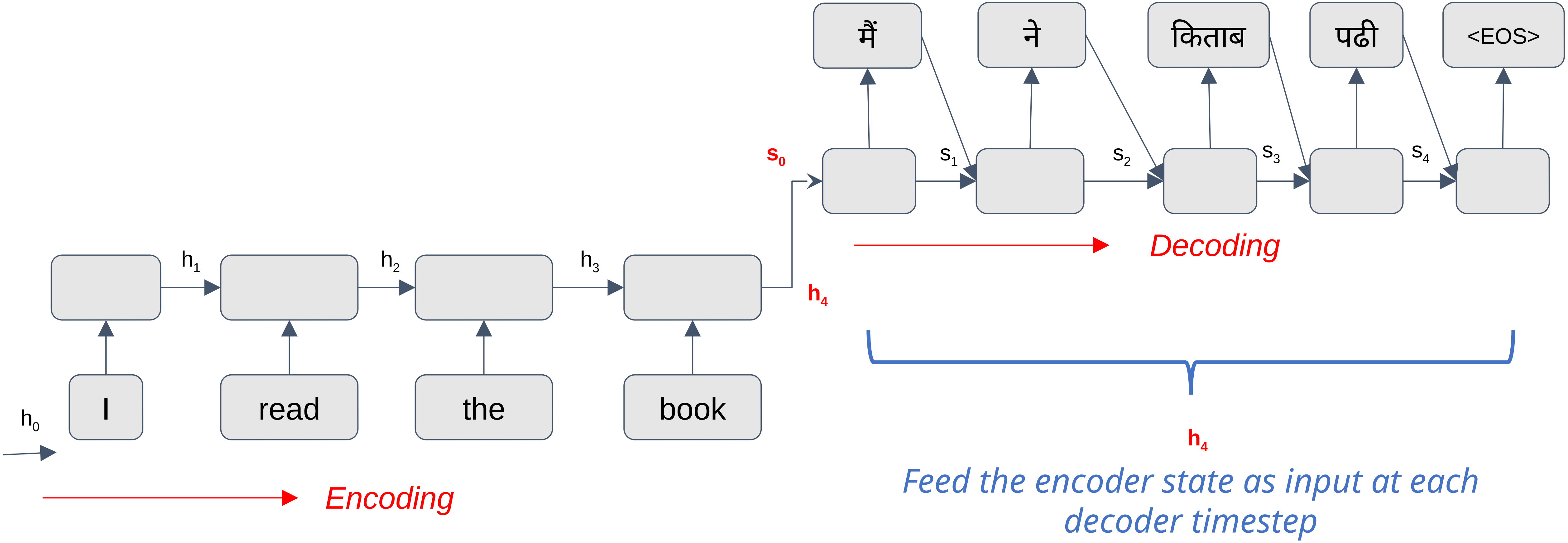
Topics

- *Encoder-Decoder Models*
- *Attention Mechanism*

The entire source sentence is represented by a single vector

Problems

- Insufficient to represent to capture all the syntactic and semantic complexities
  - *Solution: Use a richer representation for the sentences*
- Long-term dependencies: Source sentence representation not useful after few decoder time steps
  - *Solution: Make source sentence information when making the next prediction*



मैं
ने
किताब
पढी
<EOS>
s0
s1
s2
s3
s4
Decoding
h1
h2
h3
h4
h0
I
read
the
book
Encoding
h4
Feed the encoder state as input at each decoder timestep

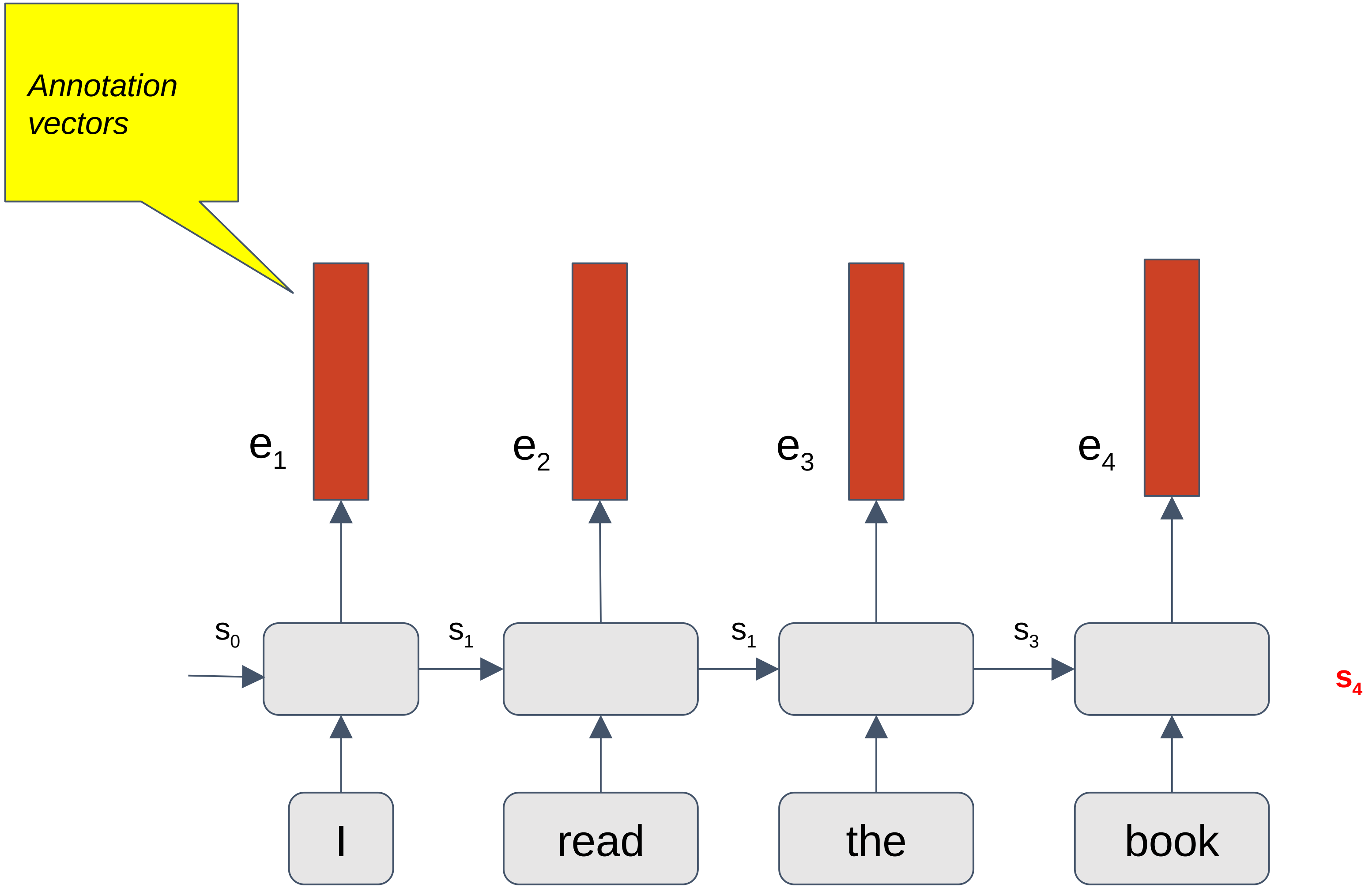
The entire source sentence is represented by a single vector

Problems

- Insufficient to represent to capture all the syntactic and semantic complexities
  - *Solution: Use a richer representation for the sentences*
- Long-term dependencies: Source sentence representation not useful after few decoder time steps
  - *Solution: Make source sentence information when making the next prediction*
  - *Even better, make **RELEVANT** source sentence information available*

These solutions motivate the next paradigm

Encode - Attend - Decode Paradigm



Represent the source sentence by the **set of output vectors** from the encoder

Each output vector at time *t* is a contextual representation of the input at time *t*

Let's call these encoder output vectors ***annotation vectors***

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *ICLR 2015.*

https://developer.nvidia.com/blog/introduction-neural-machine-translation-gpus-part-3/

How can the annotation vectors help predicting the next output?

Key Insight:

(1) Not all annotation vectors are equally important for prediction of the next element

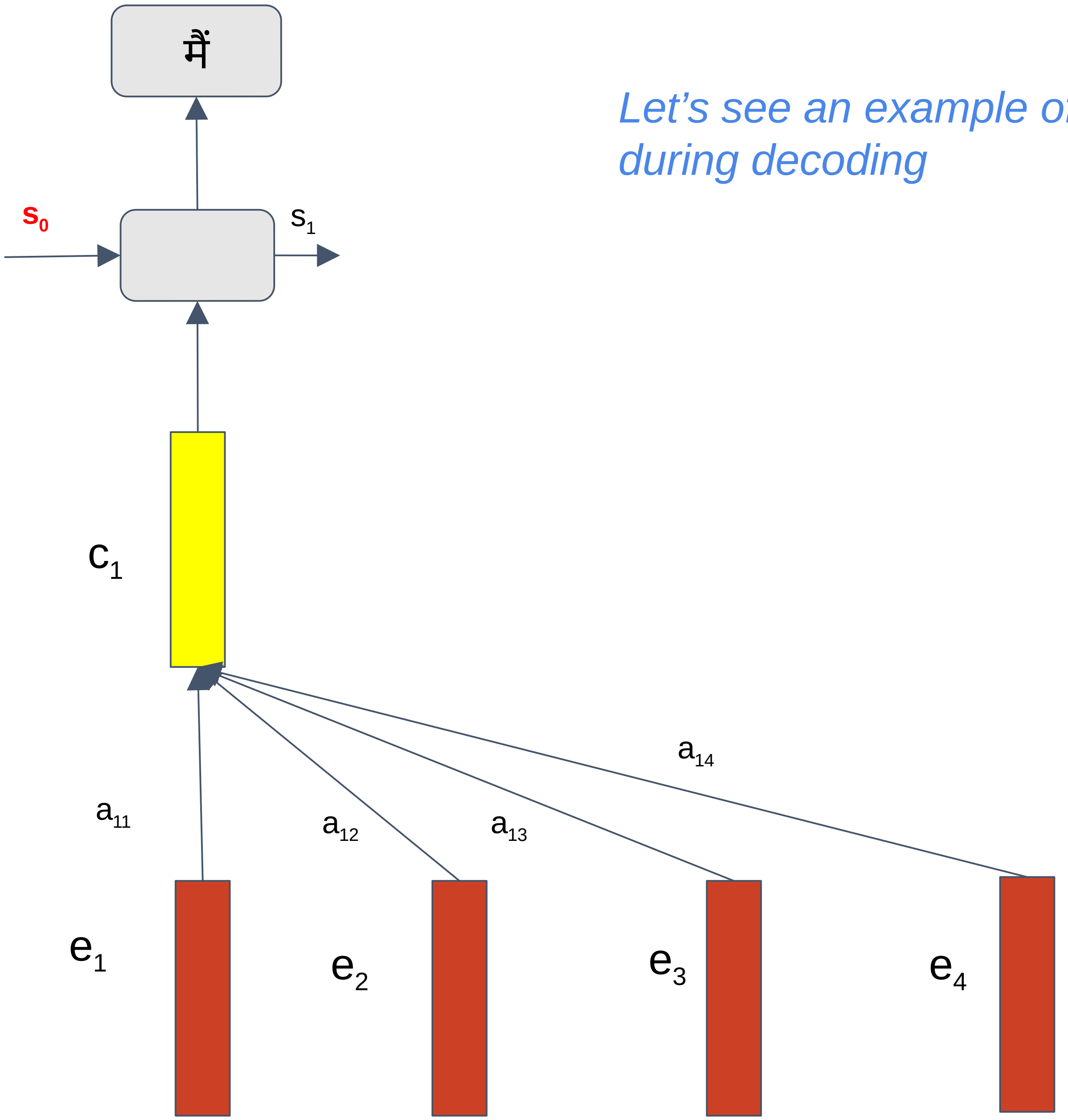
(2) The annotation vector to use next depends on what has been generated so far by the decoder

eg. To generate the 3rd target word, the 3rd source word is most important

Context vector = weighted average of the annotation vectors

More weight to annotation vectors which need more **focus or attention**

This averaged ***context vector*** is an input to the decoder



Let's see an example of how the ***attention mechanism*** *works during decoding*

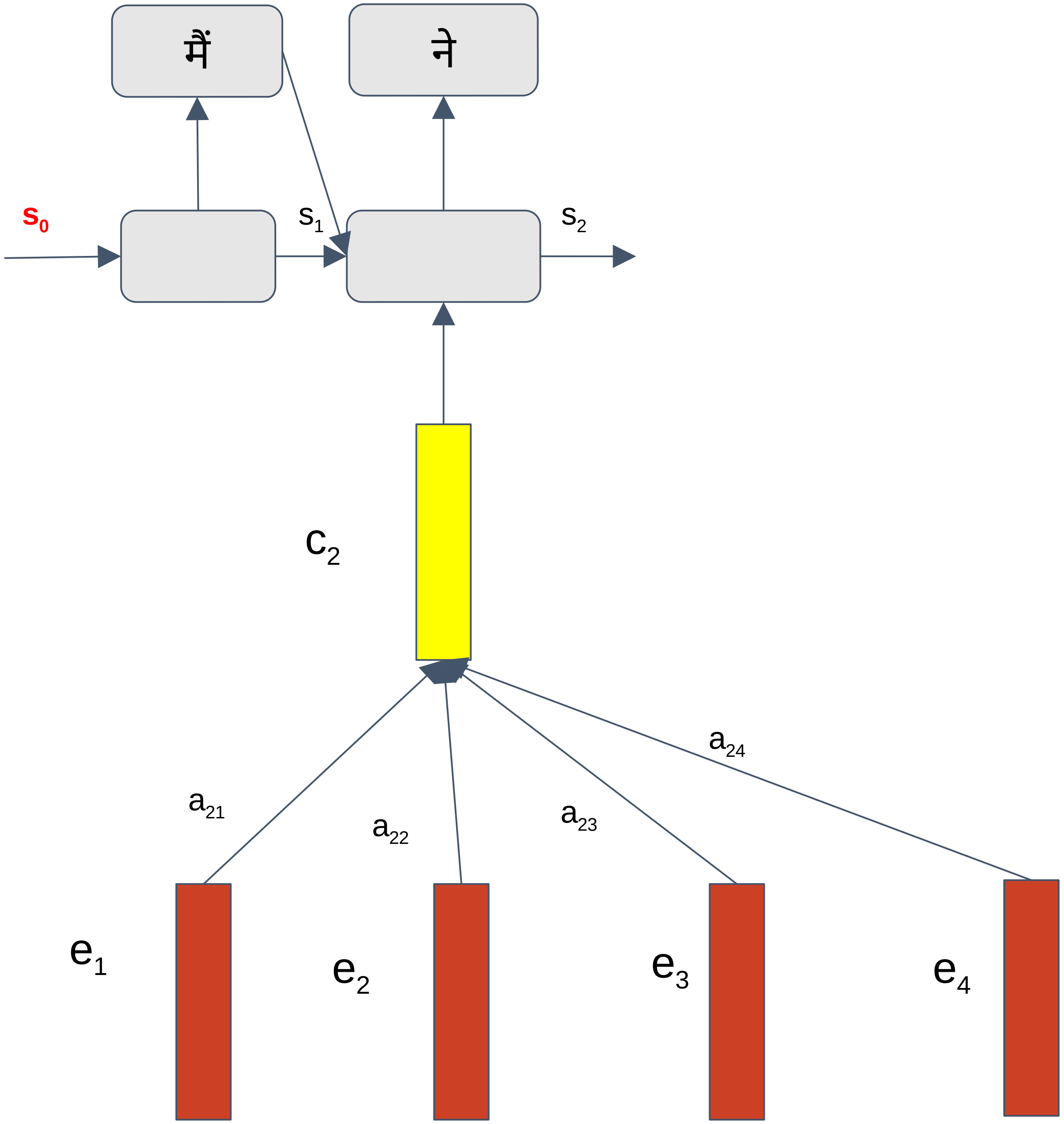
$$c_i = \sum_{j=1}^{n} a_{ij} e_j$$

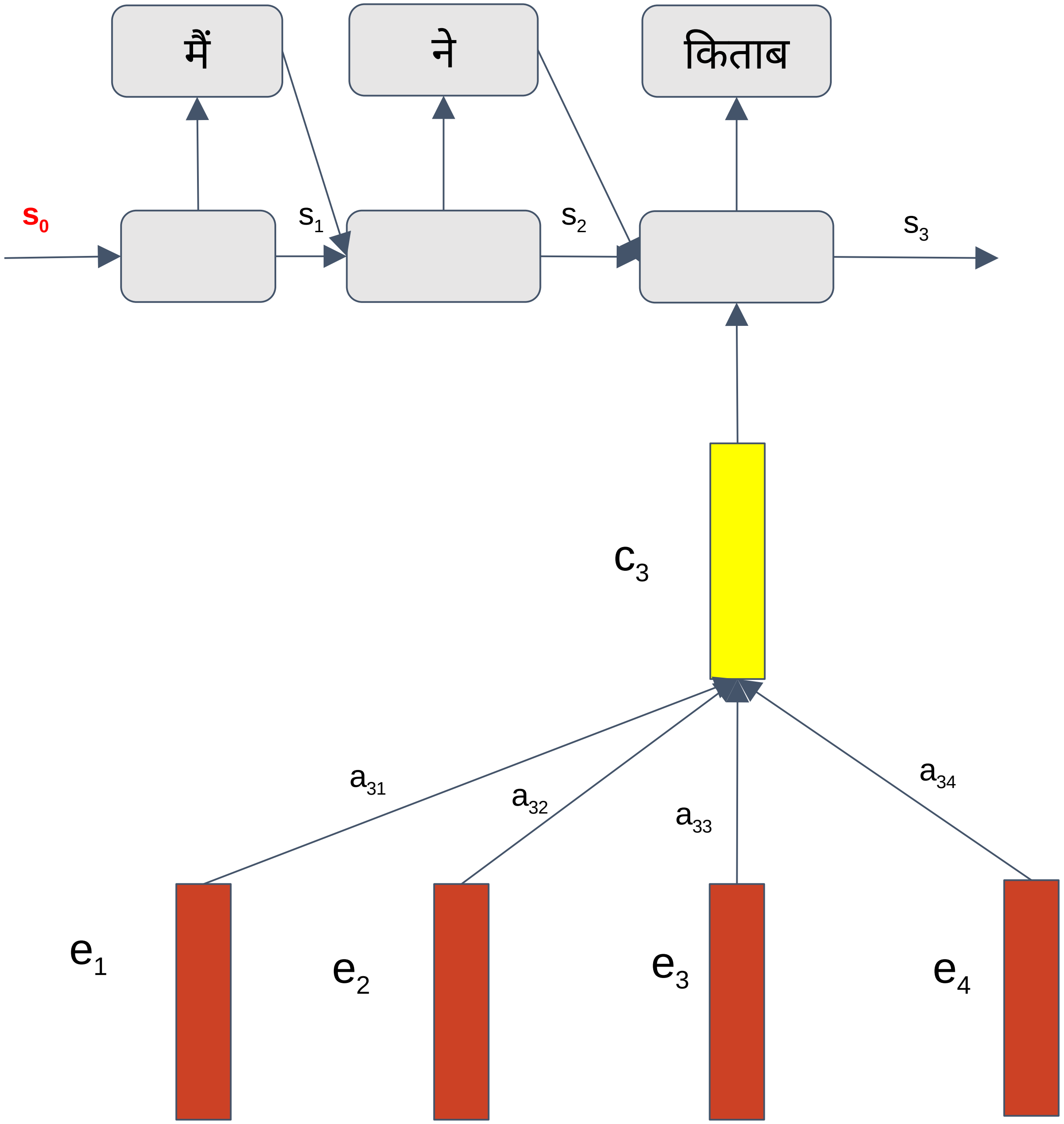
For generation of i^{th} output character:

c_i : context vector

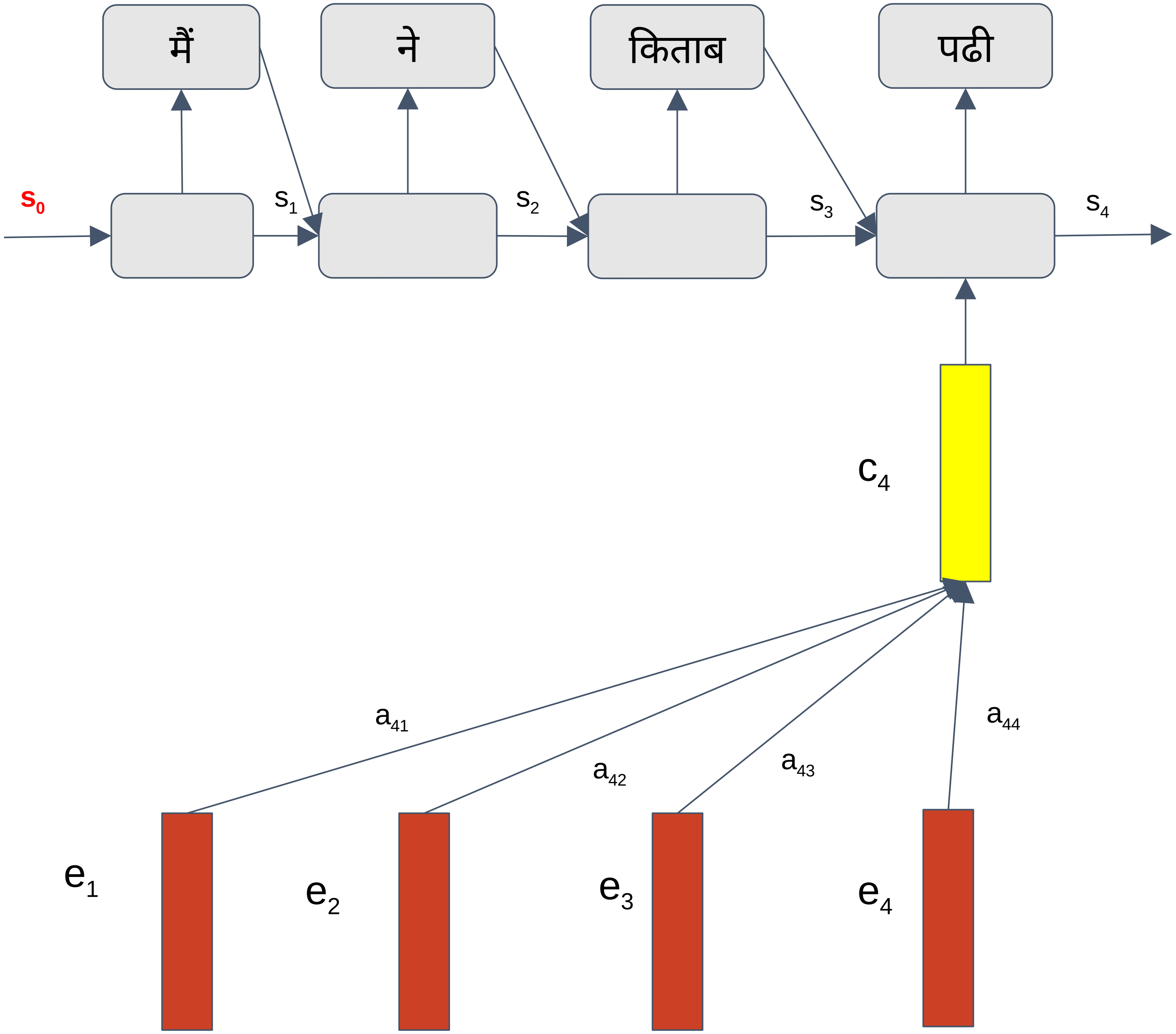
a_{ij} : annotation weight for the j^{th} annotation vector

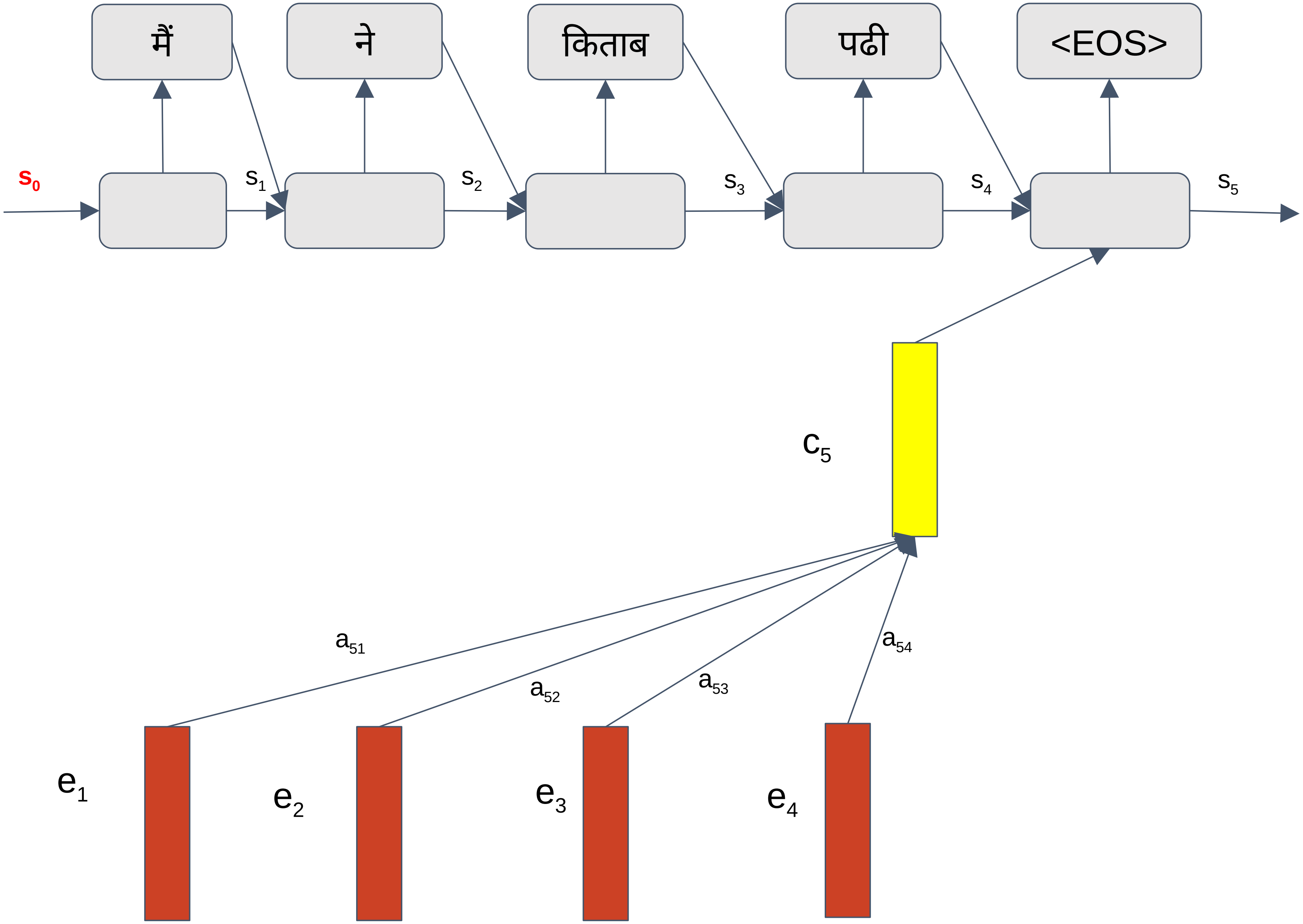
o_j: j^{th} annotation vector


मैं
ने
s0
s1
s2
c2
a21
a22
a23
a24
e1
e2
e3
e4



मैं
ने
किताब
s0
s1
s2
s3
c3
a31
a32
a33
a34
e1
e2
e3
e4


मैं
ने
किताब
पढी
s0
s1
s2
s3
s4
c4
a41
a42
a43
a44
e1
e2
e3
e4



मैं
ने
किताब
पढी
<EOS>
s_0
s_1
s_2
s_3
s_4
s_5
c_5
a_{51}
a_{52}
a_{53}
a_{54}
e_1
e_2
e_3
e_4

How do we find the attention weights?

Let the training data help you decide!!

Idea: Pick the attention weights that maximize the overall translation likelihood accuracy

Scoring function **g** to match the encoder and decoder states

$$\alpha_{ij} = g(s_{j-1}, e_i, \mathbf{emb}(y_{j-1}))$$

$$a_{ij} = \frac{\exp(\alpha_{ij})}{\sum_{i=1}^{i=N} \exp(\alpha_{kj})}$$

$$c_j = \sum_{i=1}^{i=N} a_{ij} e_i$$

How do we find the attention weights?

Let the training data help you decide!!

Idea: Pick the attention weights that maximize the overall translation likelihood accuracy

$$\alpha_{ij} = g(s_{j-1}, e_i, \mathbf{emb}(y_{j-1}))$$

g can be a feedforward network or a similarity metric like dot product

$$a_{ij} = \frac{\exp(\alpha_{ij})}{\sum_{i=1}^{i=N} \exp(\alpha_{kj})}$$

$$c_j = \sum_{i=1}^{i=N} a_{ij} e_i$$

How do we find the attention weights?

Let the training data help you decide!!

Idea: Pick the attention weights that maximize the overall translation likelihood accuracy

$$\alpha_{ij} = g(s_{j-1}, e_i, \mathbf{emb}(y_{j-1}))$$

Normalize score to obtain attention weights

$$a_{ij} = \frac{\exp(\alpha_{ij})}{\sum_{i=1}^{i=N} \exp(\alpha_{kj})}$$

$$c_j = \sum_{i=1}^{i=N} a_{ij} e_i$$

How do we find the attention weights?

Let the training data help you decide!!

Idea: Pick the attention weights that maximize the overall translation likelihood accuracy

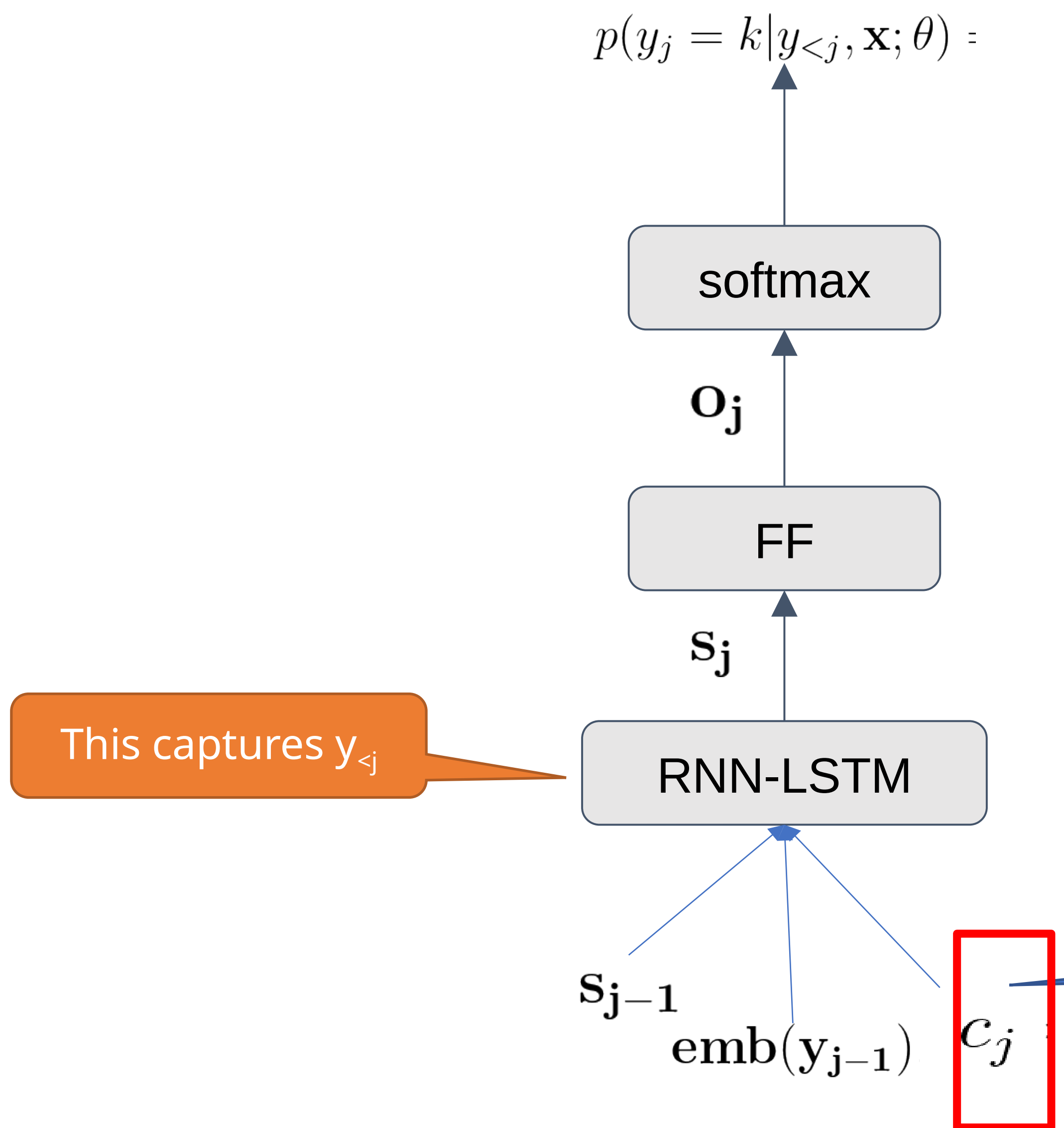
$$\alpha_{ij} = g(s_{j-1}, e_i, \mathbf{emb}(y_{j-1}))$$

$$a_{ij} = \frac{\exp(\alpha_{ij})}{\sum_{i=1}^{i=N} \exp(\alpha_{kj})}$$

Final context vector is weighted average of encoder outputs

$$c_j = \sum_{i=1}^{i=N} a_{ij} e_i$$

Let us revisit what the decoder does at time step t



$$softmax(o_{jk}) = \frac{\exp(o_{jk})}{\sum_{m=0}^{m=T} \exp(o_{jm})}$$

$$\mathbf{o_j} = FF(s_j)$$

$$\mathbf{s_j} = g(\mathbf{s_{j-1}}, \mathbf{emb(y_{j-1})}, \mathbf{c})$$

Choice of Attention Scoring Function

Feedforward	$: \alpha_{ij} = \boldsymbol{W_a}[e_j; s_i]$
Dot Product	$: \alpha_{ij} = s_i^T e_j$
Scaled Dot Product	$: \alpha_{ij} = \frac{s_i^T e_j}{\sqrt{\lvert e_j \rvert}}$
Multiplicative Attention	$: \alpha_{ij} = s_i^T \boldsymbol{W_a} e_j$
Additive Attention	$: \alpha_{ij} = \boldsymbol{W_1} s_i + \boldsymbol{W_2} e_j$

Effective Approaches to Attention-based Neural Machine Translation. Thang Luong, Hieu Pham, Christopher D. Manning. EMNLP 2015.

Attention is a general and important concept in Deep learning

Given a set of VALUES ➔ select a summary of the values that is relevant to a QUERY

Each VALUE represented by a KEY ➔ the QUERY is matched to the KEY (content similarity)

Select a summary with different focus on different values ➔ Weighted average

Associative memory read + selection

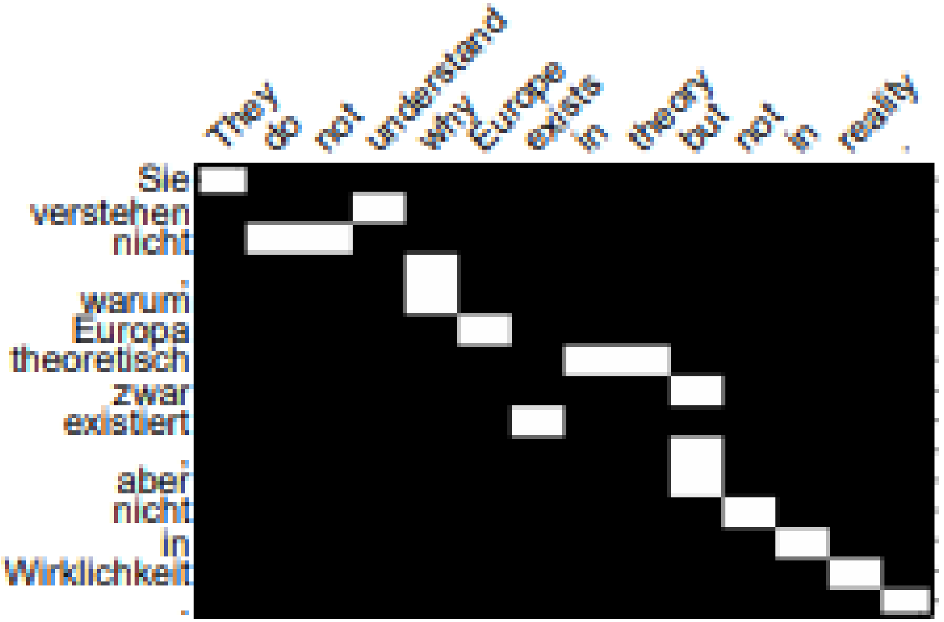
For MT

QUERY: decoder state

VALUE, KEY: encoder annotation vector

Benefits of Attention

- Significant improves in NMT quality
  - Performs better on long sentences
  - Word-order is no longer a major issu
  - Used in all NMT systems
- Attention provides some interpretability
  - Attention!=Alignment
- There is more to attention



https://arxiv.org/pdf/1508.04025.pdf

A lot of interesting work with attention

- Pointer Networks
- Pointer Generator Networks
- Modeling Coverage
- Learning word-alignments

Suggested Reading

- On the difficulty of training Recurrent Neural Networks (proof of vanishing gradient problem)
- Vanishing Gradients Jupyter Notebook (demo for feedforward networks)
- Understanding LSTM Networks (must read, blog post overview)
- https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html

Suggested Reading

- Original LSTM papers:
  - http://www.bioinf.jku.at/publications/older/2604.pdf (the original paper)
  - ftp://ftp.idsia.ch/pub/juergen/TimeCount-IJCNN2000.pdf (peephole variant that is most widely used)
- Pretrained Models
  - ELMo
  - ULMFit
- Other lectures on RNNs
  - https://github.com/oxford-cs-deepnlp-2017/lectures (Lectures 3 & 4)
  - https://www.cse.iitm.ac.in/~miteshk/CS7015.html (Lectures 14 & 15)
  - http://web.stanford.edu/class/cs224n/ (Lectures 5 & 6)

More Reading Material

This was a small introduction, you can find mode elaborate presentations, books and further references below:

SMT Tutorials & Books

- *Machine Learning for Machine Translation (An Introduction to Statistical Machine Translation)*. **Tutorial at ICON 2013** [slides]
- *Machine Translation: Basics and Phrase-based SMT*. **Talk at the Ninth IIIT-H Advanced Summer School on NLP (IASNLP 2018), IIIT Hyderabad** . [pdf]
- Statistical Machine Translation. Philip Koehn. Cambridge University Press. 2008. [site]
- Machine Translation. Pushpak Bhattacharyya. CRC Press. 2015. [site]

NMT Tutorials & Books

- Neural Machine Translation and Sequence-to-sequence Models: A Tutorial. Graham Neubig. 2017. [pdf]
- CMU CS 11-731, Fall 2019 - Machine Translation and Sequence-to-Sequence Models. [link]
- Neural Machine Translation: A Review and Survey. Felix Stahlberg. JAIR. 2020.
- Raj Dabre, Chenhui Chu, and Anoop Kunchukuttan. 2020. A Survey of Multilingual Neural Machine Translation. ACM Computing Surveys. [COLING 2020 Tutorial Material]

Other Lectures

- https://github.com/oxford-cs-deepnlp-2017/lectures (Lectures 7 & 8)
- https://www.cse.iitm.ac.in/~miteshk/CS6910.html (Lectures 16)
- http://web.stanford.edu/class/cs224n/ (Lectures 7)

Thank You!

Write to me at anoop.kunchukuttan@gmail.com in case you have any questions