Indian Institute of Technology Kharagpur
Department of Computer Science & Engineering

CS60075
Natural Language Processing
Autumn 2020

Module 7:

Machine Translation 3

Encoder Decoder

20 October 2020

Review: Recurrent Neural Networks

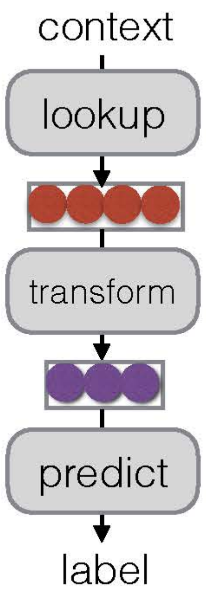
Long-distance Dependencies in Language

- Agreement in number, gender, etc.
  - **He** does not have very much confidence in **himself**.
  - **She** does not have very much confidence in **herself**.
- Selectional preference
  - The **reign** has lasted as long as the life of the **queen**.
  - The **rain** has lasted as long as the life of the **clouds**.

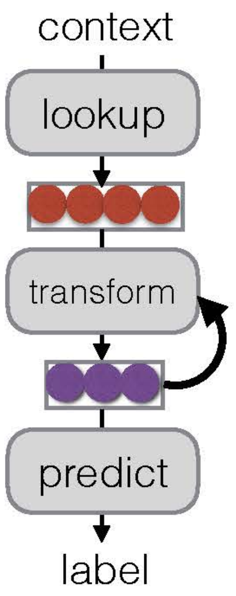
Recurrent Neural Networks
(Elman 1990)

- Tools to “remember” information

Feed-forward NN

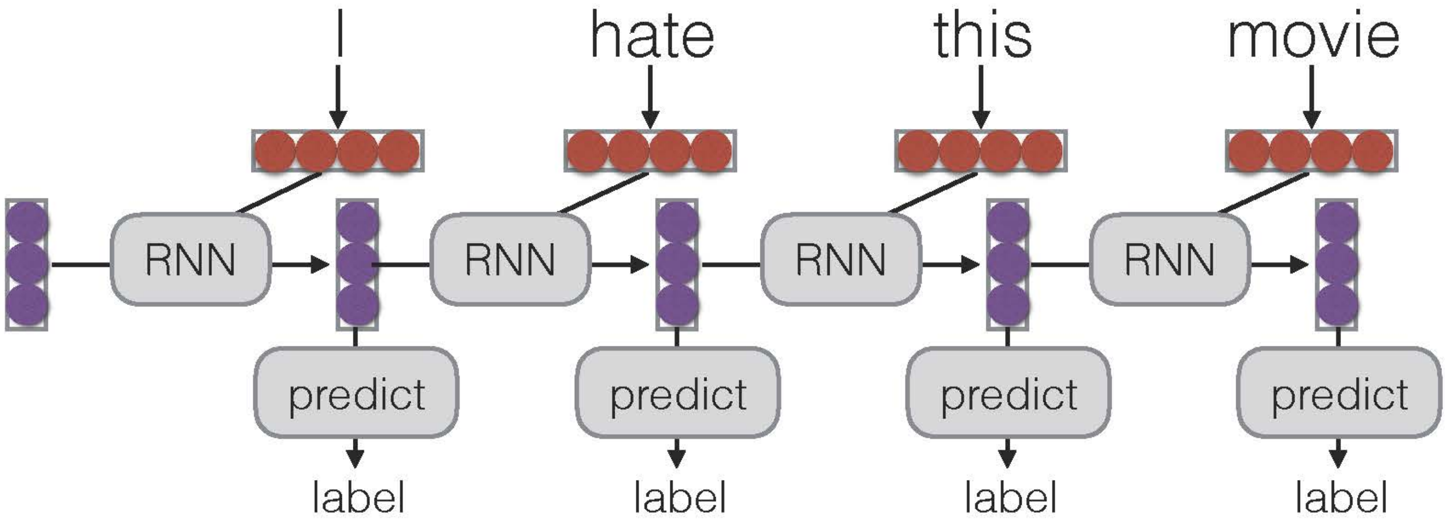


Recurrent NN

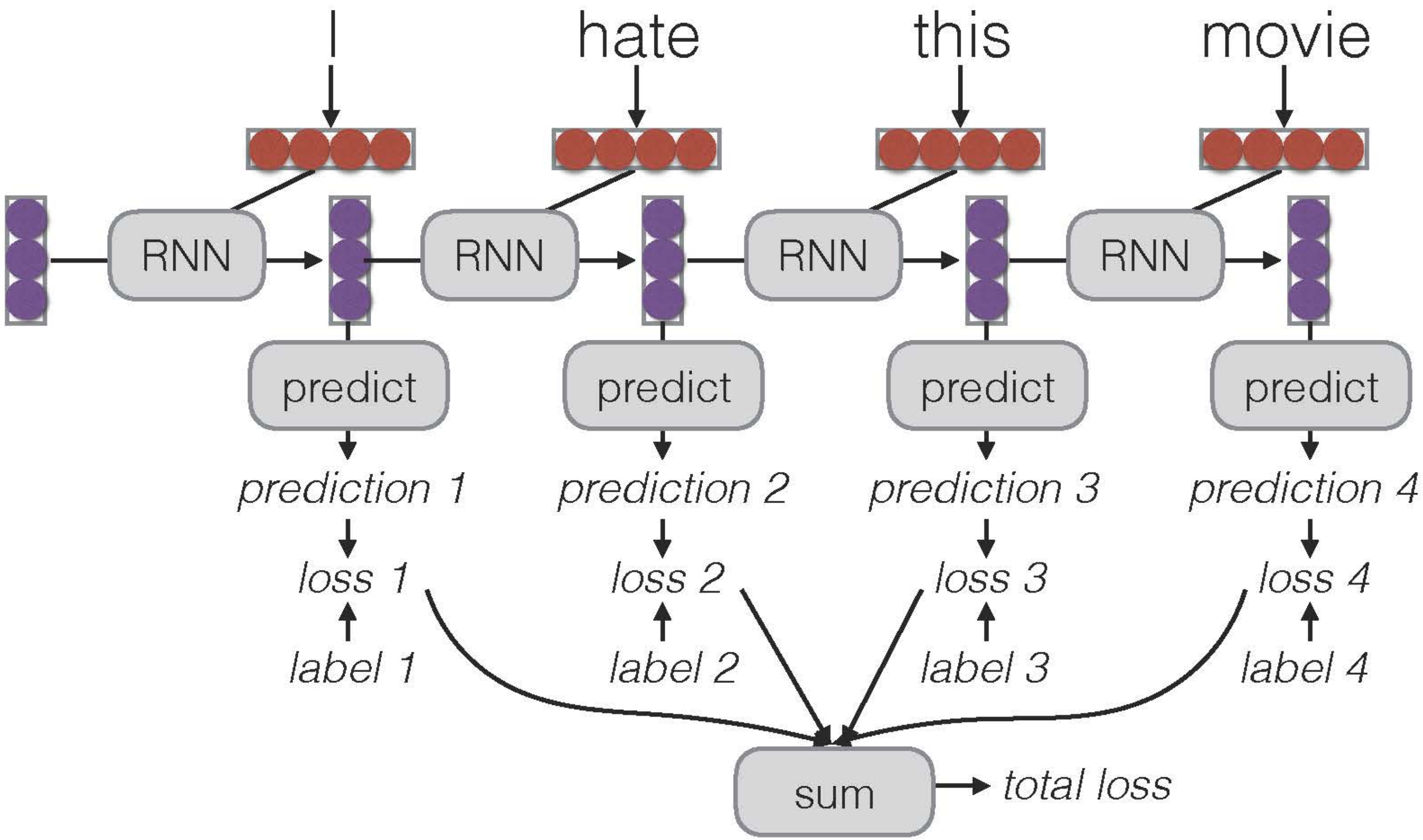


Unrolling in Time

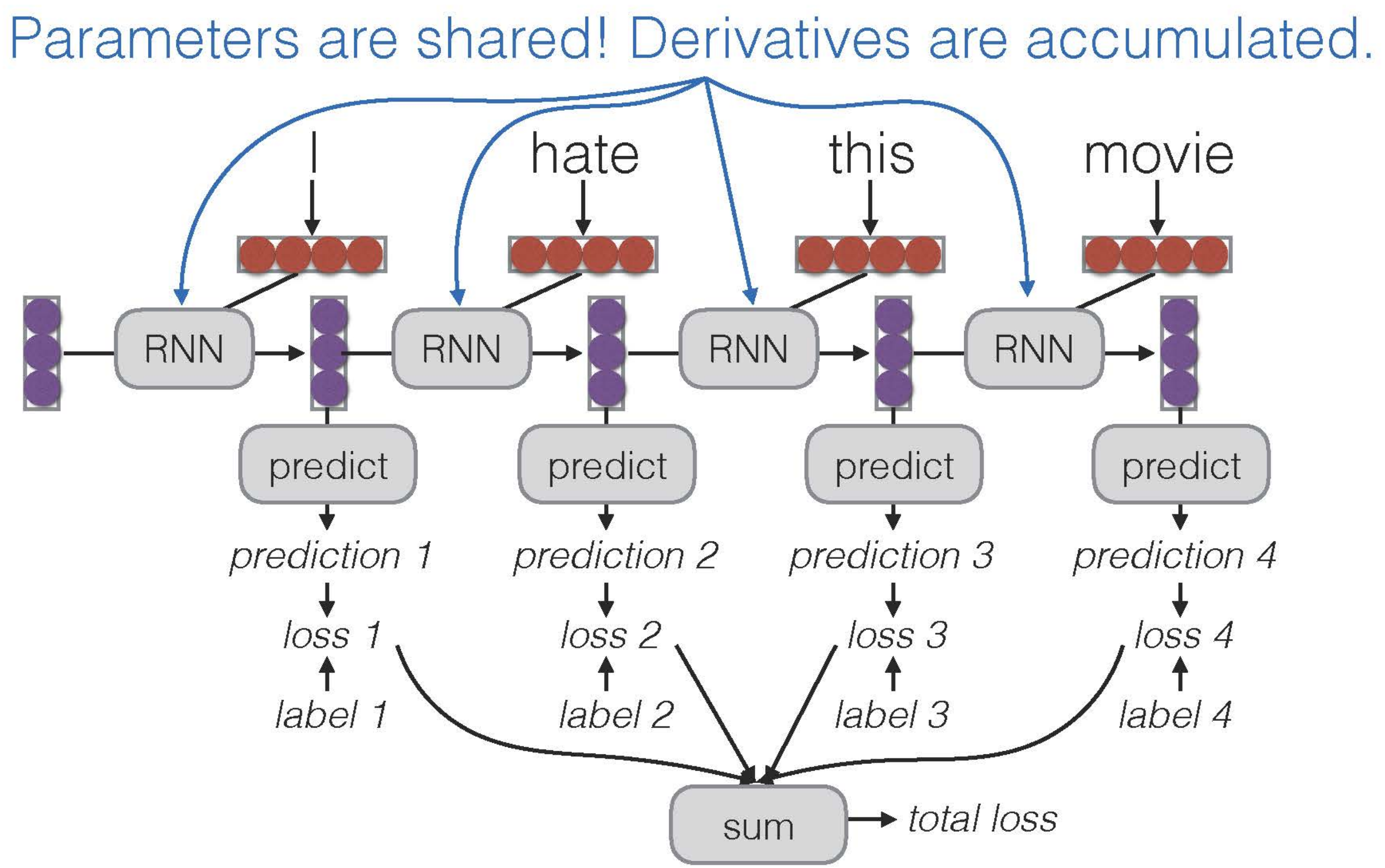
- What does processing a sequence look like?



Training RNNs



Parameter Tying



What Can RNNs Do?

- Represent a sentence
  - Read whole sentence, make a prediction
- Represent a context within a sentence
  - Read context up until that point

Representing Sentences

- Sentence Classification
- Conditioned generation

Representing Contexts

- Tagging
- Language Modeling

$$s \sim P(X)$$

- Language Modeling with RNNs
- At each step, calculate probability of next word

Conditional Language Modeling

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

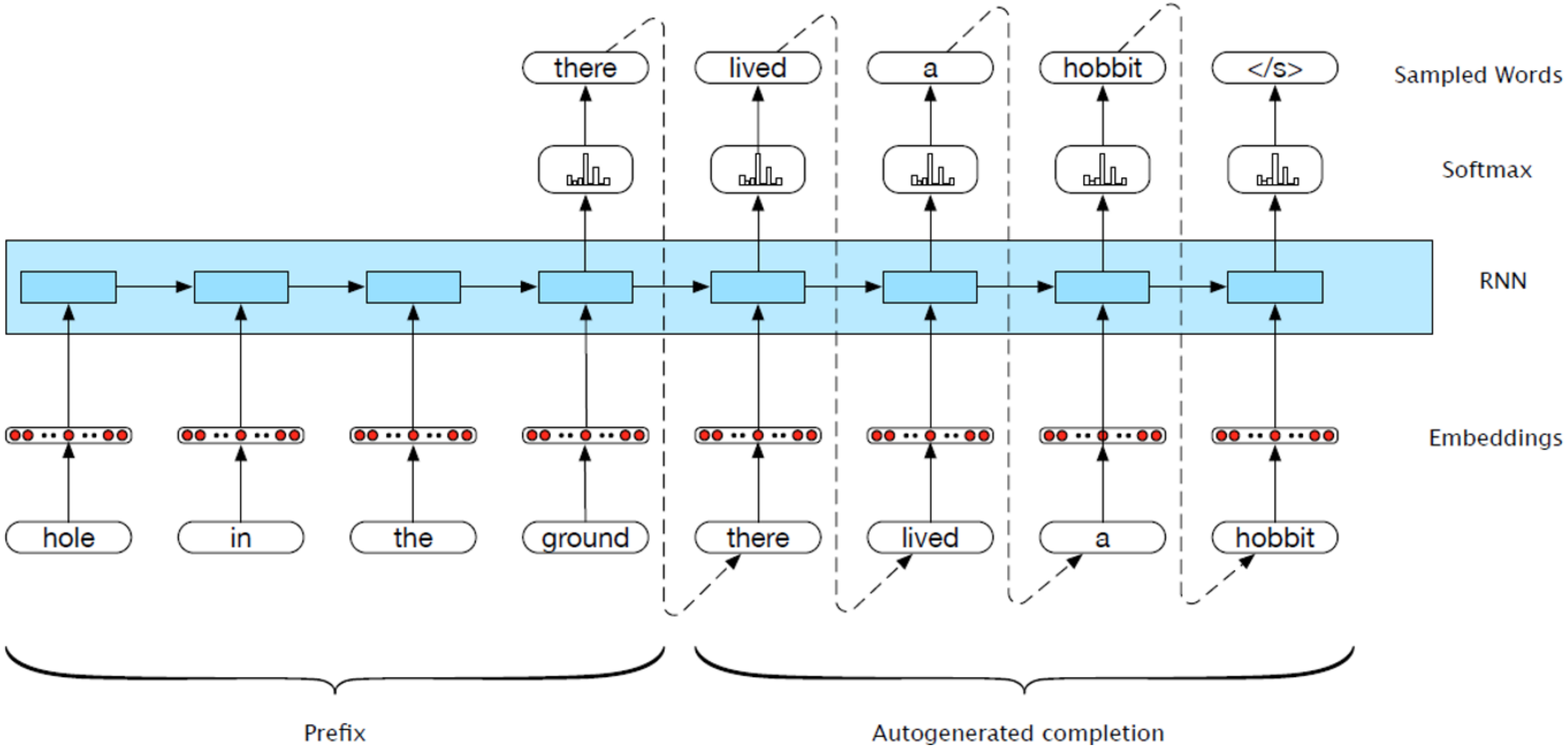
Input X	Output Y	Task
Structured data	NL Description	NL Generation
Hindi	French	Translation
Document	Short Description	Summarization
Utterance	Response	Response generation
Image	Text	Image Captioning
Speech	Transcript	Speech Recognition

Conditional Language Models

$$P(Y|X) = \prod_{j=1}^{J} P\left(y_j | \boldsymbol{X}, y_1, \dots, y_{j-1}\right)$$

Generation with prefix

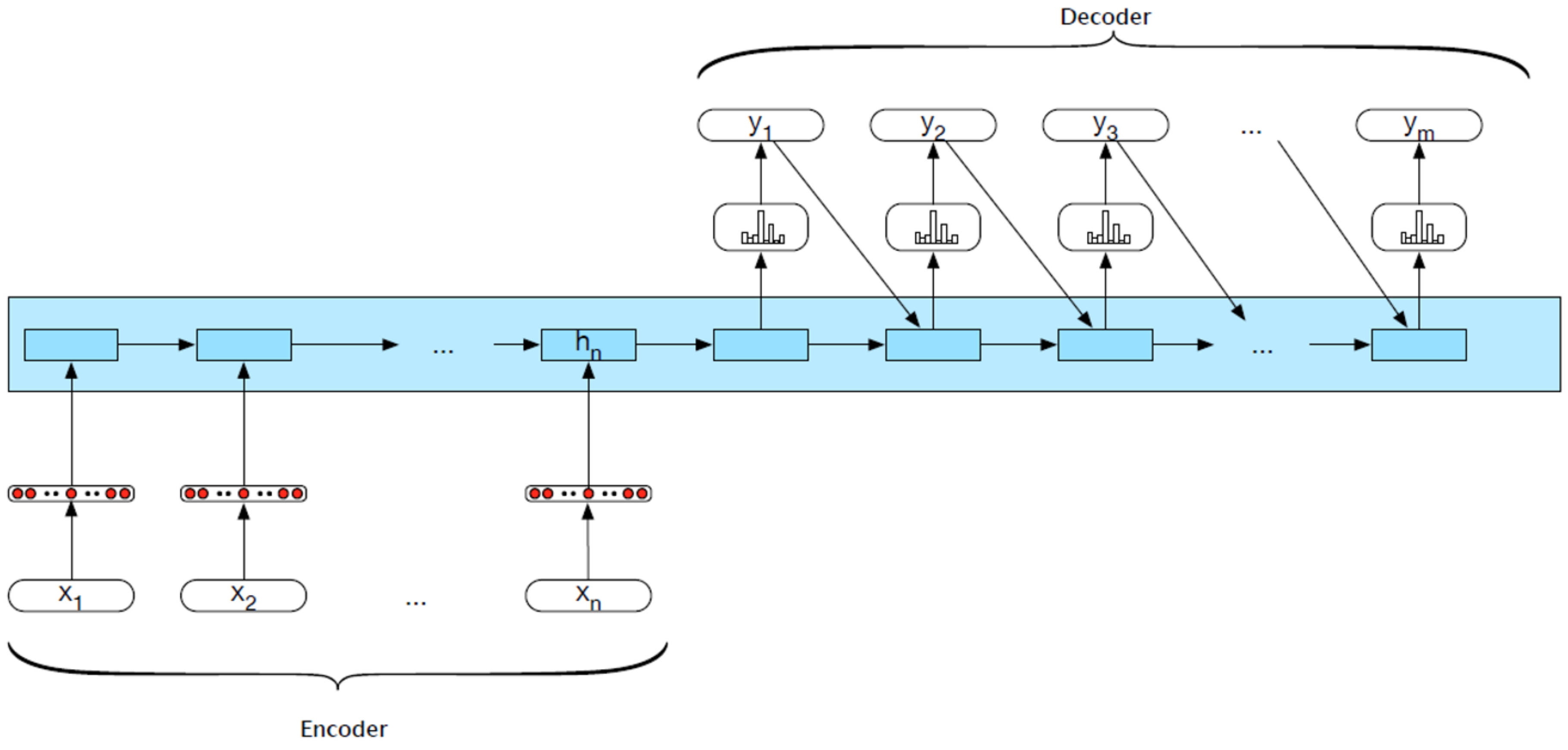
$$h_t = g(h_{t-1}, x_t)$$
$$y_t = f(h_t)$$



Encoder-Decoder Networks

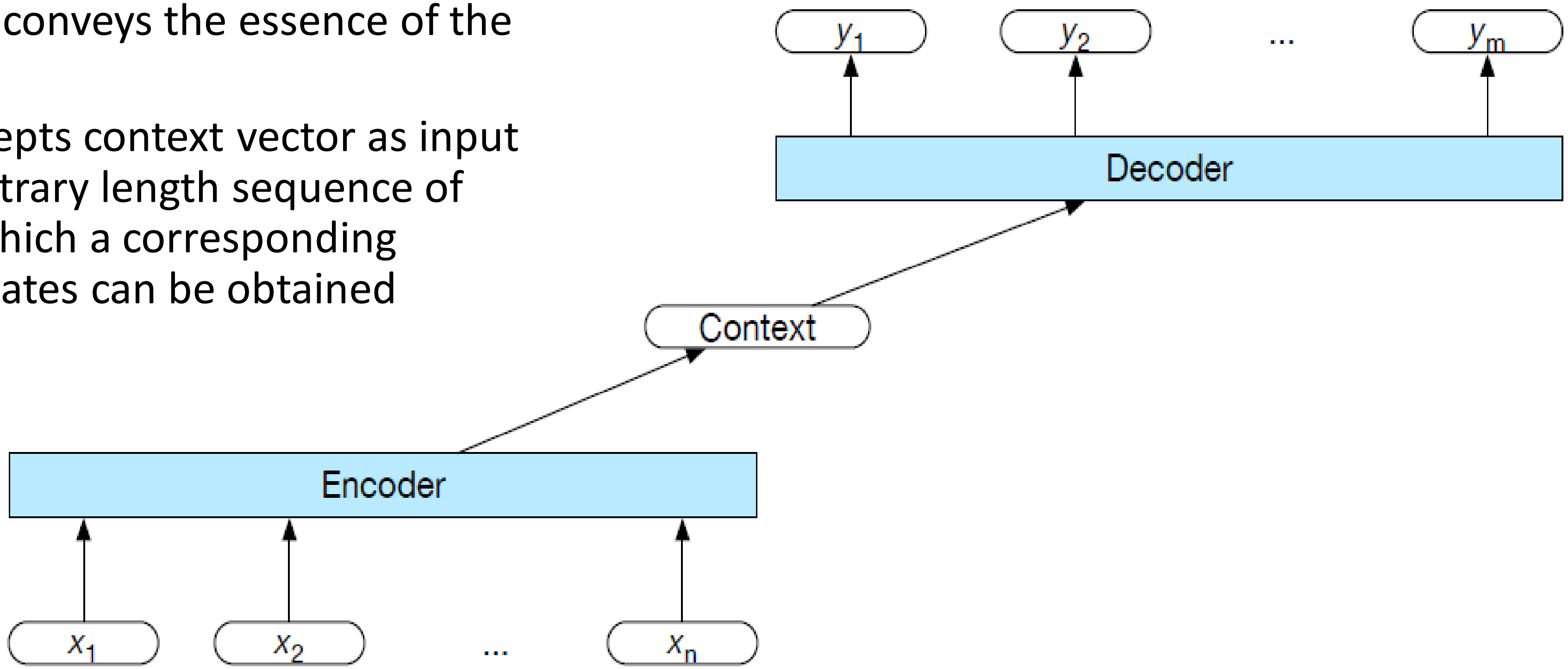
1. An **encoder** takes an input sequence $x^n{}_1$, and generates a corresponding sequence of contextualized representations, $h^n{}_1$.
2. A **context vector**, c, is a function of $h^n{}_1$, and conveys the essence of the input to the decoder.
3. A **decoder** accepts c as input and generates an arbitrary length sequence of hidden states $h^m{}_1$, from which can be used to create a corresponding sequence of output states $y^m{}_1$.

Encoder-decoder networks



Encoder-decoder networks

- An **encoder** that accepts an input sequence and generates a corresponding sequence of contextualized representations
- A **context vector** that conveys the essence of the input to the decoder
- A **decoder**, which accepts context vector as input and generates an arbitrary length sequence of hidden states, from which a corresponding sequence of output states can be obtained



Encoder

- Any kind of RNN or its variants can be used as an encoder.
  - simple RNNs, LSTMs, GRUs,
  - stacked Bi-LSTMs widely used

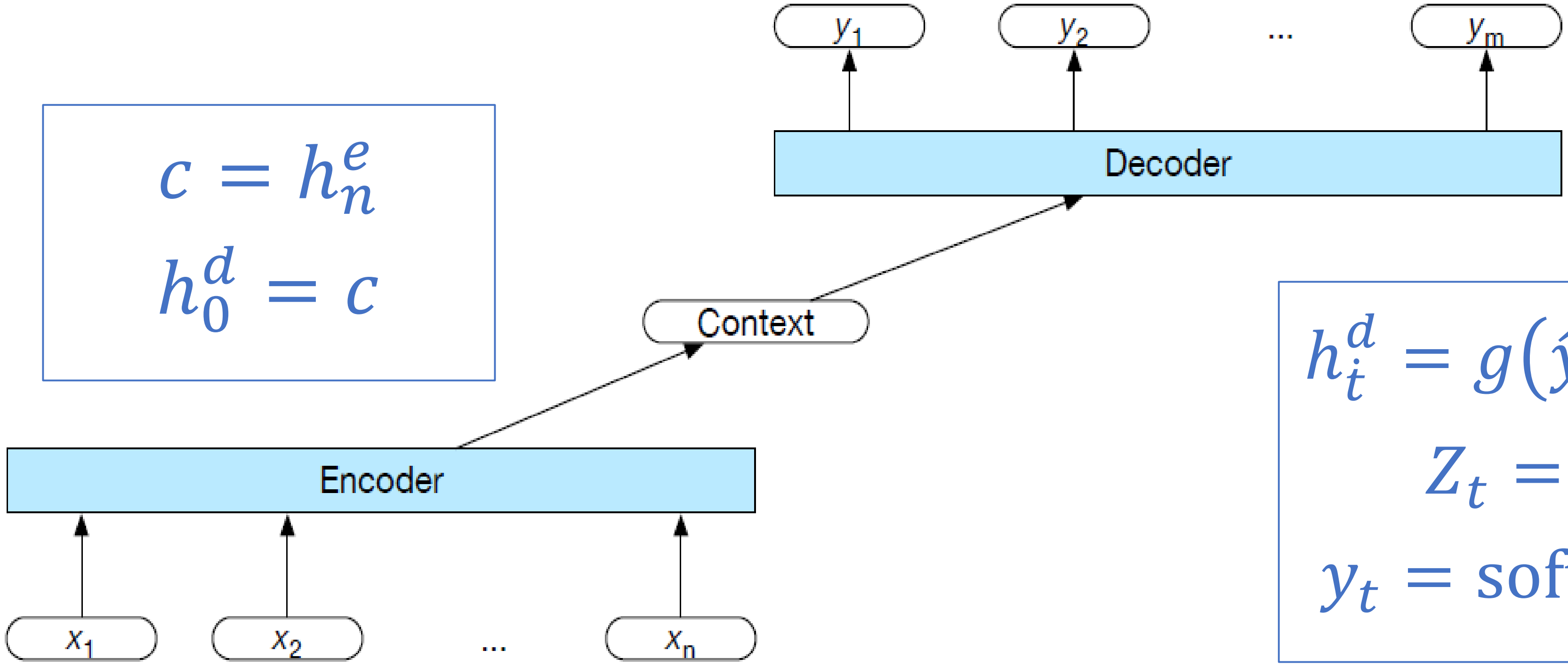
$$c = h_n^e$$
$$h_0^d = c$$

Decoder

- Autoregressive generation is used to produce an output sequence, an element at a time, until an end-of-sequence marker is generated.
- This incremental process is guided by the context provided by the encoder as well as any items generated for earlier states by the decoder.

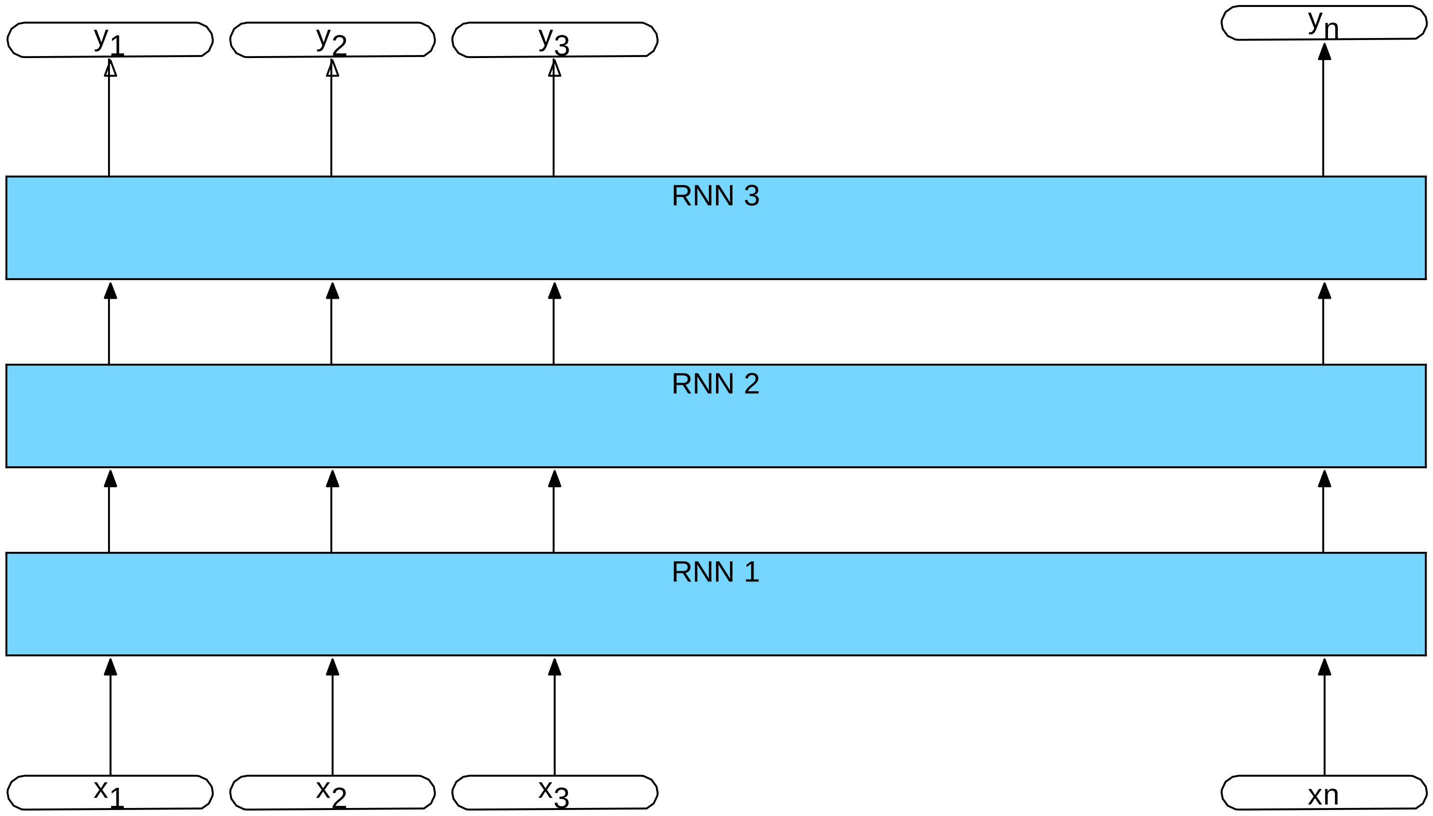
$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d)$$
$$Z_t = f(h_t^d)$$
$$y_t = \text{soft max}\ (z_t)$$

$$c = h_n^e$$
$$h_0^d = c$$

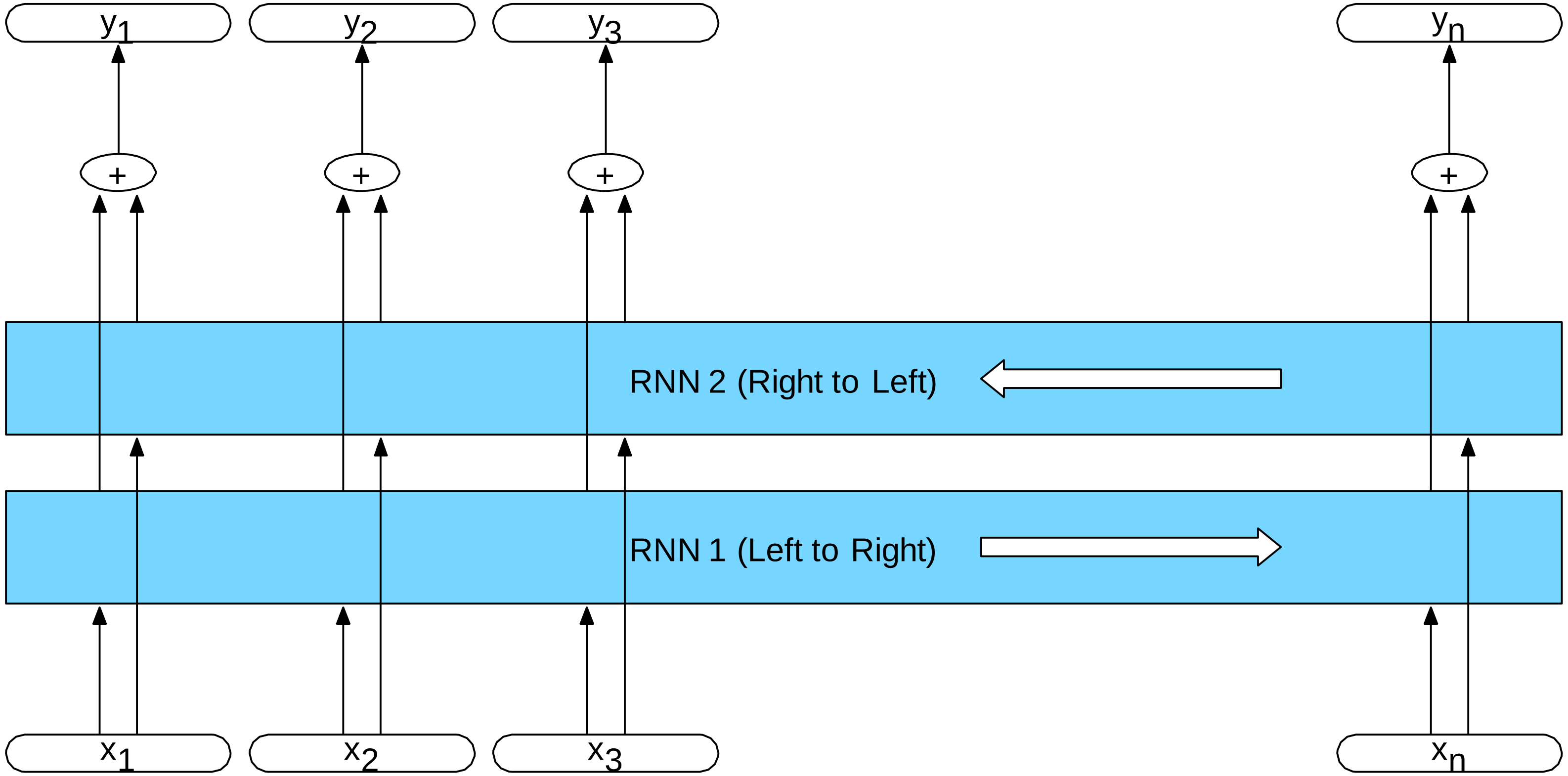


$$h_t^d = g\left(\hat{y}_{t-1}, h_{t-1}^d\right)$$
$$Z_t = f\left(h_t^d\right)$$
$$y_t = \text{soft max }(z_t)$$

Encoder: Stacked RNN



Encoder: Bidirectional RNNs



Decoder

- Autoregressive generation is used to produce an output sequence, an element at a time, until an end-of-sequence marker is generated.
- This incremental process is guided by the context provided by the encoder as well as any items generated for earlier states by the decoder.

$$c = h_n^e$$
$$h_0^d = c$$

$$h_t^d = g\left(\hat{y}_{t-1}, h_{t-1}^d\right)$$
$$Z_t = f\left(h_t^d\right)$$
$$y_t = \mathrm{soft\,max}\ (z_t)$$

Decoder Weaknesses

- The context vector *c* was only directly available at the beginning of the generation process.
- This meant that its influence became less-and-less important as the output sequence was generated.
- One solution is to make *c* available at each step in the decoding process,

1. when generating the hidden states in the deocoder, and
2. while producing the generated output.

$$c = h_n^e$$

$$h_0^d = c$$

$$h_t^d = g\left(\hat{y}_{t-1}, h_{t-1}^d, c\right)$$

$$Z_t = f\left(h_t^d\right)$$

$$y_t = \text{soft max}\ (\hat{y}_{t-1}, z_t, c)$$

Choosing the best output

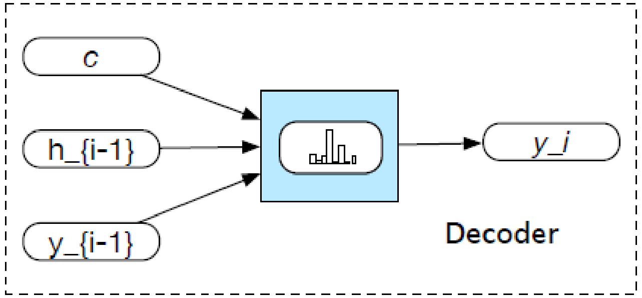
- For neural generation, where we are trying to generate novel outputs, we can simply sample from the softmax distribution.
- In MT where we're looking for a specific output sequence, sampling isn't appropriate and would likely lead to some strange output.
- Instead we choose the most likely output at each time step by taking the argmax over the softmax output

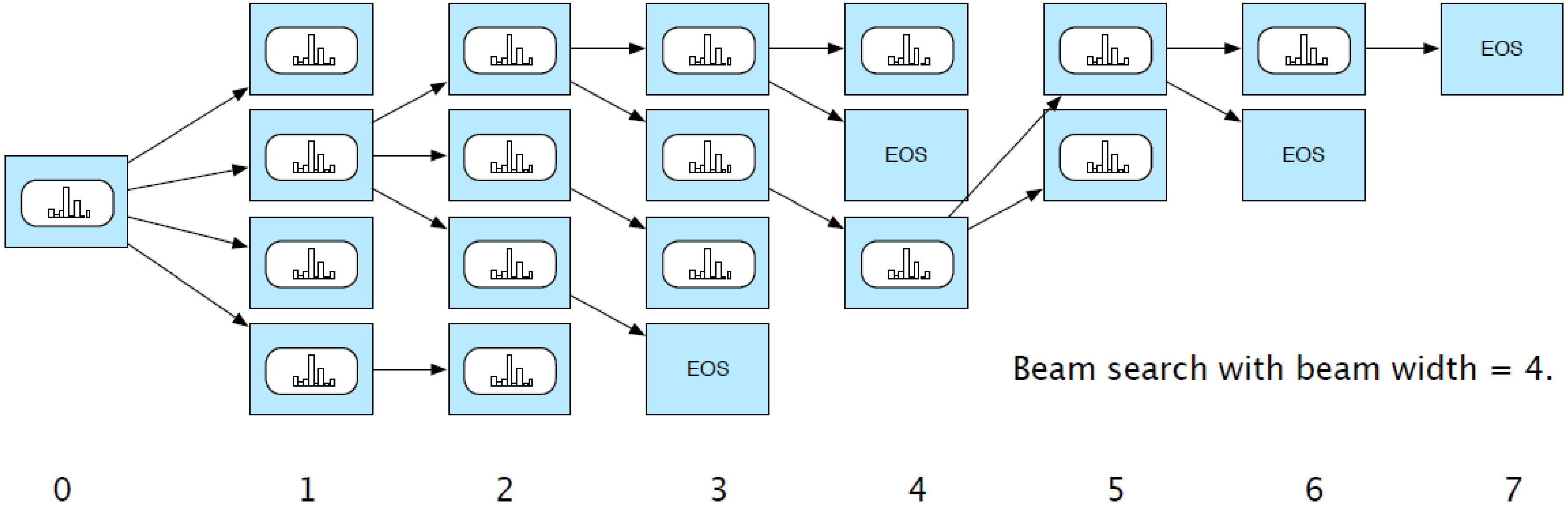
$$\hat{y} = \text{argmax} P(y_i | y_{<} i)$$

Beam search

- In order to systematically explore the space of possible outputs for applications like MT, we need to control the exponential growth of the search space.
- Beam search: combining a breadth-first-search strategy with a heuristic filter that scores each option and prunes the search space to stay within a fixed-size memory footprint, called the beam width

Beam search





Beam search with beam width = 4.

Attention

- Weaknesses of the context vector:

- Only directly available at the beginning of the process and its influence will wane as the output sequence is generated

- Context vector is a function (e.g. last, average, max, concatenation) of the hidden states of the encoder. This approach loses useful information about each of the individual encoder states

- Potential solution: **attention mechanism**

Attention

- Replace the static context vector with one that is dynamically derived from the encoder hidden states at each point during decoding
- A new context vector is generated at each decoding step and takes all encoder hidden states into derivation
- This context vector is available to decoder hidden state calculations

$$h_i^d = g\left(\hat{y}_{i-1}, h_{i-1}^d, c_i\right)$$

Attention

- To calculate c_i, first find relevance of each encoder hidden state to the decoder state. Call it $score\left(h_{i-1}^d, h_j^e\right)$ for each encoder state j
- The $score$ can simply be dot product,

$$score\left(h_{i-1}^d, h_j^e\right) = h_{i-1}^d \cdot h_j^e$$

Attention

- The score can also be parameterized with weights

$$score\left(h_{i-1}^{d}, h_{j}^{e}\right) = h_{i-1}^{d} W_s \, h_{j}^{e}$$

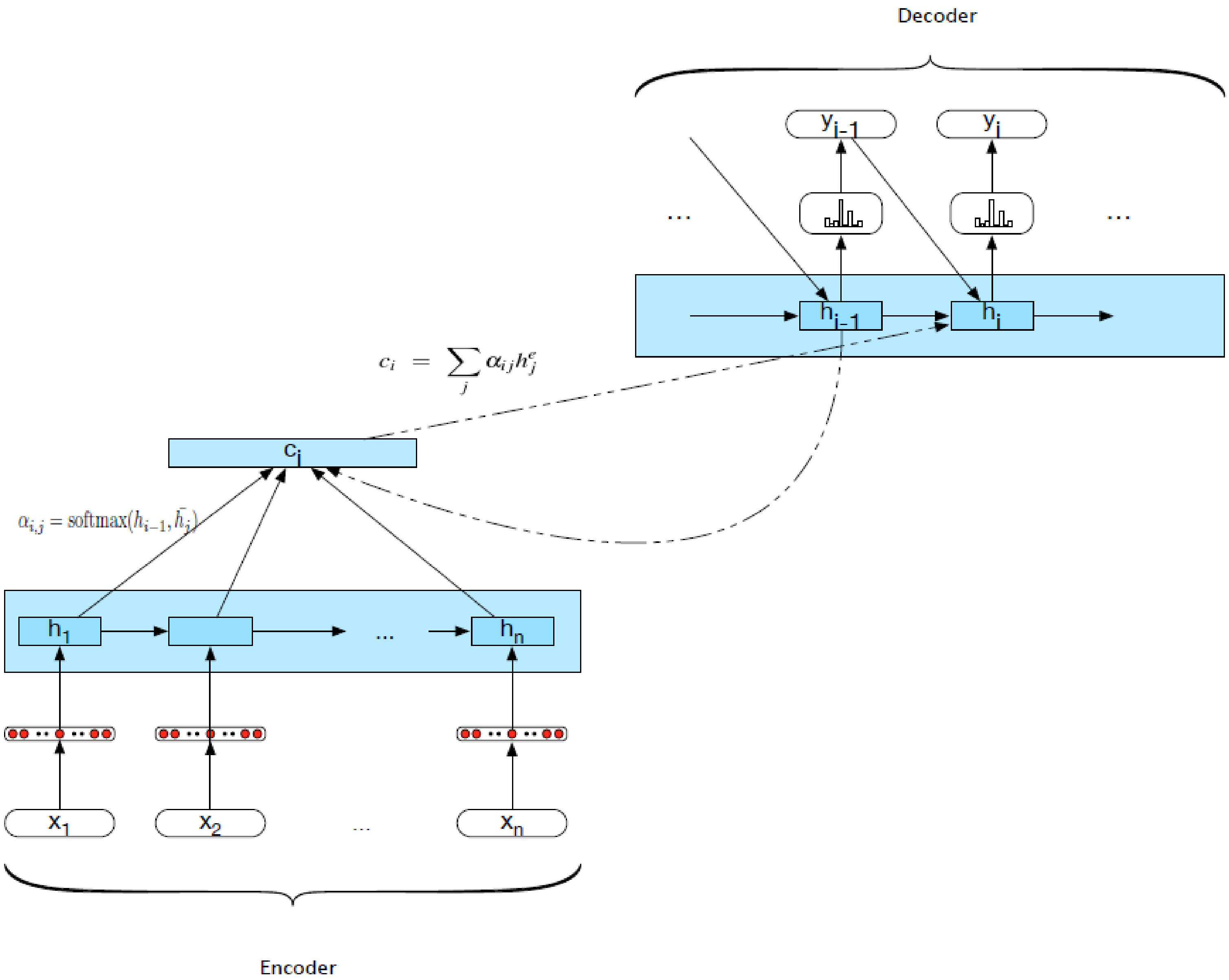
- Normalize them with a softmax to create a vector of weights $\alpha_{i,j}$ that tells us the proportional relevance of each encoder hidden state j to the current decoder state i

$$\alpha_{i,j} = softmax\left(score\left(h_{i-1}^{d}, h_{j}^{e}\right) \forall j \in e\right)$$

- Finally, context vector is the weighted average of encoder hidden states

$$c_i = \sum_{j} \alpha_{i,j} h_{j}^{e}$$

Attention mechanism



Applications of Encoder- Decoder Networks

- Text summarization
- Text simplification
- Question answering
- Image captioning
- …