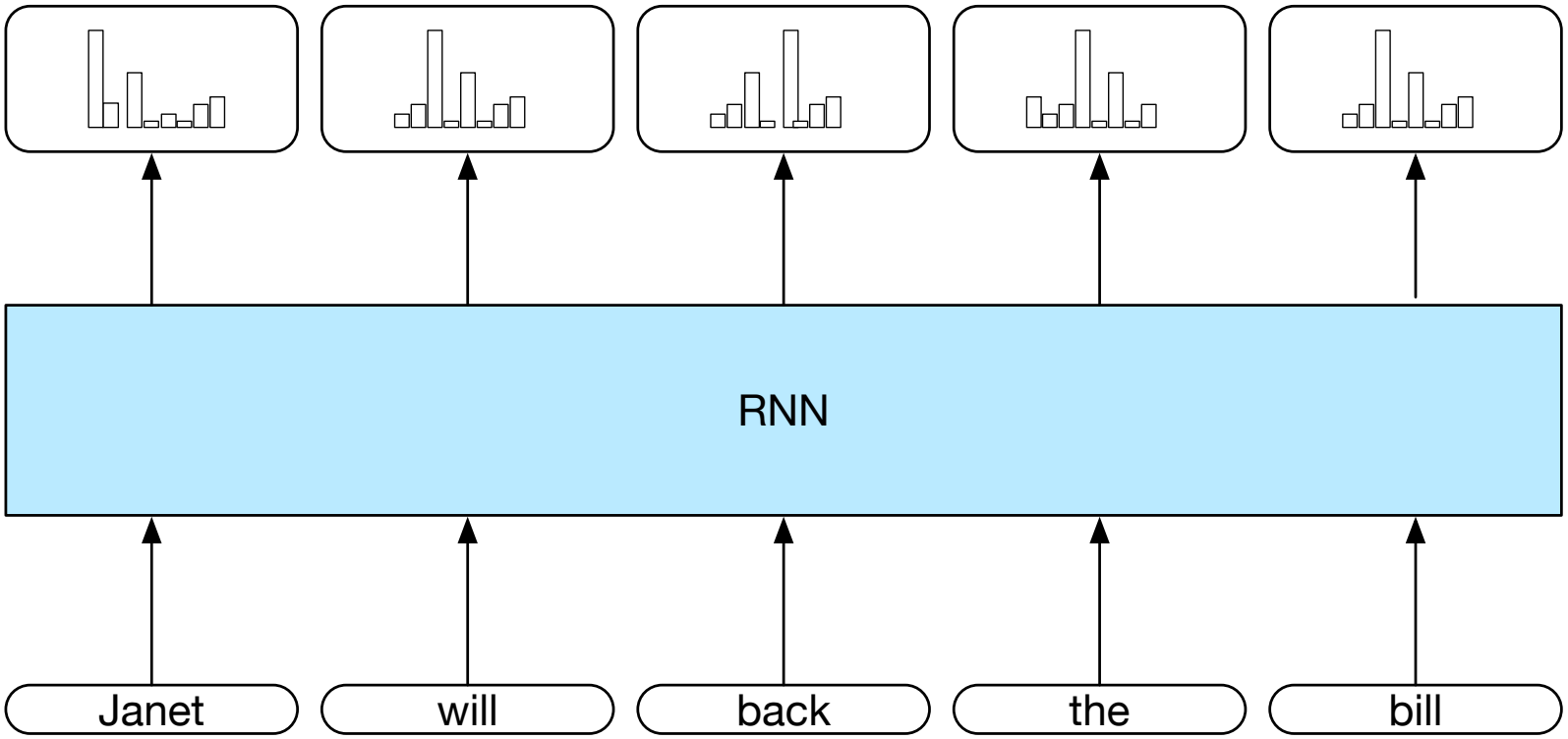
Encoder-Decoder Models

JURAFSKY AND MARTIN CHAPTER 10

Review: Recurrent Neural Networks (RNNs)

RNNs can be used for language modeling and sequence labeling. **Transduction** is the general process of taking in an input sequence and transforming it into output sequences in a one-to-one fashion.

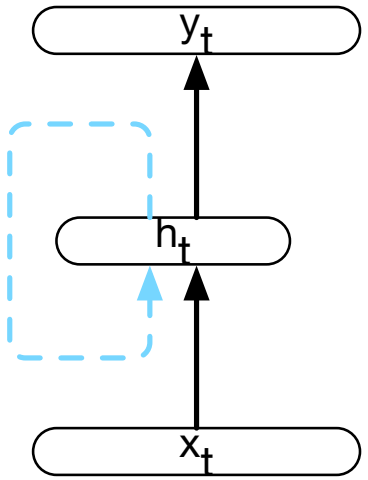


Review: Recurrent Neural Networks (RNNs)

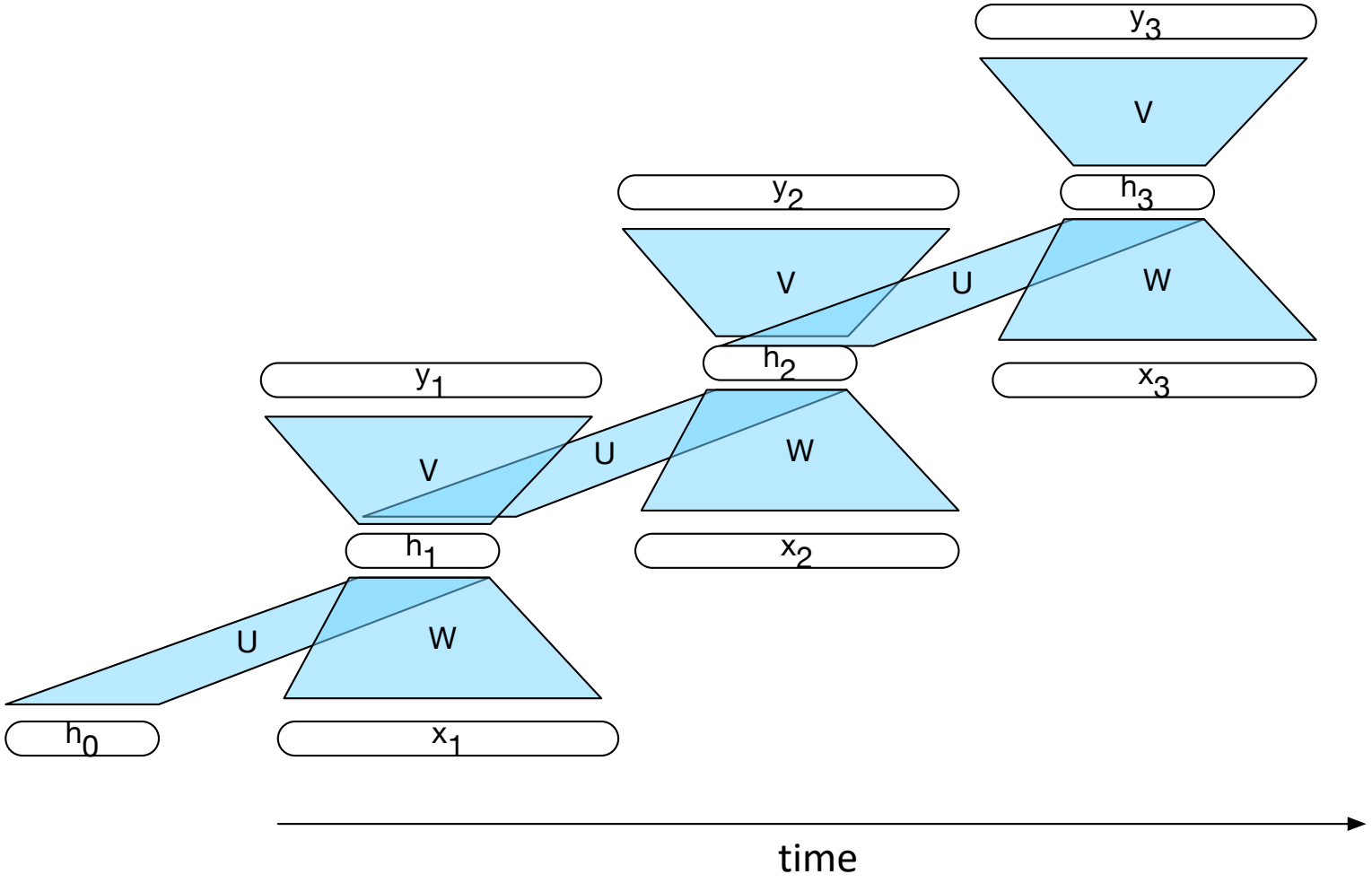
A recurrent neural network (RNN) is any network that contains a cycle within its network.

In such networks the value of a unit can be dependent on earlier outputs as an input.

RNNs have proven extremely effective when applied to NLP.



Review: Unrolled RNN



Review: Recurrent Neural Language Models

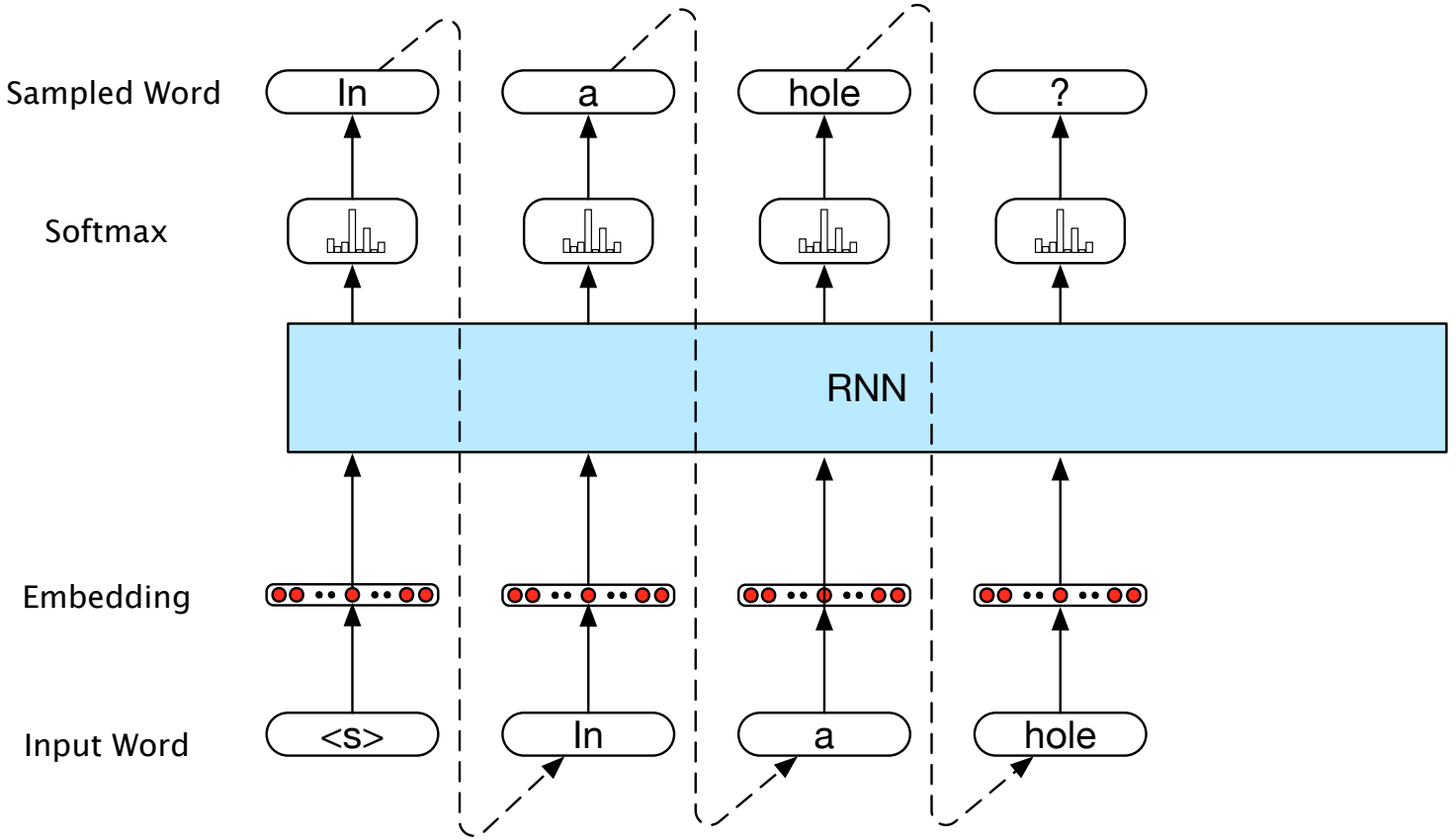
Unlike n-gram LMs and feedforward networks with sliding windows, RNN LMs don't use a fixed size context window.

They predict the next word in a sequence by using the current word and the previous hidden state as input.

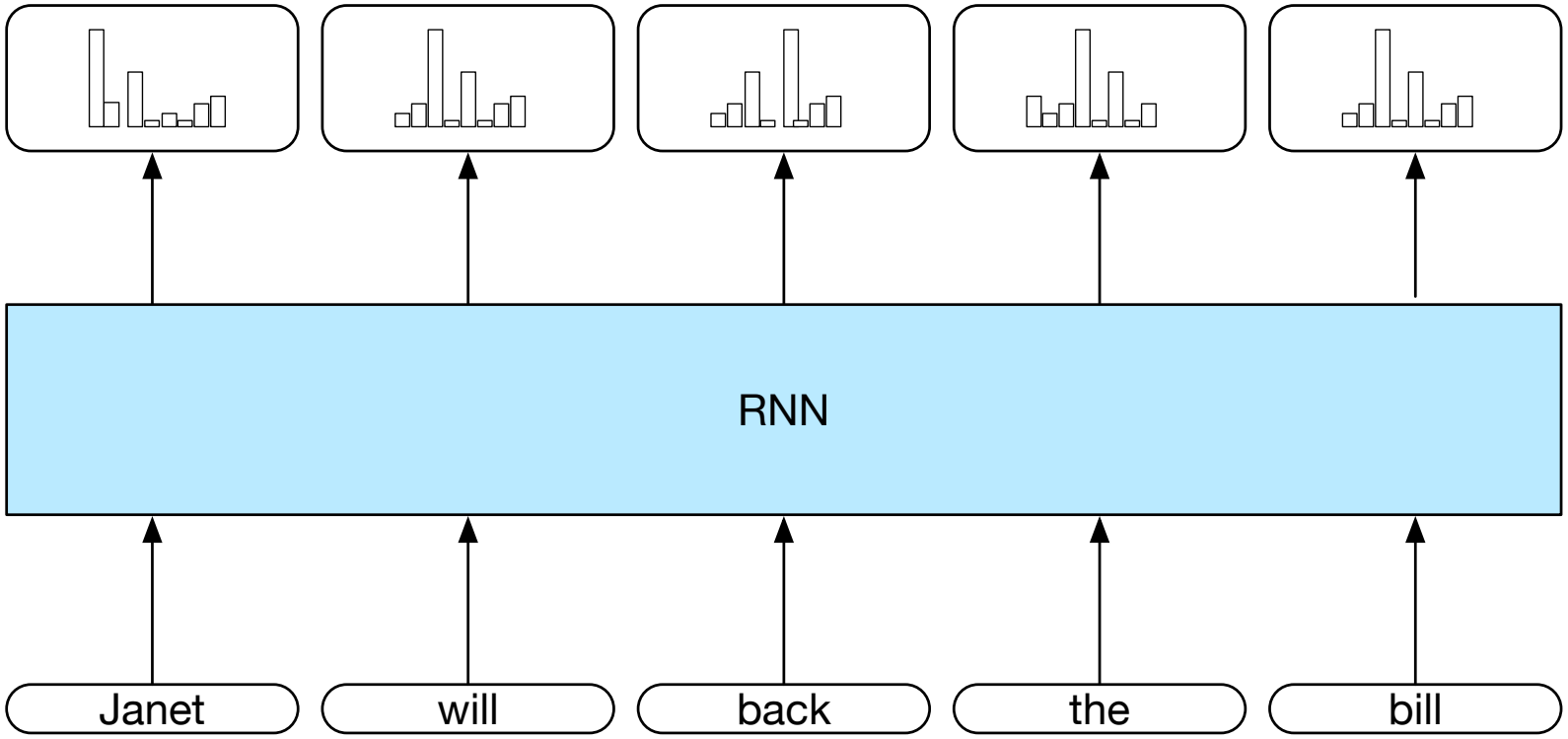
The hidden state embodies information about all of the preceding words all the way back to the beginning of the sequence.

Thus they can potentially take more context into account than n-gram LMs and NN LMs that use a sliding window.

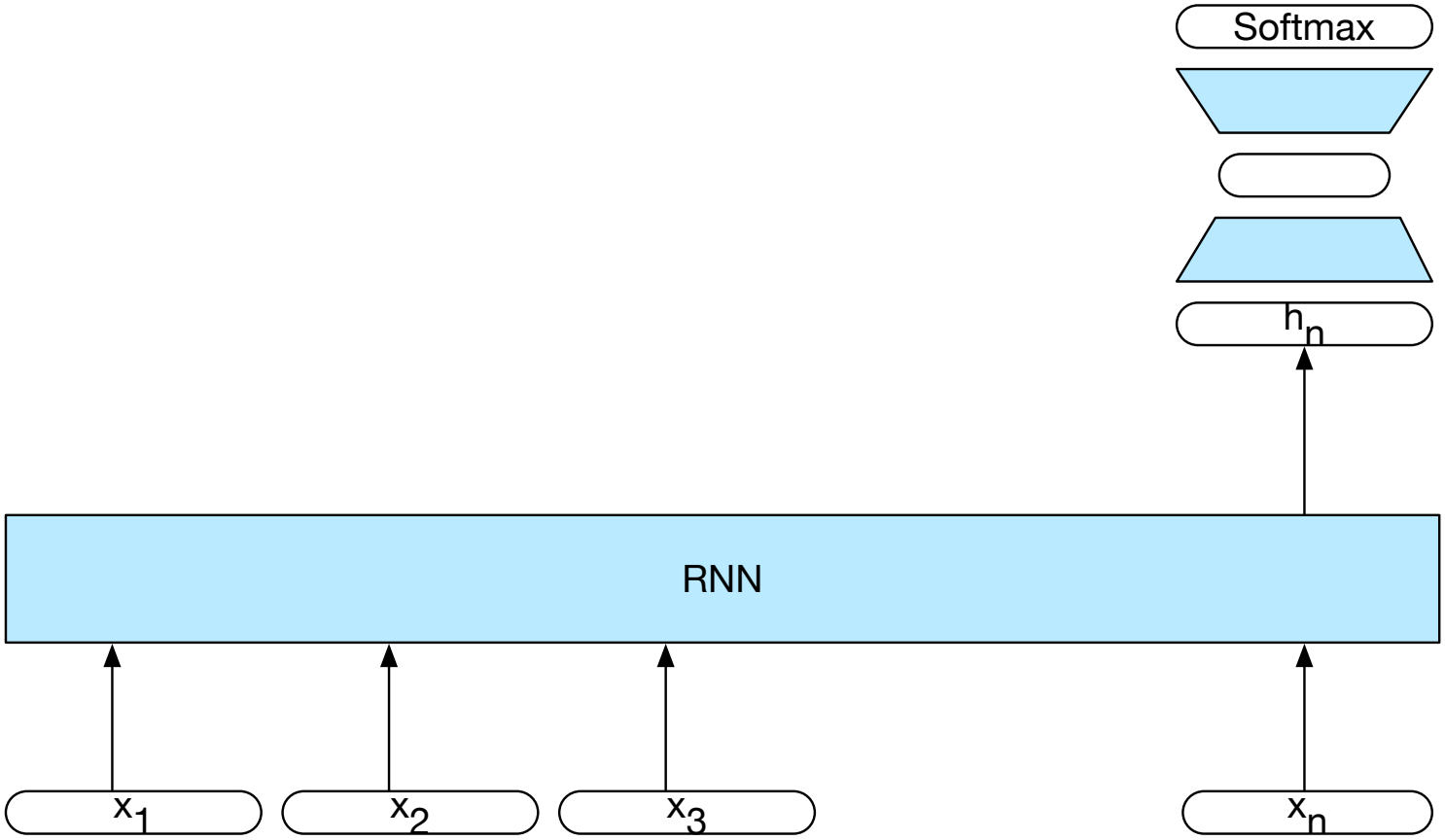
Generation with an RNN LM



Tag Sequences



Sequence Classifiers



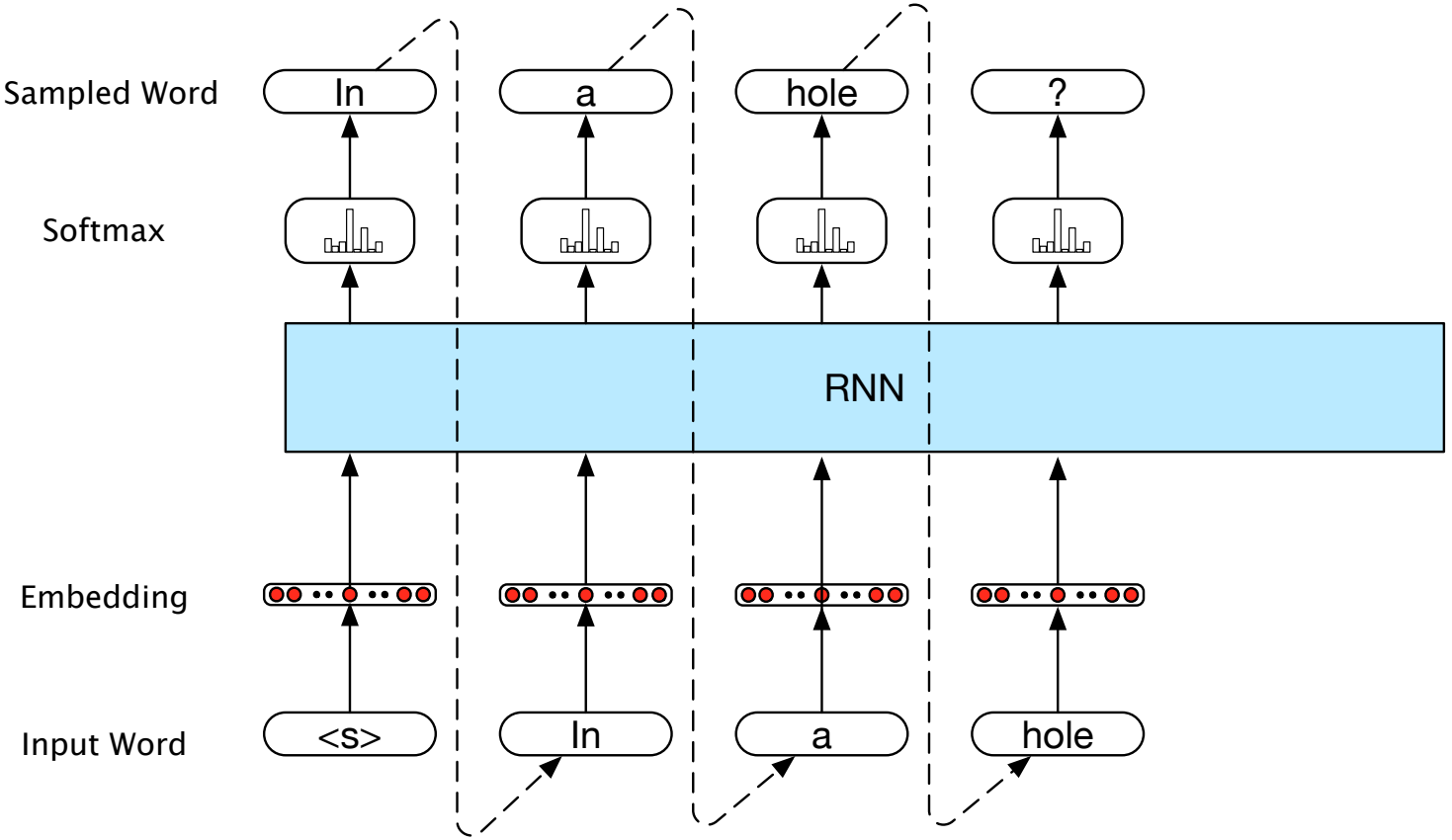
Encoder-Decoder networks

Encoder-decoder networks are a kind of **sequence-to-sequence model.** Unlike vanilla RNNs, they can generate contextually appropriate, **arbitrary length**, output sequences.

They are useful for a wide range of NLP applications including:

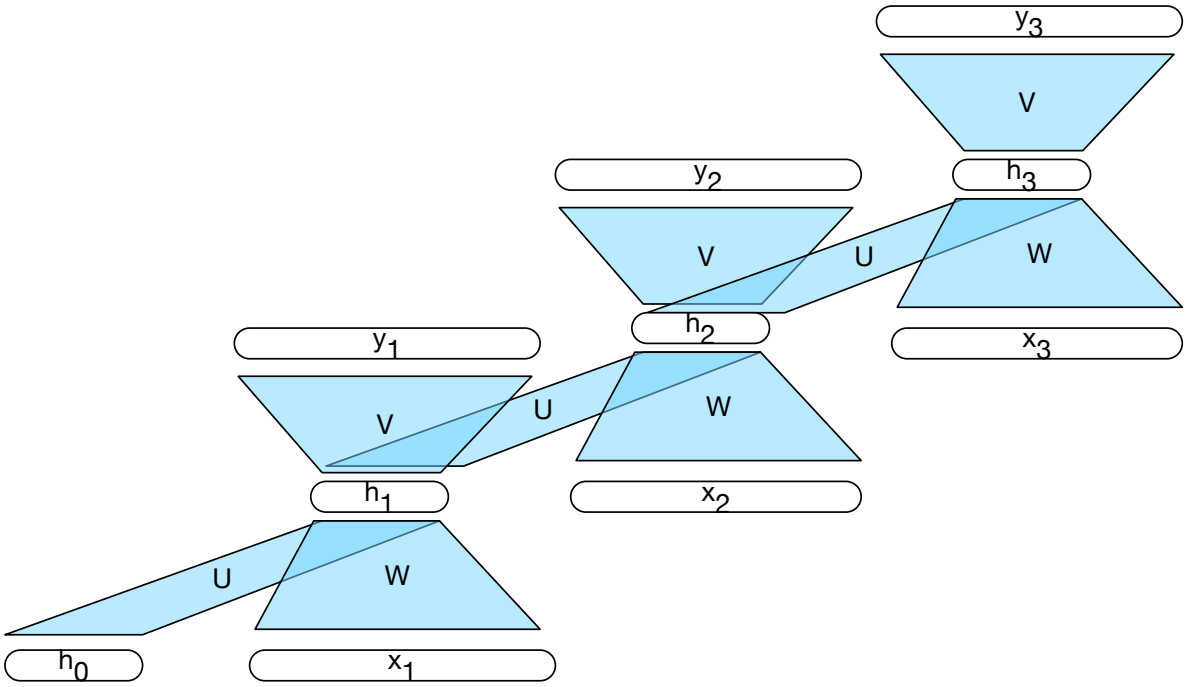
1. Machine Translation
2. Automatic summarization
3. Question answering
4. Dialog modelling

Auto-Regressive Generation with an RNN LM

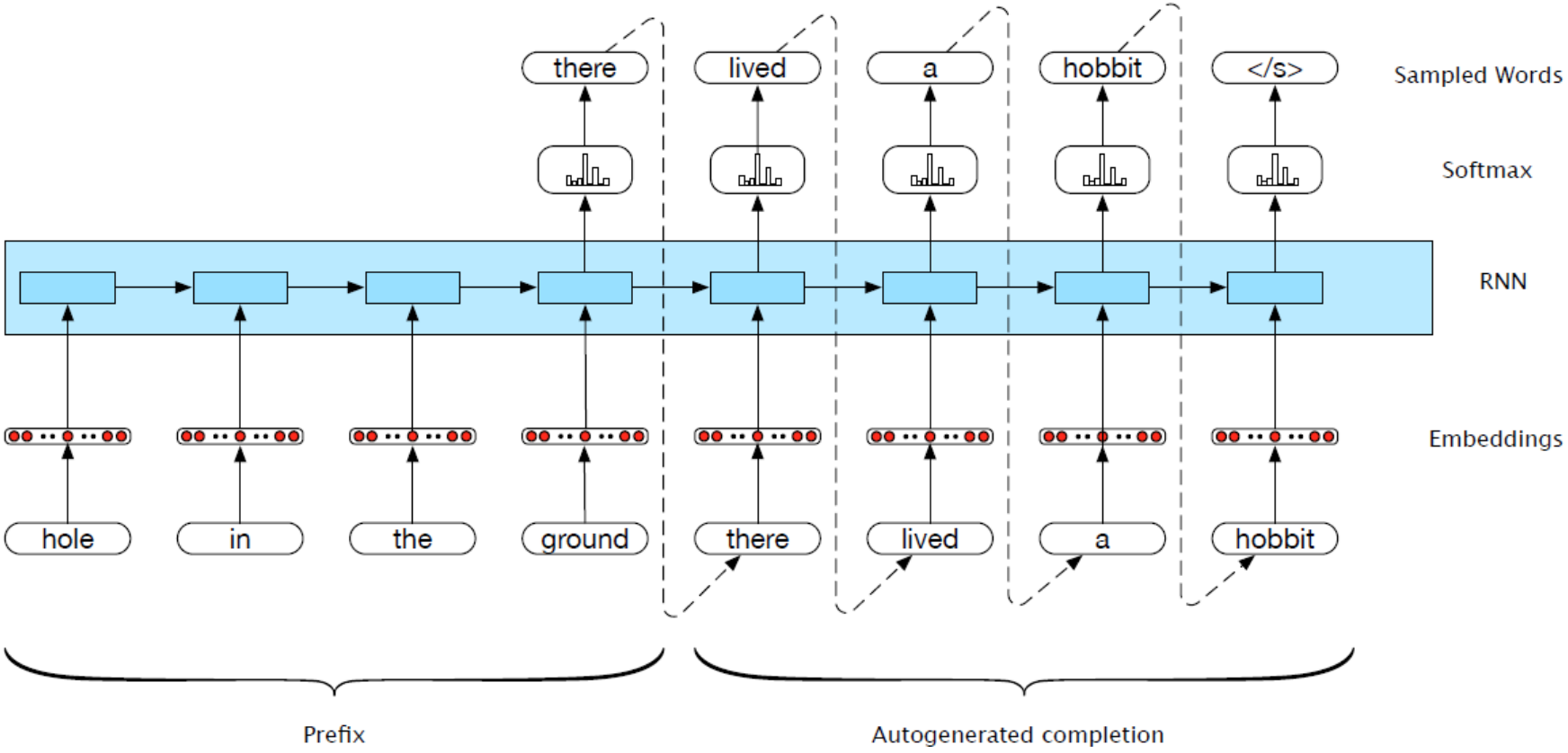


Recall: autoregressive generation

- $h_t = g(Uh_{t-1} + Wx_t),\ y_t = f(Vh_t)$
- f is a softmax over the set of possible outputs



Generation with prefix



Machine Translation (MT)

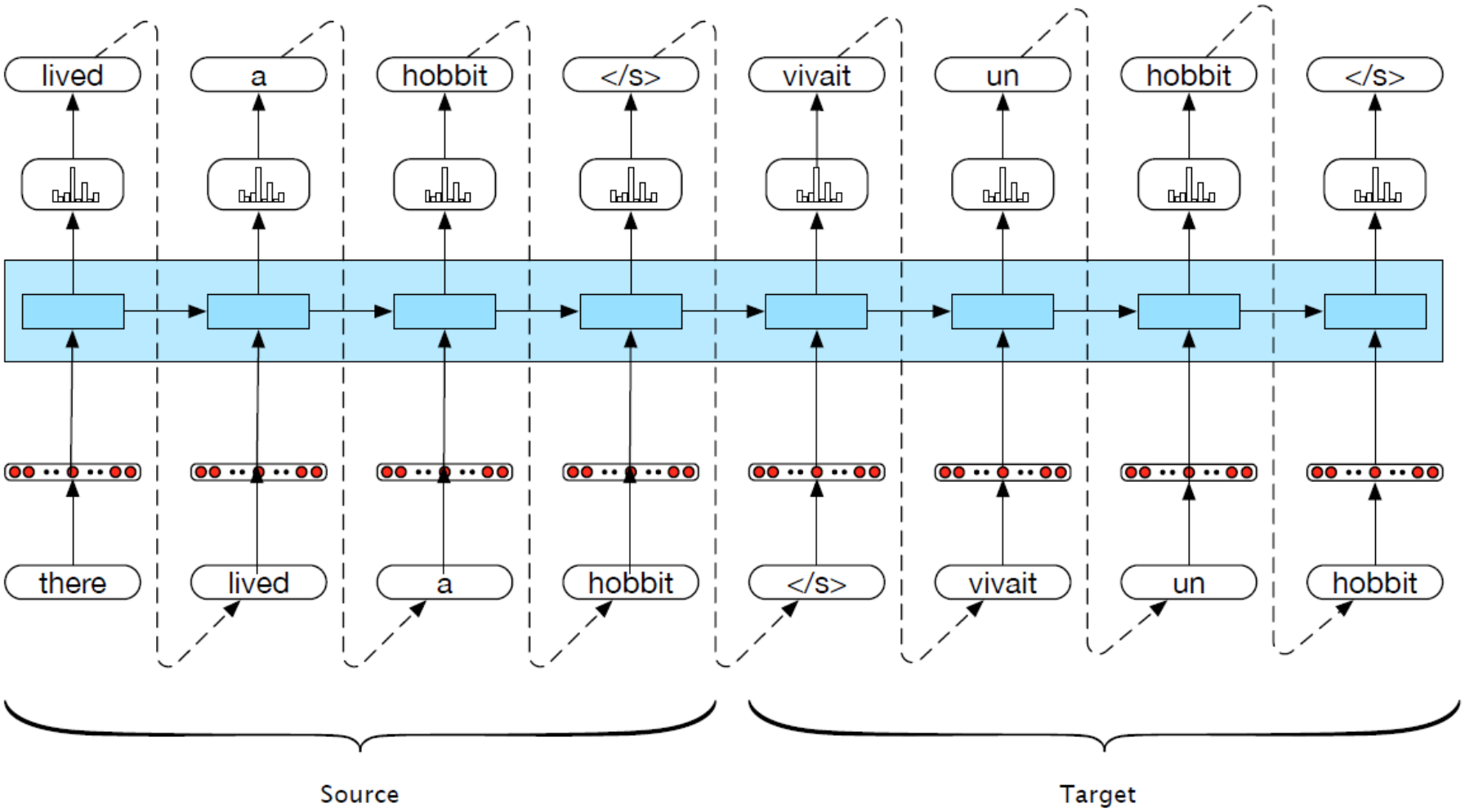
MT is the task of automatically translating sentences from one language into another.

We use bilingual parallel texts to train MT systems – pairs of **source-target** sentences that are translations of each other.

To extend LMs and autoregressive generation to MT, we will:

1. Add an end-of-sentence marker to each source sentence. Concatenate the target sentence to it.
2. Train an RNN LM based on this combined data.
3. To translate, simply treat the input sentence as a prefix, create a hidden state representation for it **(encoding step)**.
4. Use the hidden state produced by the encoder to then start generating (**decoding step)**

Machine translation

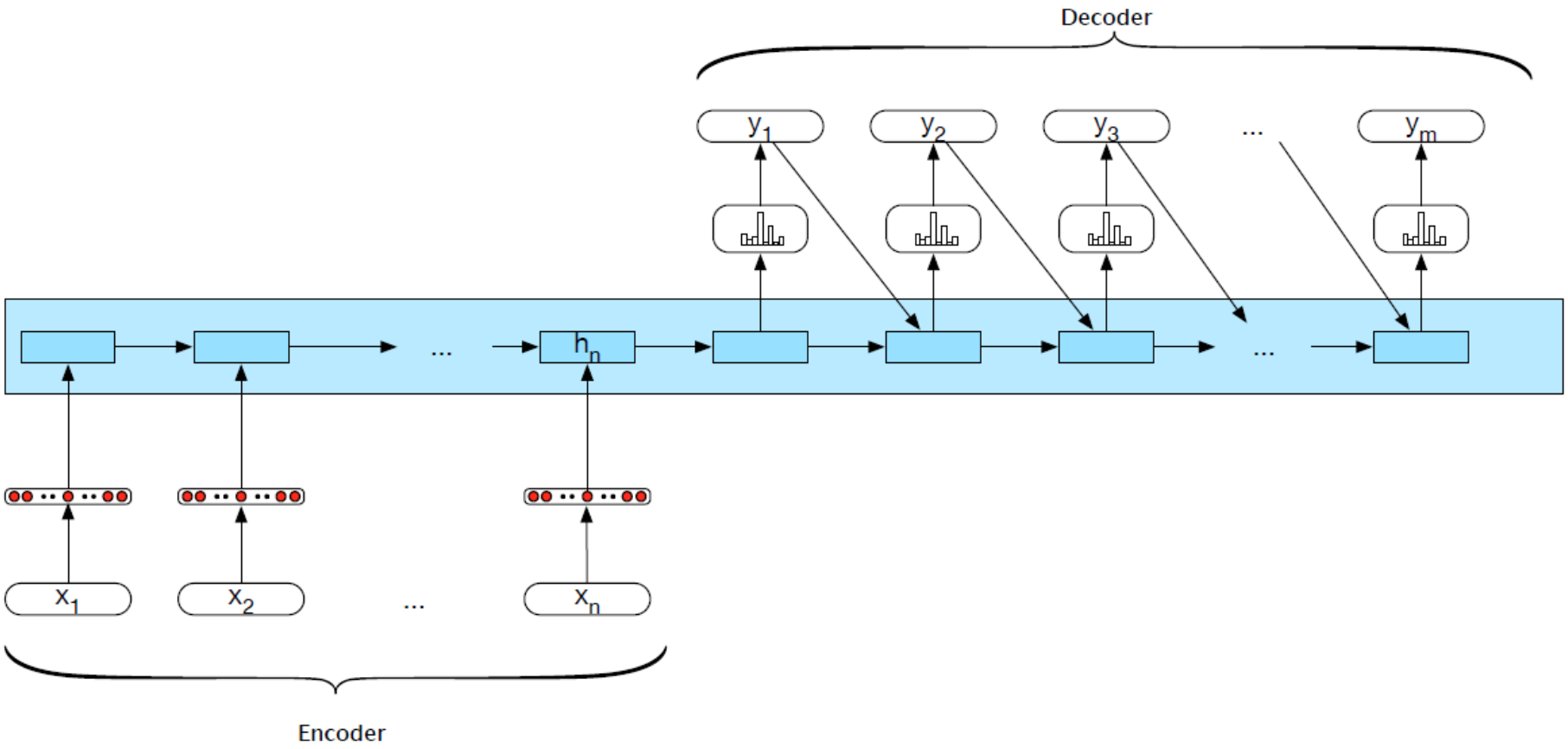


Encoder-Decoder Networks

We can abstract away from the task of MT to talk about the general **encoder-decoder architecture:**

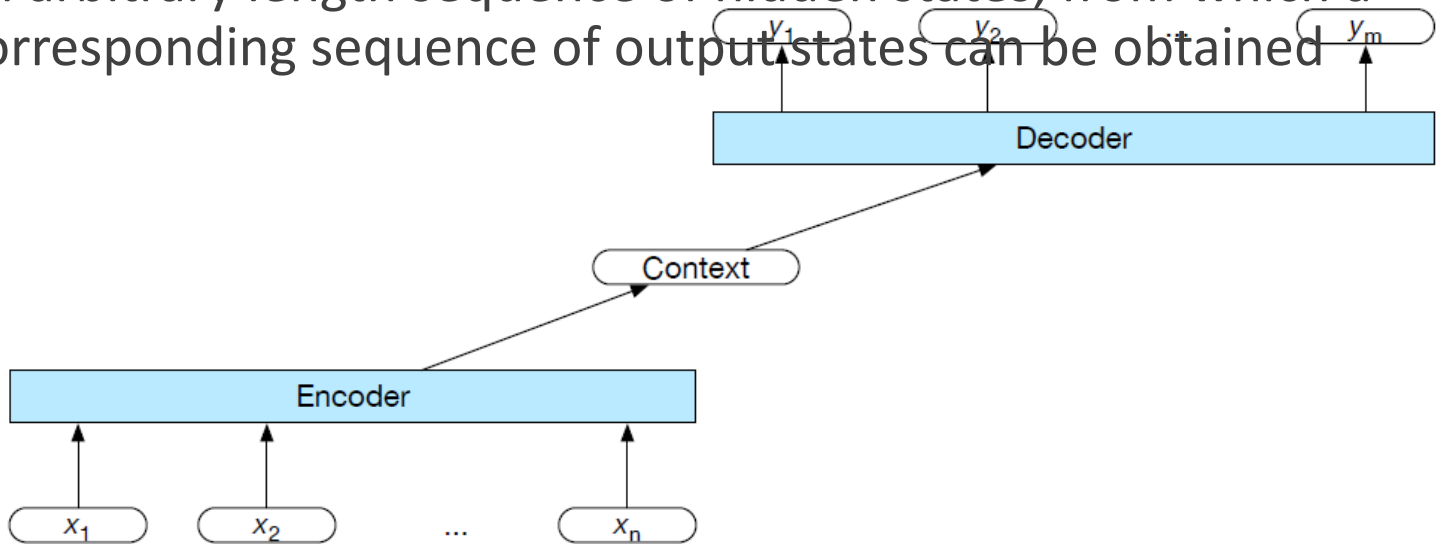
1. An **encoder** takes an input sequence x^n_1, and generates a corresponding sequence of contextualized representations, h^n_1.
2. A **context vector**, *c*, is a function of h^n_1, and conveys the essence of the input to the decoder.
3. A **decoder** accepts *c* as input and generates an arbitrary 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

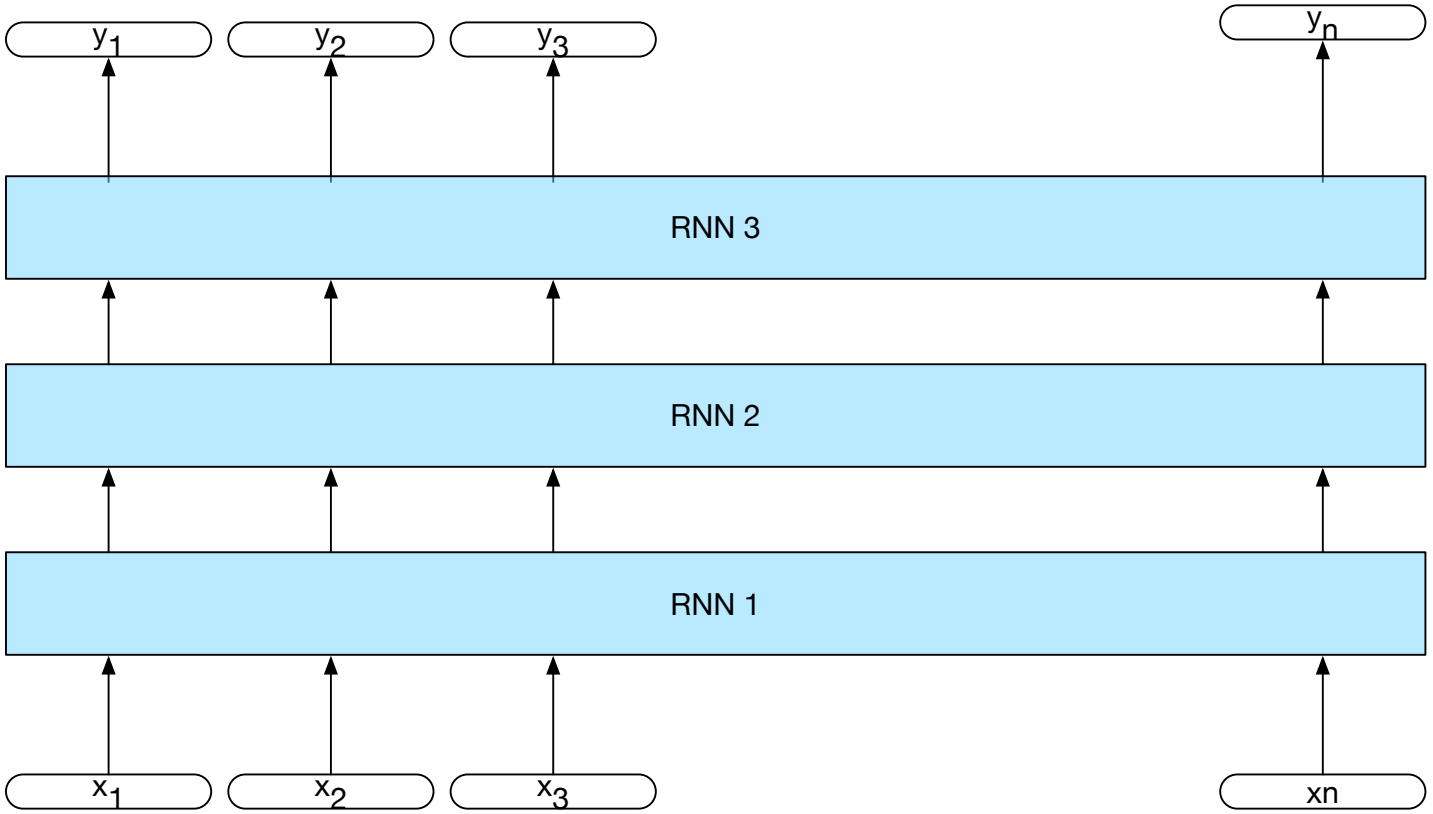


Encoder

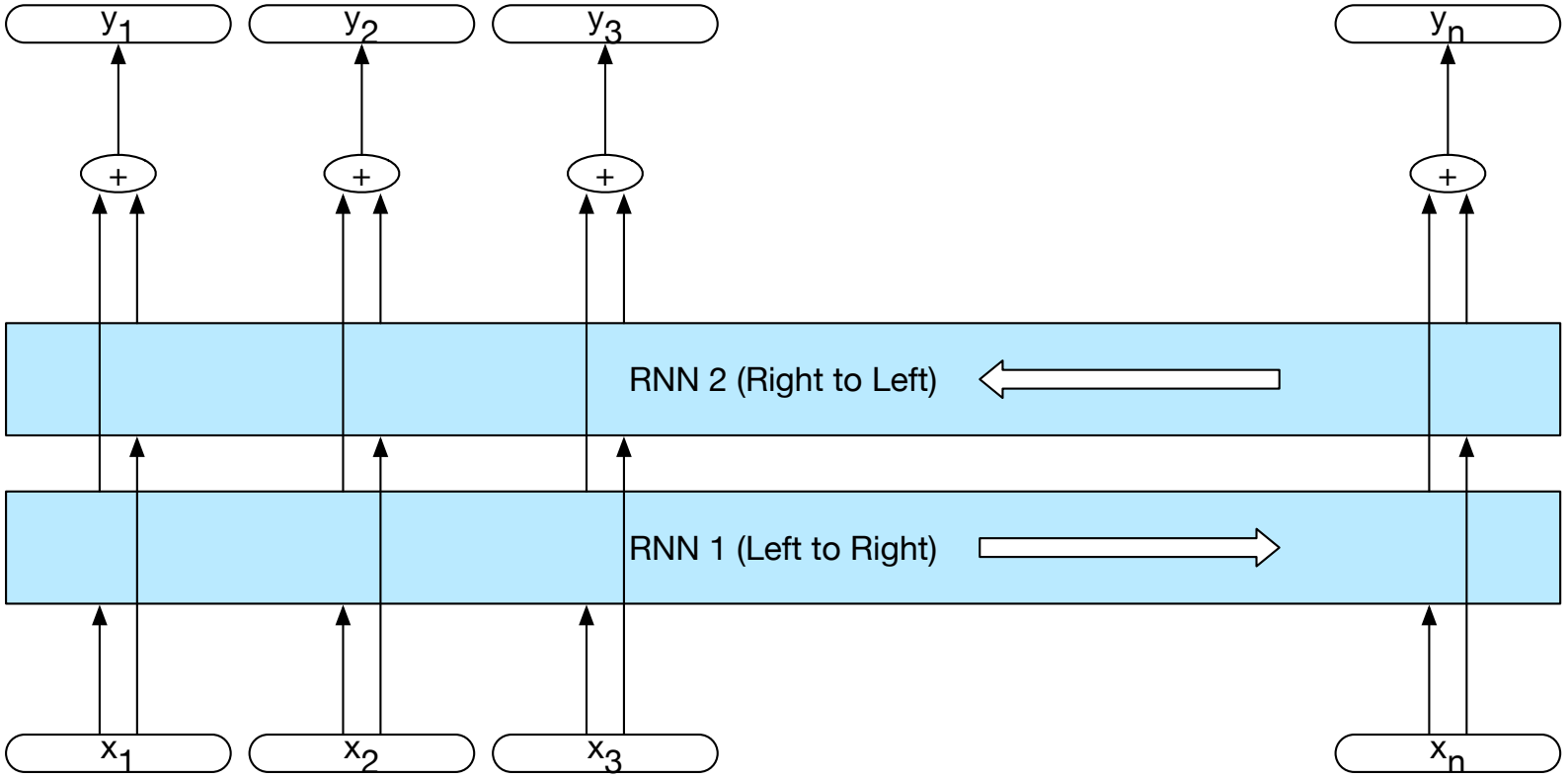
Pretty much any kind of RNN or its variants can be used as an encoder. Researchers have used simple RNNs, LSTMs, GRUs, or even convolutional networks.

A widely used encoder design makes use of stacked Bi-LSTMs where the hidden states from top layers from the forward and backward passes are concatenated

Stacked RNNs



Bidirectional RNNs



Decoder

For the 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.

Encoder

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

Decoder

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

Decoder Weaknesses

In early encoder-decoder approaches, the context vector *c* was only directly available at the beginning of the generation process.

This meant that its influence became less-and-less imporant as the output sequence was generated.

One solution is to make *c* available at each step in the decoding process, when generating the hidden states in the deocoder

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

and while producing the generated output.

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

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

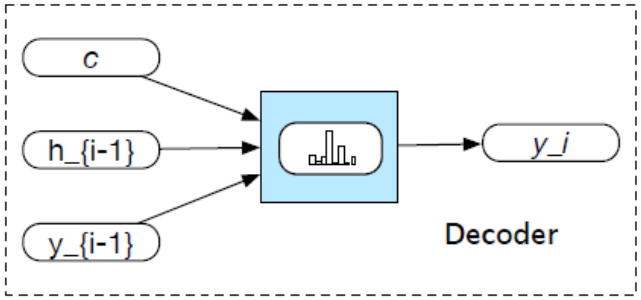
$$\hat{y} = \text{argmax} P(y_i | y_< i)$$

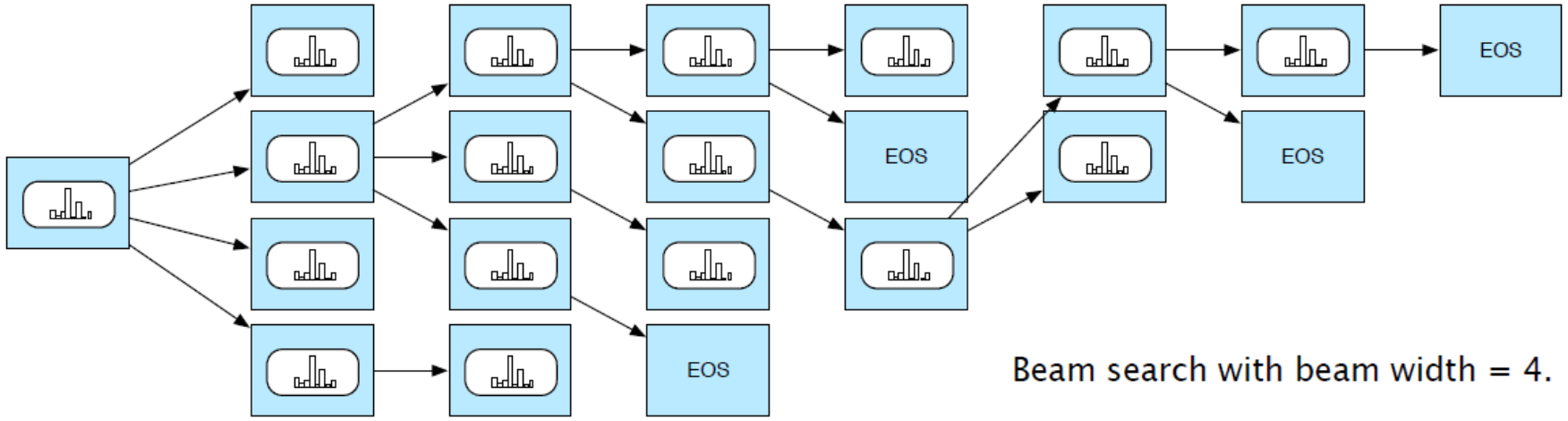
Beam search

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


c
h_{i-1}
y_{i-1}
y_i
Decoder


EOS
EOS
EOS
EOS
Beam search with beam width = 4.
0
1
2
3
4
5
6
7

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 mechanism

- 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 mechanism

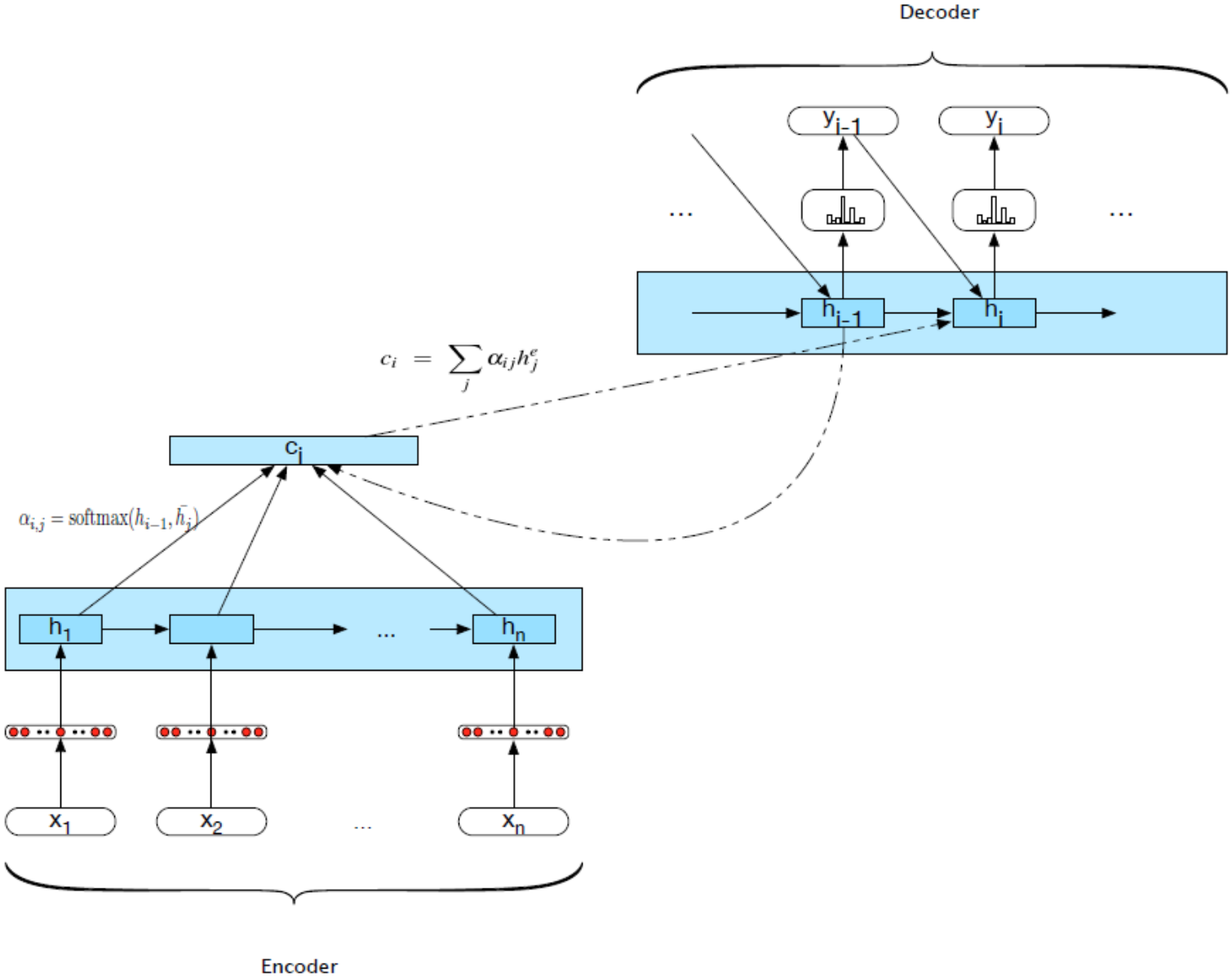
- To calculate c_i, first find relevance of each encoder hidden state to the decoder state. Call it $score(h_{i-1}^d, h_j^e)$ for each encoder state j
  - The $score$ can simply be dot product, or be parameterized with weights
- Normalize them with a softmax to create a vector of weights $\alpha_{i,j}$ that tells us the proportional relevance of each encoder hidden state j to the current decoder state i

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

- 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
- And more. What do those tasks have in common?