

Deep Learning Architecture for Sequence Processing

Kwankamol Nongpong

CS4430 Selected Topic in Natural Language Processing and Social Interaction

CSX4210 / ITX4210 Natural Language Processing and Social Interaction

Vincent Mary School of Science and Technology

Assumption University

Temporal Nature of Language

- The temporal nature of language is reflected in the metaphors we use;
 - flow of conversations
 - news feeds,
 - and twitter streams
- All of which call out the notion that language is a sequence that **unfolds in time**.
- The temporal nature is reflected in the algorithms we use to process language.

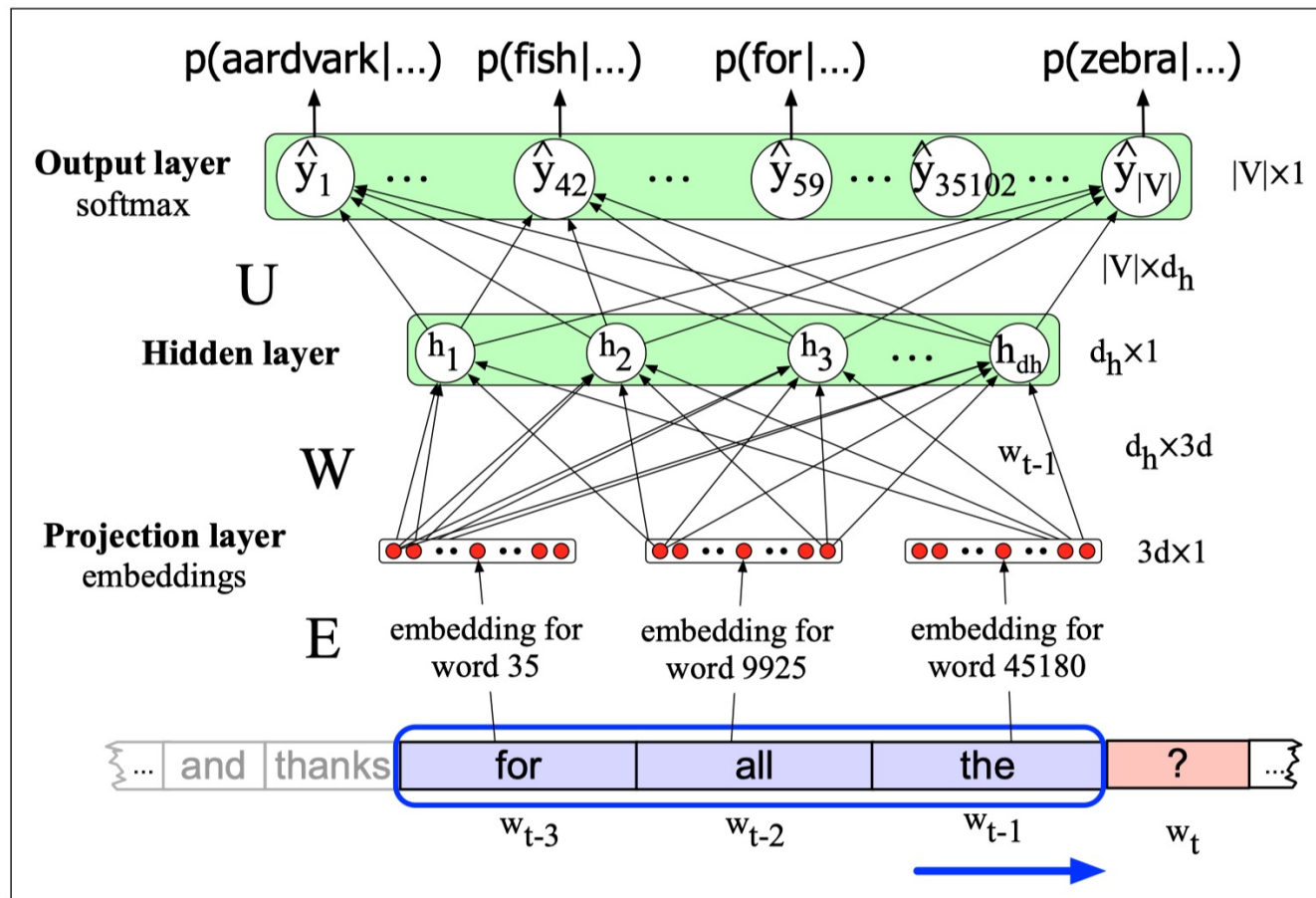
Issues with some ML Techniques

- ML approaches previously discussed (before the midterm) in Sentiment Analysis & Classification Tasks don't have temporal nature.
 - They assume **simultaneous access** to all aspects of their input.
 - Difficult to deal with sequences of **varying length**.
 - Fails to capture important **temporal aspects** of language.
- What's the workaround for such problems?
 - Sliding window

Sliding Window

- Sliding window with neural language model.
- The model accepts **fixed-sized windows** of tokens as input
 - sequences longer than the window size are processed by walking through the input making predictions along the way, with the end result being a sequence of predictions spanning the input.
- Decisions made in one window have no impact on subsequent decisions.

Sliding Window



Problems with Sliding Window

- It shares the primary weakness of our earlier Markov N-gram approaches in that it **limits the context** from which information can be extracted
 - Anything outside the context window has no impact on the decision being made.
- The use of windows makes it difficult for networks to learn systematic patterns arising from phenomena like **constituency**.

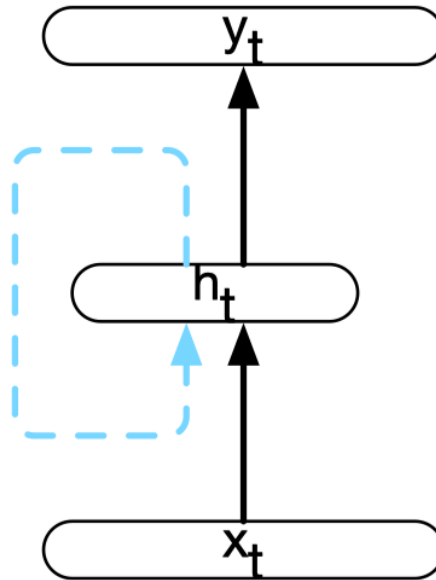
Handling such challenges

- Recurrent Neural Networks
- Transformer Networks

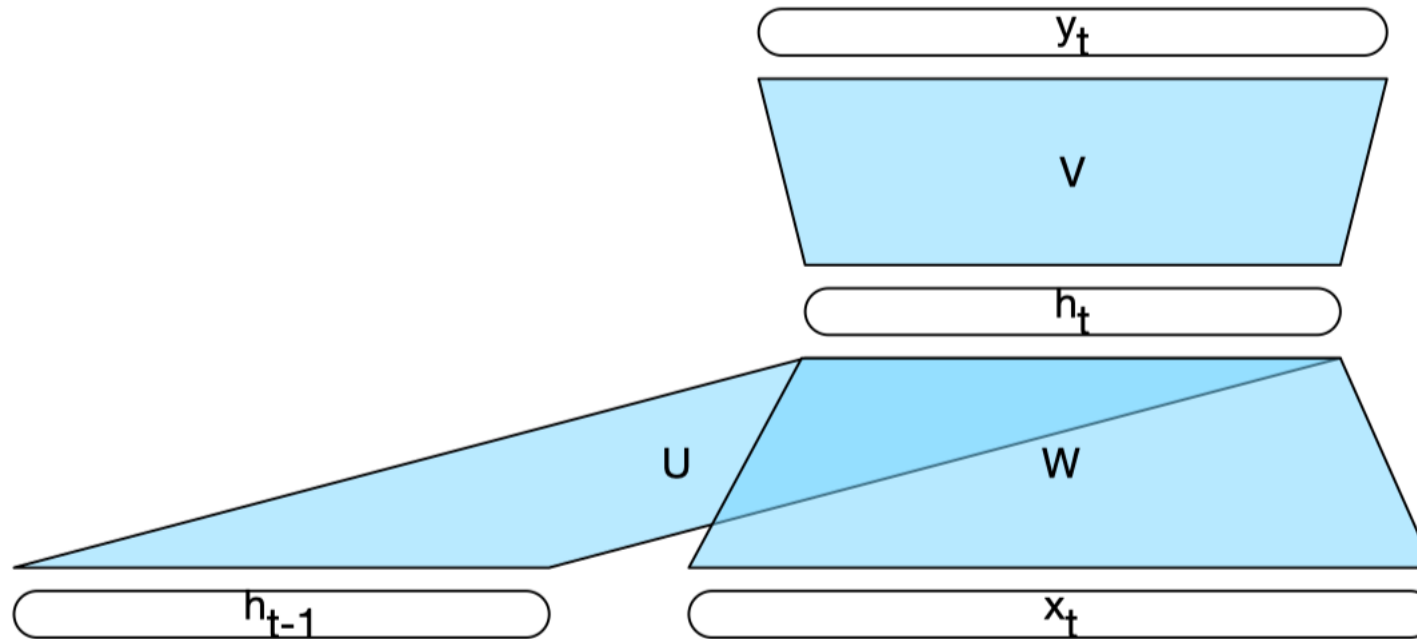
Recurrent Neural Networks

Recurrent Neural Networks

- A network that contains a **cycle** within its network connections.
- Any network where the value of a unit is directly, or indirectly, dependent on its own earlier outputs as an input.



RNN Illustrated as Simple Feedforward Network



Inference in RNNs

- Forward inference (mapping a sequence of inputs to a sequence of outputs) in an RNN is **nearly identical** to what we've already seen with feedforward networks.

$$h_t = g(Uh_{t-1} + Wx_t)$$

$$y_t = f(Vh_t)$$

$$y_t = \text{softmax}(Vh_t)$$

Forward Inference

function FORWARDRNN($x, network$) **returns** output sequence y

$h_0 \leftarrow 0$

for $i \leftarrow 1$ **to** LENGTH(x) **do**

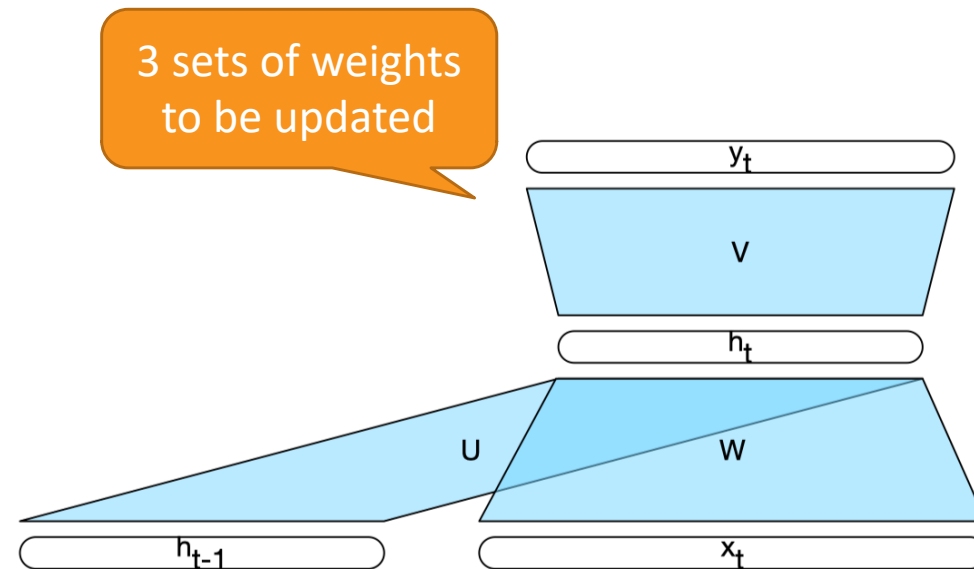
$h_i \leftarrow g(U h_{i-1} + W x_i)$

$y_i \leftarrow f(V h_i)$

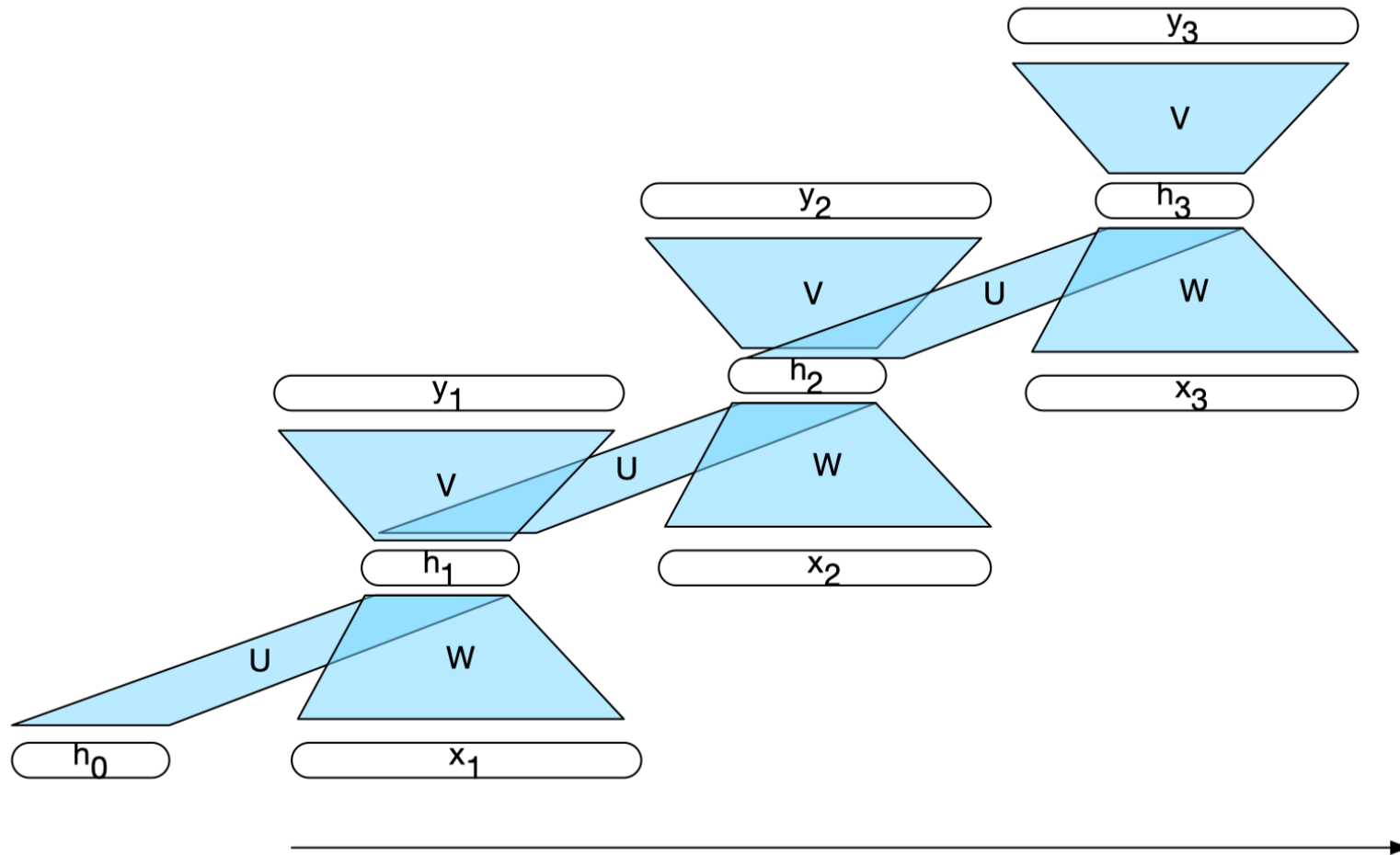
return y

Training

- The training is similar to that of feedforward networks,
 - we'll use a training set, a loss function, and backpropagation to obtain the gradients needed to adjust the weights in these recurrent networks.



RNN Unrolled in Time



Issues

- Unrolling an entire input sequence may not be feasible for applications that involve long input sequences, e.g.
 - Speech recognition
 - Character-level processing
 - Streaming of continuous inputs
- What to do then?
 - We can unroll the input into **manageable fixed-length** segments.
 - Then, treat each segment as a distinct training item.

RNNs as Language Models (1/2)

- RNN-based language models process a word at a time.
- Predict the next word in a sequence by using the **current word** and the **previous hidden state** as inputs.
- How is the limited context constraint avoided in this approach?
 - The hidden state!

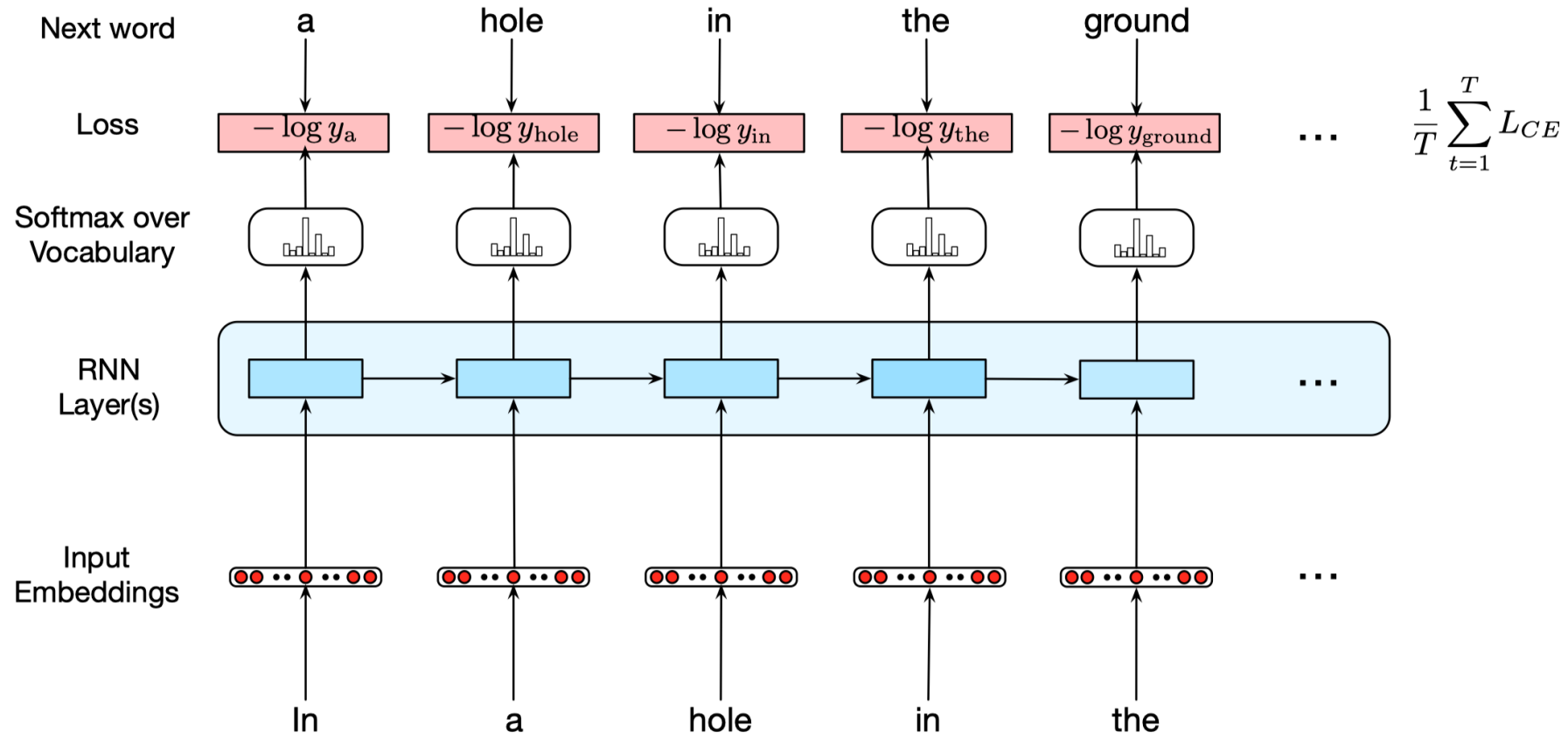
RNNs as Language Models (2/2)

- The **input sequence** x consists of word embeddings represented as one-hot vectors of size $|V| \times 1$
- The **output predictions**, y , are represented as vectors representing a probability distribution over the vocabulary.
- At each step, the model uses the **word embedding matrix** E to retrieve the embedding for the current word, and then combines it with the **hidden layer from the previous step** to compute a new hidden layer.
- The hidden layer is then used to generate an output layer which is passed through a softmax layer to generate a probability distribution over the entire vocabulary.

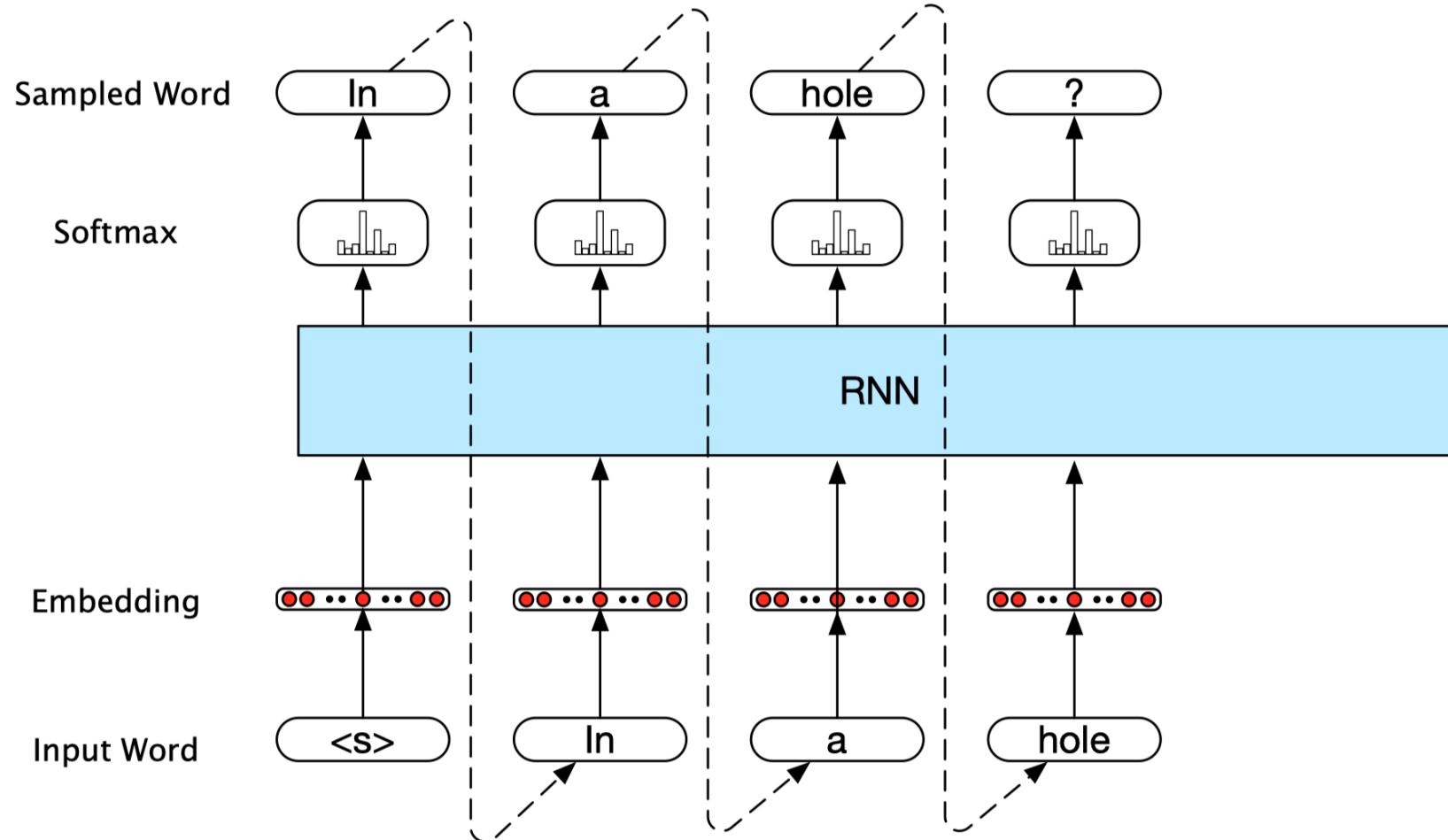
Training RNN as a Language Model

- Use a **corpus** of text and a **training regimen** (teacher forcing).
- Teacher forcing uses **ground truth** instead of the output from prior time step **as input**.
- The weights in the network are adjusted to minimize the average CE loss over the training sequence via gradient descent.

Training RNN-Based Language Models



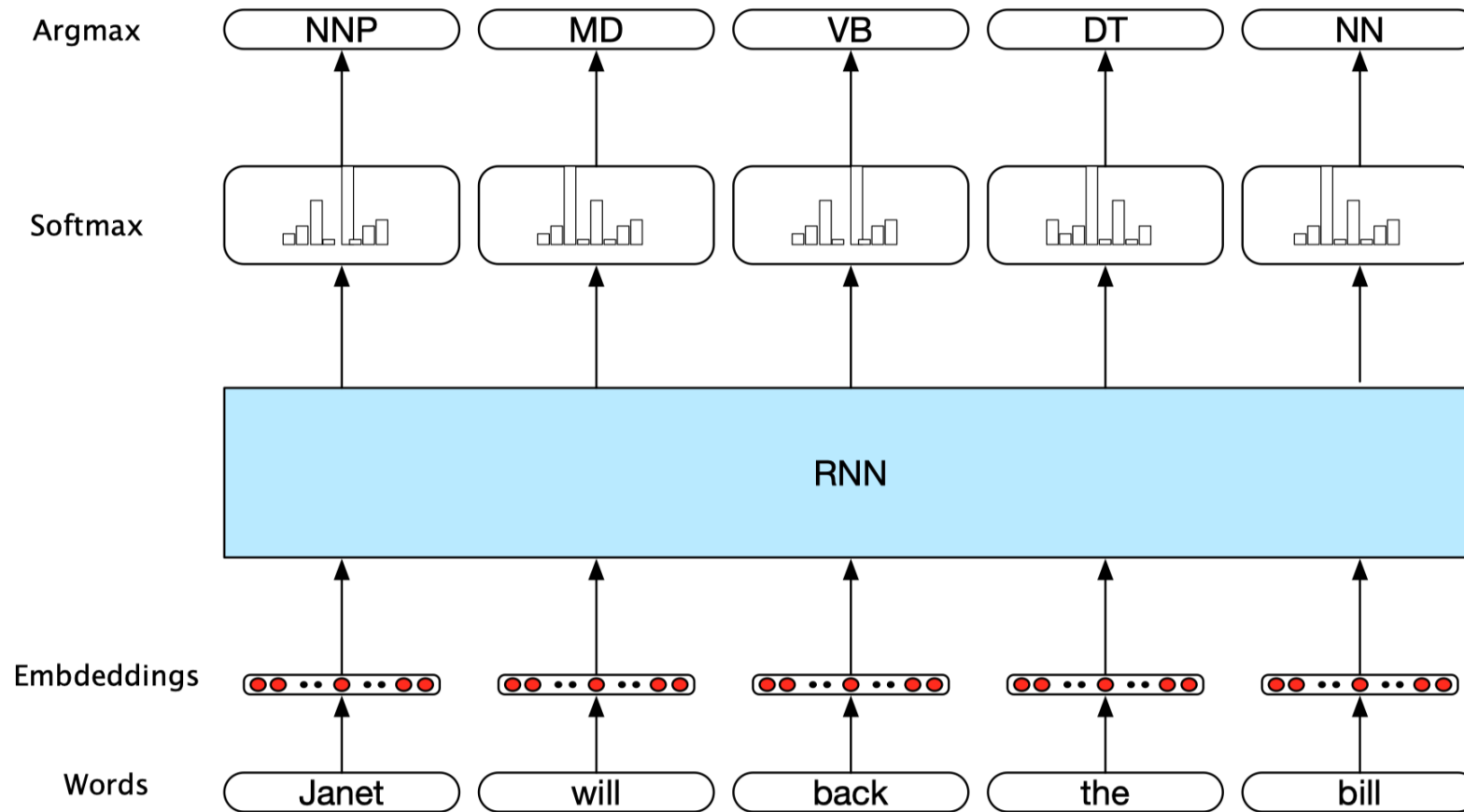
Autoregressive Generation



RNNs for Sequence Labeling

- Recall that in sequence labeling, the task is to assign a label chosen from a fixed set of labels to each token/element of a sequence.
 - POS tagging
 - NER
- In RNN, inputs are word embeddings and the outputs are tag probabilities generated by a softmax layer.

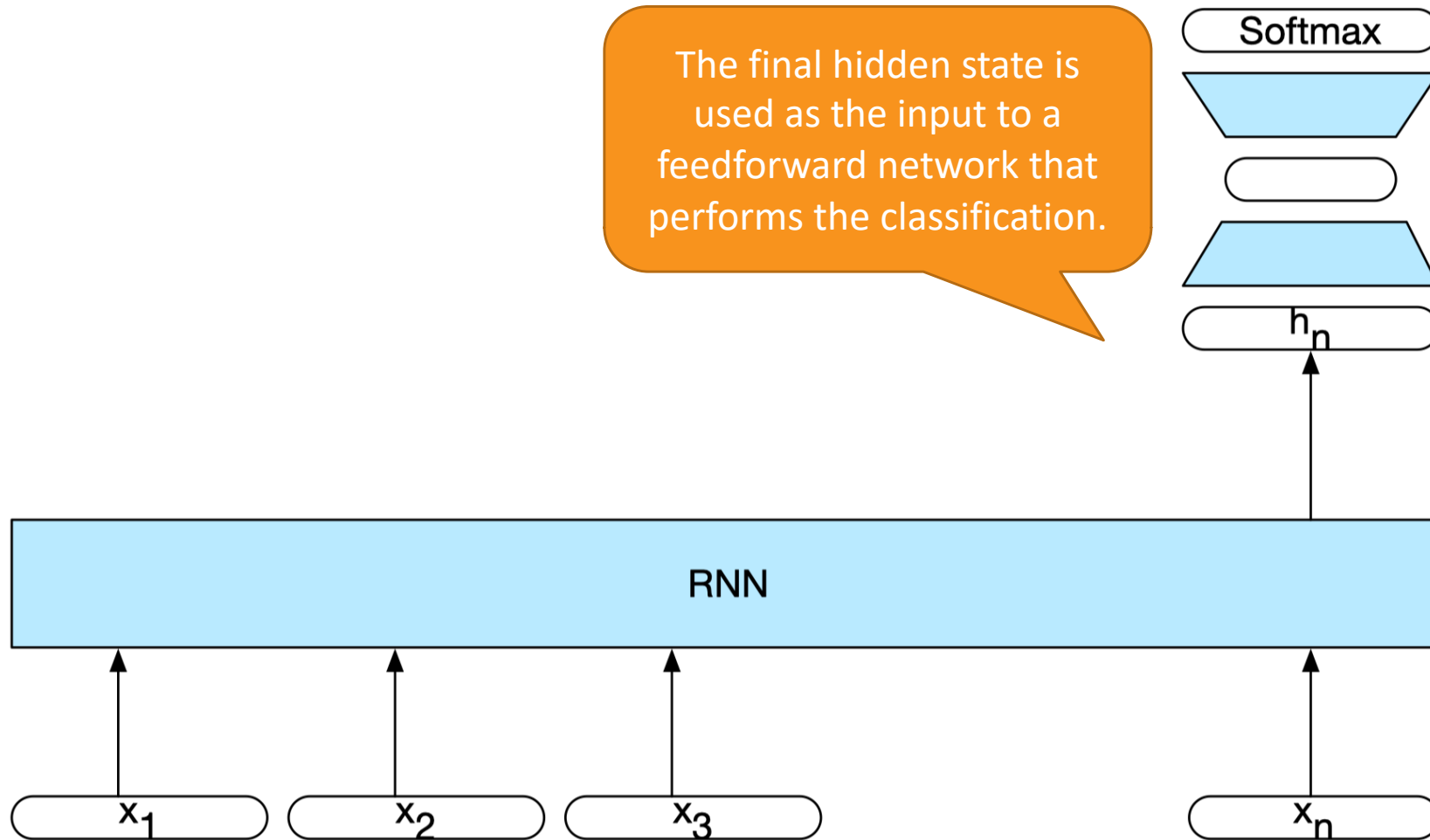
Simple RNN for POS Tagging



RNNs for Sequence Classification

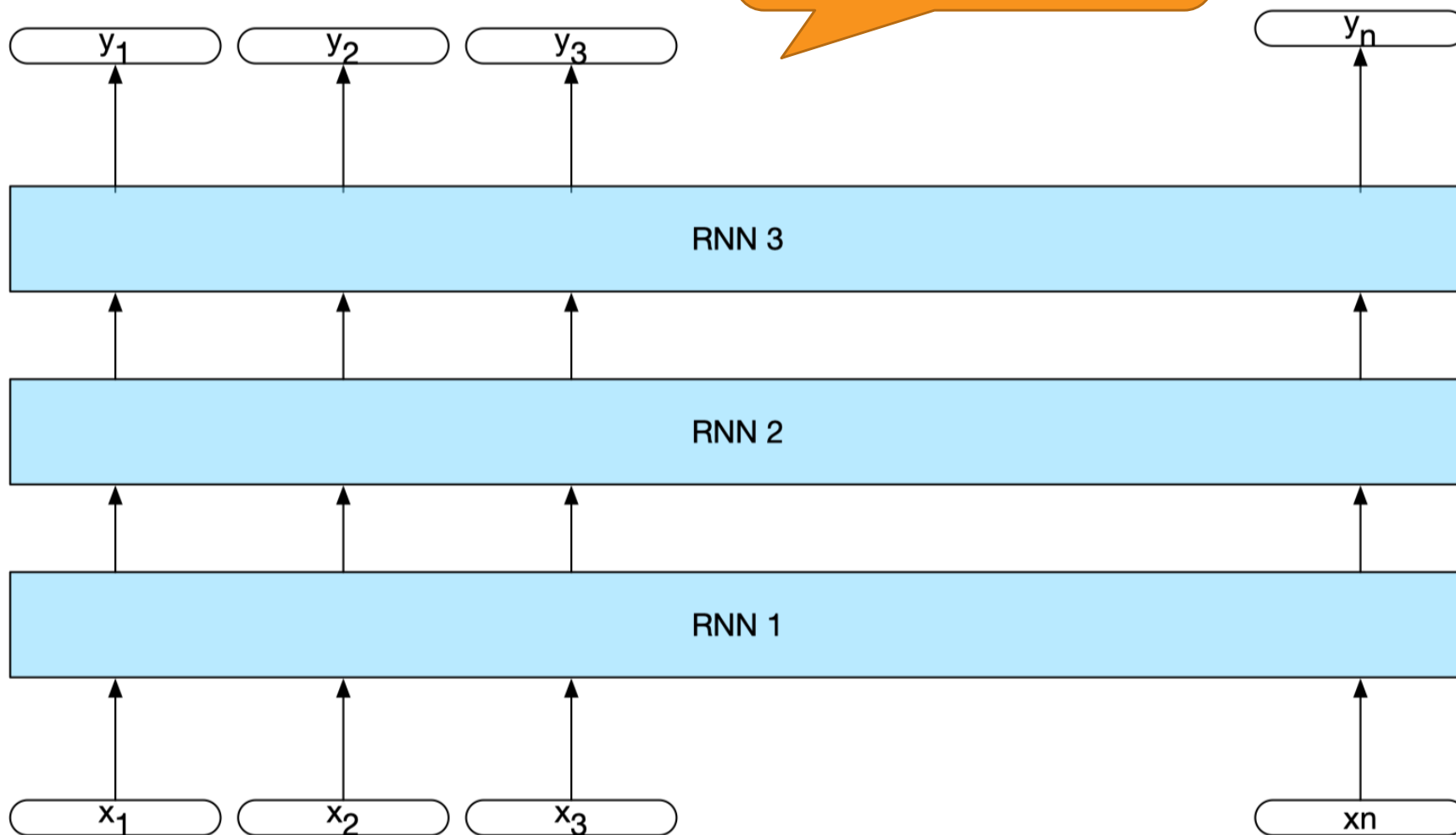
- Sequence classification aims to classify **entire sequences** rather than the tokens, e.g.
 - Document-Level Topic Classification
 - Spam Detection
- Sequences of text are classified as belonging to one of (pre-defined) categories.

RNNs for Sequence Classification



Stacked RNNs

The output of one layer serves as the input to a subsequent layer



Why Bi-Directional?

- RNNs capture the context to the **left** of the input.
- Suppose we have access to the entire input sequence at once
 - Will it be helpful to know the context to the right of the input as well?
- Train RNN on an input sequence **in reverse**.

Bi-Directional RNNs (1/2)

$$h_t^f = RNN_{forward}(x_1^t)$$

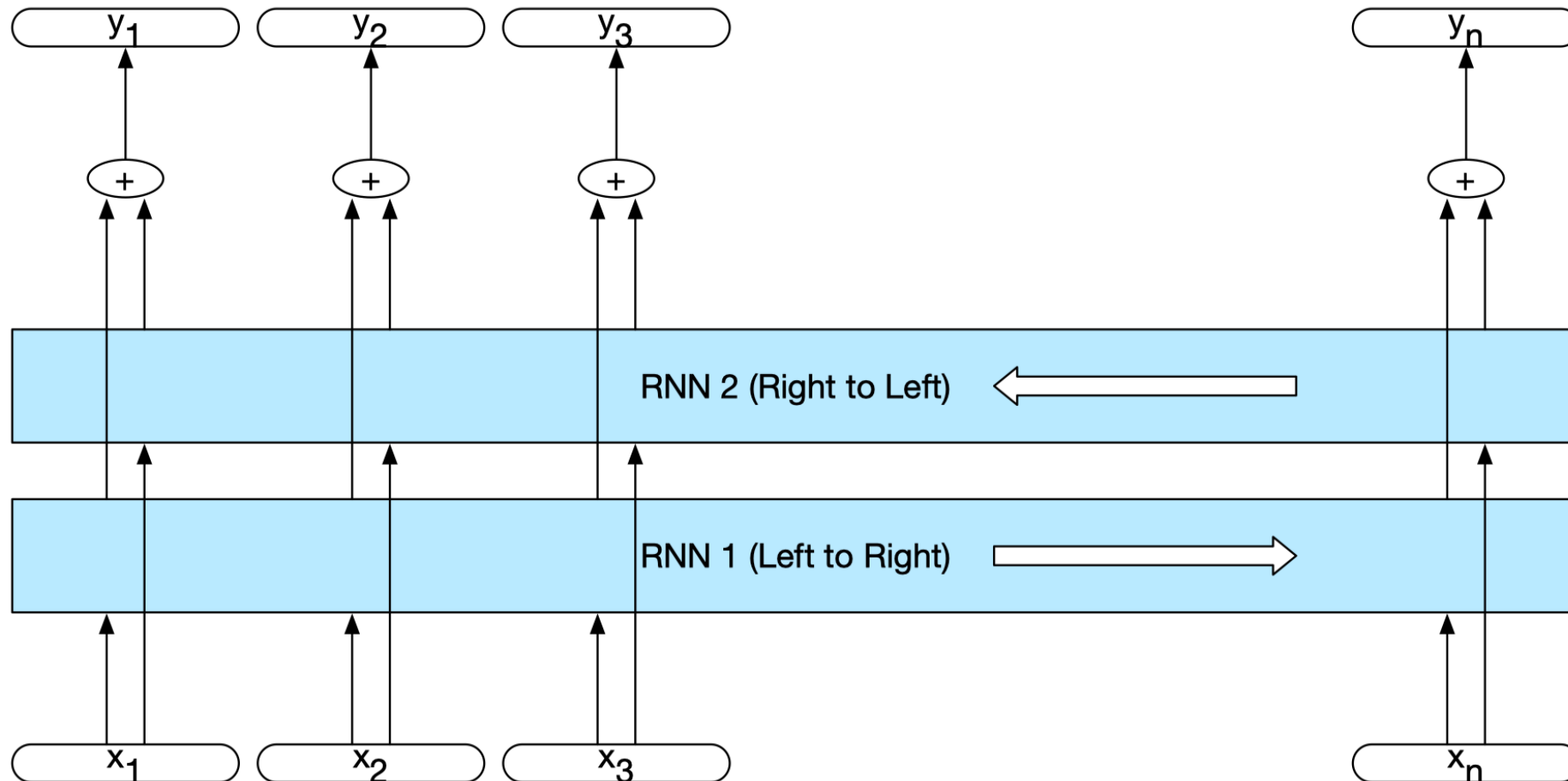
$$h_t^b = RNN_{backward}(x_t^n)$$

- h_t^f corresponds to the normal hidden state at time t
- h_t^b corresponds to all the information we have discerned about the sequence from t to the end of the sequence

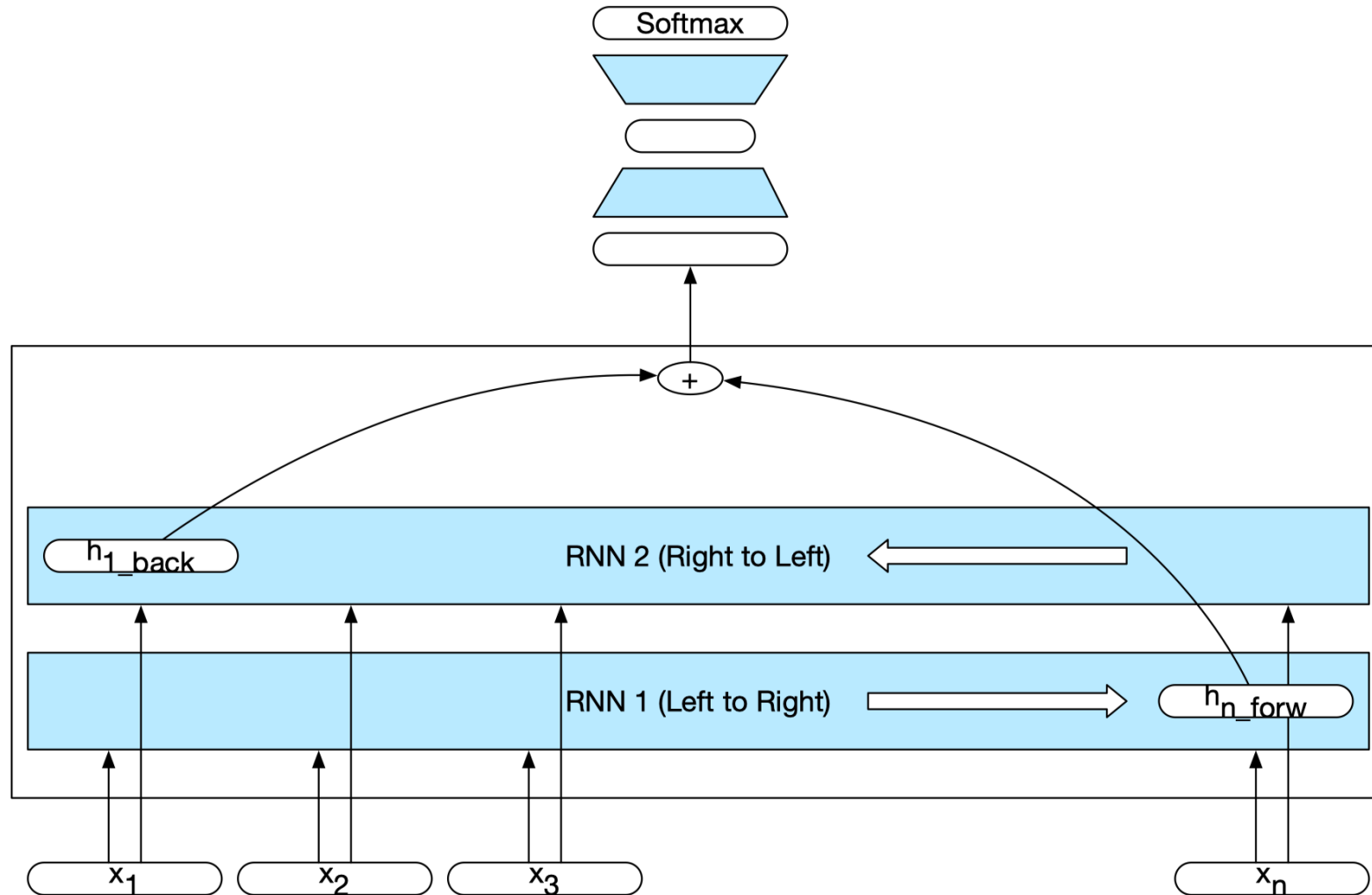
$$h_t = h_t^f \oplus h_t^b$$

Combining the left and right contexts

Bi-Directional RNNs (2/2)




Bi-Directional RNNs for Sequence Classification



Context Management

- Despite having access to the entire preceding sequence, the information encoded in hidden states tends to be fairly **local**.
 - more relevant to the most recent parts of the input sequence and recent decisions
- However, distant information is crucial to many applications.

The flights the airline was cancelling  were full.

Issues with RNNs

- RNNs can't carry forward critical information.
- The weights that determine the values in the hidden layer are performing two tasks simultaneously.
 - Providing information that is useful for the current decision
 - Updating and carrying forward information required for future decisions
- The need to backpropagate the error signal back through time.
 - The hidden layers are subject to repeated multiplications (by the length of the sequence).
 - The gradients are eventually driven to zero or **vanishing gradients** problem.
- We need network architectures that can **maintain relevant context** over time.

Long Short-Term Memory

- Long short-term memory (LSTM) networks divide the context management problem into two sub-problems:
 - **removing** information no longer needed from the context
 - **adding** information likely to be needed for later decision making
- LSTMs learn how to manage this context rather than hard-coding:
 - add an **explicit context layer** to the architecture (in addition to the usual recurrent hidden layer)
 - use **specialized neural units** that make use of gates to control the flow of information into and out of the units that comprise the network layers

LSTM Gates

- The gates in an LSTM share a common **design pattern**:
 - a feedforward layer
 - a sigmoid activation function
 - a pointwise multiplication with the layer being gated
- **Forget gate**
 - delete information from the context that is no longer needed.
- **Add gate**
 - add selected information to the context
- **Output gate**
 - decide what information is required for the current hidden state

LSTM Gates

8 sets of weights to learn

Forget gate

$$f_t = \sigma(U_f h_{t-1} + W_f x_t)$$

$$k_t = c_{t-1} \odot f_t$$

$$g_t = \tanh(U_g h_{t-1} + W_g x_t)$$

Add gate

$$i_t = \sigma(U_i h_{t-1} + W_i x_t)$$

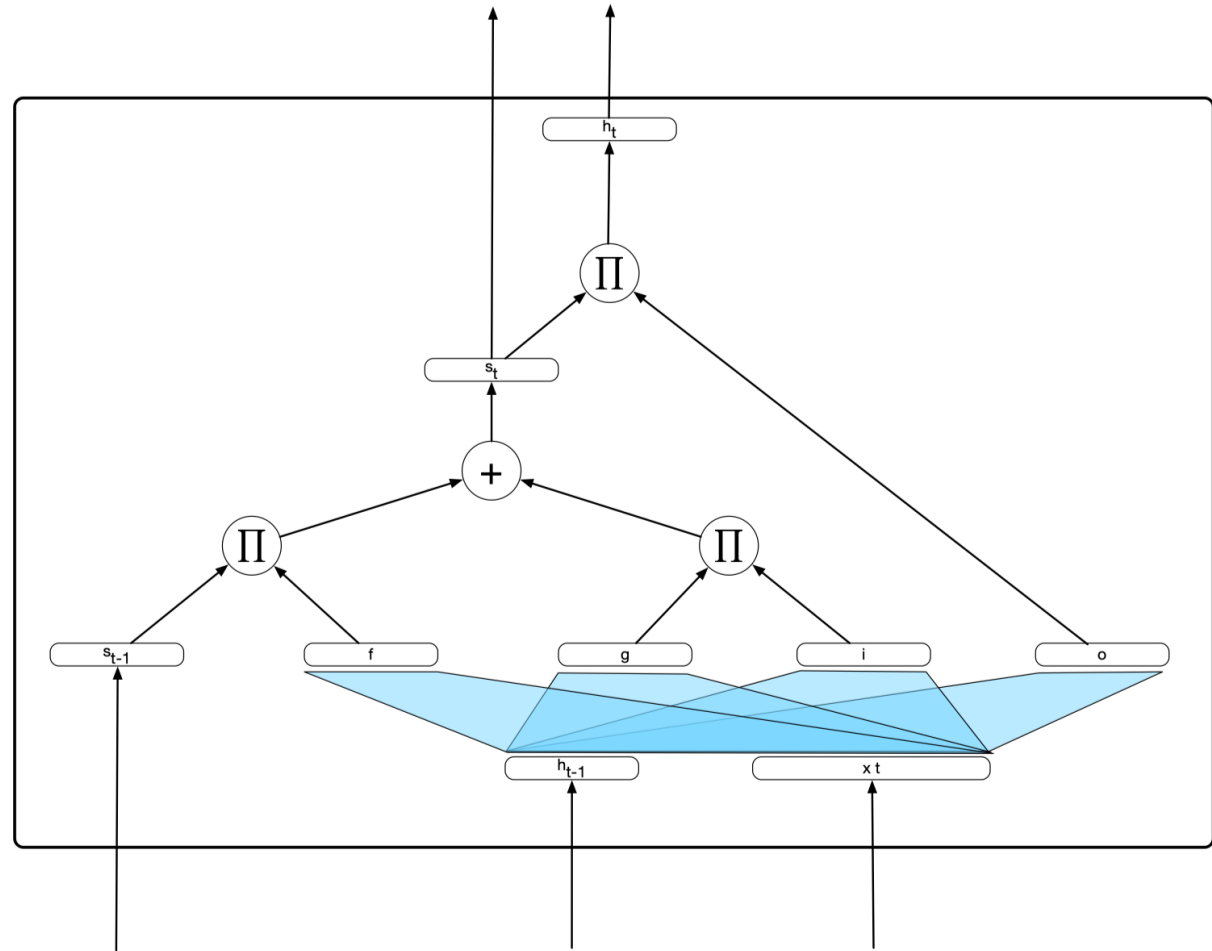
$$j_t = g_t \odot i_t$$

$$c_t = j_t + k_t$$

Output gate

$$o_t = \sigma(U_o h_{t-1} + W_o x_t)$$

$$h_t = o_t \odot \tanh(c_t)$$



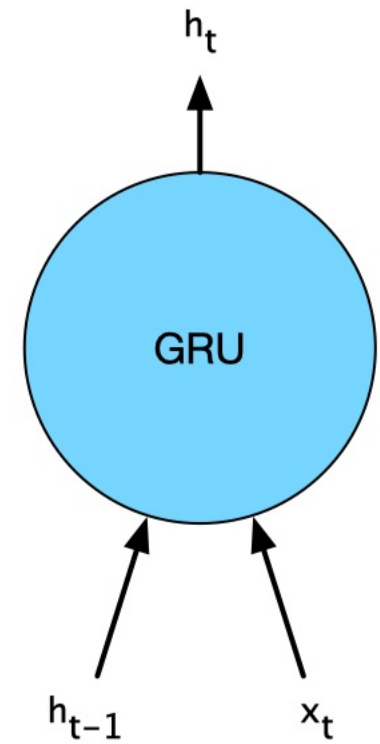
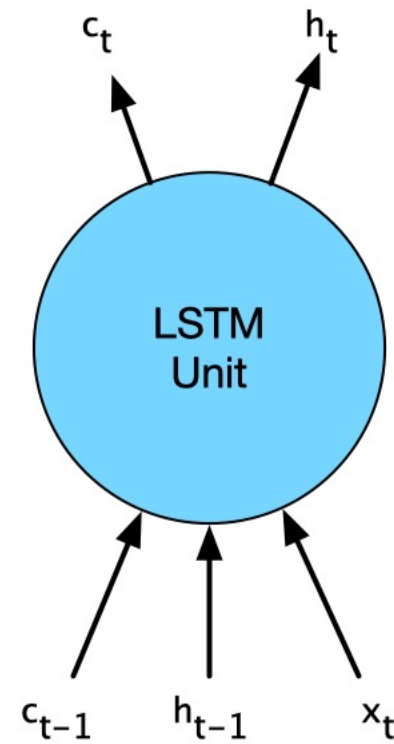
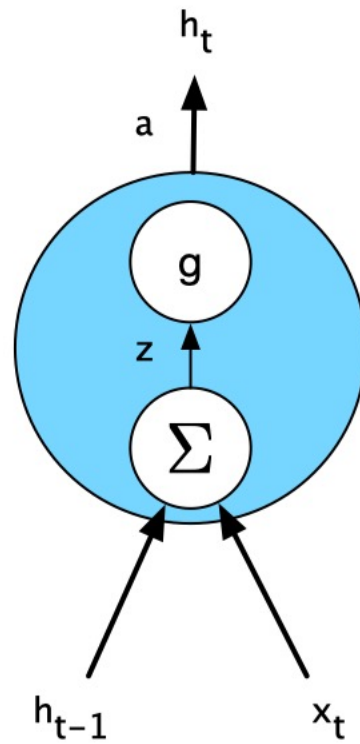
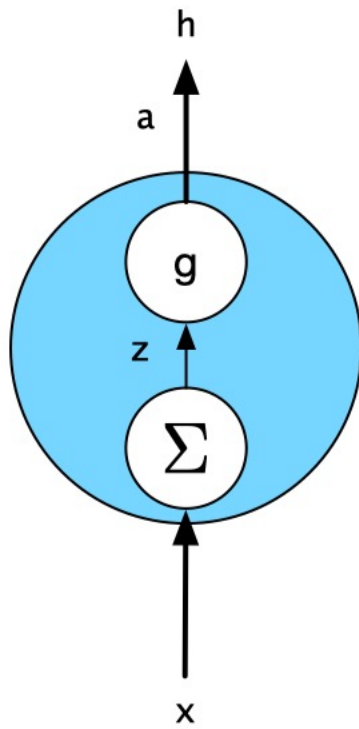
Gated Recurrent Units

- No separate context vector
- **Reset gate r**
 - which aspects of the previous hidden state are relevant to the current context
 - what can be ignored
- **Update gate z**
 - which aspects of this **new state** will be used directly in the new hidden state
 - which aspects of the **previous state** need to be preserved for future use

$$\begin{aligned} r_t &= \sigma(U_r h_{t-1} + W_r x_t) & h_t &= (1 - z_t) h_{t-1} + z_t \tilde{h}_t \\ z_t &= \sigma(U_z h_{t-1} + W_z x_t) & \tilde{h}_t &= \tanh(U(r_t \odot h_{t-1}) + W x_t) \end{aligned}$$

Element-wise
multiplication

Various Neural Units



Self-Attention Networks: Transformers

Issues with RNNs

- RNNs pass information forward through a series of recurrent connections:
 - loss of relevant information
 - difficulties in training
- Parallel computing resources needed.

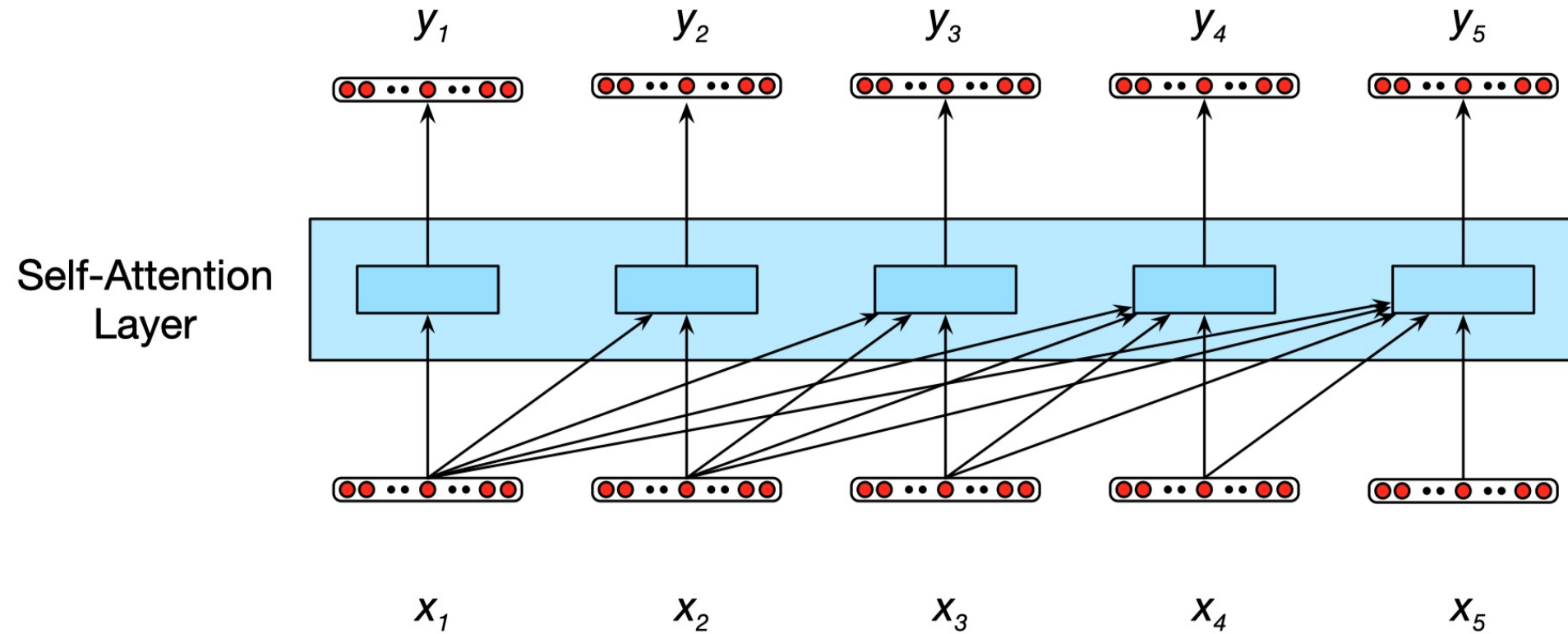
Transformers

- Eliminate recurrent connections
- Similar architecture to fully connected networks
- Map sequences of input vectors (x_1, \dots, x_n) to sequences of output vectors (y_1, \dots, y_n) of the same length
- Made up of stacks of network layers:
 - simple linear layers
 - feedforward networks, and
 - custom connections

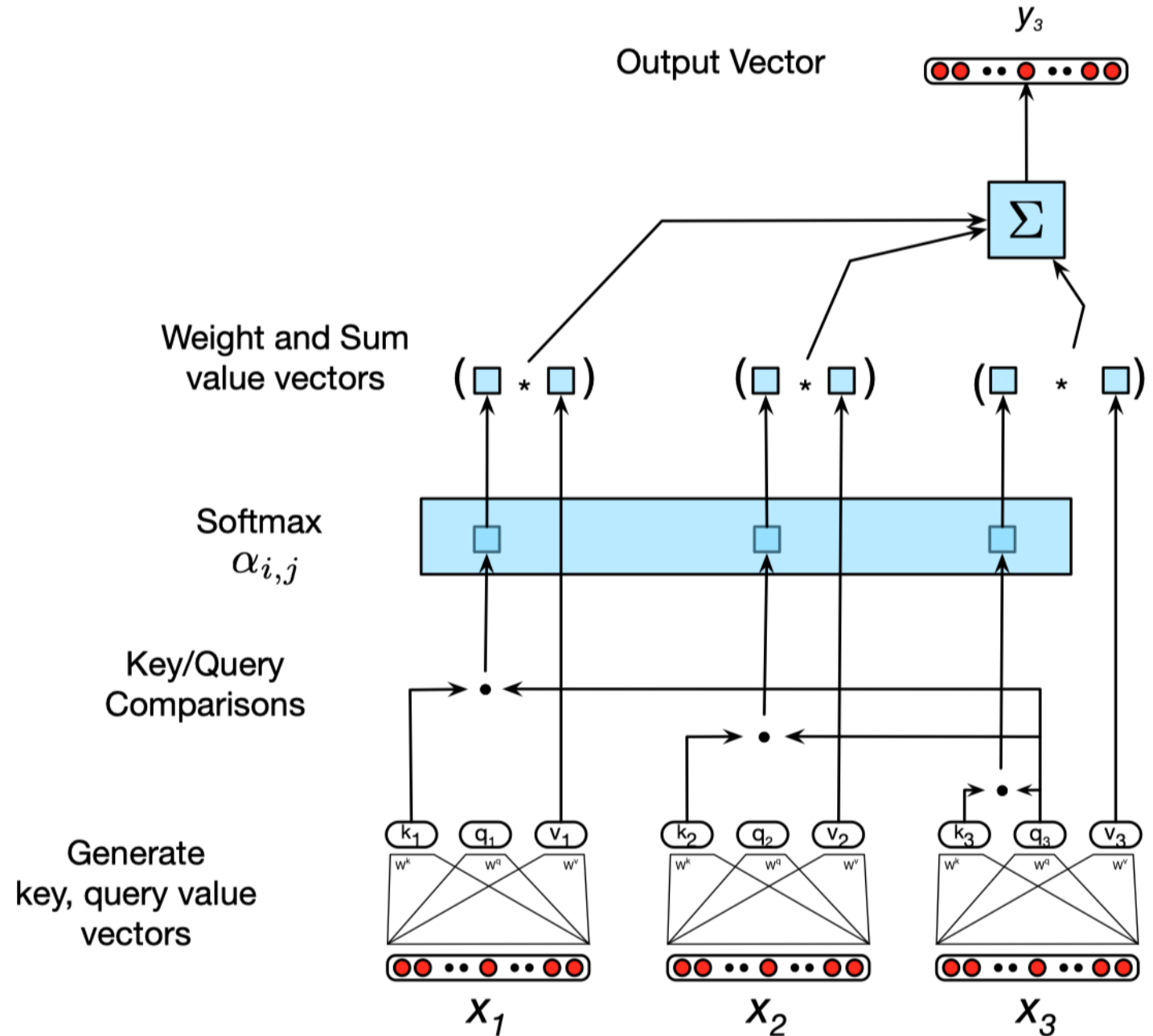
An orange speech bubble with a tail pointing towards the bottom-left, containing the text "Self-Attention Layer".

Self-Attention
Layer

Self-Attention Layer



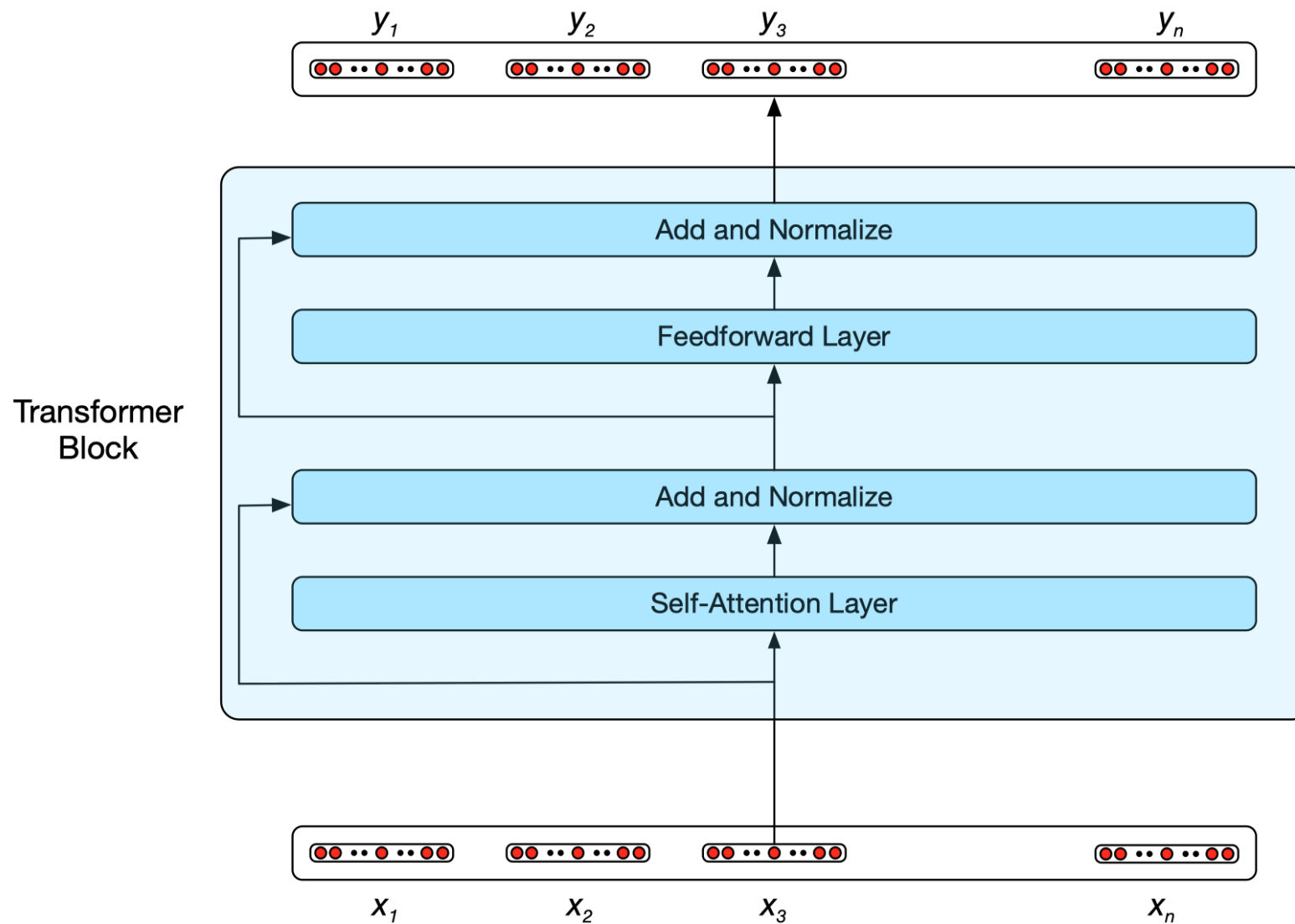
Calculation of the Third Element (y_3)



Transformer Blocks

- Core units for calculation.
- These blocks can be stacked.

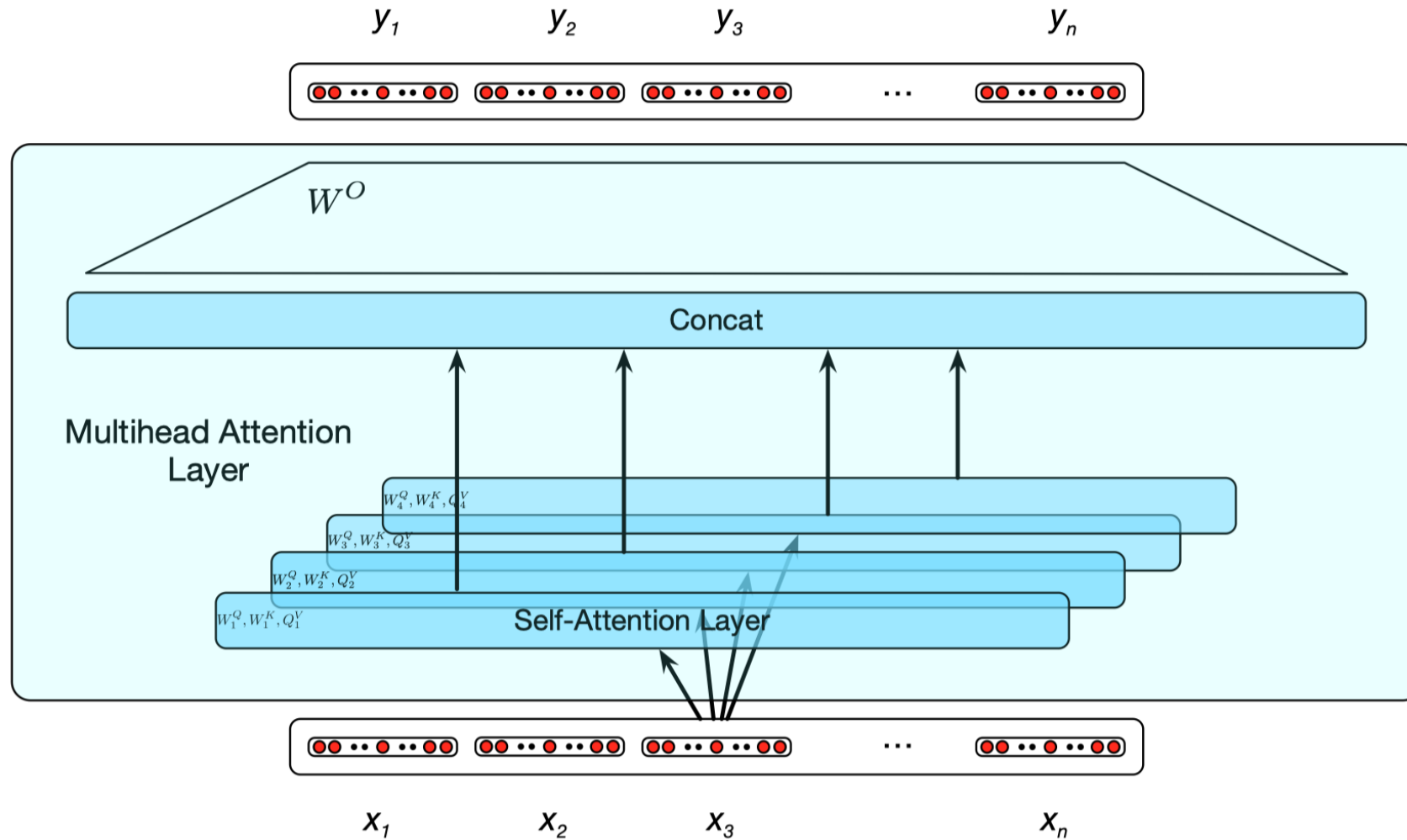
Transformer Block



Multihead Attention

- The different words in a sentence can relate to each other in different ways simultaneously.
- A single transformer block can't capture all of the different kinds of parallel relations among its inputs.
- Multihead self-attention layers are sets of self-attention layers, called heads, that reside in parallel layers at the same depth in a model, each with its own set of parameters.

Multihead Self-Attention



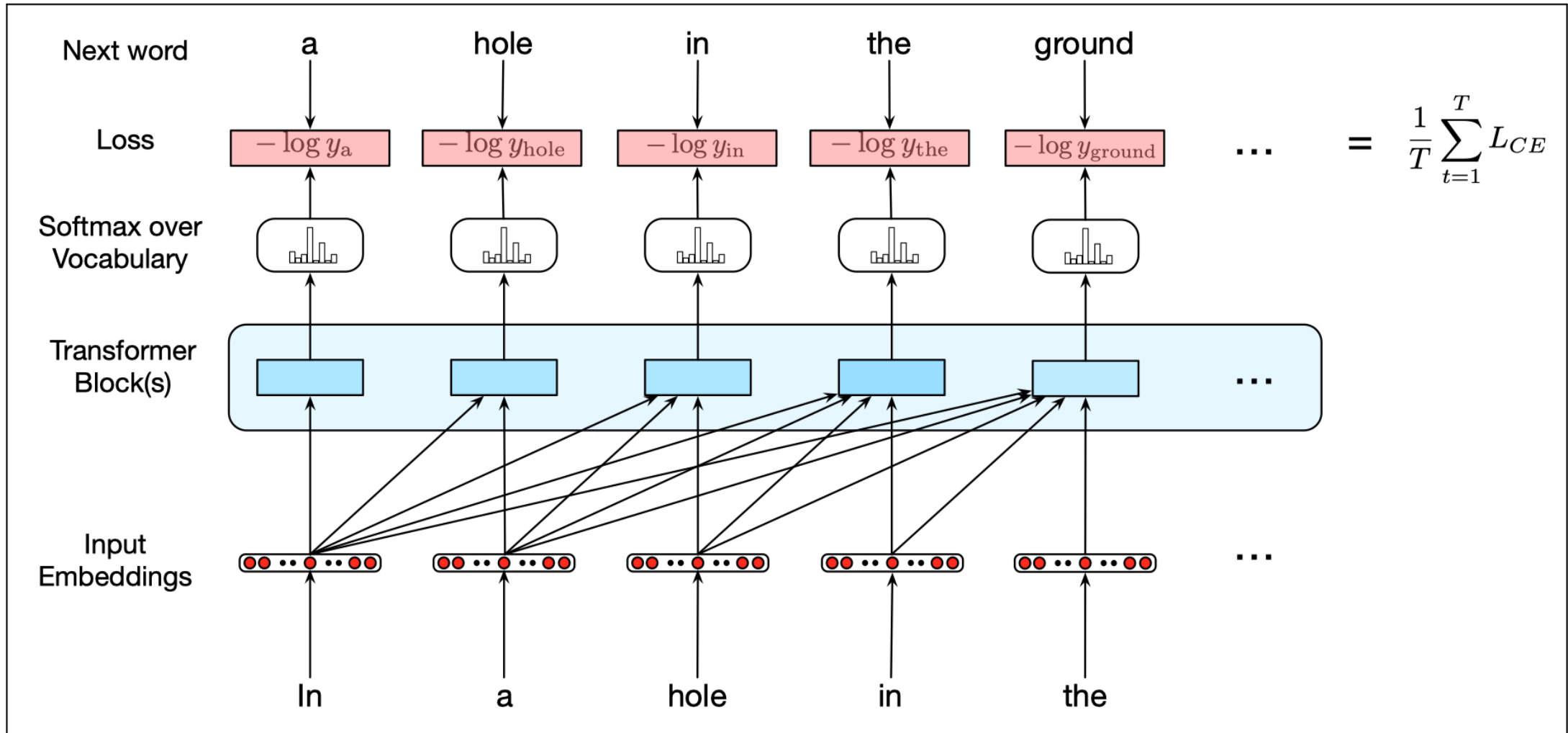
Positional Embeddings

- In RNNs, order of inputs are embedded by nature of the models.
- Transformer inputs are combined with **positional embeddings** specific to each position in an input sequence.

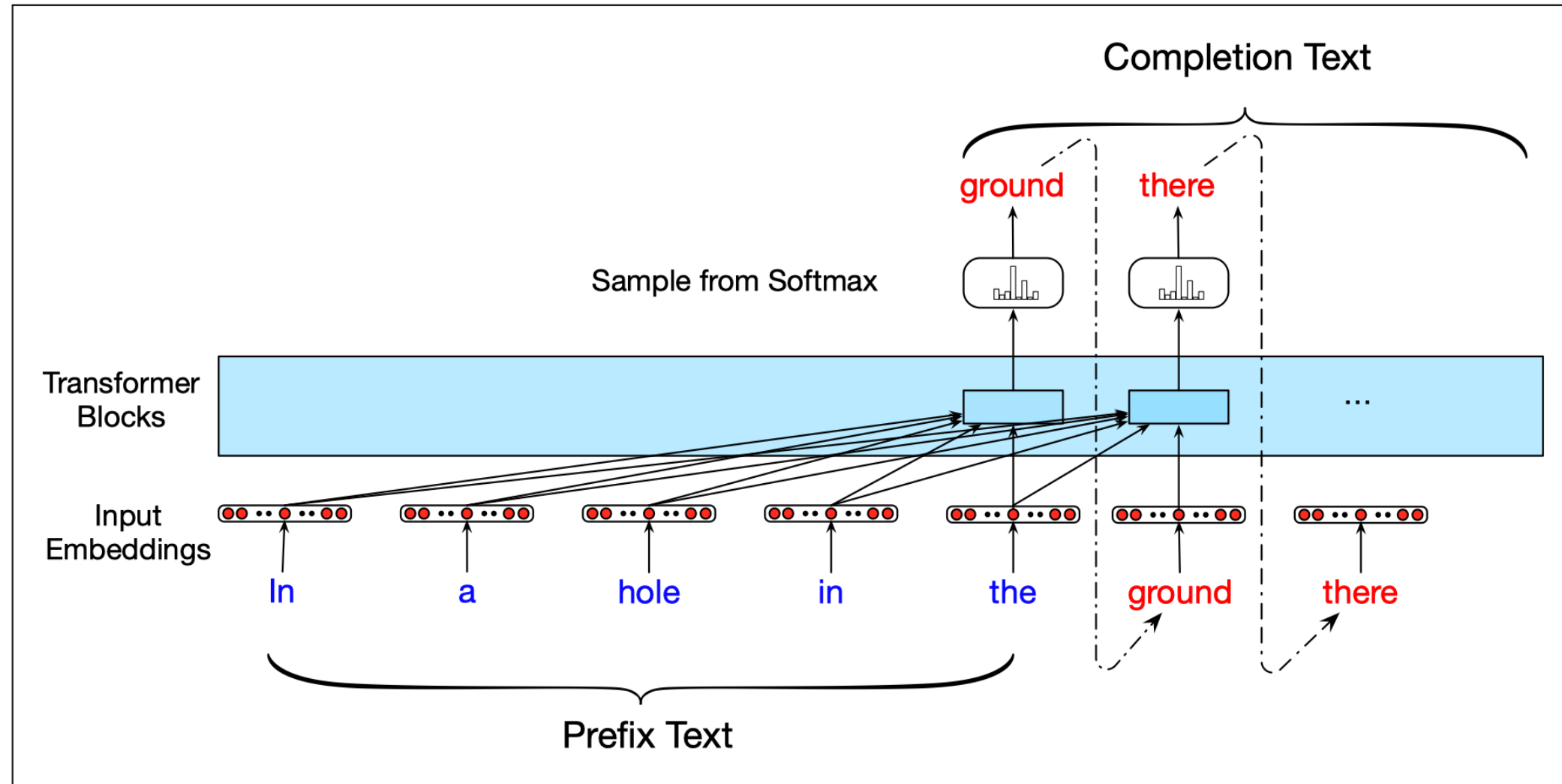
How to Obtain Positional Embeddings?

- Start with randomly initialized embeddings corresponding to each possible input position up to some maximum length.
- Positional embeddings are learned along with other parameters during training.
- Word embedding for each input is then added to its corresponding positional embedding.

Self-Attention Network as Autoregressive LM



Autoregressive Text Completion



Text Summarization

Original Article

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For \$89, self-styled entrepreneur Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says.

But not if you live in New England or surrounding states. “We will not ship snow to any states in the northeast!” says Waring’s website, ShipSnowYo.com. “We’re in the business of expunging snow!”

His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity.

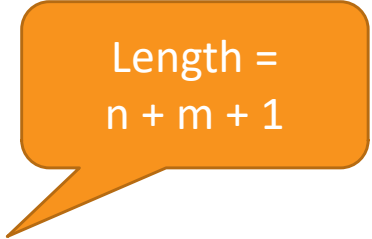
According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston. He joked about shipping the stuff to friends and family in warmer states, and an idea was born. His business slogan: “Our nightmare is your dream!” At first, ShipSnowYo sold snow packed into empty 16.9-ounce water bottles for \$19.99, but the snow usually melted before it reached its destination...

Summary

Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.

Text Summarization using Transformers

- We need a corpus of full-length articles with their corresponding summaries.
- Append a summary to each full-length article in a corpus with a unique marker.
- For each article summary pair (x_1, \dots, x_m) and (y_1, \dots, y_n) , we convert it into a single training instance $(x_1, \dots, x_m, \delta, y_1, \dots, y_n)$
- Then train an autoregressive language model using teacher forcing.



Length =
 $n + m + 1$

What have we learned so far?

- Temporal Nature of the Language
- RNNs
 - Language Model
 - Sequence Labeling
 - Sequence Classification
- Stacked RNNs
- Bi-directional RNNs
- LSTM
- Gated Recurrent Units (GRUs)
- Transformers