Deep Learning Architectures for Sequence Processing

T-622-ARTI

Stefán Ólafsson
Spring 2023

Sequence Modeling

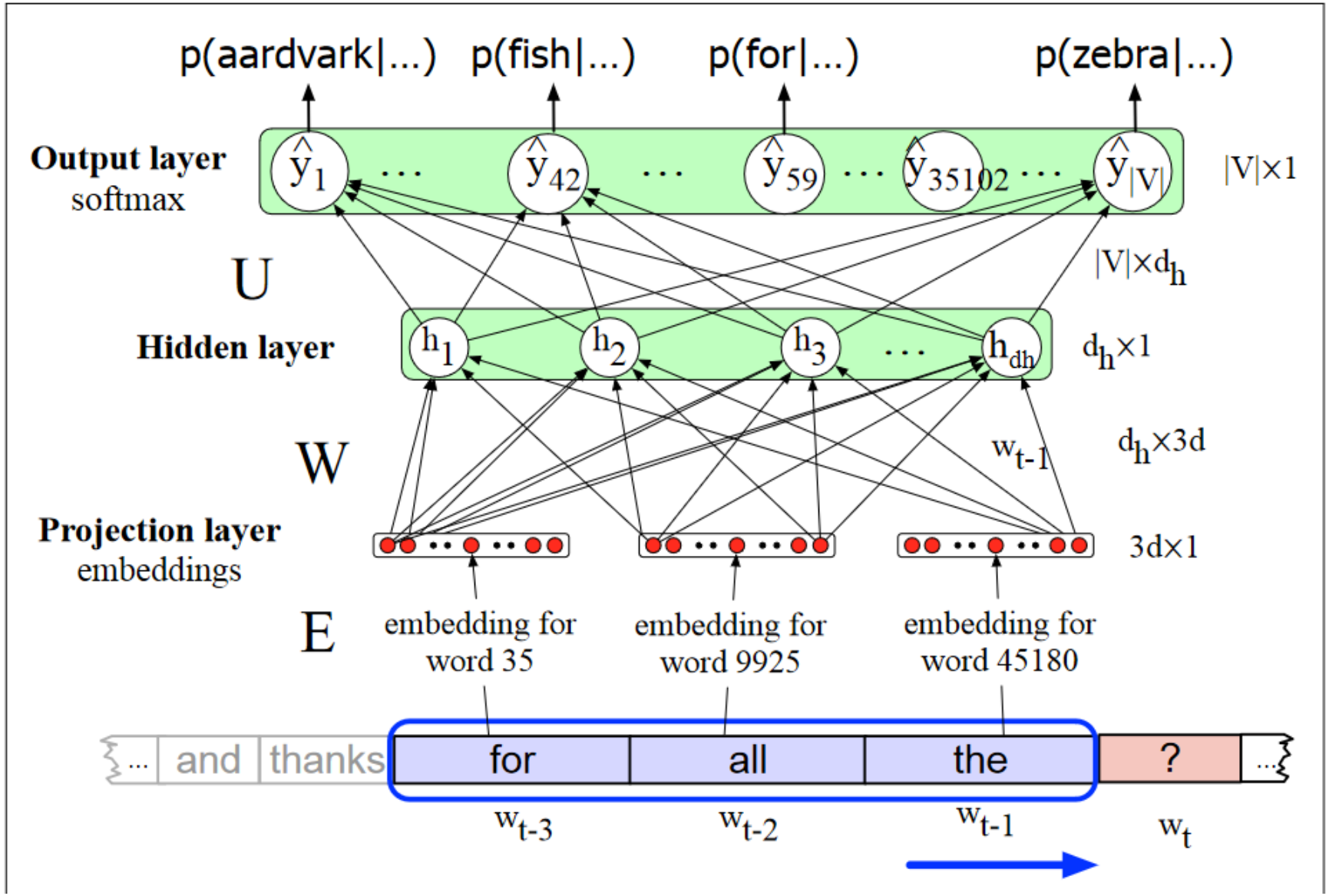
- Predict/generate next steps given previous steps
- Learn something about observations by modeling their sequence
- Models that input or output sequences of data
- Examples:
  - Text streams, audio clips, video clips, time-series data, etc.

Goal: Model sequences of varying length

- Feed-forward networks
- Logistic regression
- Naïve Bayes
- …

- No sequential information!

Issues?


p(aardvark|...)
p(fish|...)
p(for|...)
p(zebra|...)
Output layer
softmax
U
Hidden layer
W
Projection layer
embeddings
E
|V|×1
|V|×dh
dh×1
dh×3d
3d×1
wt-1
embedding for word 35
embedding for word 9925
embedding for word 45180
... and thanks
for
all
the
?
wt-3
wt-2
wt-1
wt

Models for sequences of varying length

- **Recurrent Neural Networks**
  - LSTM, GRU
- **Transformers**
  - Encoder, Decoder, Self-attention

Recurrent Neural Network (RNN)

- Contains a cycle within its network connections
- Value of a unit depends on its earlier outputs as input

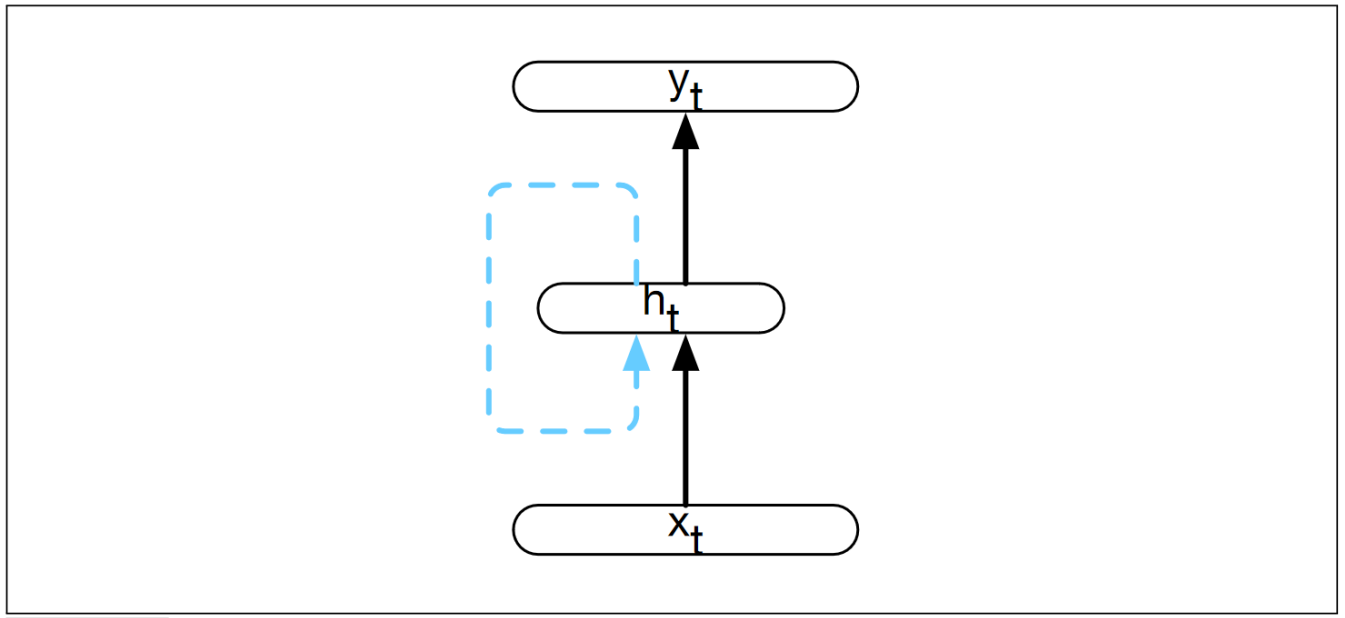
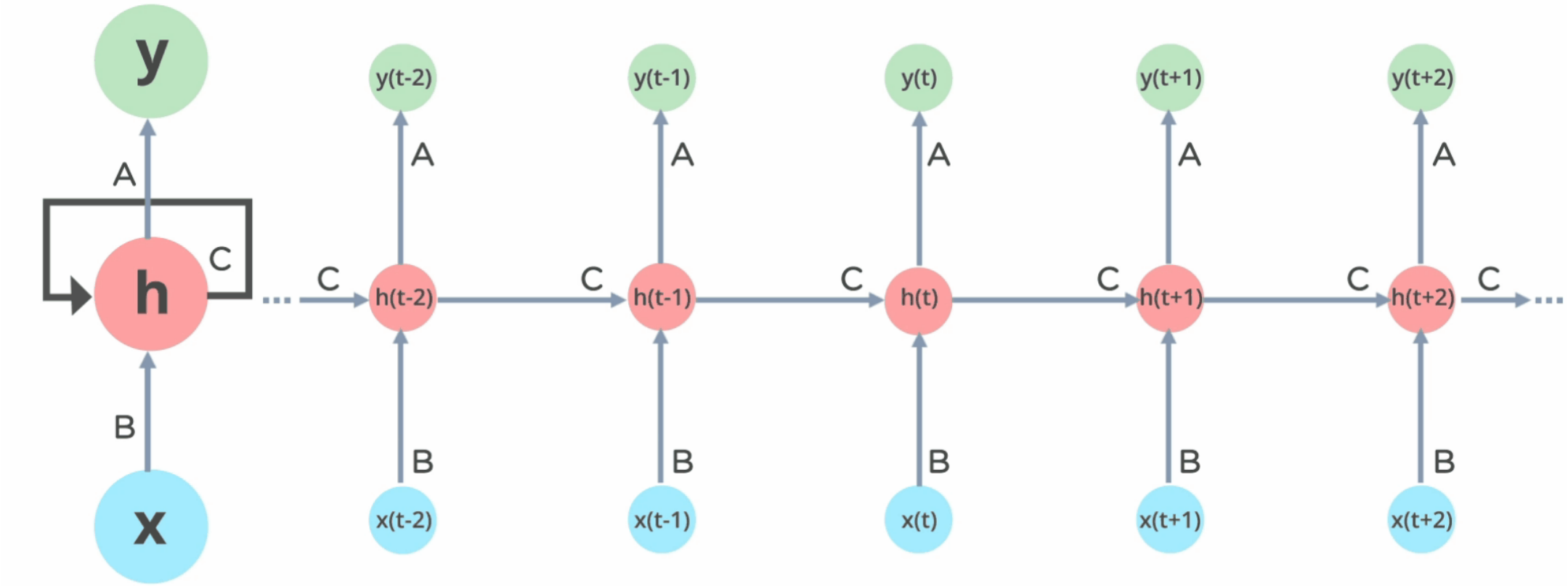


Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous time step.

RNN – Sequential processing

- Sequences are processed by presenting one item at a time to the network
- The hidden layer from previous time step provides “memory” or context:
  - Informs the decisions made at later points in time
- No fixed-length limit on the prior context
- Hidden layer embodies information going back to the beginning of the sequence

RNN - Animated



RNN - Unrolled

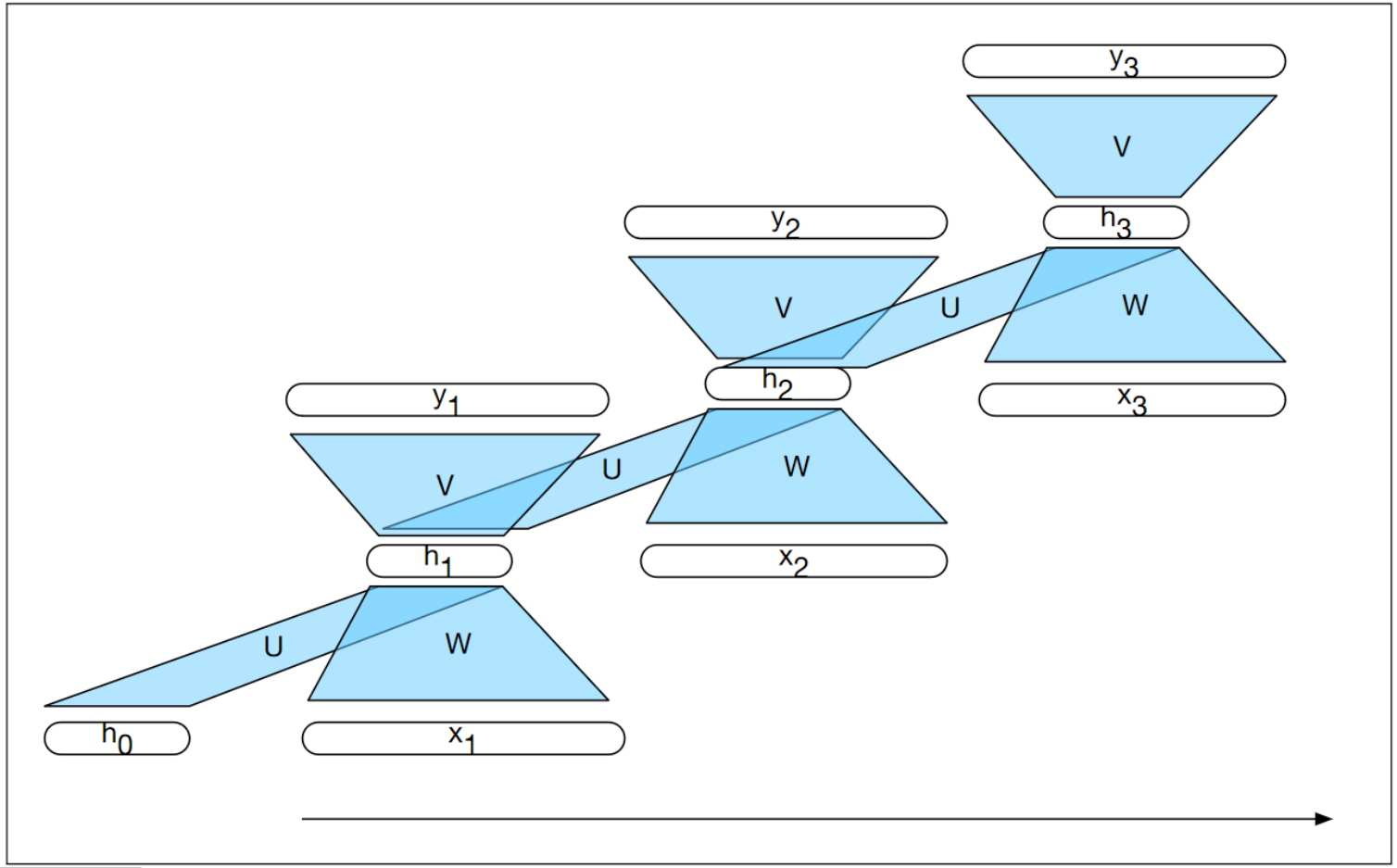


Figure 9.5 A simple recurrent neural network shown unrolled in time. Network layers are copied for each time step, while the weights U, V and W are shared in common across all time steps.

Forward Inference in RNNs

- Map a sequence of inputs to a sequence of outputs
- Output y_t is calculated for an input x_t using the activation value for the hidden layer h_t

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

$$y_t = f(Vh_t)$$

RNN Training

- Training set
- Loss function
- Back-propagation:
  - Unrolling the network

RNN Language Models

- Process sequences of words, one word at a time
- Predict the next word using current word and hidden state

$$e_t = E^T x_t$$

$$h_t = g(U h_{t-1} + W e_t)$$

$$y_t = softmax(V h_t)$$

RNN Language Model Training

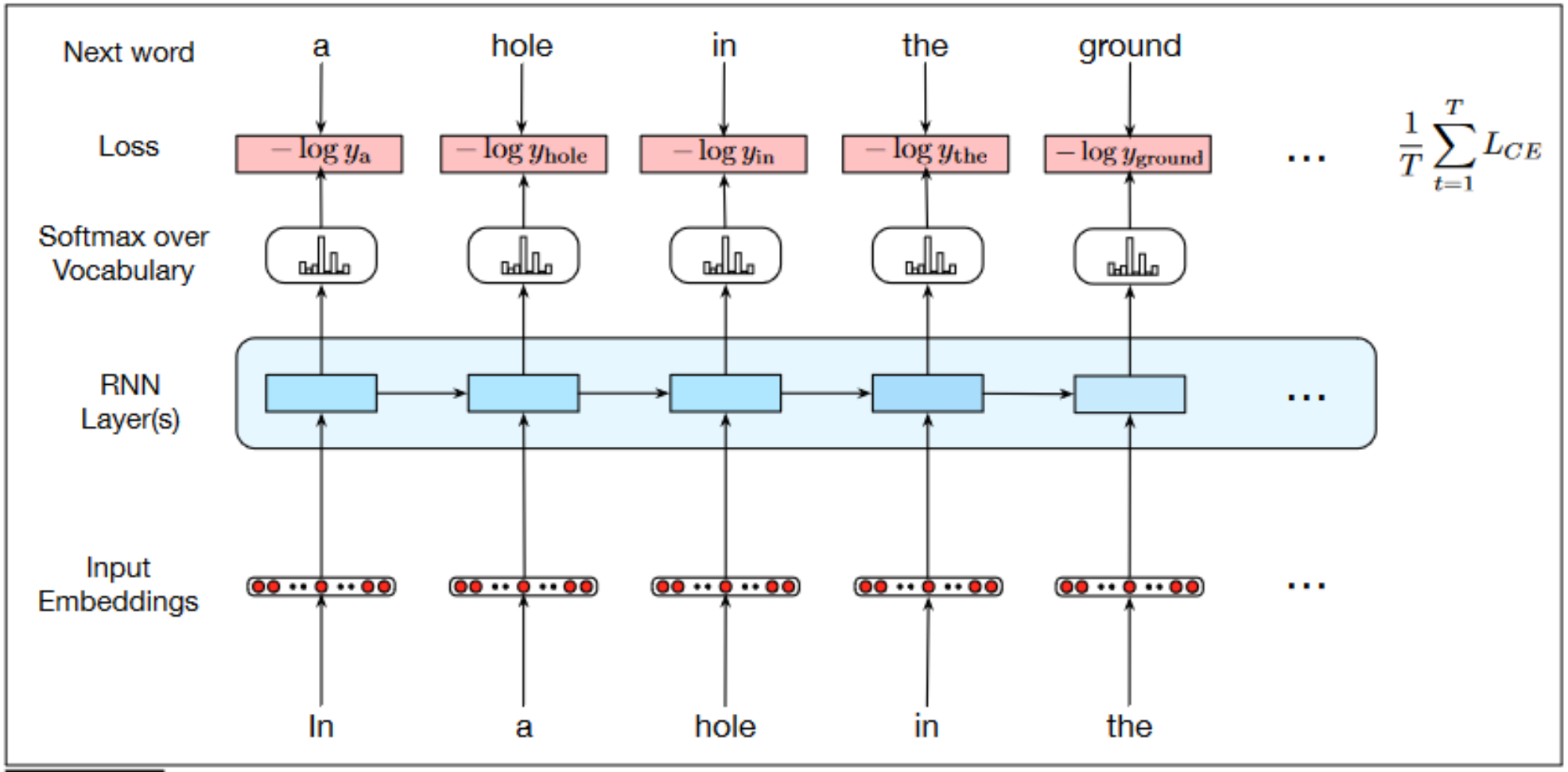


Figure 9.6 Training RNNs as language models.

Language model evaluation

- **Perplexity (PP):**
  - The probability the model assigns to an unseen test set, normalized by its length.

$$PP_\theta(w_{1:n}) = P(w_{1:n})^{\frac{1}{n}}$$

- **Autoregressive generation:**
  - Generate novel sequences conditioned on previous choices

Other Applications of RNNs for Sequential Tasks

Many-to-One

Classification

End-to-end training

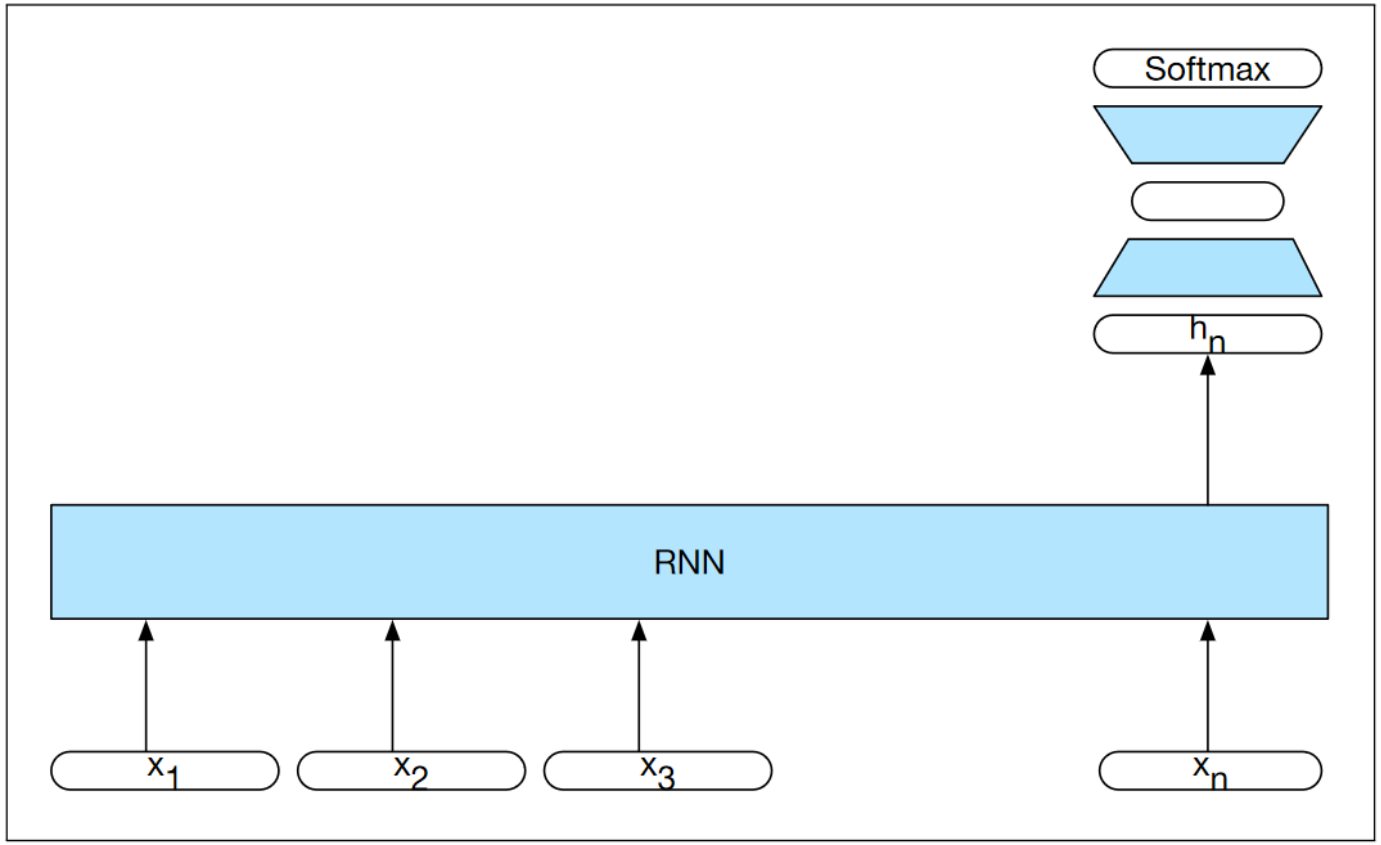


Figure 9.9 Sequence classification using a simple RNN combined with a feedforward network. The final hidden state from the RNN is used as the input to a feedforward network that performs the classification.

Other Applications of RNNs for Sequential Tasks

One-to-One

Part-of-speech tagging

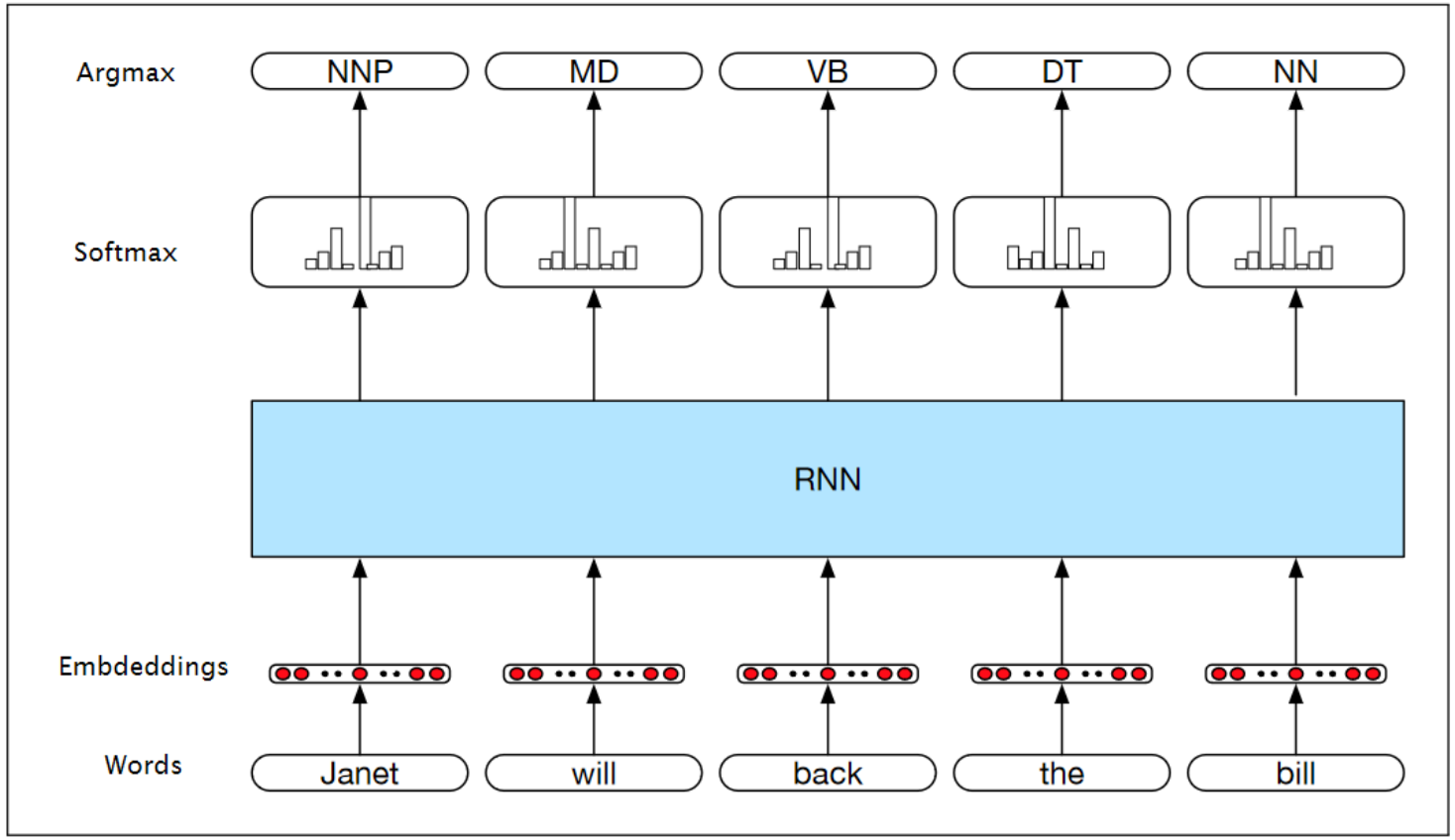


Figure 9.8 Part-of-speech tagging as sequence labeling with a simple RNN. Pre-trained word embeddings serve as inputs and a softmax layer provides a probability distribution over the part-of-speech tags as output at each time step.

Other Applications of RNNs for Sequential Tasks

Language Generation

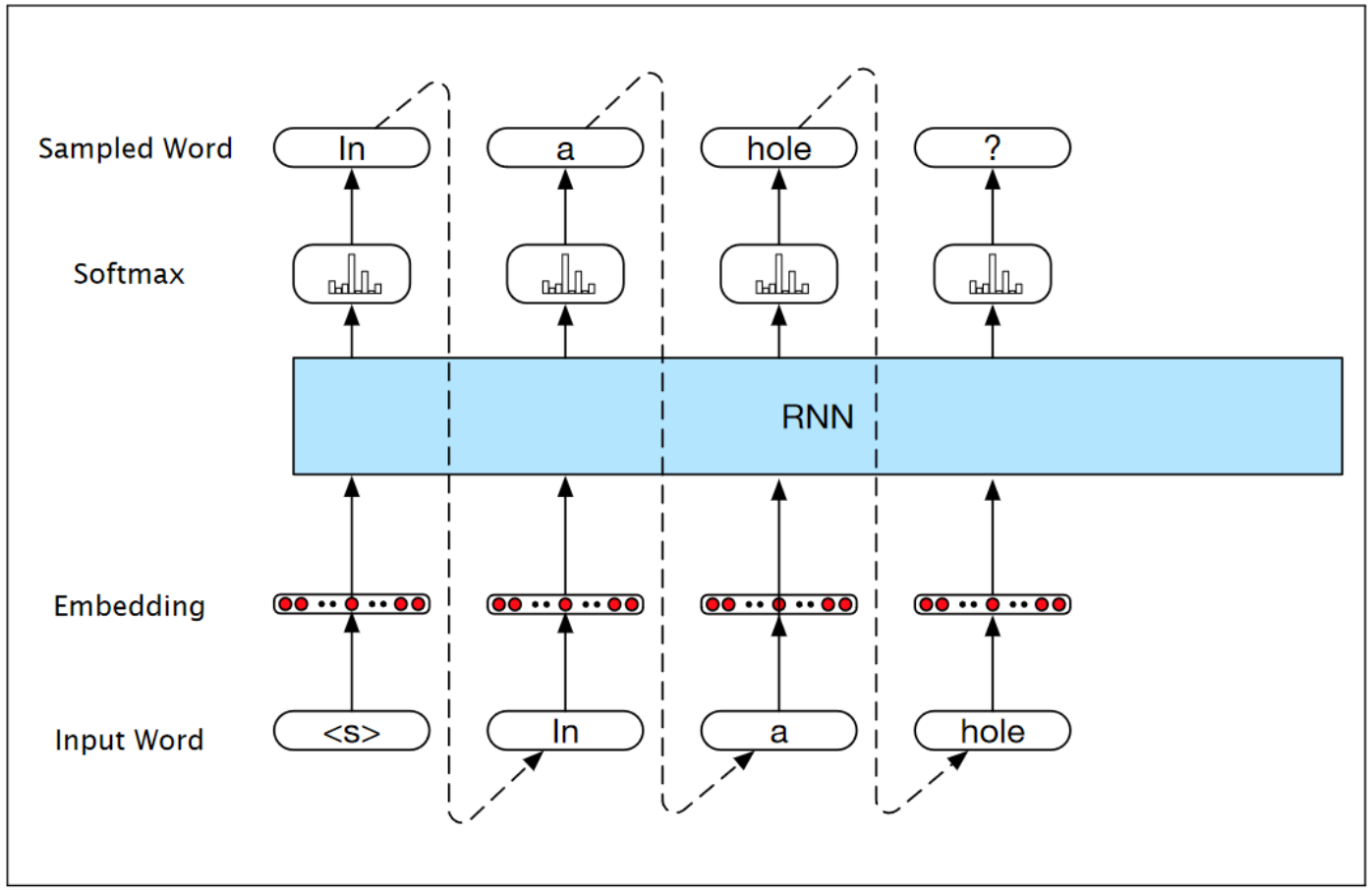


Figure 9.7 Autoregressive generation with an RNN-based neural language model.

Bidirectional RNNs

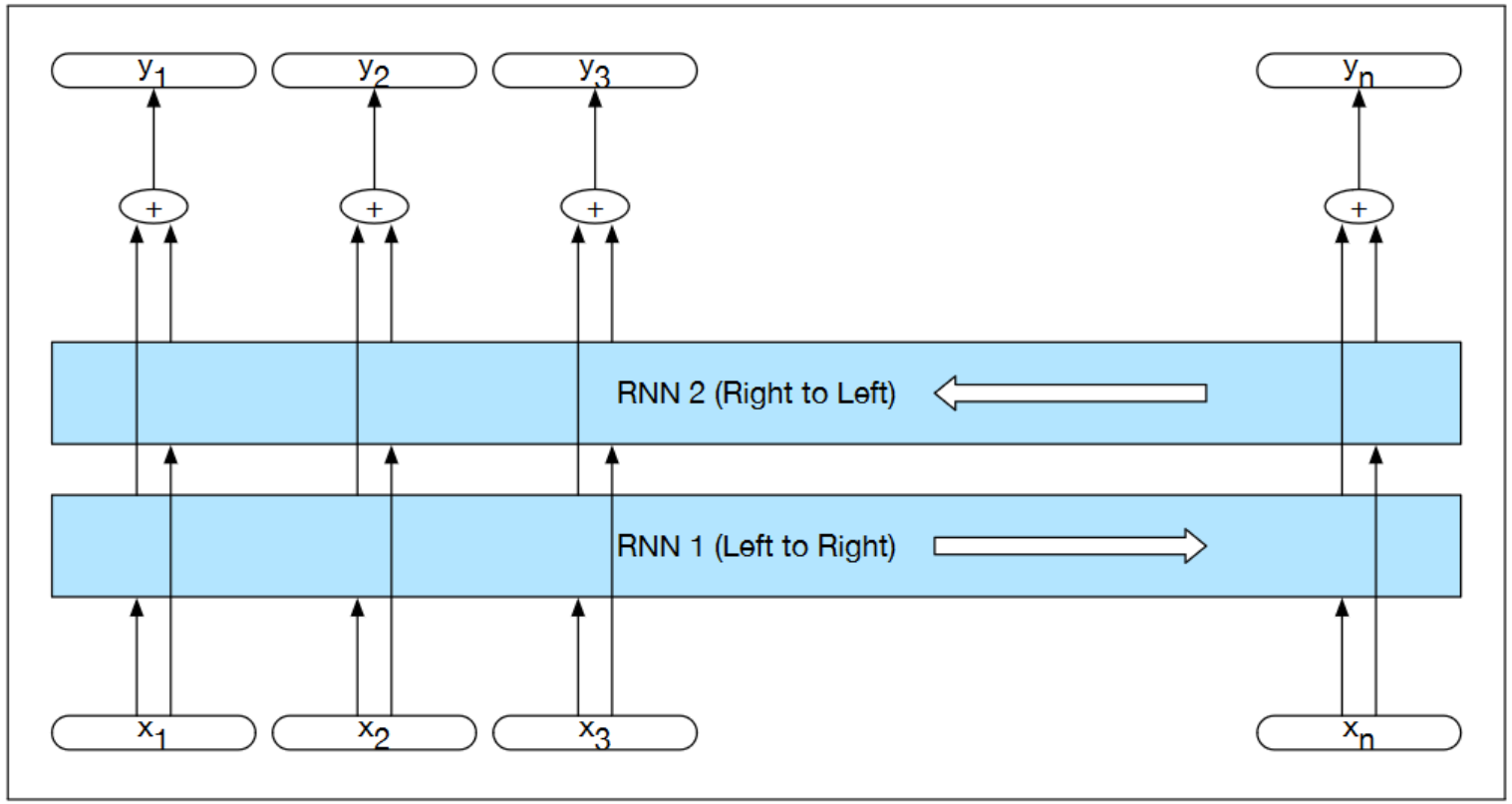


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions with the output of each model at each time point concatenated to represent the state of affairs at that point in time. The box wrapped around the forward and backward network emphasizes the modular nature of this architecture.

Bidirectional RNNs

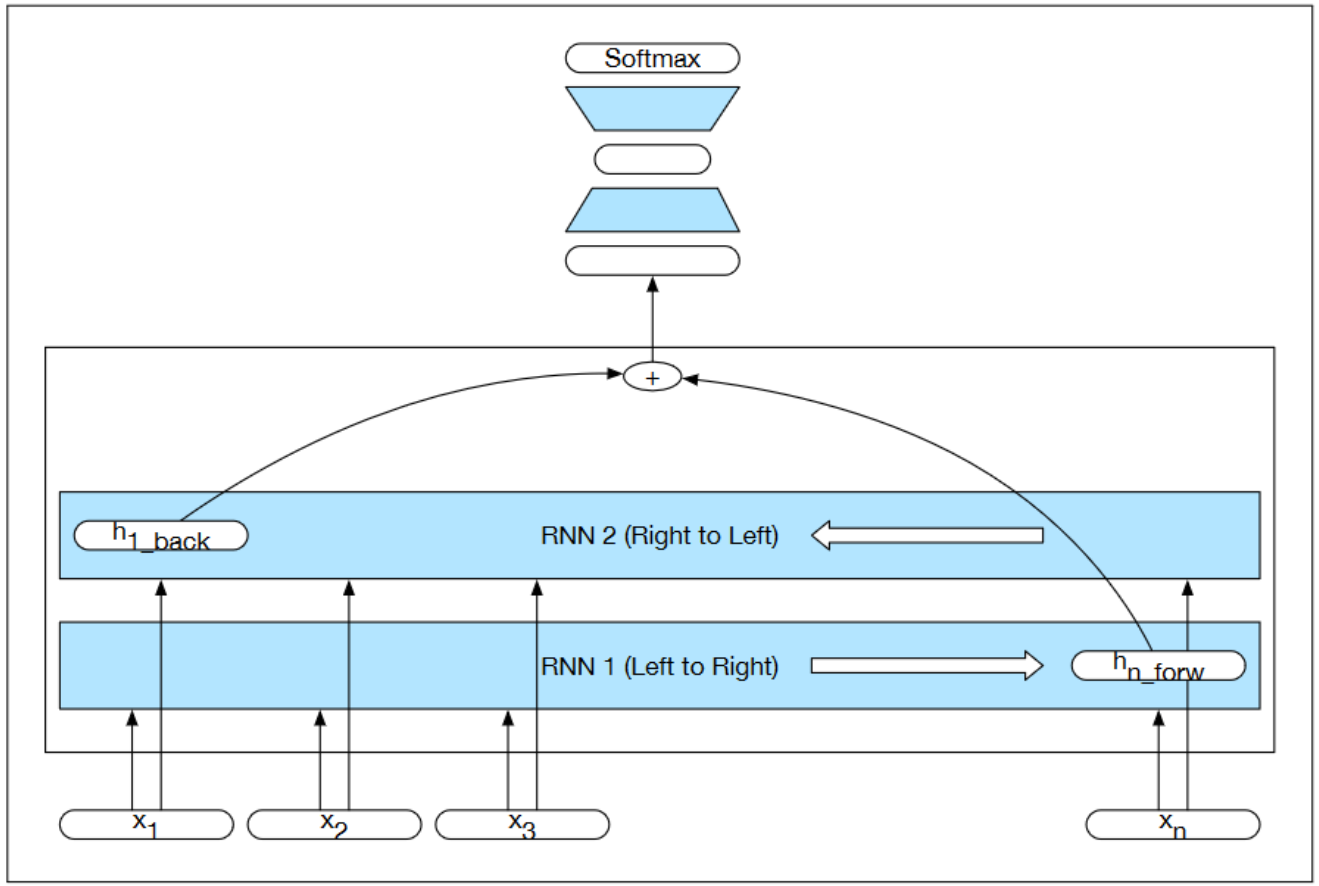
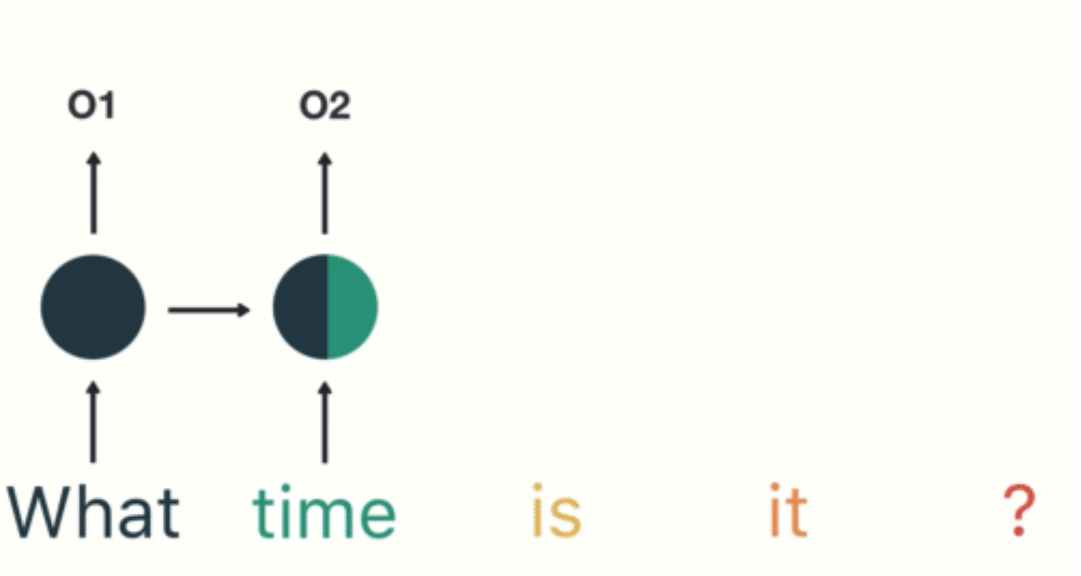


Figure 9.12 A bidirectional RNN for sequence classification. The final hidden units from the forward and backward passes are combined to represent the entire sequence. This combined representation serves as input to the subsequent classifier.

RNNs have “memory loss”

- Problem doing both:
  - Carrying forward critical information
  - Providing useful information for the current decision
- Vanishing gradients problem



Long Short-Term Memory (LSTM)

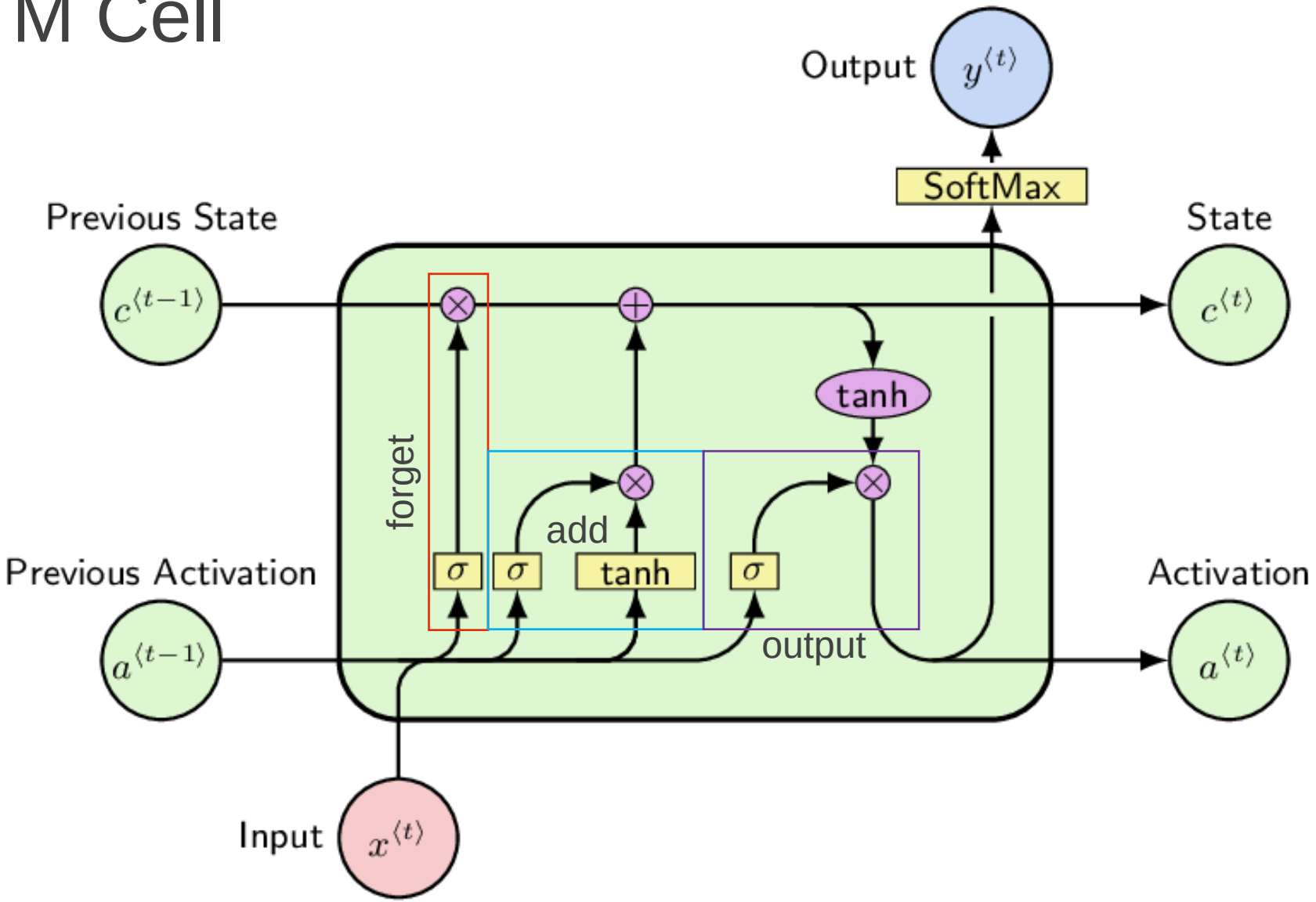
- RNNs with neural units including a special *cell state* and a series of *gates*
- Learns what to add and remove from the network
- Cell state is updated (long-term memory)
- The gates protect and control the cell state

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.

LSTM Gates

- Forget:
  - Delete information from the context that is no longer needed
- Add:
  - Select past information to add to the current context
- Output:
  - Keep information needed for the current hidden state
- Each gate:
  - Feed-forward layer
  - Sigmoid activation function
  - Pointwise multiplication with the layer being gated

LSTM Cell


Output
$y^{\langle t \rangle}$
SoftMax
Previous State
$c^{\langle t-1 \rangle}$
State
$c^{\langle t \rangle}$
tanh
forget
add
Previous Activation
$a^{\langle t-1 \rangle}$
σ
σ
tanh
σ
output
Activation
$a^{\langle t \rangle}$
Input
$x^{\langle t \rangle}$

LSTM Calculations per Timestep

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

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

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

$$c'_t = tanh(W_c x_t + U_c h_{t-1} + b_c)$$

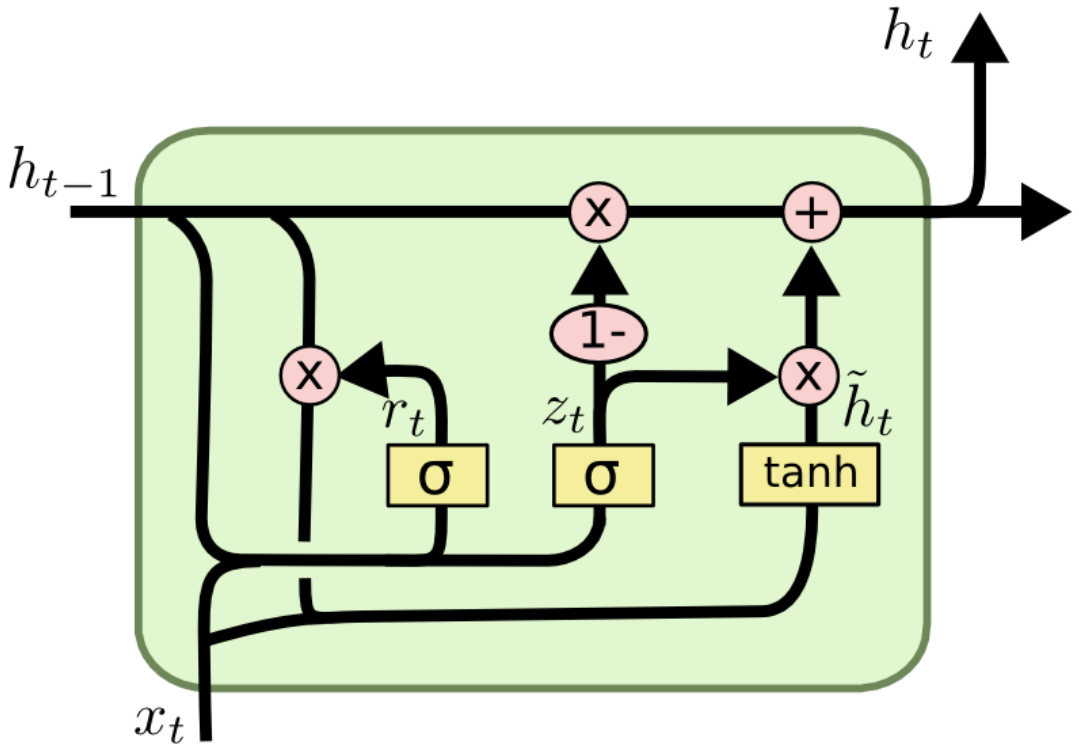
$$c_t = f_t \cdot c_{t-1} + i_t \cdot c'_t$$

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

Gated Recurrent Unit (GRU)

- LSTMs have a considerable training cost:
  - 8 matricies etc.
- GRUs are like LSTMs but have only two gates:
  - Update and reset
- And no special *cell state*

GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Neural Units

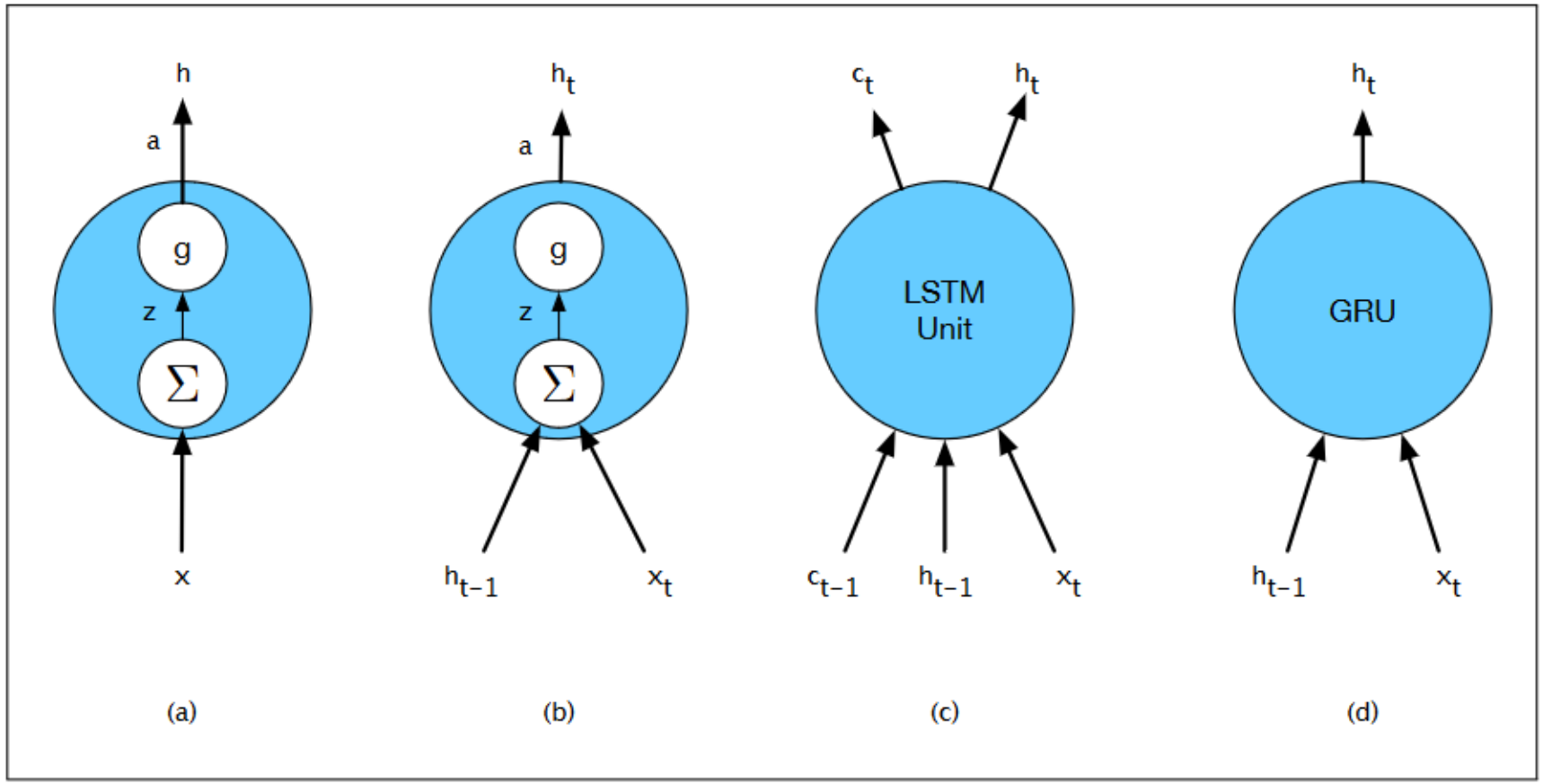


Figure 9.14 Basic neural units used in feedforward, simple recurrent networks (SRN), long short-term memory (LSTM) and gate recurrent units.

State-of-the-art PoS Tagging for Icelandic

Augmenting a BiLSTM Tagger with a Morphological Lexicon and a Lexical Category Identification Step

Steinþór Steingrímsson, Örvar Kárason, Hrafn Loftsson

	Acc.	Known	Unknown
TnT	90.45%	91.82%	71.82%
IceTagger	92.73%	93.84%	**77.47%**
+ DMII	93.48%	93.85%	60.50%
IceStagger	92.82%	93.97%	77.03%
+ DMII	93.70%	94.02%	61.45%
+ DMII,WE	93.84%	94.15%	61.99%
Our model	**95.15%**	**95.48%**	54.06%

	Acc.	Known	Unknown
Baseline	93.25%	95.19%	**66.84%**
+ DMII	94.84%	95.17%	54.61%
+ LC	**95.15%**	**95.48%**	54.06%

Self-Attention Networks: Transformers

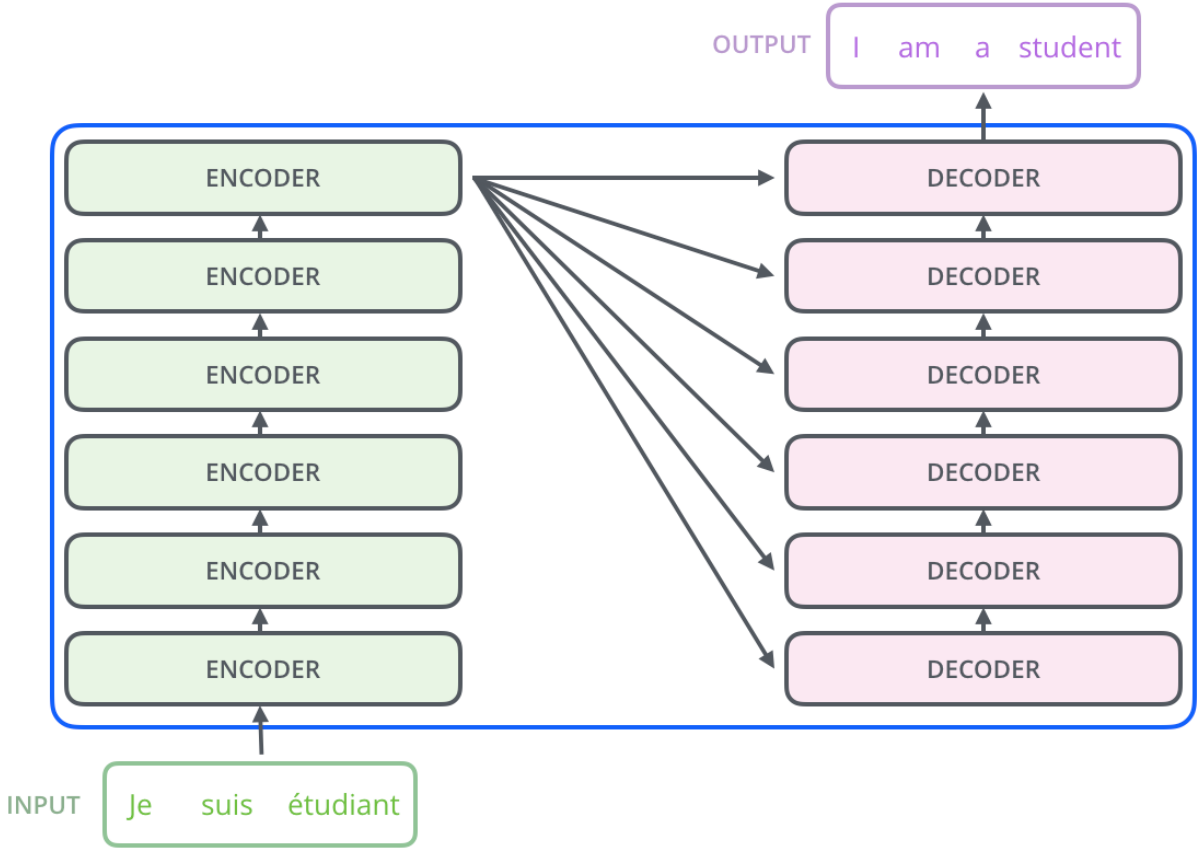
Improving on Recurrent Networks

- Passing information forward is still a problem, even for LSTMs
- RNNs inhibit use of parallel computing

- Transformers eliminate recurrent connections

Transformers

- Take sequences of vectors as input and output sequences of vectors
- Example: Machine translation
- Stacks of linear layers, feed forward networks, and custom connections



Key Innovation: Self-Attention

- Extracts information from arbitrarily large contexts
- No need for intermediate recurrent connections

- **Focus today:**
  - Transformer language models and autoregressive generation

Self-Attention

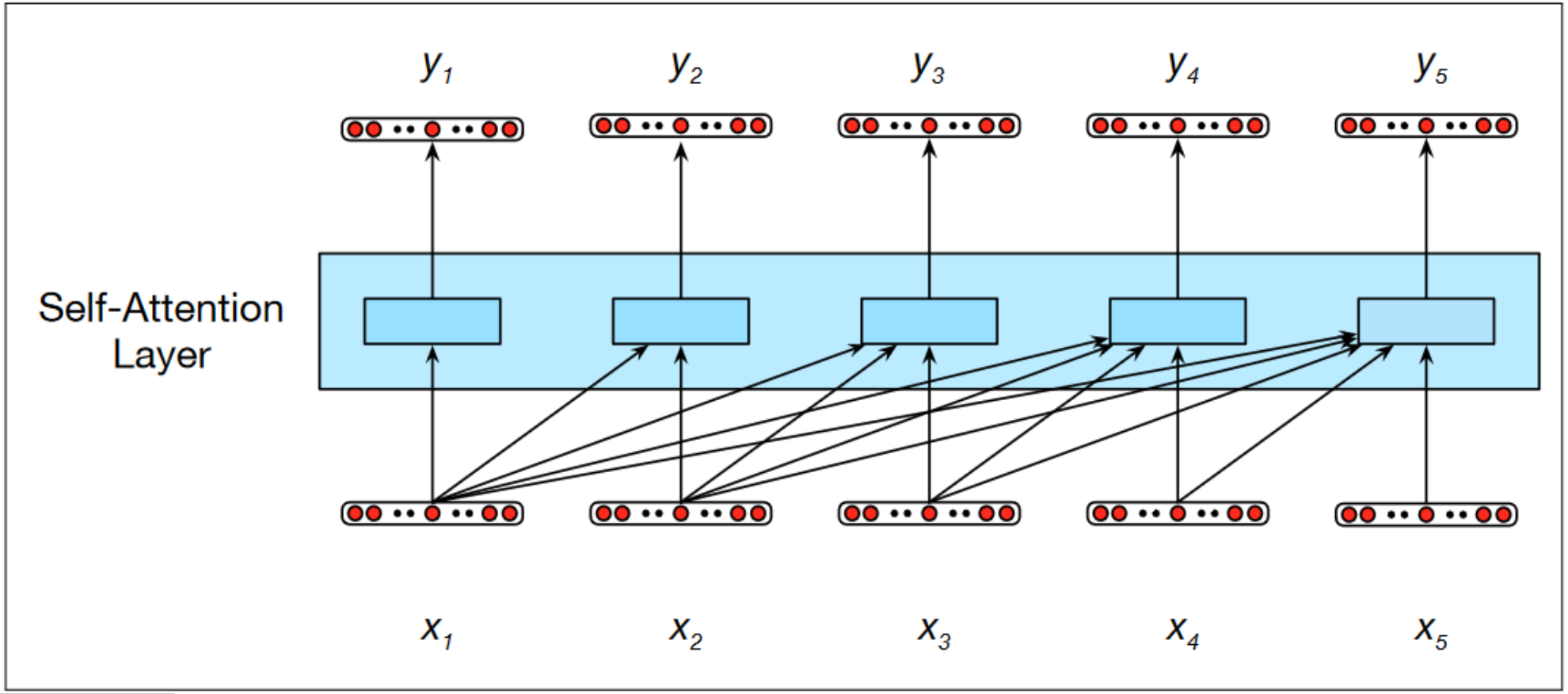


Figure 9.15 Information flow in a causal (or masked) self-attention model. In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one. Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

Simple Self-Attention

- Compare current item to previous ones to reveal its importance
- The result from a set of comparisons is used to compute the next output

$$score(x_i, x_j) = x_i \cdot x_j$$

$$\alpha_{ij} = softmax\left(score(x_i, x_j)\right) \forall j \leq i$$

$$= \frac{\exp(score(x_i, x_j))}{\sum_{k=1}^{i} \exp(\text{score}(x_i, x_k))} \forall j \leq i$$

$$y_i = \sum_{j \leq i} \alpha_{ij} x_j$$

Learned Self-Attention

- Use a set to weight matrices to operate over input embeddings
- Input embeddings play different roles:
  - *Key* - The current focus of attention
  - *Query* - A preceding input
  - *Value* - The output for the current focus of attention

$$q_i = W^Q x_i$$

$$k_i = W^K x_i$$

$$v_i = W^V x_i$$

Updated Equations for Self-Attention

$$score(x_i, x_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = softmax\left(score(x_i, x_j)\right) \forall j \leq i$$

$$y_i = \sum_{i \leq j} \alpha_{ij} v_j$$

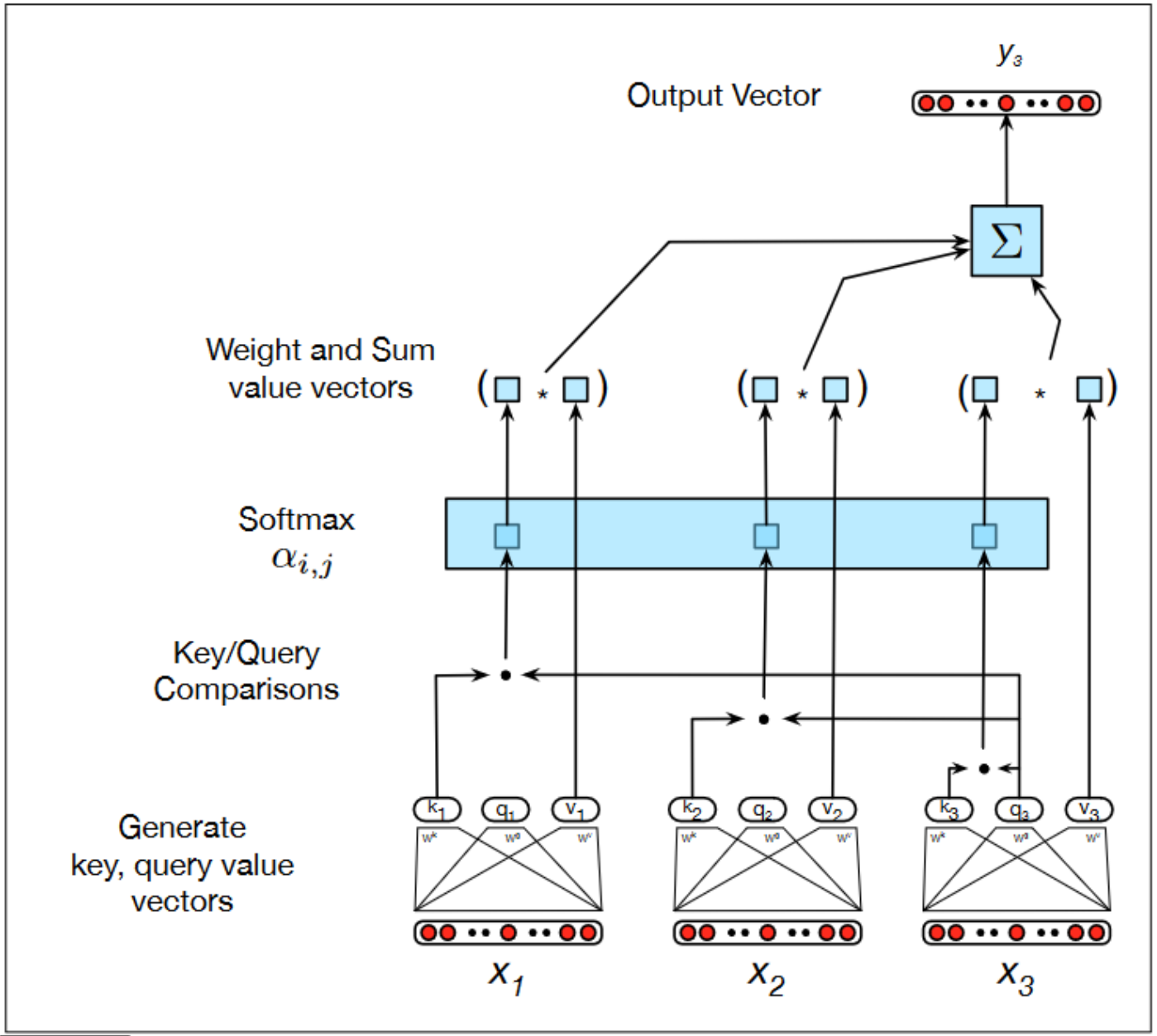


Figure 9.16 Calculation of the value of the third element of a sequence using causal self-attention.

Parallelized Self-Attention

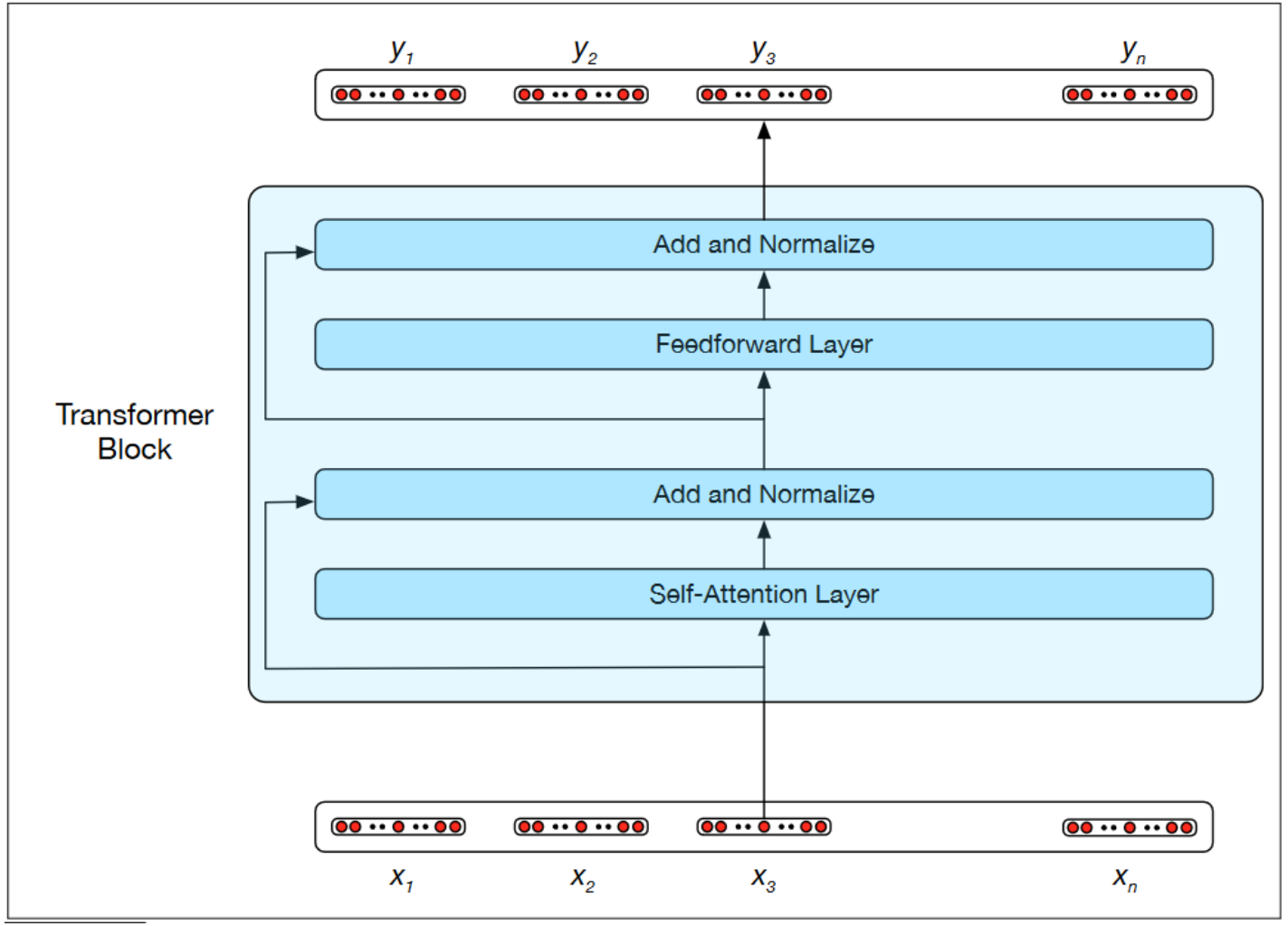
- Input embeddings packed into a single matrix X

$$Q = W^Q X \qquad K = W^K X \qquad V = W^V X$$

$$SelfAttention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Transformer Blocks

- Self-attention layer
- Feedforward layers
- Residual connections
- Normalizing layers



Multihead Attention

- Each block has multiple self-attention heads/layers
- Each has its own set of key, query and value matrices

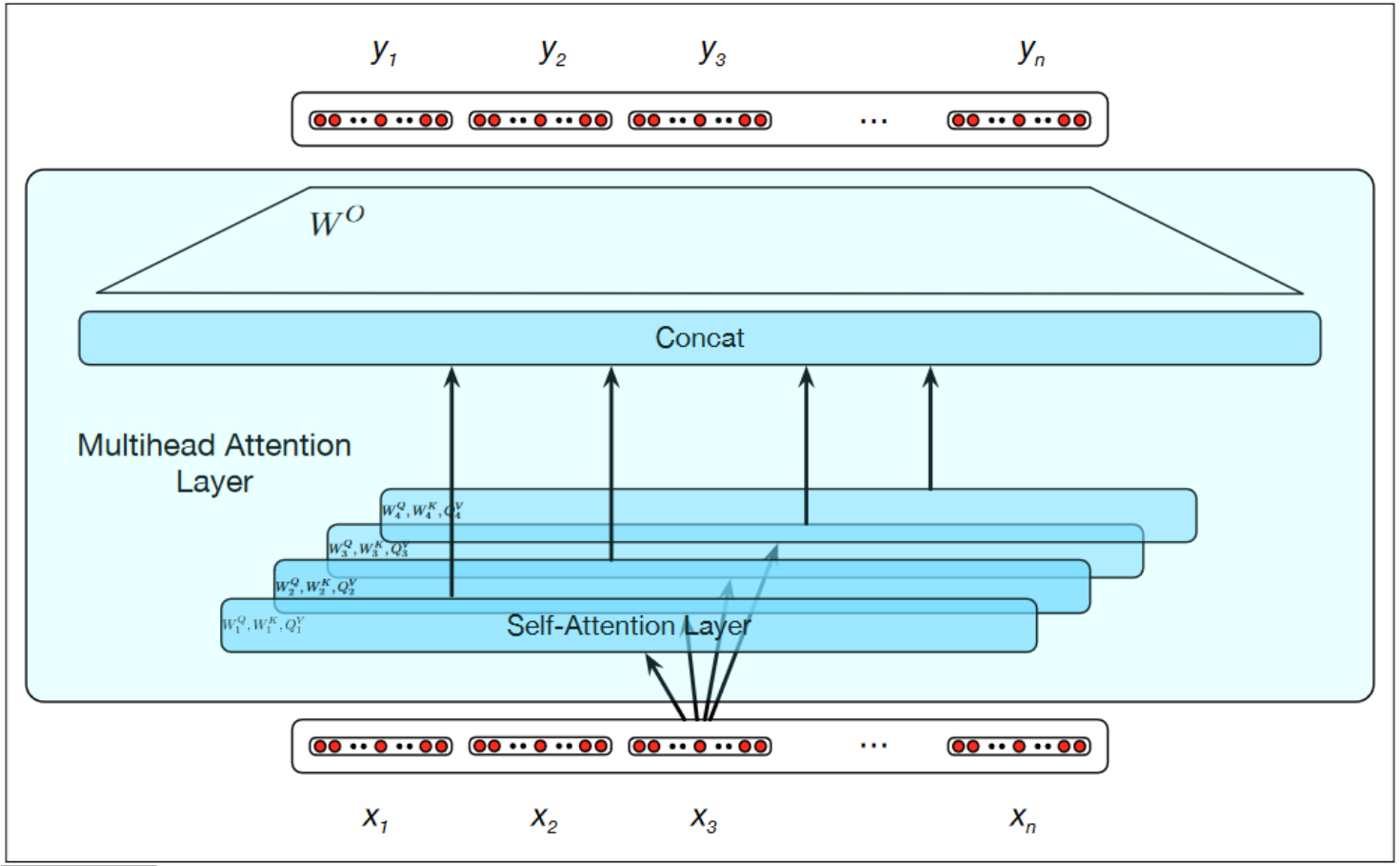
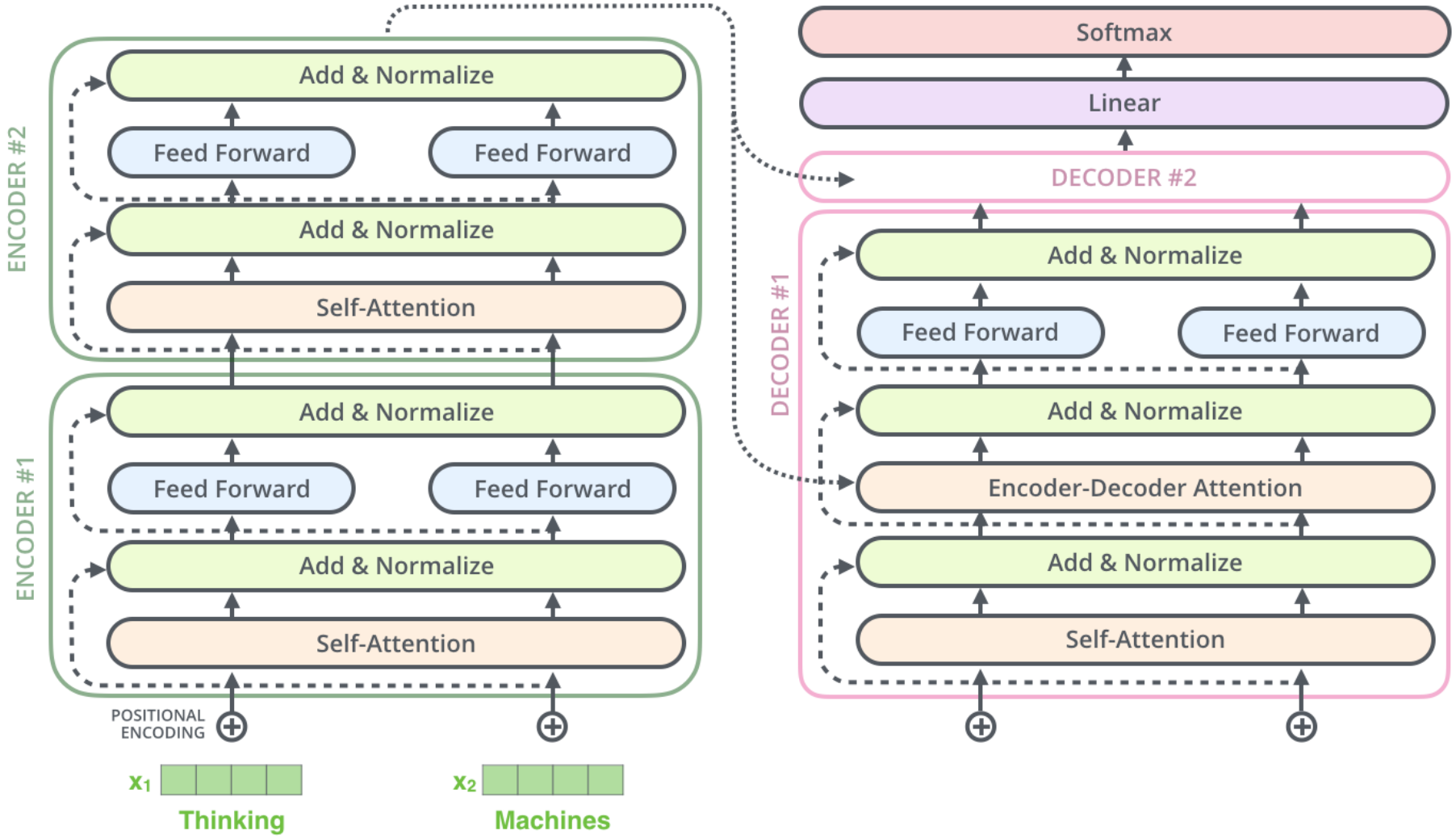


Figure 9.18 Multihead self-attention: Each of the multihead self-attention layers is provided with its own set of key, query and value weight matrices. The outputs from each of the layers are concatenated and then projected down to d_{model}, thus producing an output of the right size.

Positional Embeddings

- The order of inputs is not self-evident as in RNNs
- Information about the position of elements in the input are added to the input embeddings
- Various methods like using **sine** and **cosine** functions with differing frequencies

Transformer Guts



Training a Transformer Language Model

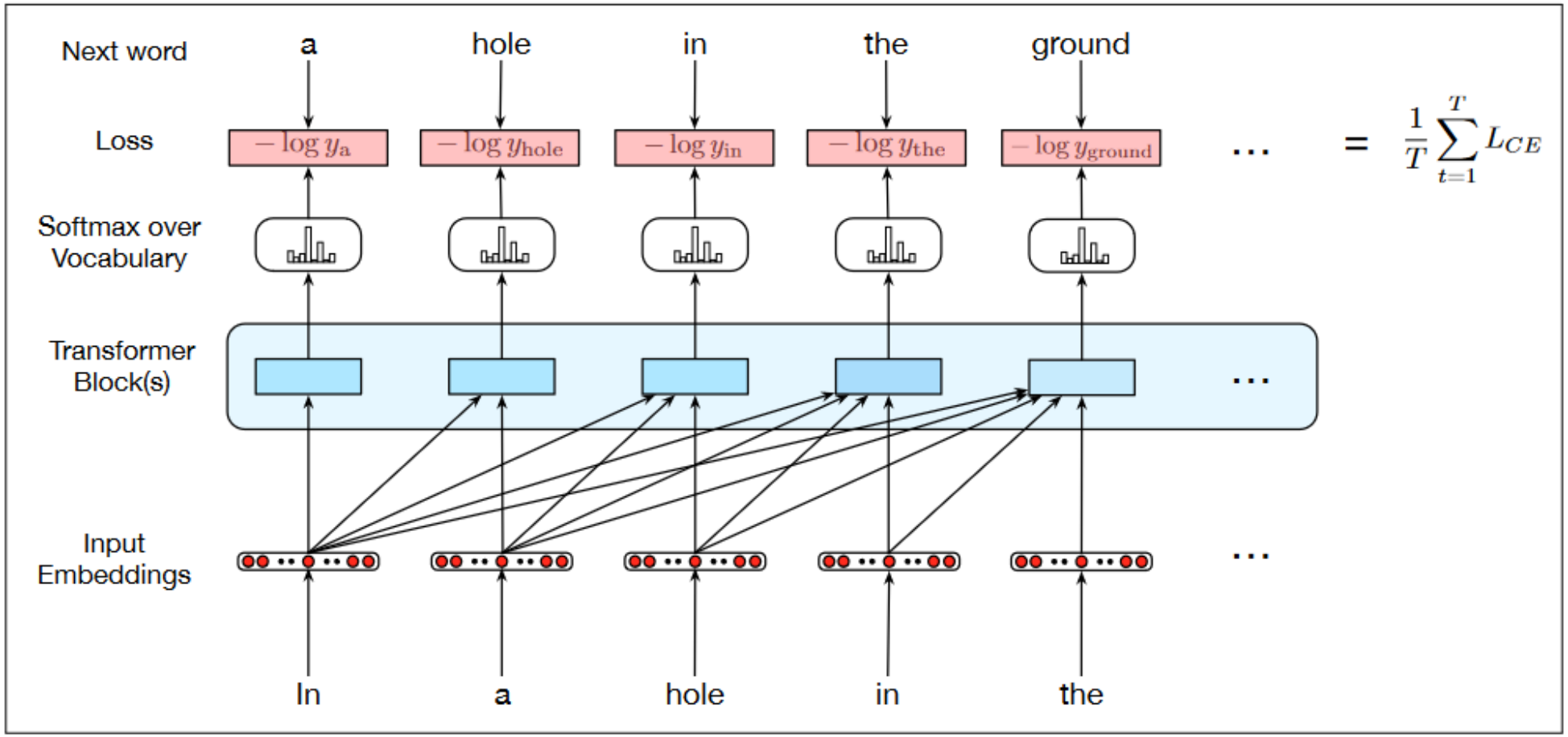
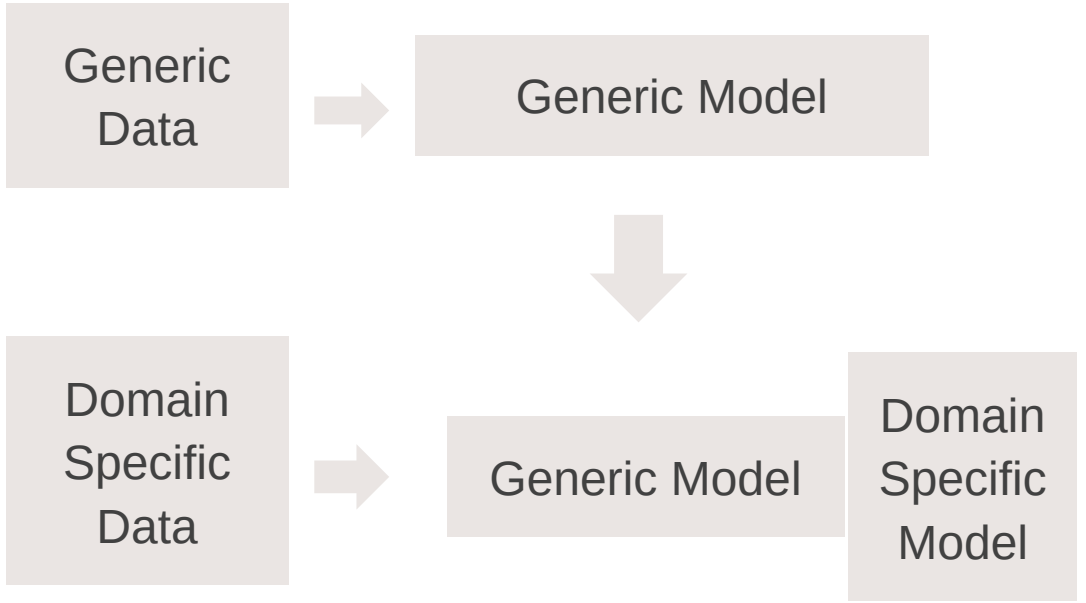


Figure 9.19 Training a Transformer as a language model.

Transformers for Transfer Learning

- Train a giant model of a language, for example English
- Have a smaller dataset for a specific domain
- Use the giant model as a starting point and fine-tune it to your smaller dataset



Use Case – LSTMs and Transformers for Dialog Move Prediction

Model	F1	Precision	Recall	Accuracy
7 Labels Task				
Majority label only	0.07	0.05	0.14	0.34
Linear SVC + TF-IDF	0.26	0.25	0.28	0.37
CRF + Doc2Vec	0.28	0.26	0.29	0.46
ConLSTM	0.32	0.35	0.32	0.42
BERT-base w/fine-tuning	0.37	0.34	0.37	0.47
ConLSTM-CRF	0.41	0.44	0.4	0.5
ConLSTM-CRF + BERT	0.41	0.44	0.39	0.5
5 Labels Task				
Majority label only	0.11	0.07	0.2	0.37
ConLSTM-CRF	0.62	0.65	0.61	0.59

Table 5. Experimental results using various methods for predicting the next counselor move, given dialog context, and a comparison to predicting the majority label only. The columns show macro-averages.

Use Case – Transformer Transfer-Learning for Contextual Language Generation

- GPT-2 large language model of English fine-tuned on counselor-client dialog about alcohol use
- Task: Generate counselor *reflections* automatically given dialog context

> *Context:* Yeah, I'm good with that. I've got to make some big changes. I've been thinking about this for a while though. This just came at a good time. So, yeah, I've been just, you know what I mean? I've been thinking about doing a lot of different stuff.<reflect><3.53>
> GENERATED: So, drinking is not something you want to do in your life.
>
> *Context:* Then everybody's on their way over, and I change my mind because I'm not even into it anymore. It's only a spur of the ... if I'm gonna drink and it's not a birthday or a celebration of some kind, then it's spur of the moment. I never plan it I'm just like, Eh, let's go to the liquor store and grab a couple of things and that's it.<reflect><3.1>
> GENERATED: So, you had said, you don't plan for it.