Natural Language Processing (CSE 447/517)

Winter 2026 ● Noah Smith

Many figures from Jurafsky and Martin ch. 7-8

“One-hot” vectors allow lookup of a word’s embedding

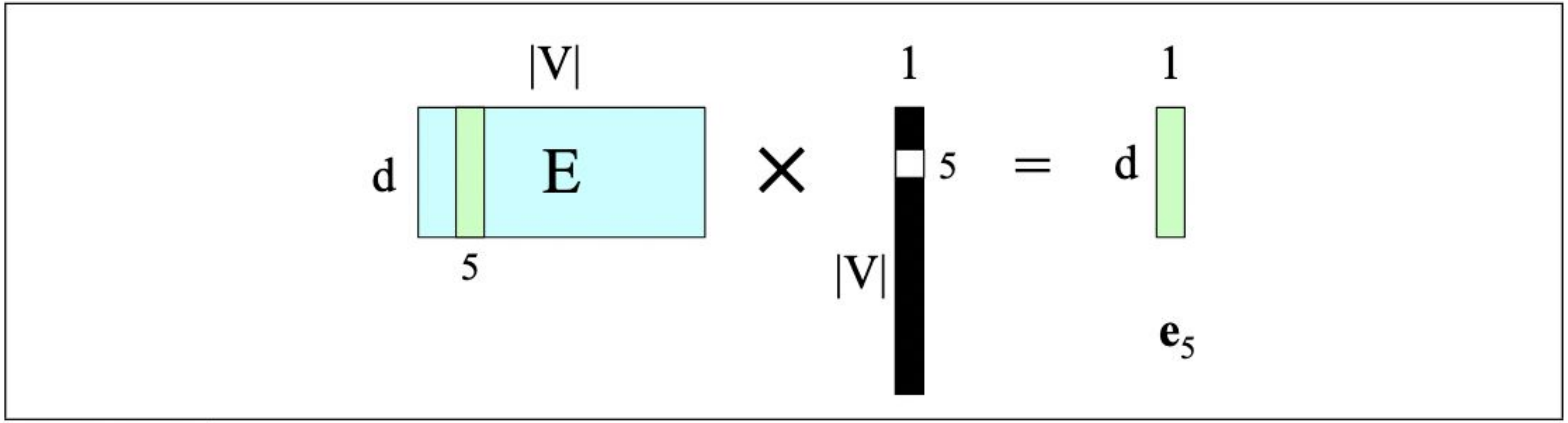


Figure 7.16 Selecting the embedding vector for word V_5 by multiplying the embedding matrix **E** with a one-hot vector with a 1 in index 5.

Feedforward neural language model

$$
\begin{aligned}
\mathbf{e} &= [\mathbf{E}\mathbf{x}_{t-3}; \mathbf{E}\mathbf{x}_{t-2}; \mathbf{E}\mathbf{x}_{t-1}] \\
\mathbf{h} &= \sigma(\mathbf{W}\mathbf{e} + \mathbf{b}) \\
\mathbf{z} &= \mathbf{U}\mathbf{h} \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{z})
\end{aligned}
$$

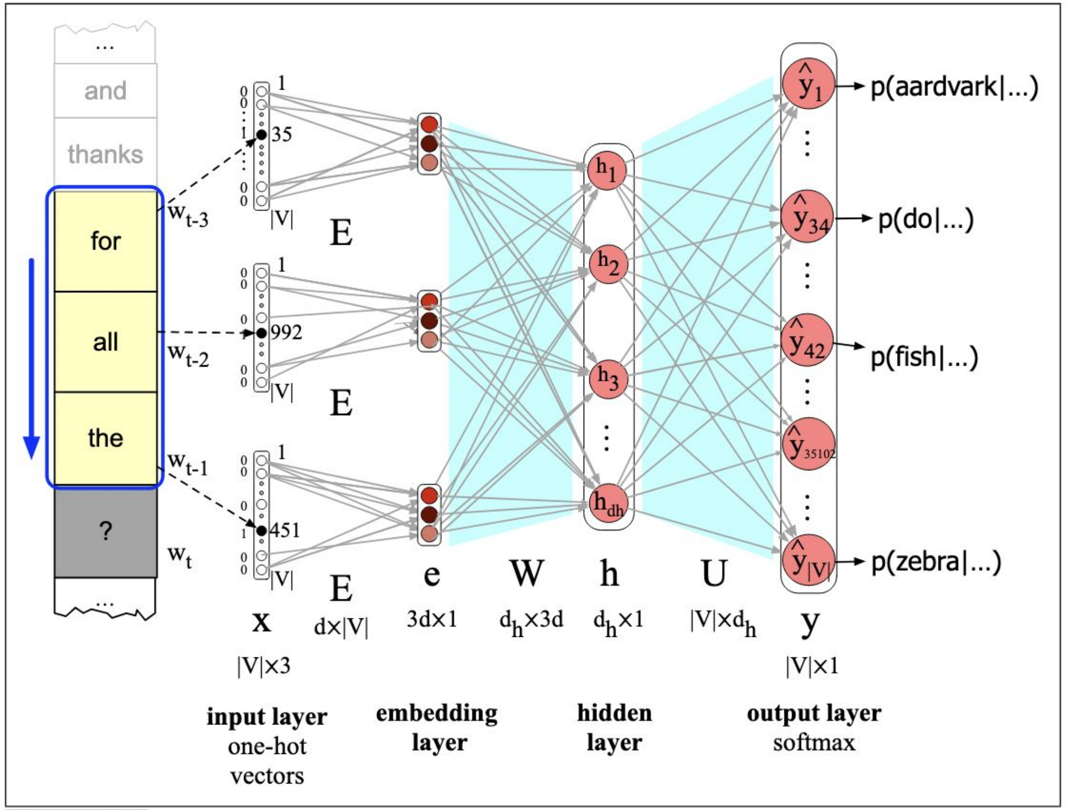


Figure 7.17 Forward inference in a feedforward neural language model. At each timestep *t* the network computes a *d*-dimensional embedding for each context word (by multiplying a one-hot vector by the embedding matrix **E**), and concatenates the 3 resulting embeddings to get the embedding layer **e**. The embedding vector **e** is multiplied by a weight matrix **W** and then an activation function is applied element-wise to produce the hidden layer **h**, which is then multiplied by another weight matrix **U**. Finally, a softmax output layer predicts at each node *i* the probability that the next word w_t will be vocabulary word V_i.

Learning a neural 4-gram model (including embeddings)

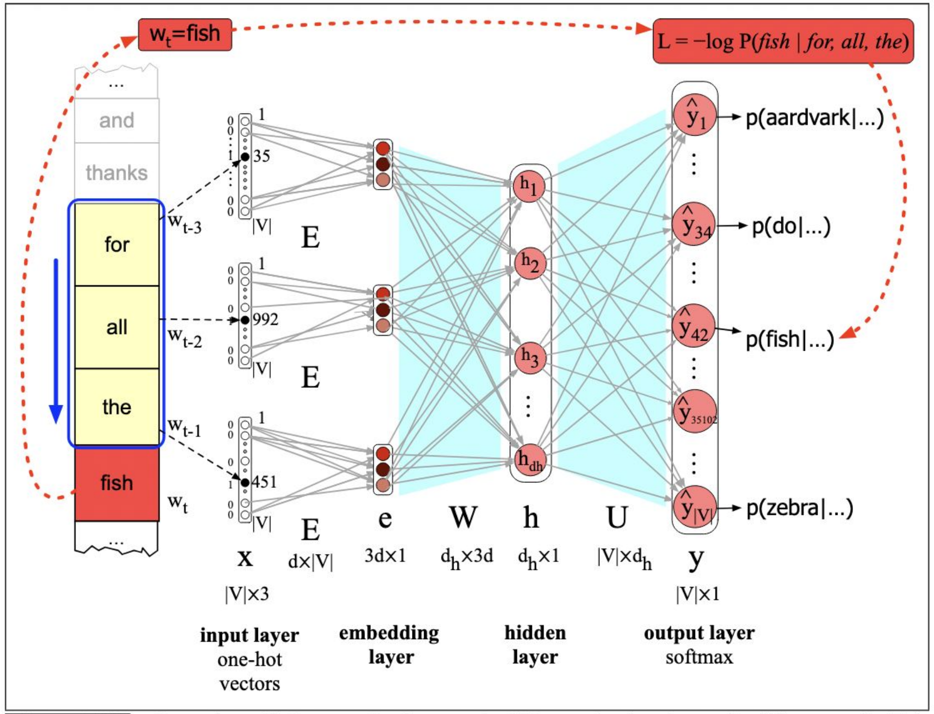


Figure 7.18 Learning all the way back to embeddings. Again, the embedding matrix **E** is shared among the 3 context words.

Simple recurrent neural network

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$$
$$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$$

for LMs, softmax

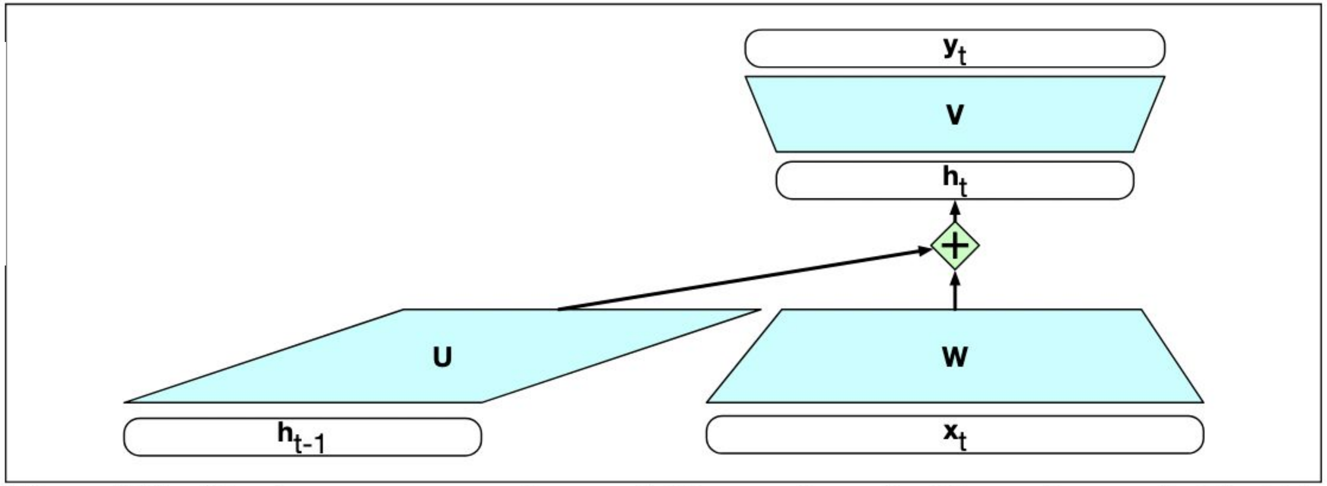


Figure 8.2 Simple recurrent neural network illustrated as a feedforward network. The hidden layer $\mathbf{h}_{t-1}$ from the prior time step is multiplied by weight matrix $\mathbf{U}$ and then added to the feedforward component from the current time step.

“Unrolled” computation in a simple RNN

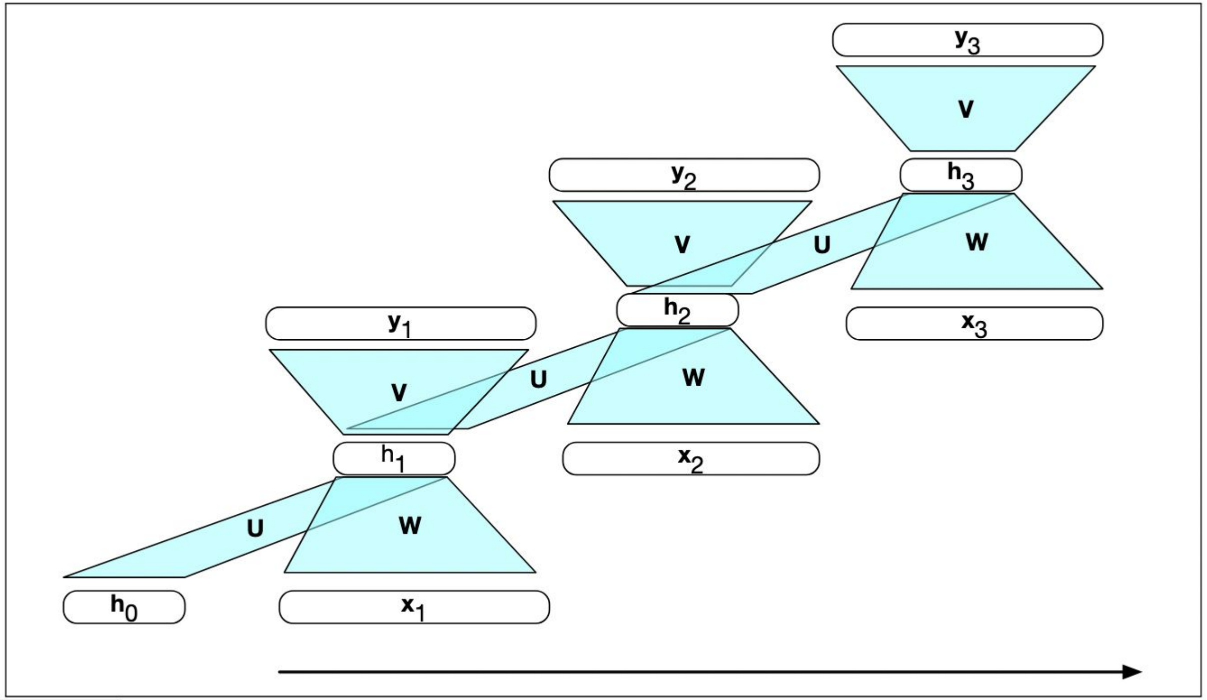


Figure 8.4 A simple recurrent neural network shown unrolled in time. Network layers are recalculated for each time step, while the weights **U**, **V** and **W** are shared across all time steps.

Comparing feedforward (a) and recurrent (b) LMs

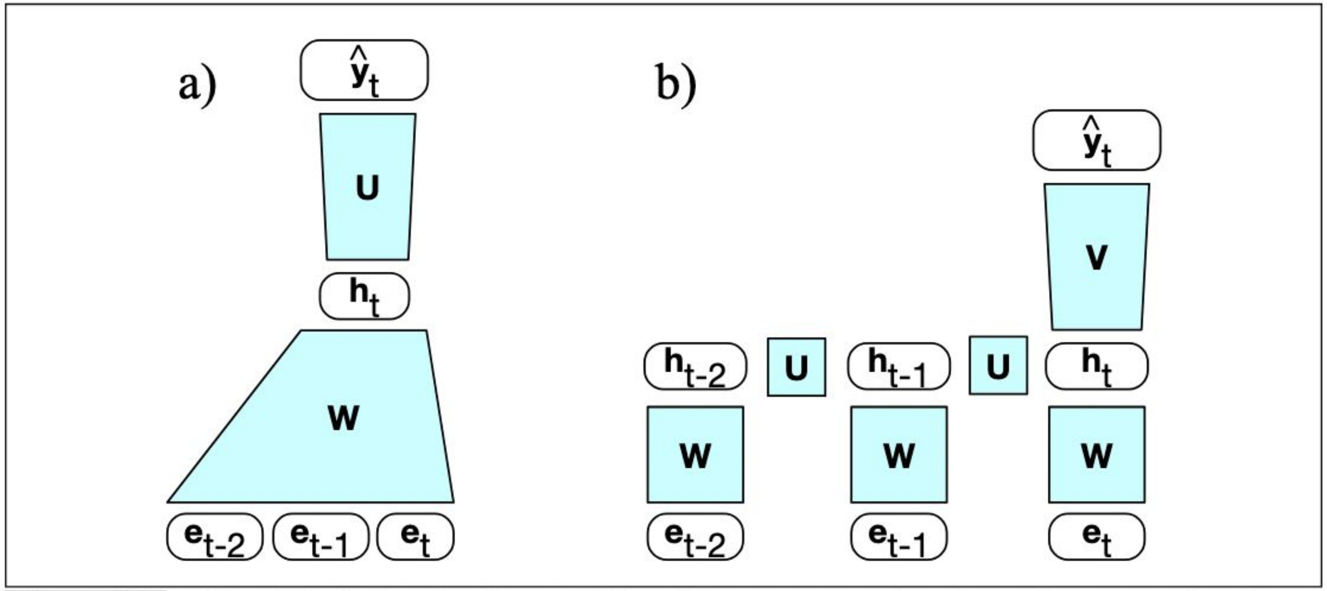


Figure 8.5 Simplified sketch of two LM architectures moving through a text, showing a schematic context of three tokens: (a) a feedforward neural language model which has a fixed context input to the weight matrix $\mathbf{W}$, (b) an RNN language model, in which the hidden state $\mathbf{h}_{t-1}$ summarizes the prior context.

$$\begin{aligned} \mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\ \mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\ \hat{\mathbf{y}}_t &= \text{softmax}(\mathbf{V}\mathbf{h}_t) \end{aligned}$$

Training an RNN language model

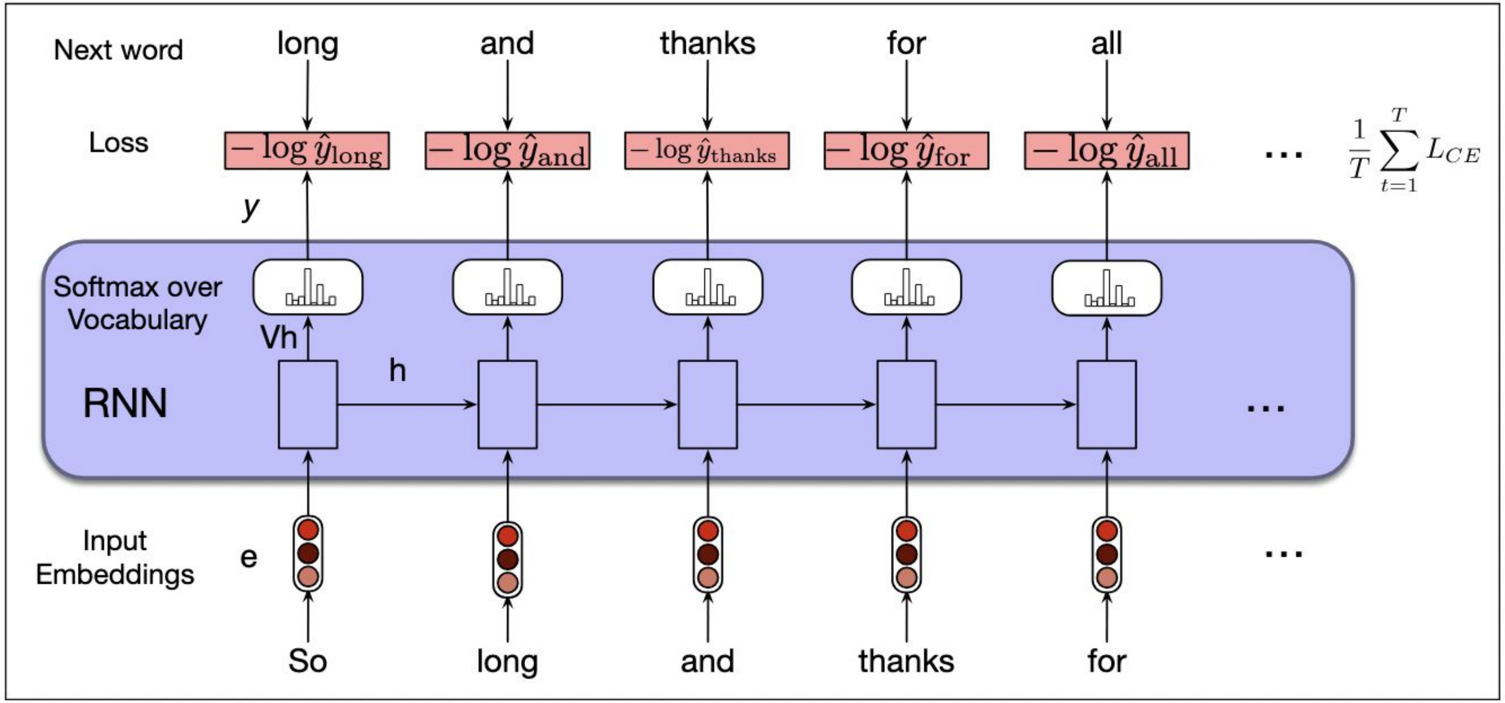

Next word
long
and
thanks
for
all
Loss
$-\log \hat{y}_{\text{long}}$
$-\log \hat{y}_{\text{and}}$
$-\log \hat{y}_{\text{thanks}}$
$-\log \hat{y}_{\text{for}}$
$-\log \hat{y}_{\text{all}}$
...
$\frac{1}{T}\sum_{t=1}^{T} L_{CE}$
y
Softmax over Vocabulary
Vh
h
RNN
...
Input Embeddings
e
...
So
long
and
thanks
for

Figure 8.6 Training RNNs as language models.

Weight tying (reusing the embedding matrix)

$$\begin{aligned} \mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\ \mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\ \hat{\mathbf{y}}_t &= \text{softmax}(\mathbf{E}^\intercal \mathbf{h}_t) \end{aligned}$$

RNN for classification

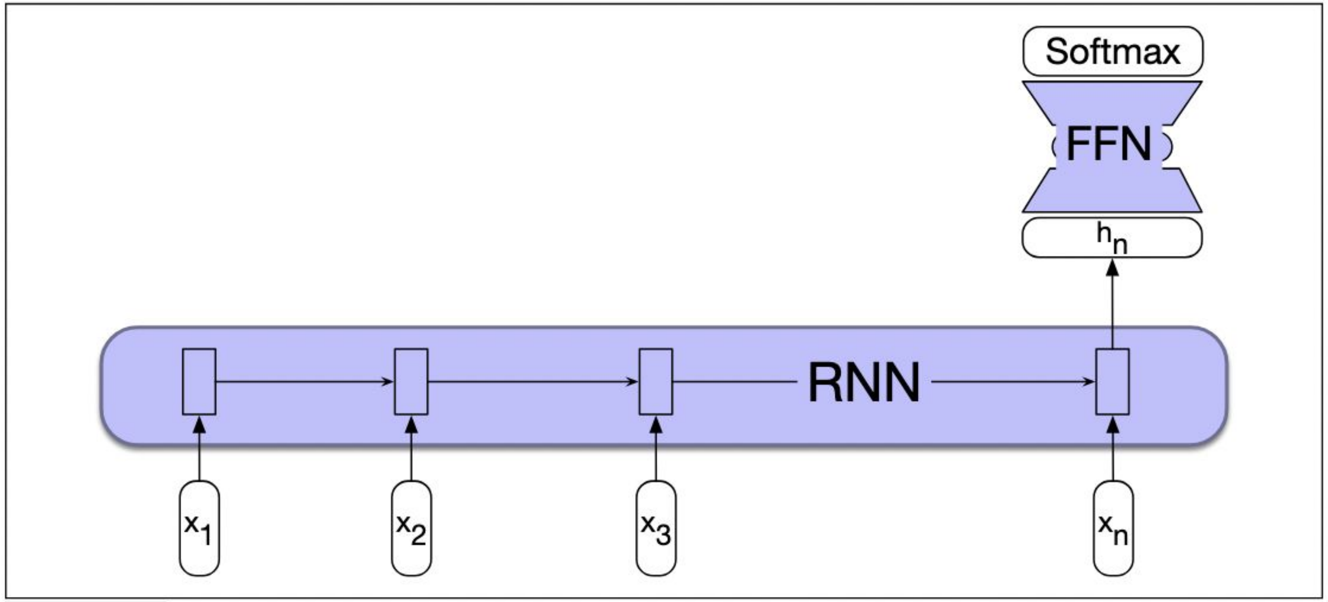


Figure 8.8 Sequence classification using a simple RNN combined with a feedforward network. The final hidden state from the RNN is used as the input to a feedforward network that performs the classification.

Bidirectional RNN

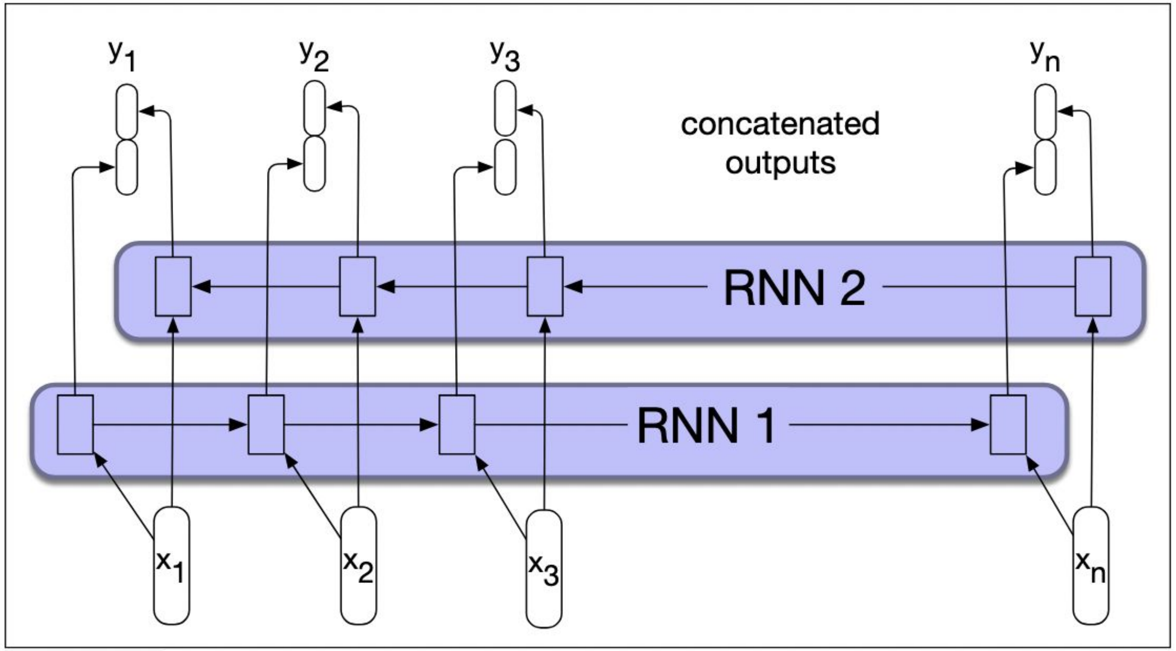


Figure 8.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

RNNs for text generation

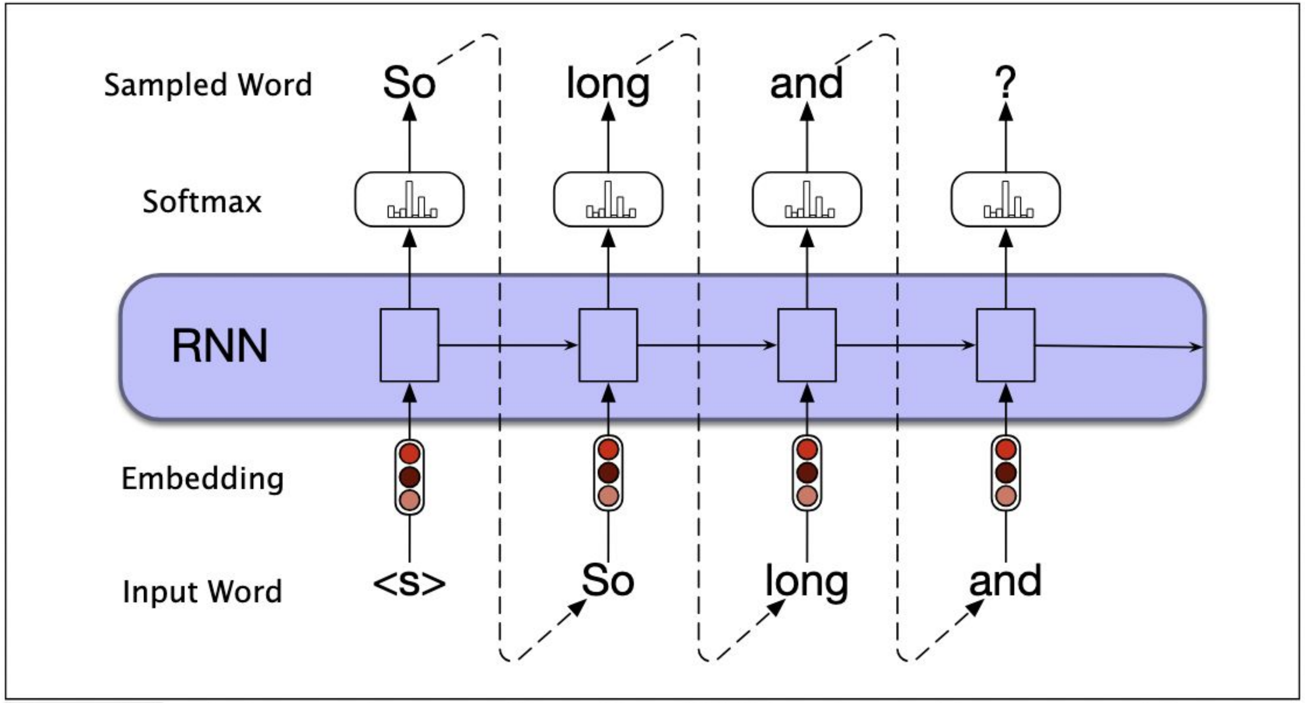


Figure 8.9 Autoregressive generation with an RNN-based neural language model.

Stacked RNN

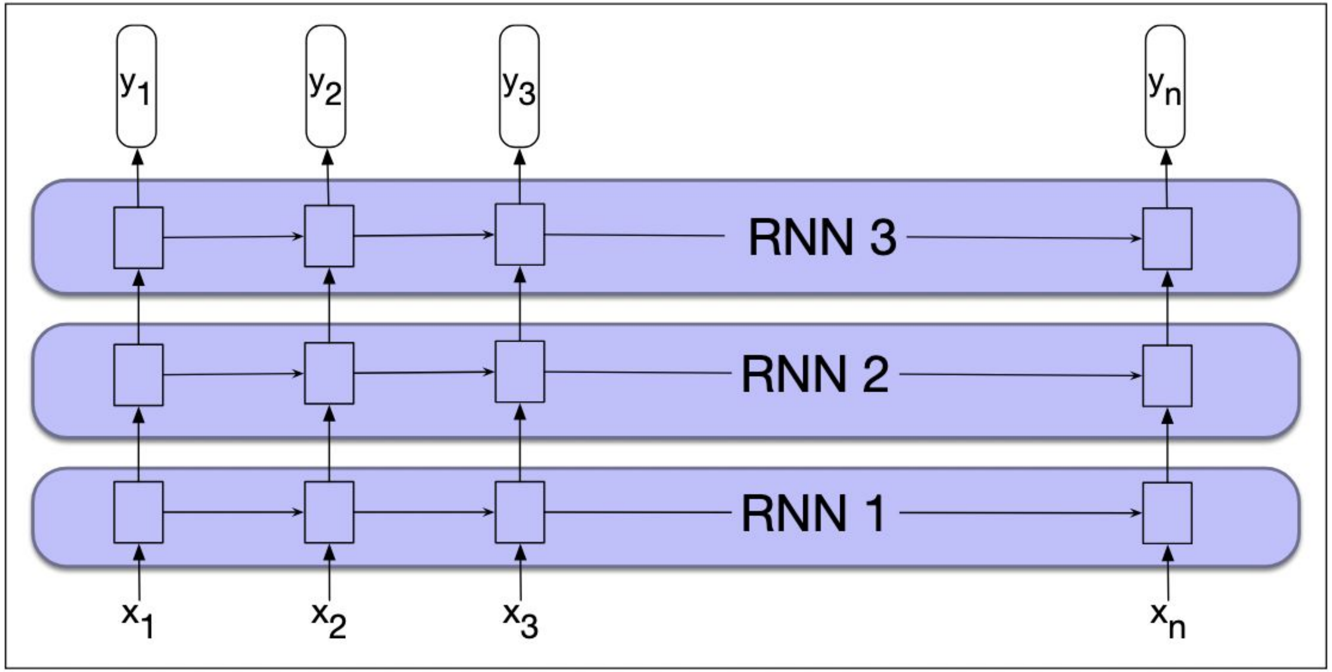


Figure 8.10 Stacked recurrent networks. The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output.

Long short-term memory (single unit)

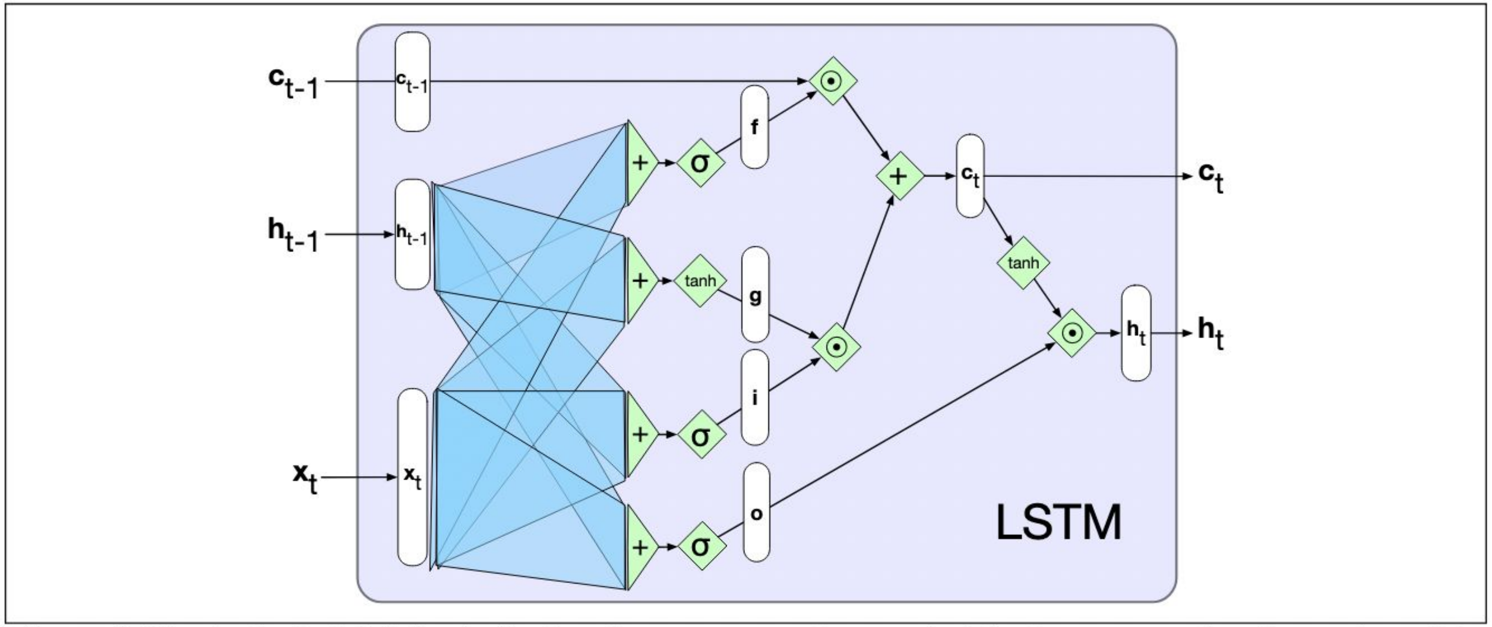


Figure 8.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x, the previous hidden state, h_{t-1}, and the previous context, c_{t-1}. The outputs are a new hidden state, h_t and an updated context, c_t.

Operations inside the LSTM cell

Forgetting some info from the past context:

$$\mathbf{f}_t = \sigma(\mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t)$$

$$\mathbf{k}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t$$ "keep"

Extract information from hidden state and current input:

$$\mathbf{g}_t = \tanh(\mathbf{U}_g \mathbf{h}_{t-1} + \mathbf{W}_g \mathbf{x}_t)$$

Select information to add to the current context:

$$\mathbf{i}_t = \sigma(\mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{W}_i \mathbf{x}_t)$$

$$\mathbf{j}_t = \mathbf{g}_t \odot \mathbf{i}_t$$ "update"

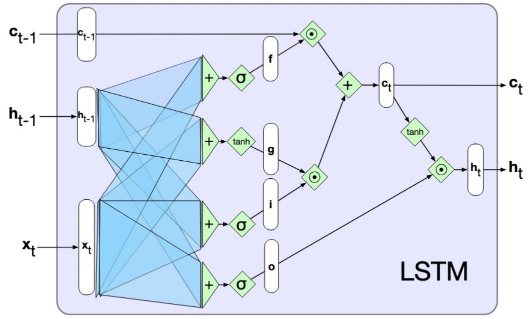
"update" "keep"

Generate current context: $\mathbf{c}_t = \mathbf{j}_t + \mathbf{k}_t$

Output gate to build current hidden state:

$$\mathbf{o}_t = \sigma(\mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{W}_o \mathbf{x}_t)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$



Comparing neural units

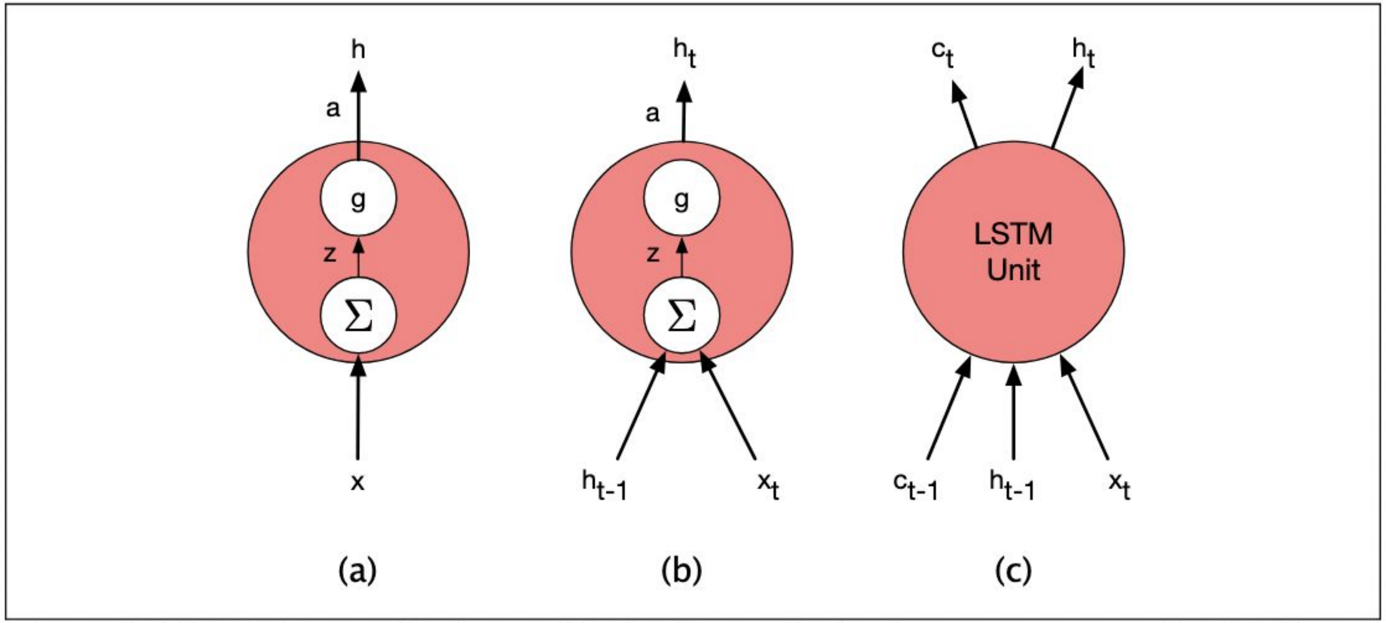


Figure 8.14 Basic neural units used in feedforward, simple recurrent networks (SRN), and long short-term memory (LSTM).

RNN architectures

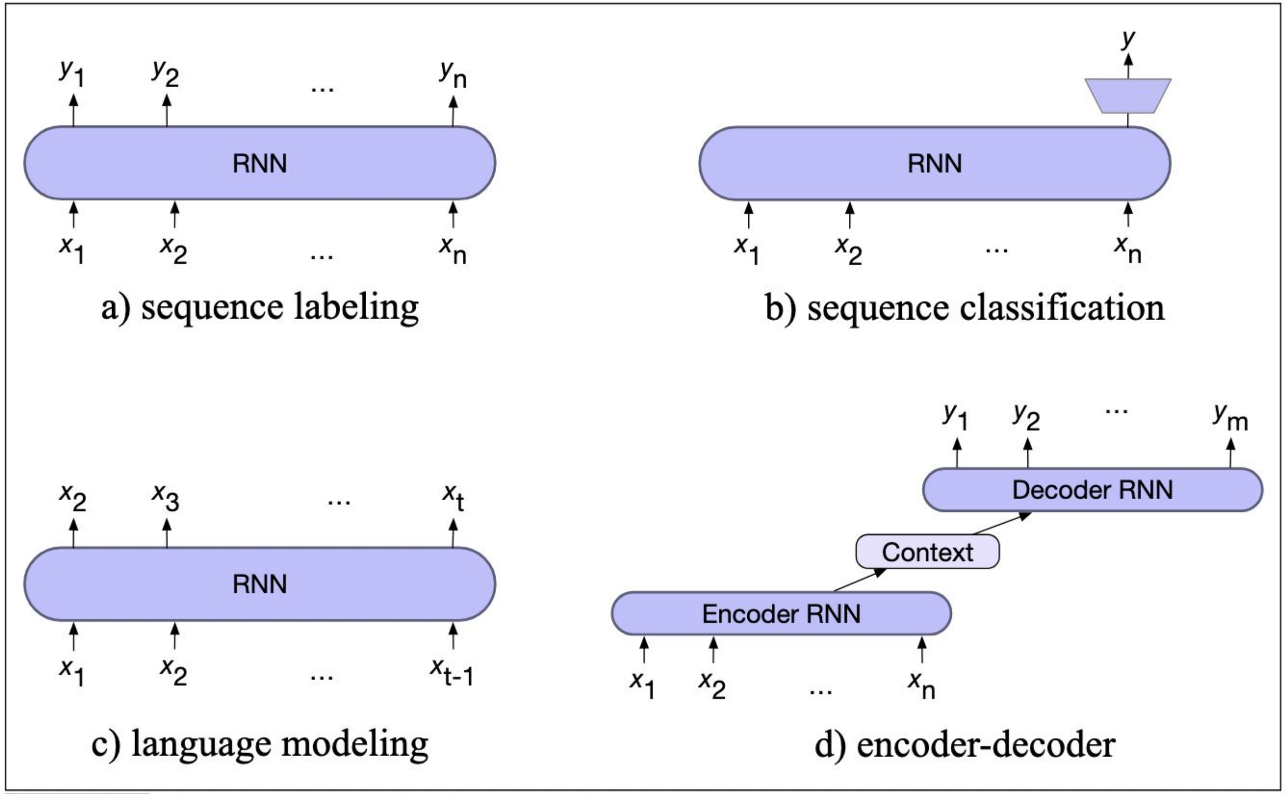


Figure 8.15 Four architectures for NLP tasks. In sequence labeling (POS or named entity tagging) we map each input token x_i to an output token y_i. In sequence classification we map the entire input sequence to a single class. In language modeling we output the next token conditioned on previous tokens. In the encoder model we have two separate RNN models, one of which maps from an input sequence $\mathbf{x}$ to an intermediate representation we call the **context**, and a second of which maps from the context to an output sequence $\mathbf{y}$.

Translation with an RNN encoder-decoder

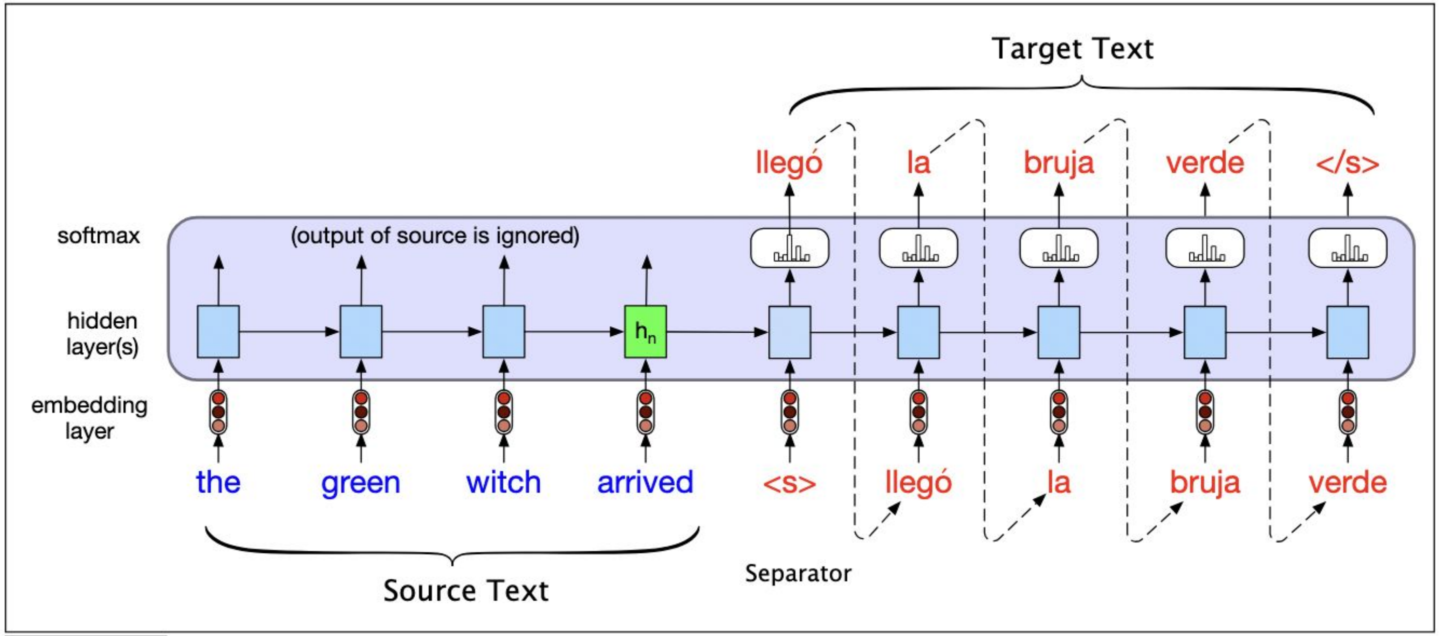


Figure 8.17 Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

Translation with an RNN encoder-decoder

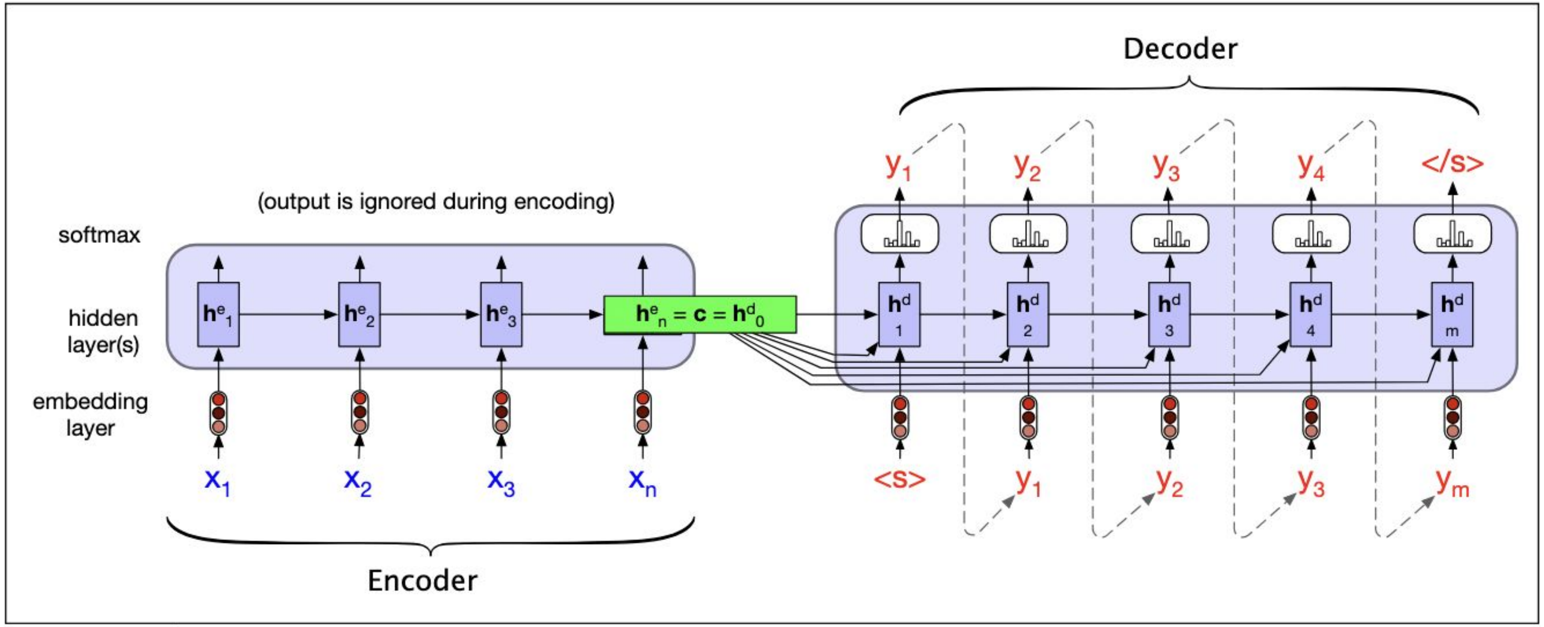


Figure 8.18 A more formal version of translating a sentence at inference time in the basic RNN-based encoder-decoder architecture. The final hidden state of the encoder RNN, h_n^e, serves as the context for the decoder in its role as h_0^d in the decoder RNN, and is also made available to each decoder hidden state.

Training an RNN encoder-decoder

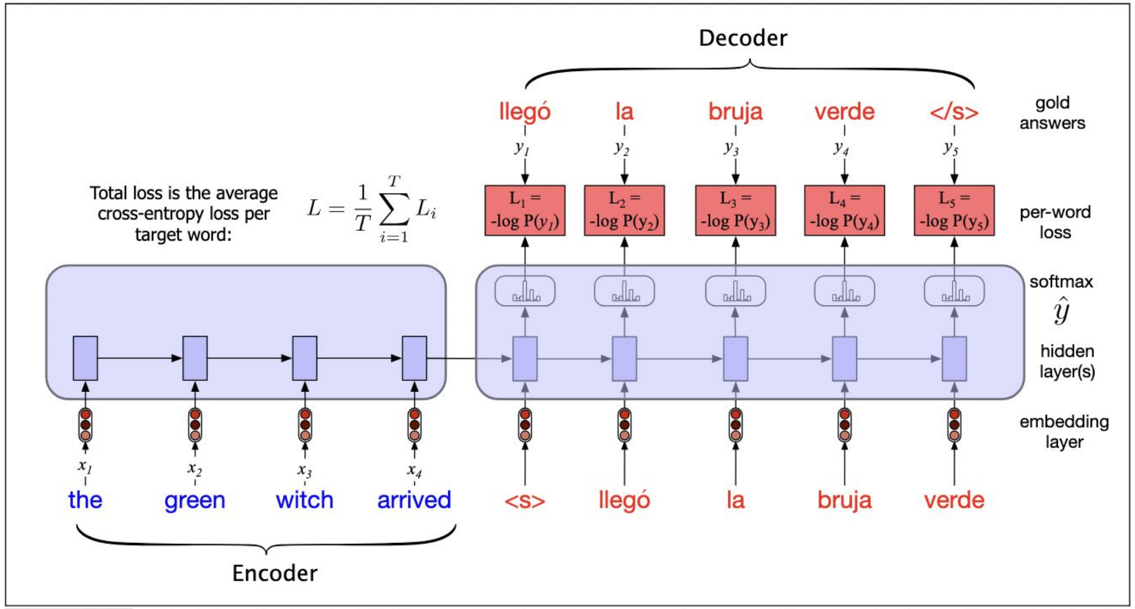


Figure 8.19 Training the basic RNN encoder-decoder approach to machine translation. Note that in the decoder we usually don't propagate the model's softmax outputs $\hat{y}_t$, but use **teacher forcing** to force each input to the correct gold value for training. We compute the softmax output distribution over $\hat{y}$ in the decoder in order to compute the loss at each token, which can then be averaged to compute a loss for the sentence. This loss is then propagated through the decoder parameters and the encoder parameters.