Long-short Term Memory
Encoder-Decoder
Attention

MICHAEL WOLLOWSKI

SUMMARY OF CHAPTER 9: RNNS AND LSTMS

FROM: SPEECH AND LANGUAGE PROCESSING.

BY JURAFSKY AND MARTIN. HTTPS://WEB.STANFORD.EDU/~JURAFSKY/SLP3/

Limitations of RNNs

RNNs are pretty powerful.

However, they have a drawback.

Consider the statement: "The flights the airline was cancelling were full."

What does "was" refer to?
- "airline" i.e. the prior word

What does "were" refer to?
- "flights" i.e. a word much earlier in the sentence

Limitations of RNNs

The recurrent units of an RNN carry state information.

By this we mean that they can “remember” information that may be useful for processing the next or next few pieces of input.

Think about the task of predicting the next word.

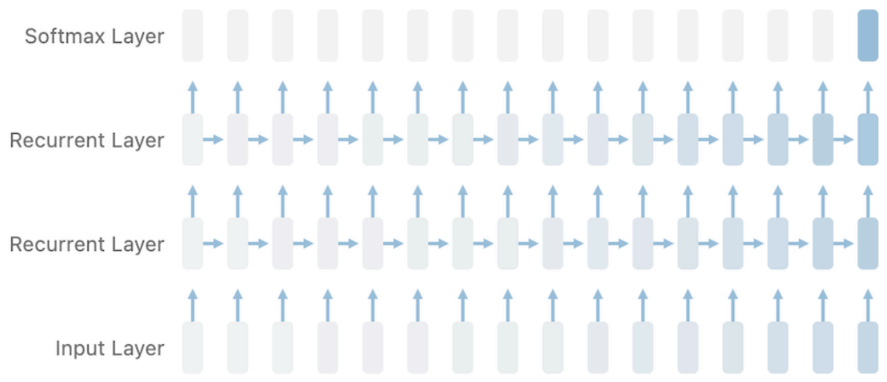
This depends on the prior few words.

The “challenge” is that it has to remember data:

- from the recent past as well as
- potentially from the more distant past.

Limitations of RNNs

Vanishing gradients.



Source: https://distill.pub/2019/memorization-in-rnns/

Long-Short Term Memory (LSTM) Nets

To address the limitations of RNNs, more complex units were developed.

Those units are designed to explicitly manage context

They have two inputs:

- the data pushed through the network and
- context data, maintained by the network.

In Long short-term memory (LSTM) networks, the units are designed to:

- remove information that is no longer needed from the context, and
- adding information to the context that is likely to be relevant for later processing.

Long-Short Term Memory (LSTM) Nets

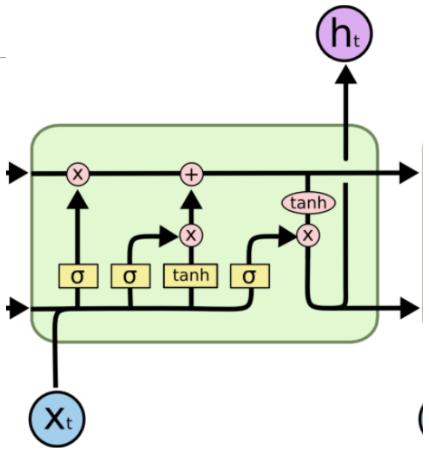
The units use *gates* to control the flow of information into and out of the units.

These gates are implemented through the use of additional weights that operate sequentially on the input, the previous hidden layer and the previous context layers.

LSTM Units in Detail

Let's zoom in and talk about some detail.

Btw. the images are from the fabulous blog entry referenced below.



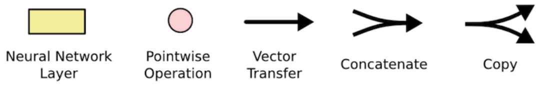


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LONG SHORT-TERM MEMORY NEURAL NETWORKS

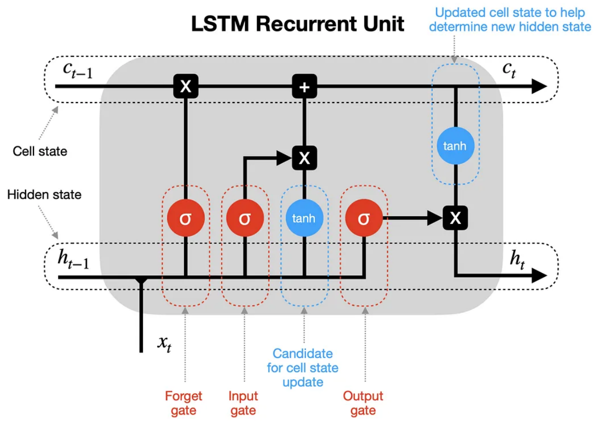


Image source: https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e

Long-Short Term Memory (LSTM) Nets

Below is an LSTM unit shown in time.

Notice the input x_t, output h_t context (upper arrows) and hidden state (lower arrows)

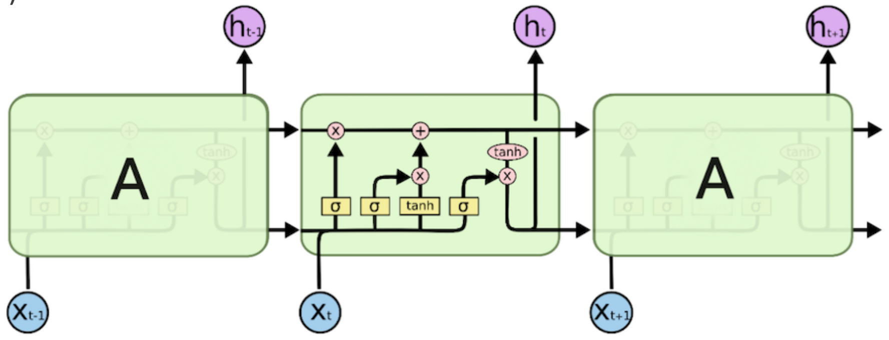


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Units in Detail

Let's have a look at the context data.

Early on, the unit performs multiplication on context vector *C* and soon afterwards the unit performs addition on it.

The first operation is designed to remove data from the context vector.

The second operation is designed to add data to the context vector.

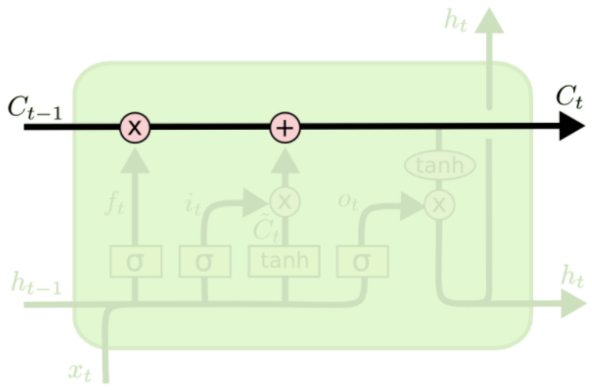


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Units in Detail: The Forget Gate

Let’s have a look at how to “remove” from the context vector.

At first, the unit concatenates the input and hidden state vectors.

The weights into the sigmoid unit will be trained to determine information that is and is not relevant given the current input x and the hidden state.

Vector *f* is multiplied with vector *C*.

f acts as a mask, “zeroing” out information that should be forgotten.

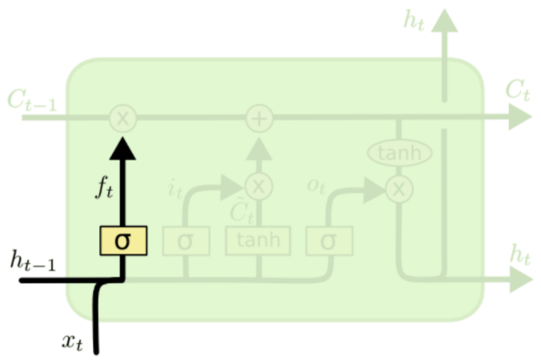
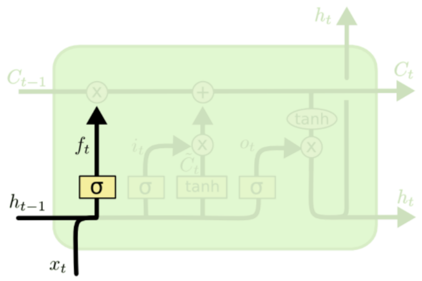


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Units in Detail: The Forget Gate

The weights for the forget gate.



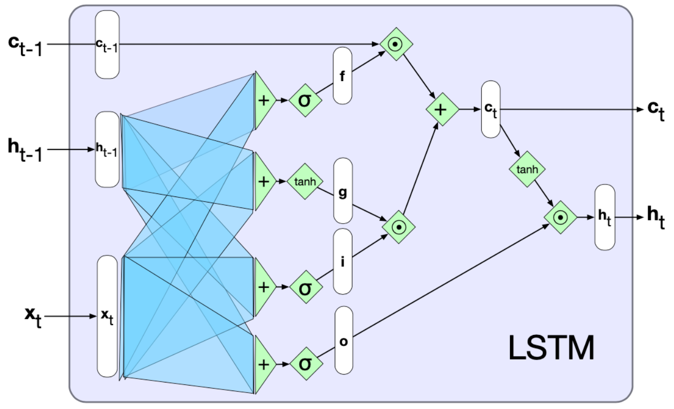


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Units in Detail: Input Gate

Let's have a look at how to "add" to the context vector.

The tanh activation function creates a vector of new candidate values.

An additional sigmoid activation function creates a mask to determine which values to add to the context vector.

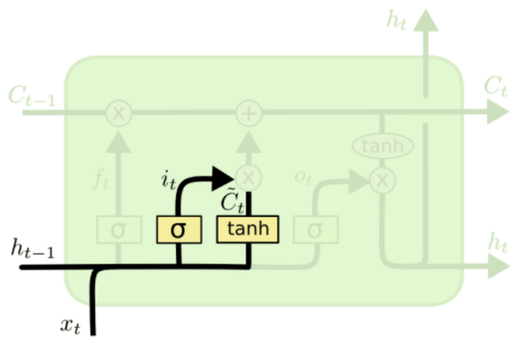


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Units in Detail: Input Gate

Next, perform multiplication on the vector produced by the input gate and the candidate values produced by tanh.

The resulting vector is **added** to the context vector.

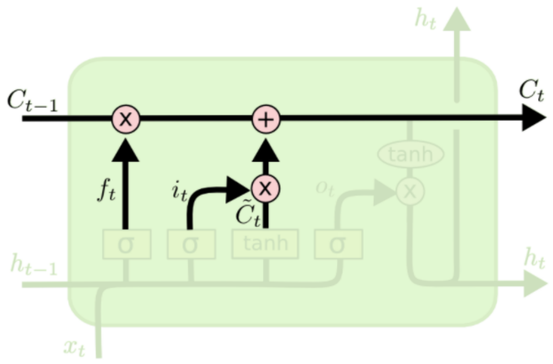
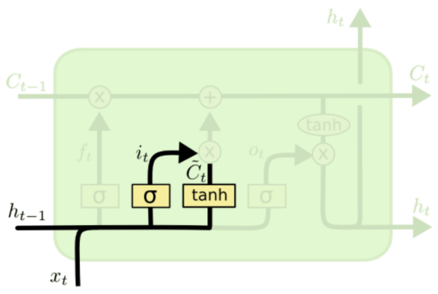


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Units in Detail: Input Gate

The weights for the input gate.



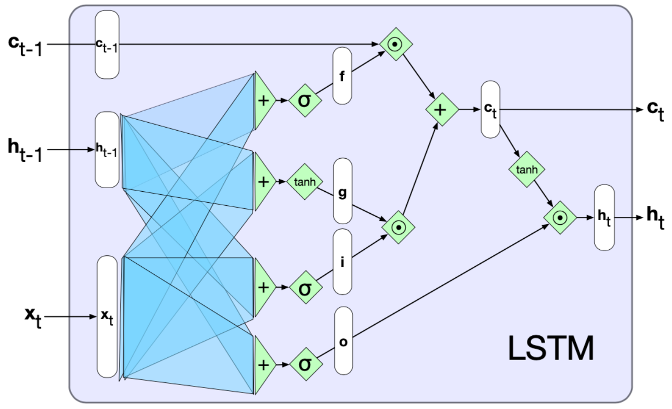


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Units in Detail: Output Gate

We are done with maintaining the context.

Next, let us have a look at calculating the output and updated hidden state.

The output gate is used to decide what data is required for the current hidden state.

Using yet another sigmoid activation function, the unit determines which values are relevant.

Before using the sigmoid function as a mask on the context vector, the unit runs it through tanh.

This is necessary because the addition to the context vector may have produces values outside of the range [-1 .. 1]

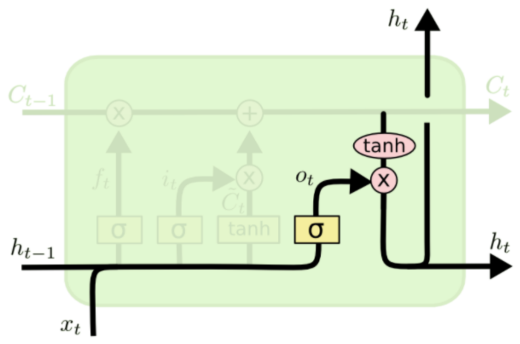


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs – A different perspective

On the next slide, you see an image of an LSTM unit.

It highlights the matrices used for running it.

As you may imagine the more matrices, the more weights that need to be learned, the more computing time it takes to train the network.

Weight matrices in an LSTM Unit

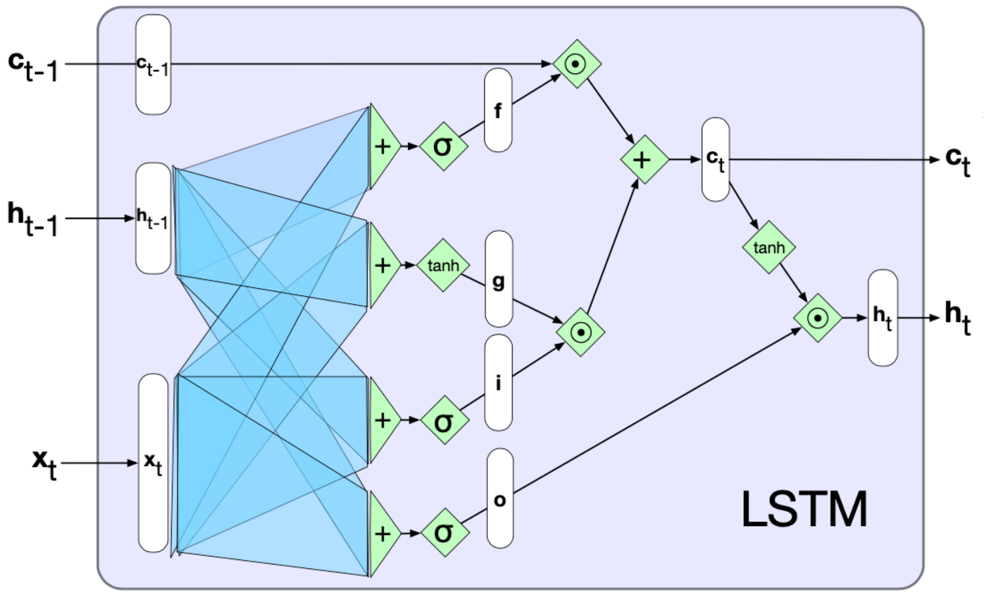


Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of January 21, 2022.

Review of NN units

On the right, you see units and the sort of input they take.

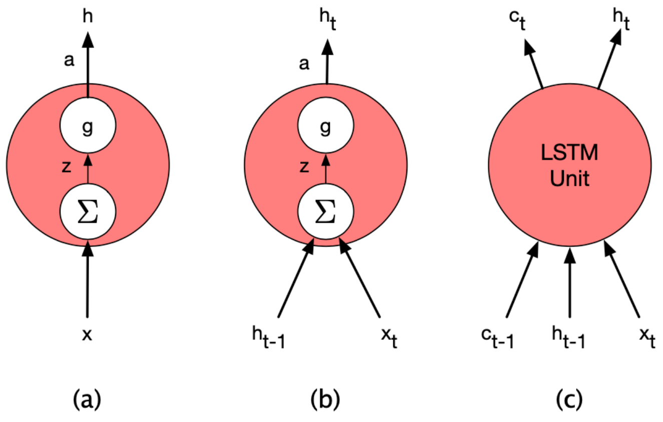


Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of January 12, 2022.

Common RNN NLP Architectures

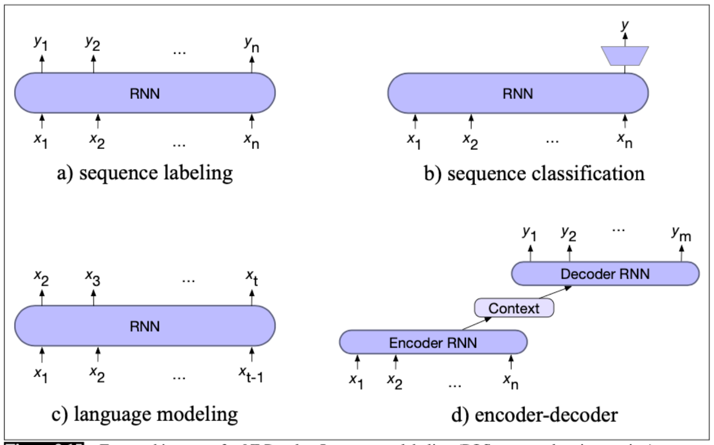


Figure 9.15 Four architectures for NLP tasks. In sequence labeling (POS or named entity tagging) we map each input token x_i to an output token y_i. In sequence classification we map the entire input sequence to a single class. In language modeling we output the next token conditioned on previous tokens. In the encoder model we have two separate RNN models, one of which maps from an input sequence **x** to an intermediate representation we call the **context**, and a second of which maps from the context to an output sequence **y**.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Encoder-Decoder

- Also called sequence-to-sequence network
- Used when an input sequence is to be translated to an output sequence.
- Especially when they are of a different lengths.
- Example: machine translation

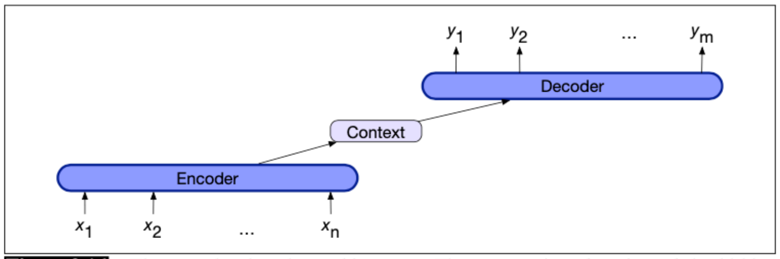


Figure 9.16 The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Encoder-Decoder

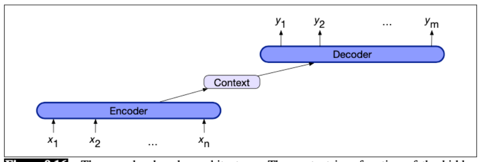


Figure 9.16 The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.

The encoder network takes an input sequence and creates a contextualized representation of it, called the *context*.

Encoder-decoder networks consist of three conceptual components:

1. An encoder that accepts an input sequence, $x_{1:n}$, and generates a corresponding sequence of contextualized representations, $h_{1:n}$.
2. A context vector, c, which is a function of $h_{1:n}$, conveys the essence of the input to the decoder.
3. A decoder, which accepts c as input generates an arbitrary length sequence of hidden states $h_{1:m}$, from which a corresponding sequence of output states $y_{1:m}$, can be obtained.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Translation with a Basic RNN version

<s> is the sentence separator token.

Translate the English source text "the green witch arrived"

to a Spanish sentence

"llego´ la bruja verde"

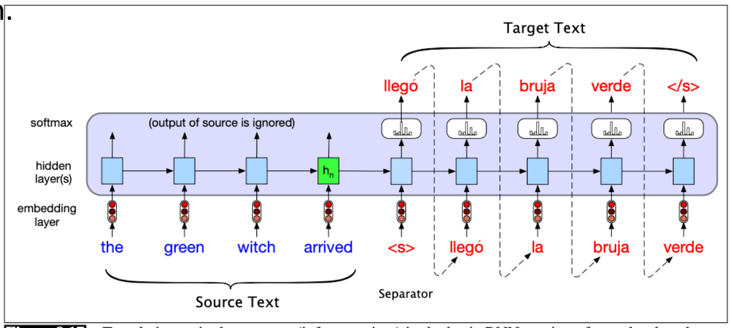


Figure 9.17 Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Translation with a Basic RNN version

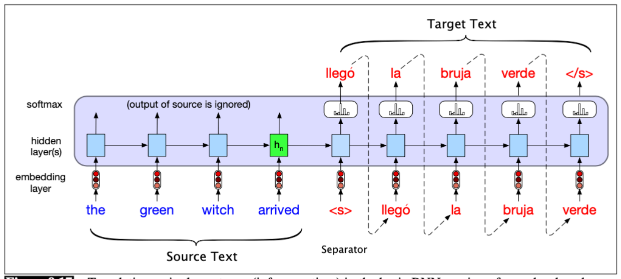


Figure 9.17 Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

To translate a source text, we run it through the RNN to generate hidden states.

In this version, the context is simply h_n

Then begins the generation of the output.

This is done one word at a time.

Initially, i.e. <s> is given the context, i.e. h_n

From then on, the context (or h_n) is passed on.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

A Closer Look at the Basic RNN version

In the figure on the right, the context vector is made available to **all** of the decoders hidden states.
This is done to ensure that the influence of the context vector does not wane as the output sequence is generated.

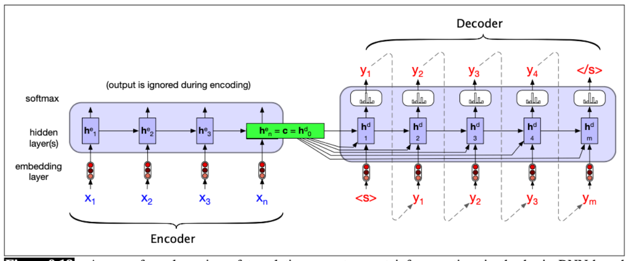


Figure 9.18 A more formal version of translating a sentence at inference time in the basic RNN-based encoder-decoder architecture. The final hidden state of the encoder RNN, h^e_n, serves as the context for the decoder in its role as h^d_0 in the decoder RNN, and is also made available to each decoder hidden state.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Training the Encoder-Decoder Model

Each training example is a tuple of paired strings: a source and a target.

For Machine translation, the training data typically consists of sets of sentences and their translations.

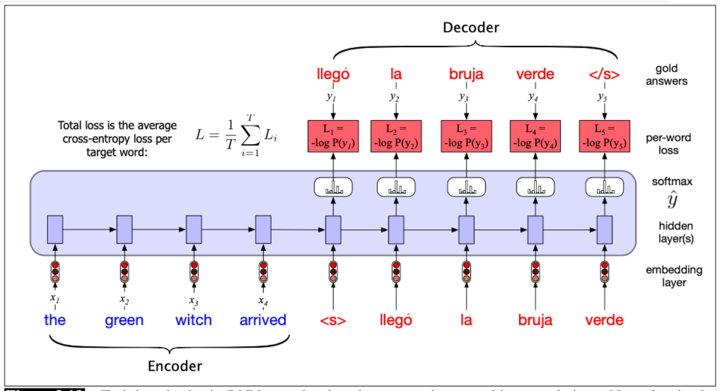


Figure 9.19 Training the basic RNN encoder-decoder approach to machine translation. Note that in the decoder we usually don't propagate the model's softmax outputs $\hat{y}_t$, but use **teacher forcing** to force each input to the correct gold value for training. We compute the softmax output distribution 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.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Training the Encoder-Decoder Model

The source and target are separated by <s> the sentence token.

Starting with the seperator token, the network is trained autoregressively to predict the next word.

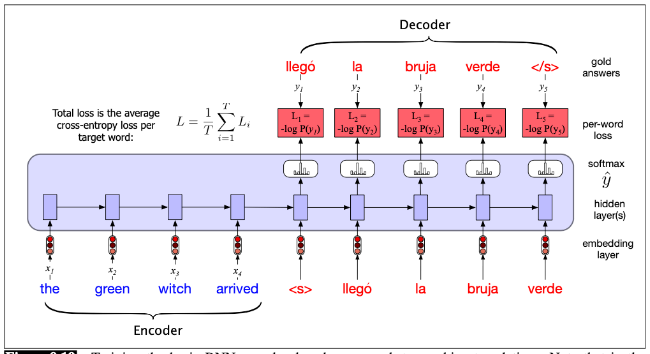


Figure 9.19 Training the basic RNN encoder-decoder approach to machine translation. Note that in the decoder we usually don't propagate the model's softmax outputs $\hat{y}_t$, but use **teacher forcing** to force each input to the correct gold value for training. We compute the softmax output distribution 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.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

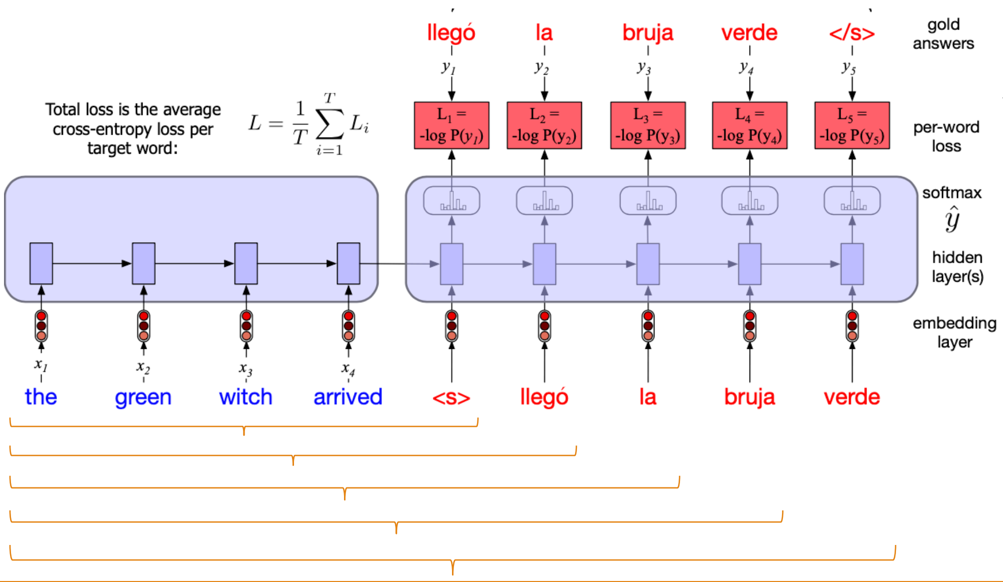


Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Attention

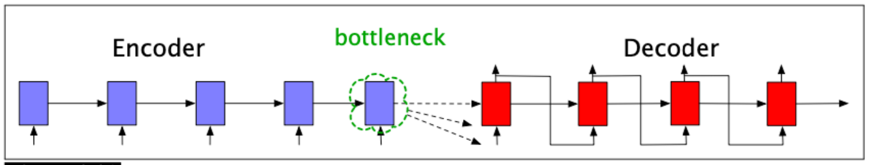


Figure 9.20 Requiring the context *c* to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck.

The encoder-decoder model is appealing because of its clean separation of the encoder and decoder.
The encoder builds a representation of the source text.
The decoder uses this context to generate a target text.
The context vector is h_n, the hidden state of the last (nth) time step of the source text.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Attention

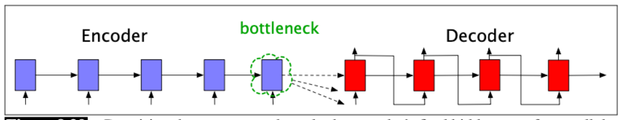


Figure 9.20 Requiring the context *c* to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck.

Challenge: The final hidden state must represent absolutely everything about the meaning of the source text.
This is because it is the only thing the decoder knows about the source text.
It acts somewhat as a bottleneck:
- It has to contain information at the beginning of the sentence,
- As well as from the more recent portion of the sentence.

In other words, how much information can one pack into this vector?

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Attention

The attention mechanism is a solution to the bottleneck problem.

It collects information from *all* the hidden states of the encoder, not just the last hidden state.

In the attention mechanism, the context vector c is a function of the hidden states of the encoder: c=f(h$^e_1$…h^e_n).

We create a single fixed-length vector *c* by taking a weighted sum of all the encoder hidden states.

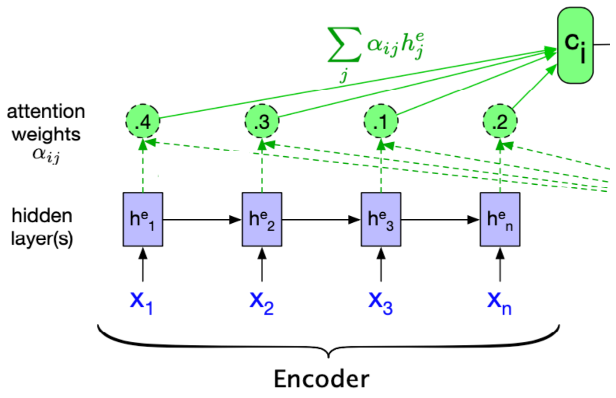
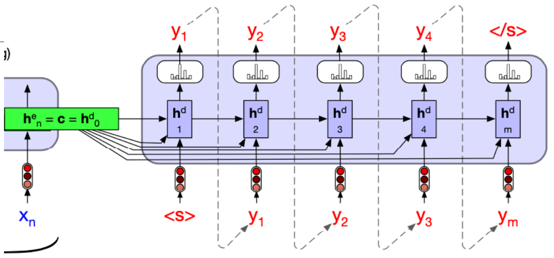


Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Attention



This context vector, c_i, is generated anew with each decoding step.

This context is made available during decoding, along with the prior hidden state and the previous output generated by the decoder.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Attention

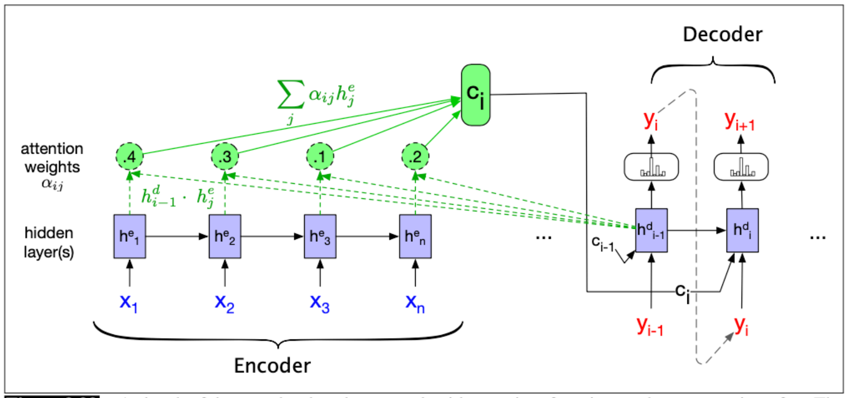


Figure 9.22 A sketch of the encoder-decoder network with attention, focusing on the computation of $\mathbf{c}_i$. The context value $\mathbf{c}_i$ is one of the inputs to the computation of $\mathbf{h}^d_i$. It is computed by taking the weighted sum of all the encoder hidden states, each weighted by their dot product with the prior decoder hidden state $\mathbf{h}^d_{i-1}$.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.