Long-short Term Memory NN 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/

Long-Short Term Memory (LSTM) Nets

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 doe “were” refer to?
- “flights” i.e. a word much earlier in the sentence

Long-Short Term Memory (LSTM) Nets

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.

Long-Short Term Memory (LSTM) Nets

To address this problem, more complex units were developed.

Those units are designed to explicitly manage context

As such, 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 likely to be needed for later decision making.

Long-Short Term Memory (LSTM) Nets

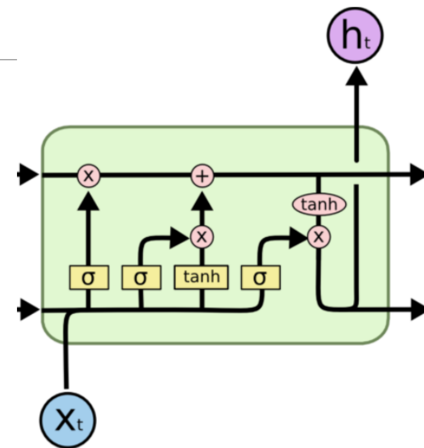
The units use of *gates* to control the flow of information into and out of the units.

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.



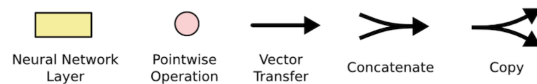


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LONG SHORT-TERM MEMORY NEURAL NETWORKS

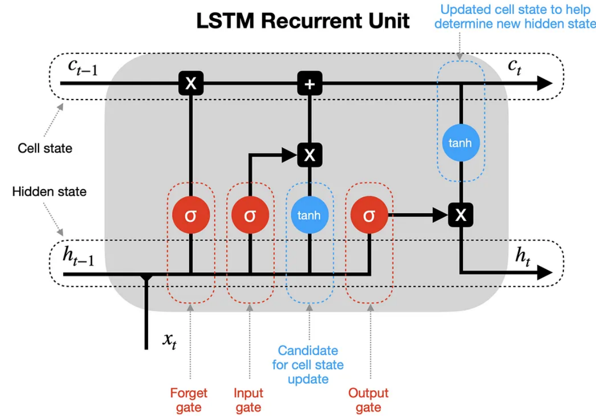


Image source: https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e

Clarification on RNN

The diagram suggests that the hidden layer information is made available to the next unit.

This is incorrect.

The diagram is unfolded in time.

It makes the unrealistic assumption that we only have one input word.

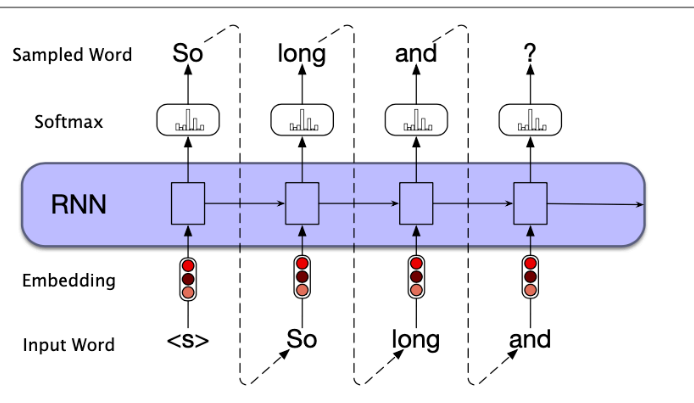


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

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

Clarification on RNNs

Here is an RNN with three input nodes.

Notice that the hidden layers info is not shared.

Recall that the recurrent input to a node is simply the prior output.

Hence a fairly myopic memory.

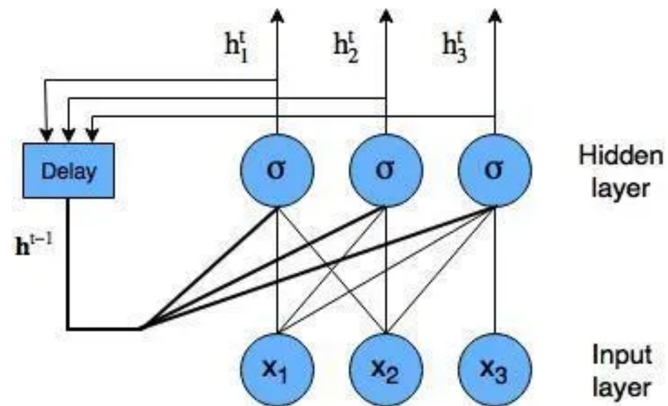


Image source: https://medium.com/@humble_bee/rnn-recurrent-neural-networks-lstm-842ba7205bbf

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)

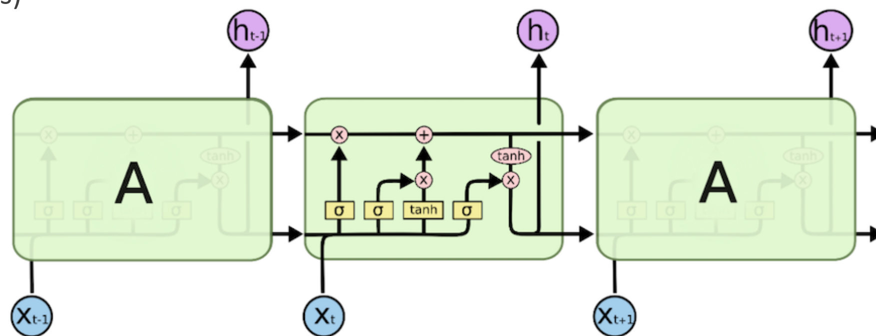


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

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.

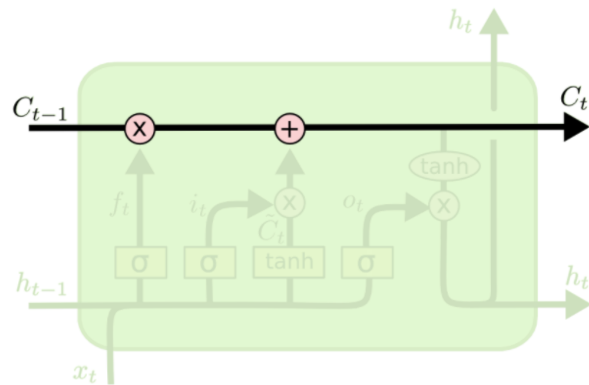


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Units in Detail

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

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

Using the weights and resulting vector, the unit calculates the weighted sum of its inputs and runs it through a sigmoid function to produce a vector *f*.

This is called the *forget gate*.

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

As such *f* acts as a mask, zeroing out information that should be forgotten and keeping information that should be kept.

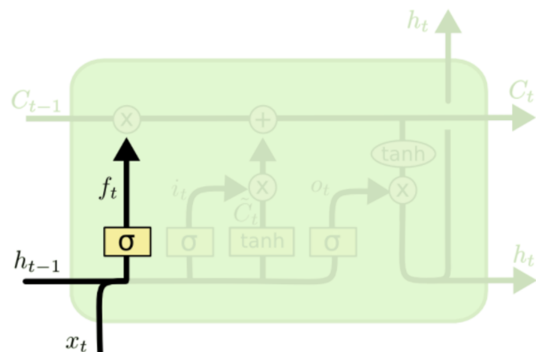


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Units in Detail

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

Using a separate sigmoid activation function, the unit determines which values to update.

This is called the *input gate*.

Next, a tanh activation function creates a vector of new candidate values.

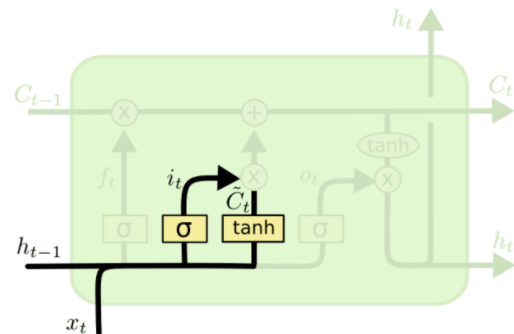


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Units in Detail

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

As with the forget gate, the output vector produced by the sigmoid function acts as a mask.

The resulting vector is added to the context vector.

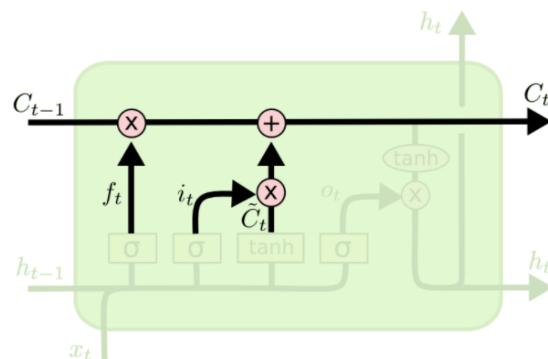


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Units in Detail

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]

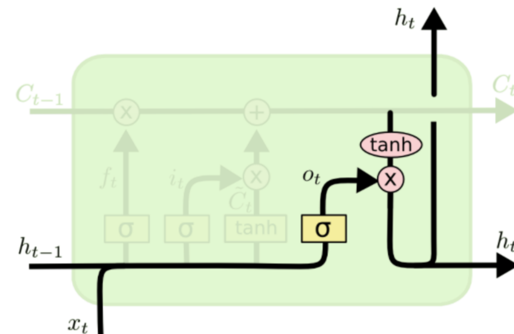


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

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.

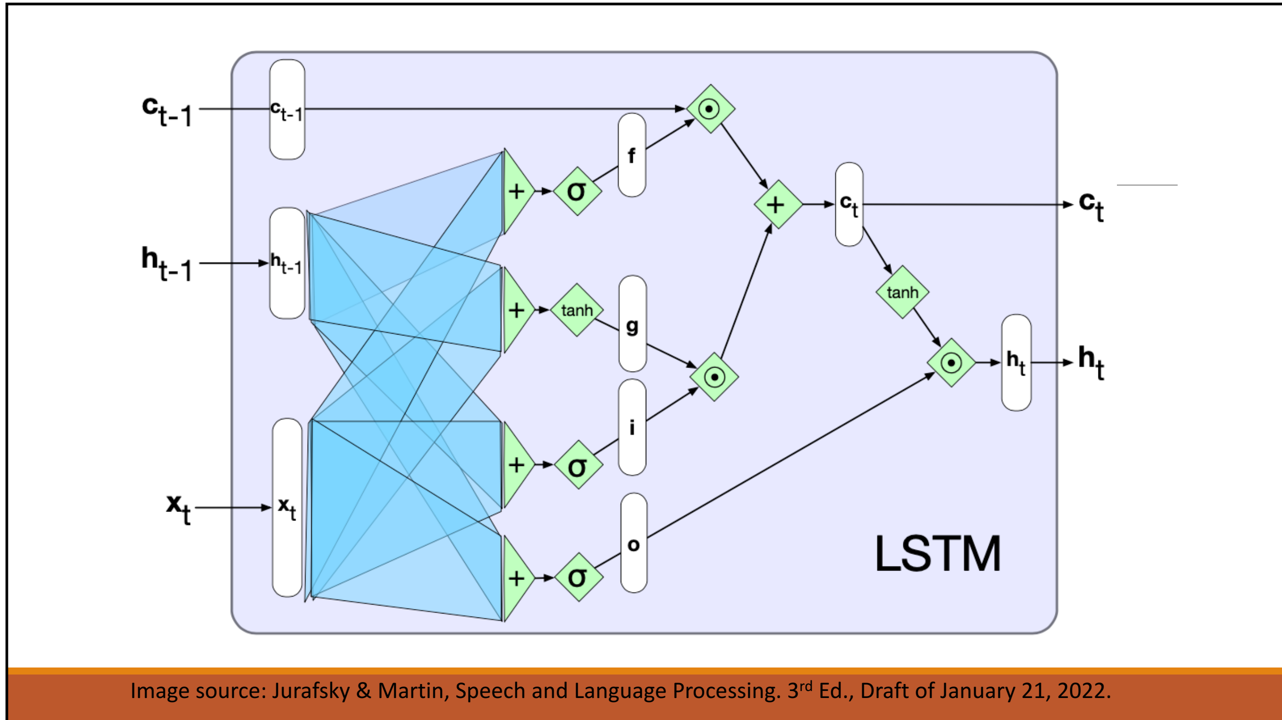


Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of January 21, 2022.

Review of NN units

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

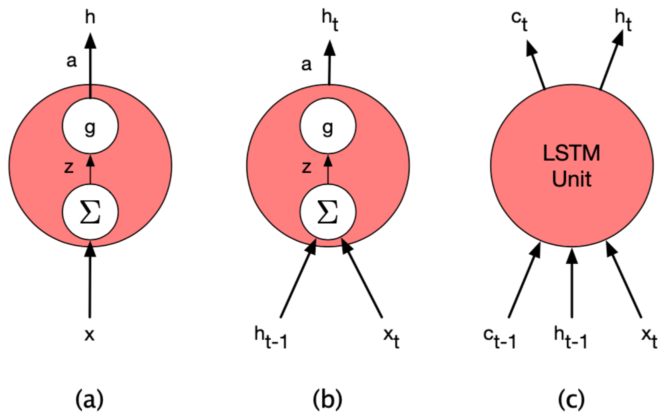


Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of January 12, 2022.

Common RNN NLP Architectures

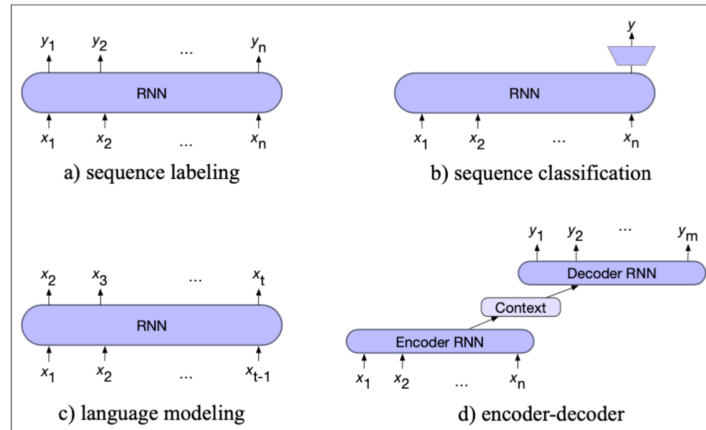


Figure 9.15 Four architectures for NLP tasks. In sequence labeling (POS or named entity tagging) we map each input token x_i to an output token y_i. In sequence classification we map the entire input sequence to a single class. In language modeling we output the next token conditioned on previous tokens. 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 that is of a different length than the input, and does not align with it in a word-to-word way.

Example: machine translation, where the input sequence and output sequence can have different lengths

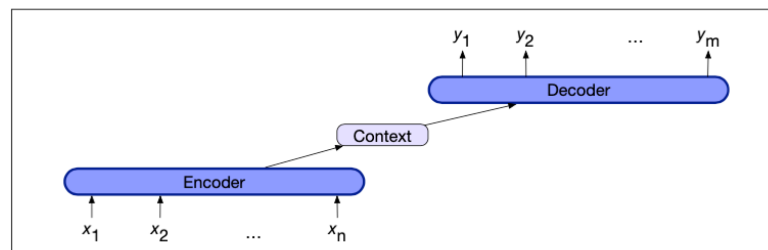


Figure 9.16 The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.

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

Encoder-Decoder

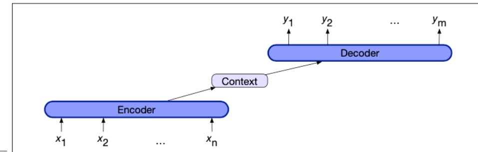


Figure 9.16 The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.

Encoder network that 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 $h_{1:n}$, and conveys the essence of the input to the decoder.
3. A decoder, which accepts c as input and 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.

LSTMs, convolutional networks, and transformers can all be employed as encoders or decoders.

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

Translation with a Basic RNN version

We use <s> for the sentence separator token.

We wish to translate the English source text "the green witch arrived"

to a Spanish sentence "*llegó la bruja verde*"

The latter can be glossed word-by-word as 'arrived the witch green'.

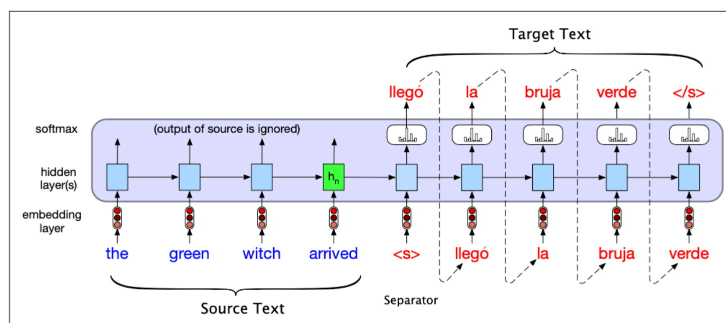


Figure 9.17 Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

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

Translation with a Basic RNN version

To translate a source text, we run it through the network performing forward inference to generate hidden states until we get to the end of the source.

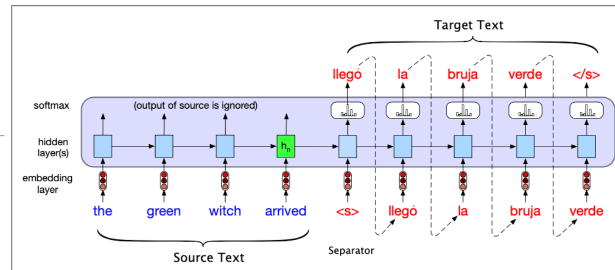


Figure 9.17 Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

Then we begin autoregressive generation, asking for a word in the context of the hidden layer from the end of the source input as well as the end-of-sentence marker.

Subsequent words are conditioned on the previous hidden state and the embedding for the last word generated.

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

A Closer Look at the Basic RNN version

The prior figure shows only a single network layer for the encoder.
Stacked architectures are the norm.
The output states from the top layer of the stack are taken as the final representation.

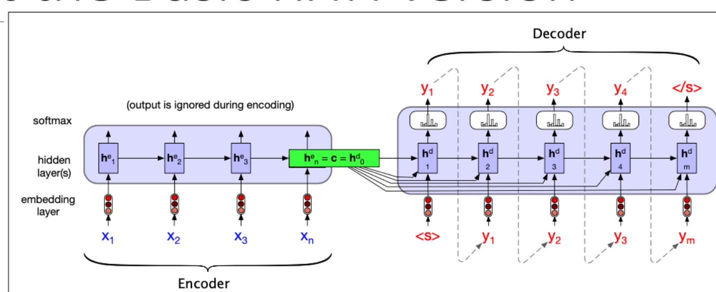


Figure 9.18 A more formal version of translating a sentence at inference time in the basic RNN-based encoder-decoder architecture. The final hidden state of the encoder RNN, h_n^e, serves as the context for the decoder in its role as h_0^d in the decoder RNN, and is also made available to each decoder hidden state.

The encoder consists of stacked biLSTMs where the hidden states from top layers from the forward and backward passes are concatenated to provide the contextualized representations for each time step.

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

A Closer Look at the Basic RNN version

The purpose of the encoder is to generate a contextualized representation of the input.
This representation is embodied in the final hidden state of the encoder, h^e_n.
This representation, the context, is then passed to the decoder.

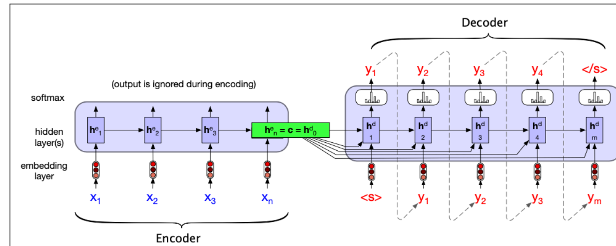


Figure 9.18 A more formal version of translating a sentence at inference time in the basic RNN-based encoder-decoder architecture. The final hidden state of the encoder RNN, h^e_n, serves as the context for the decoder in its role as h^d_0 in the decoder RNN, and is also made available to each decoder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

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

A Closer Look at the Basic RNN version

The simplest version of the decoder network takes this state and uses it to initialize just the first hidden state of the decoder
The first decoder RNN cell would use *c* as its prior hidden state h^d_0.

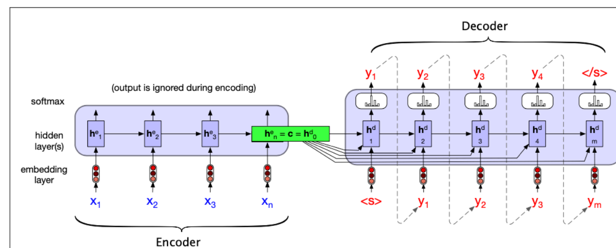


Figure 9.18 A more formal version of translating a sentence at inference time in the basic RNN-based encoder-decoder architecture. The final hidden state of the encoder RNN, h^e_n, serves as the context for the decoder in its role as h^d_0 in the decoder RNN, and is also made available to each decoder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

The decoder would then autoregressively generates a sequence of outputs, an element at a time, until an end-of-sequence marker is generated.

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.

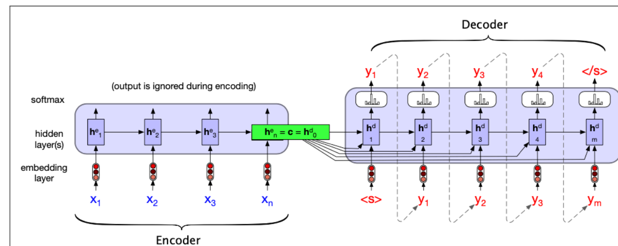


Figure 9.18 A more formal version of translating a sentence at inference time in the basic RNN-based encoder-decoder architecture. The final hidden state of the encoder RNN, h_n^e, serves as the context for the decoder in its role as h_0^d in the decoder RNN, and is also made available to each decoder hidden state.

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

Training the Encoder-Decoder Model

Encoder-decoder architectures are trained end-to-end.
Each training example is a tuple of paired strings, a source and a target.
They are concatenated with a separator token.
For MT, the training data typically consists of sets of sentences and their translations.

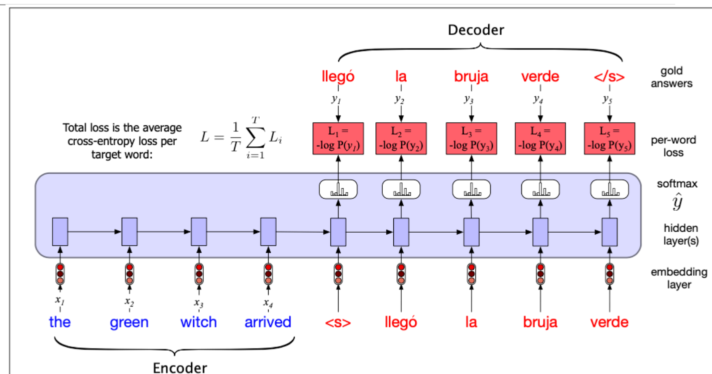


Figure 9.19 Training the basic RNN encoder-decoder approach to machine translation. Note that in the decoder we usually don't propagate the model's softmax outputs $\hat{y}_t$, but use **teacher forcing** to force each input to the correct gold value for training. We compute the softmax output distribution 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 network is given the source text
Starting with the separator token, it is trained autoregressively to predict the next word.

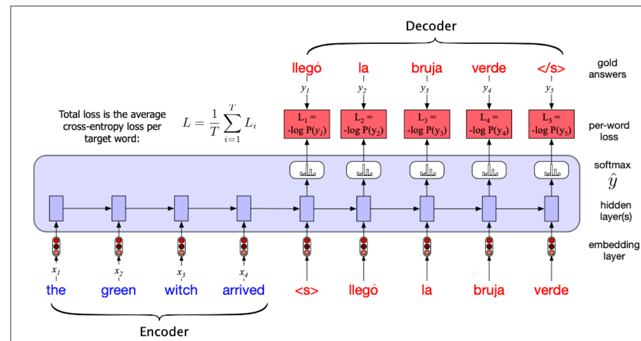


Figure 9.19 Training the basic RNN encoder-decoder approach to machine translation. Note that in the decoder we usually don't propagate the model's softmax outputs $\hat{y}_t$, but use **teacher forcing** to force each input to the correct gold value for training. We compute the softmax output distribution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

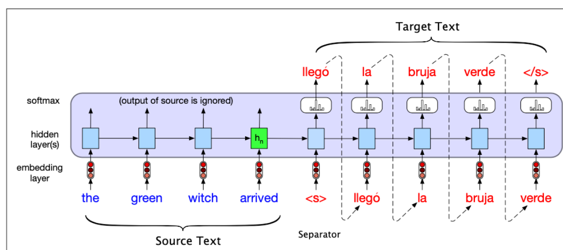


Figure 9.17 Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

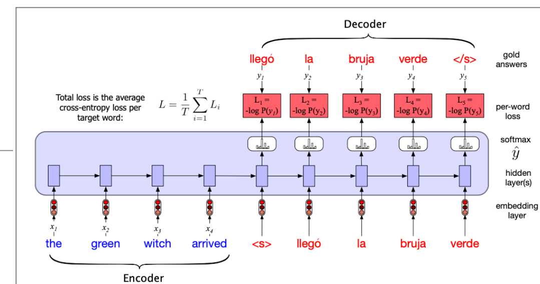


Figure 9.19 Training the basic RNN encoder-decoder approach to machine translation. Note that in the decoder we usually don't propagate the model's softmax outputs $\hat{y}_t$, but use **teacher forcing** to force each input to the correct gold value for training. We compute the softmax output distribution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

Note the differences between training and inference with respect to the outputs at each time step.
The decoder during **inference** uses its own estimated output *yˆ* as the input for the next time step *x+1* .
Thus the decoder will tend to deviate more and more from the gold target sentence as it keeps generating more tokens.
In **training**, therefore, it is more common to use teacher forcing in the decoder.
Teacher forcing means that we force the system to use the gold target token from training as the next input *xt*+1, rather than allowing it to rely on the (possibly erroneous) decoder output *yˆ* .
This speeds up training.

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

Attention

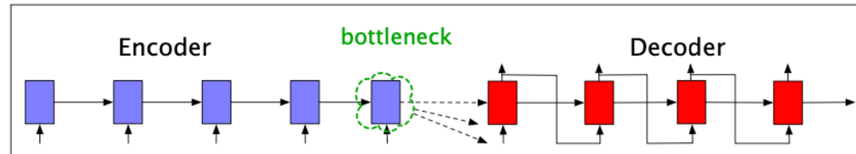


Figure 9.20 Requiring the context c to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck.

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

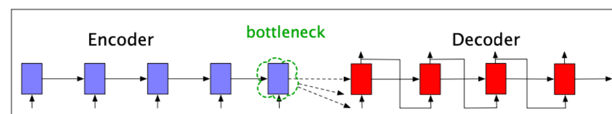


Figure 9.20 Requiring the context c to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck.

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.
Similar to RNNs, 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 single vector that is a function of the hidden states of the encoder: $c=f(h^e_1 \ldots h^e_n)$.

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

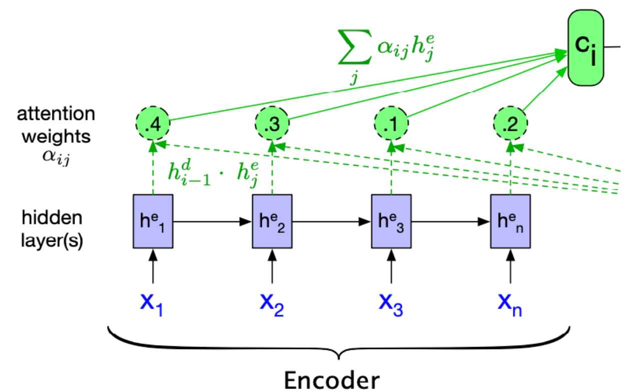
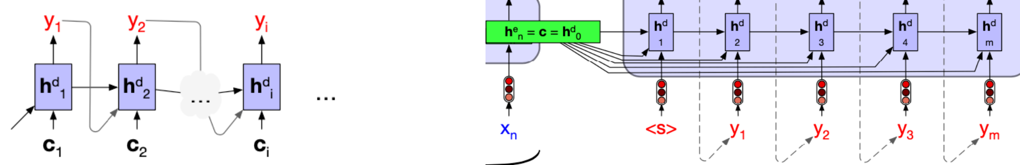


Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Attention



This context vector, c_i, is generated anew with each decoding step *i* and takes all of the encoder hidden states into account.

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

This is represented in the figure on the left side. On the right side, you see the single, last context vector approach.

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

Attention

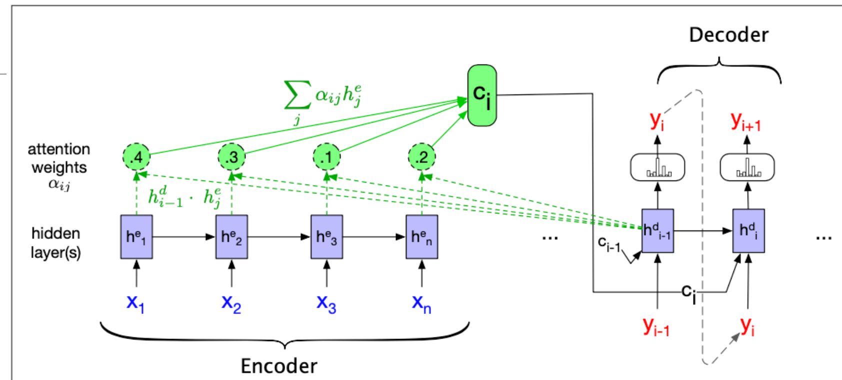


Figure 9.22 A sketch of the encoder-decoder network with attention, focusing on the computation of $\mathbf{c}_i$. The context value $\mathbf{c}_i$ is one of the inputs to the computation of $\mathbf{h}^d_i$. It is computed by taking the weighted sum of all the encoder hidden states, each weighted by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

Here is a diagram of the attention mechanism.

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