Natural Language Processing

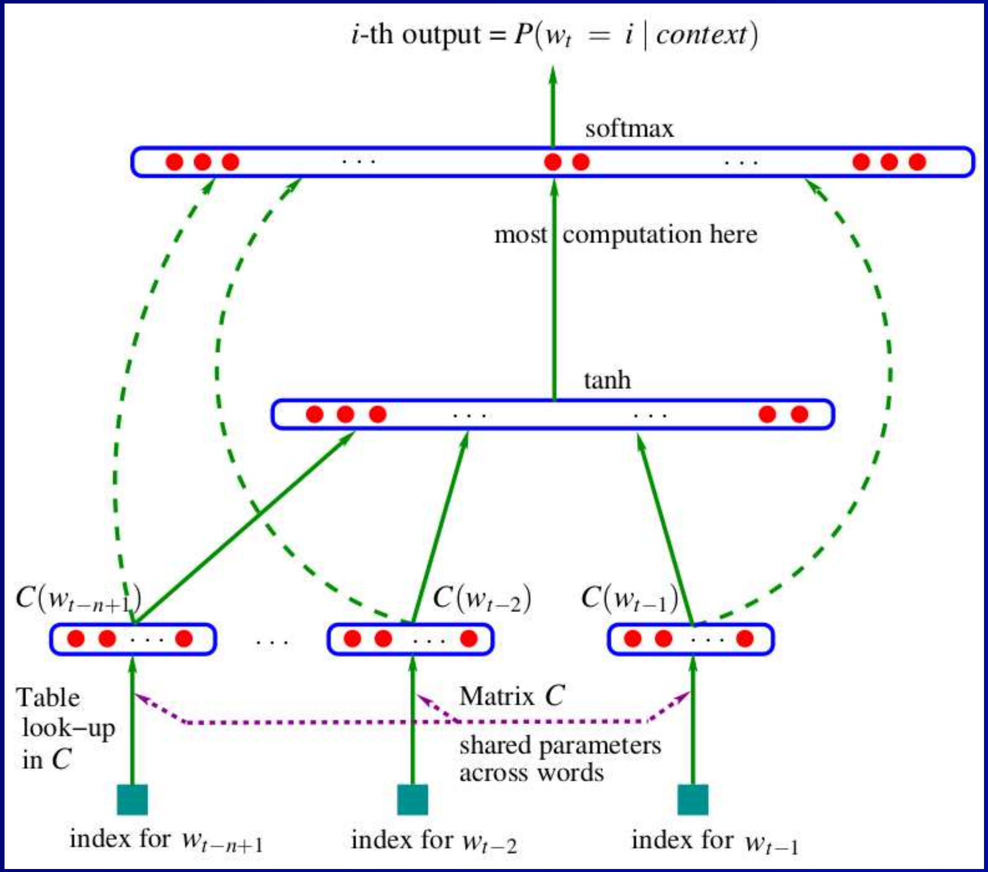
Lecture 9
RNNs and language modeling

2021

Introduction

Language modeling with NNs

As we have seen, feedforward NN language models with word embeddings already performed better than traditional n-gram models, e.g., Bengio et al. [2003]:



Language modeling with NNs cont.

Bengio et al. [2003] reports 24% improvement in perplexity compared to the best n-gram model.

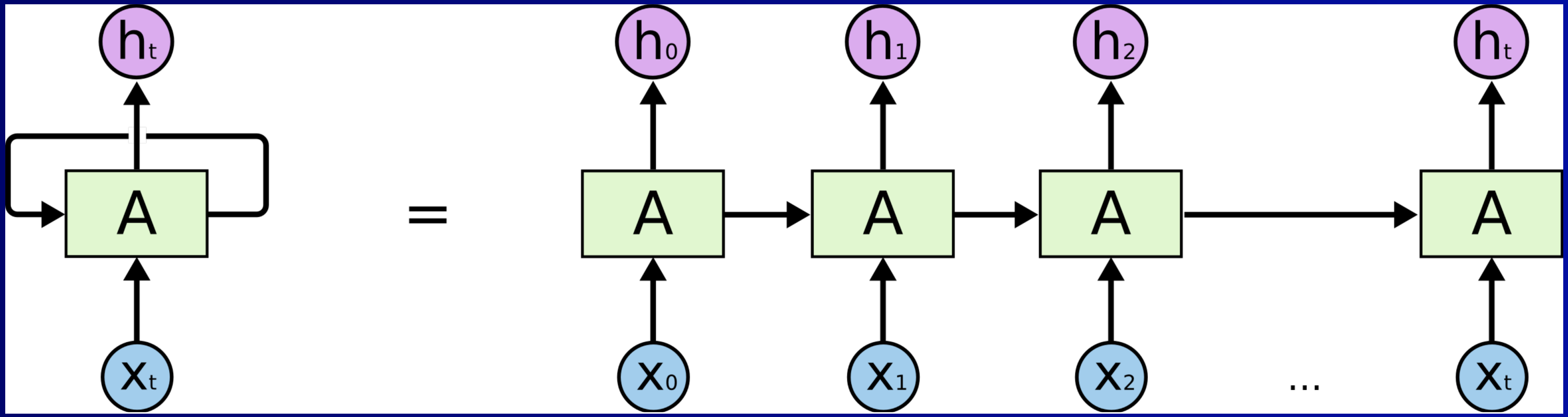
But these models still share an important limitation of n-gram models: the continuation predictions are based on a *fixed size context window*, without any information on earlier history:

$$\hat{P}(w_t \mid w_0, \ldots, w_{t-1}) = \phi(w_{t-k}, \ldots, w_{t-1}),$$

where $\phi(\cdot)$ is a function computed by a feedforward neural network.

Recurrent Neural Networks (RNNs)

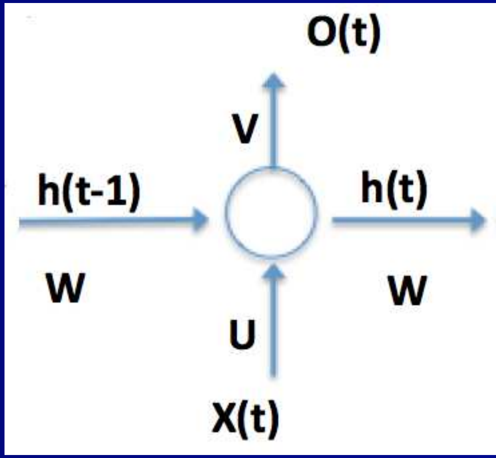
Recurrent neural networks, in contrast, are not limited to fixed-length input sequences, and can form, at least in theory, useful internal representations of *arbitrary long* histories. They can process sequential input step-by-step and keep an internal state which can be updated at each step:



If not otherwise indicated, figures in this and the next section are from Olah [2015].

Recurrent Neural Networks cont.

RNNs can have rather simple internal structure, e.g., the once widely used Elman network[1] has a structure



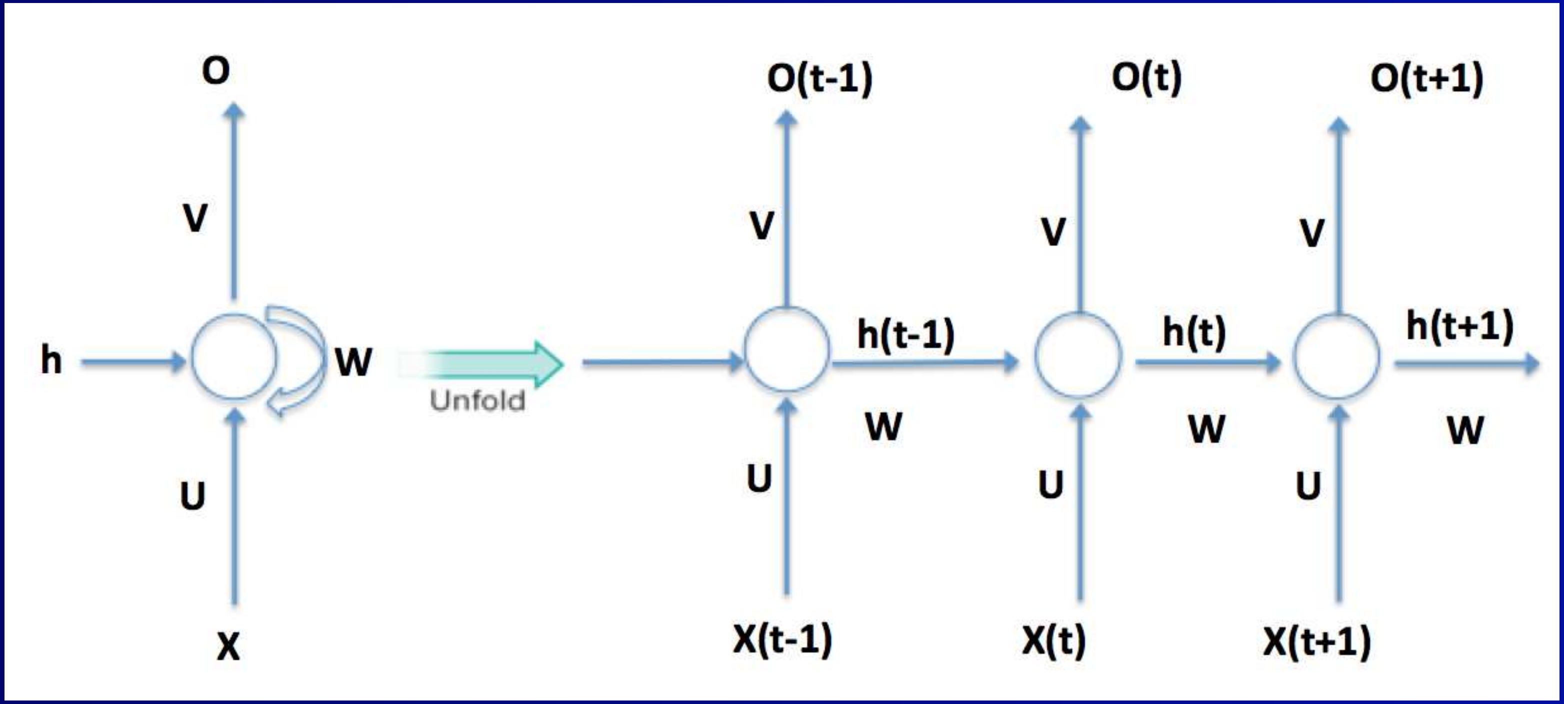
$$h_t = a_h(Ux_t + Wh_{t-1} + b_h),$$

$$o_t = a_o(Vh_t + b_o).$$

[1] Elman [1990].

Backpropagation through time

The standard optimization method for RNNs is *backpropagation through time* (BPTT), which is backpropagation applied to the time-unfolded network:



Backpropagation through time cont.

Since the depth of an unrolled RNN grows linearly with the number of time steps through which it is unrolled, it is often unfeasible to do backpropagation through all time steps until the first one.

In these cases, unrolling and backpropagation of error is only done for a certain number of time steps – *backpropagation is truncated*. In practice, most neural network frameworks implement truncated backpropagation.

RNN training challenges

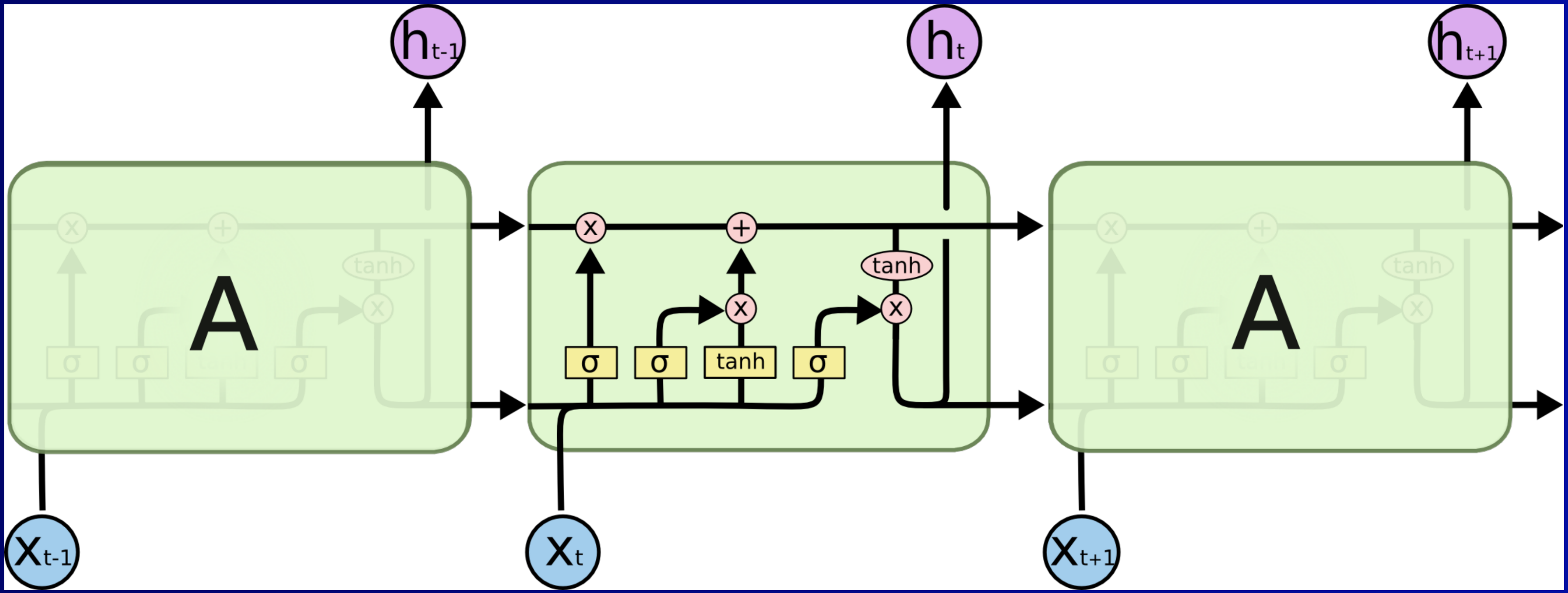
Training RNNs poses significant challenges:

- An RNN unrolled through several timesteps is behaving like a deep feedforward network with respect to backpropagation, so both *vanishing* and *exploding gradients* can be a problem, exacerbated by the fact that the exact same layers are repeated.
- Vanishing gradients, in particular, mean that the RNN does not learn *long-term dependencies*, which, in theory, should be its strength.

Long Short-Term Memory Networks

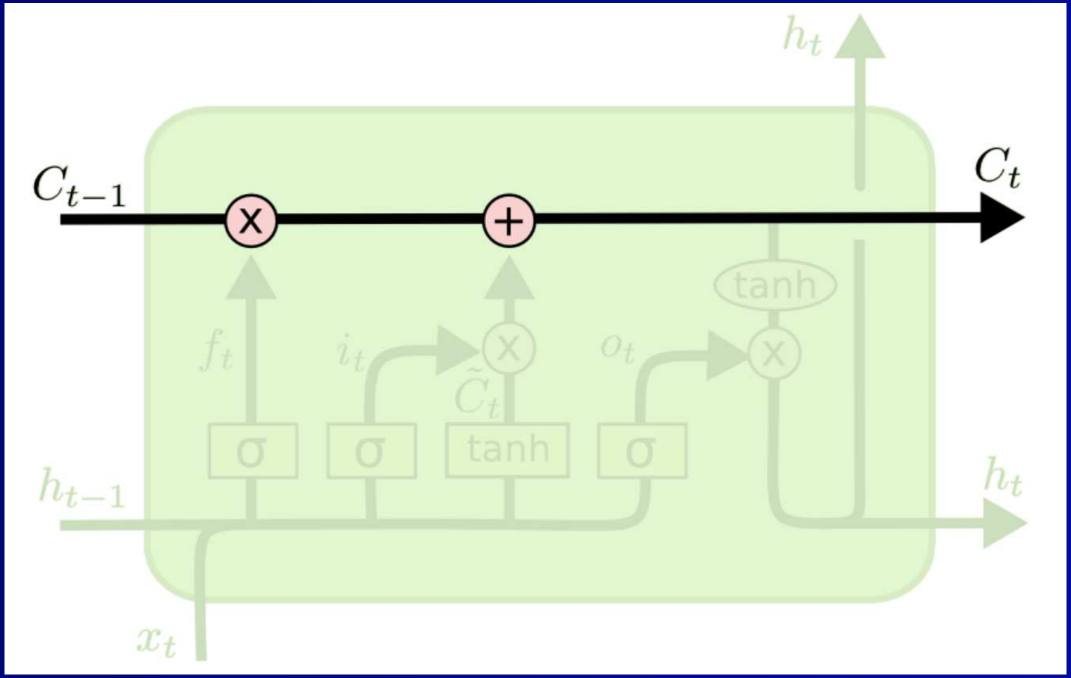
Long Short-Term Memory (LSTM)

Hochreiter and Schmidhuber [1997] introduced an elaborate gated topology to endow RNNs with long-term memory and solve the vanishing/exploding gradients problem.



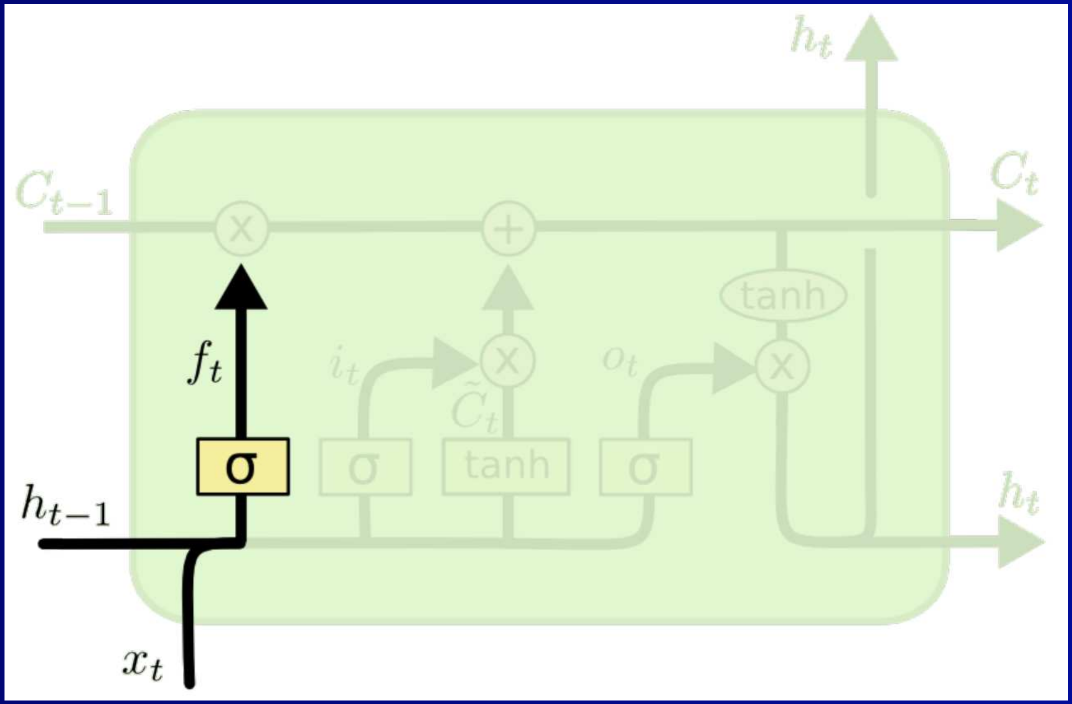
Cell state

The LSTM's cell state acts as an "information conveyor belt", on which information can travel across time steps.



Forget gate

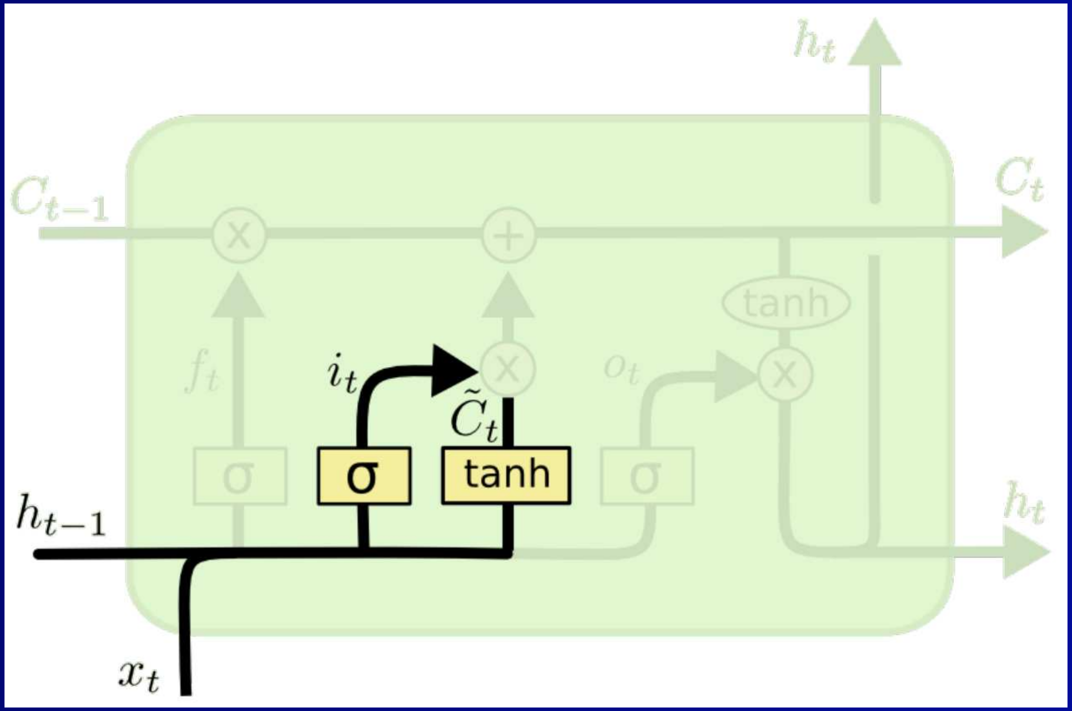
The forget gate calculates an $f_t \in (0,1)^d$ mask for removing information from the cell state:



$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f).$$

Input gate and update vector

An i_t input mask and a $\tilde{C}_t$ update vector is calculated:

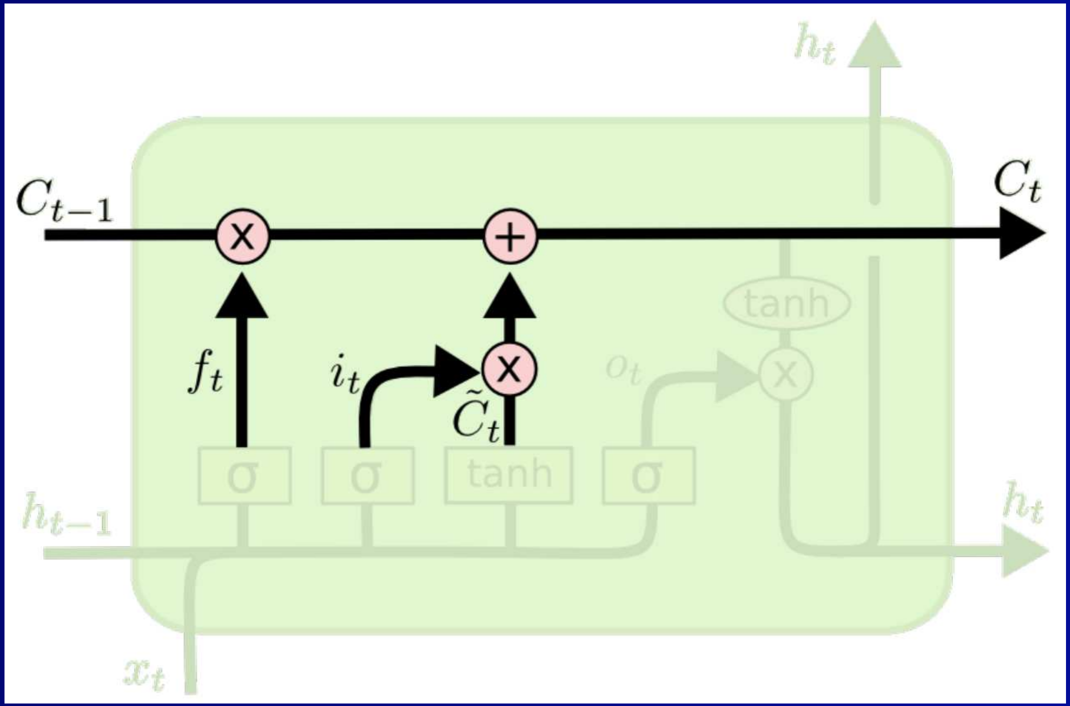


$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i),$$

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C).$$

Computing the new cell state

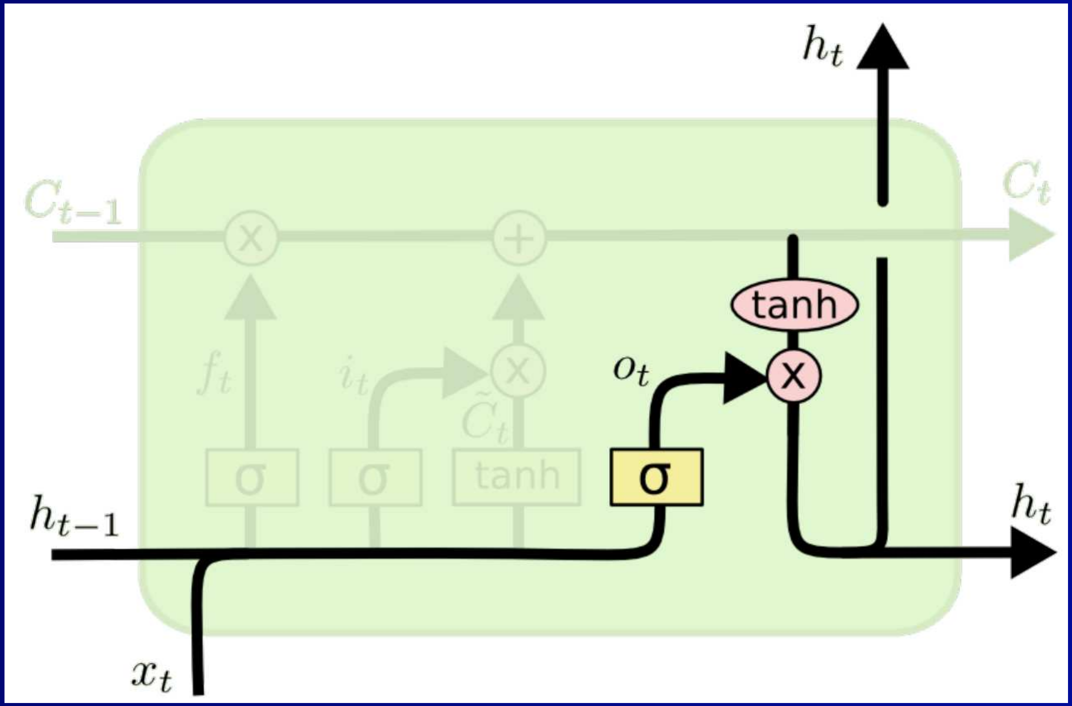
The new cell state can be computed using f_t, i_t and $\tilde{C}_t$:



$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t.$$

Output

Finally, an output, h_t is generated:



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

$$h_t = o_t \odot \tanh(C_t).$$

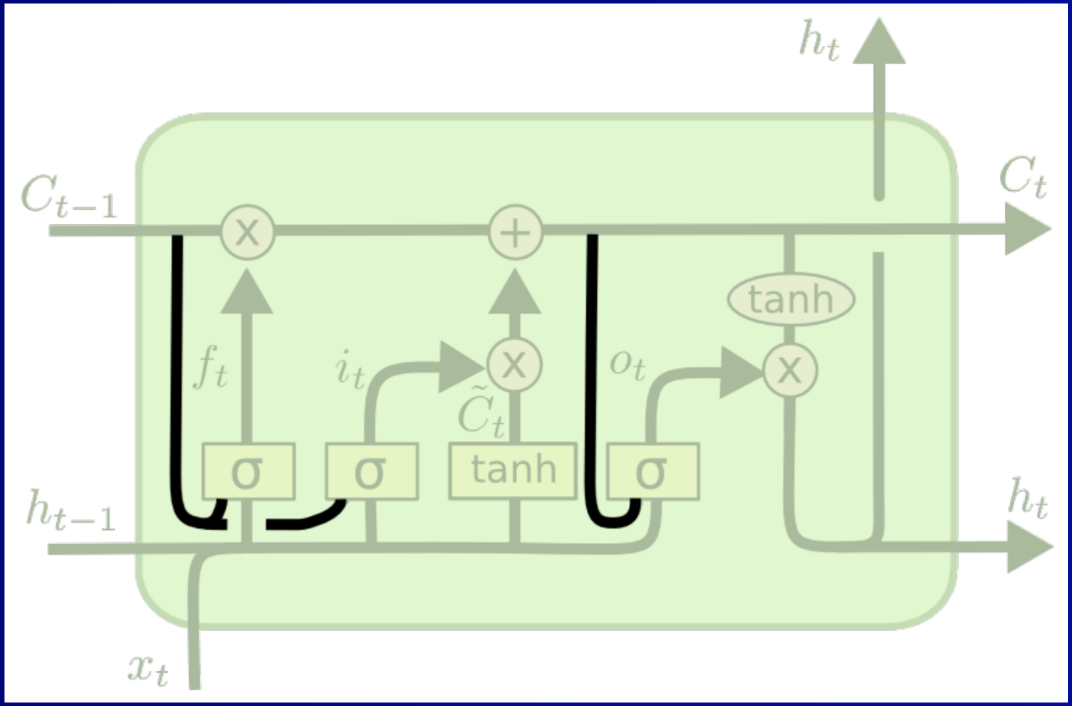
LSTM advantages

The gated LSTM architecture solves the problem of vanishing/exploding gradients by ensuring that the gradient can flow to distant time steps.

The fact that the updates are *additive* means that gradients are not multiplied as in the Elman network's case, and the gates can acquire weights during training that allow the network to exhibit long-range dependencies between input and output values.

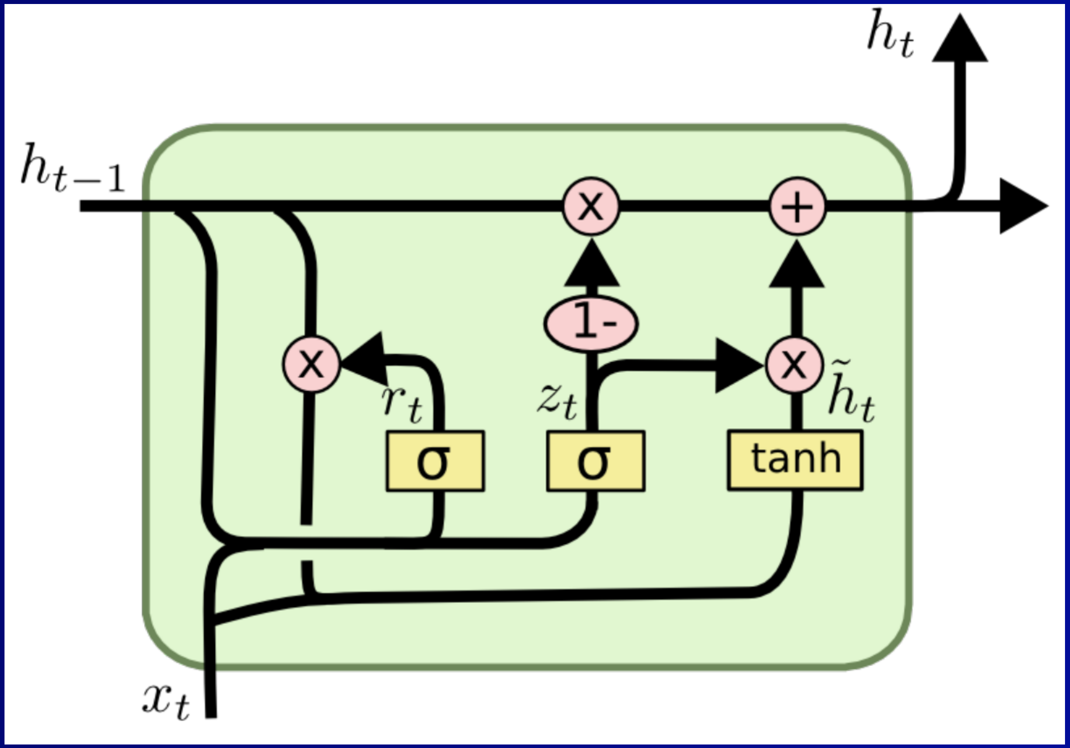
LSTM variants: Peephole connections

Peephole connections extend the LSTM architecture by giving access to the actual cell state to the gates:



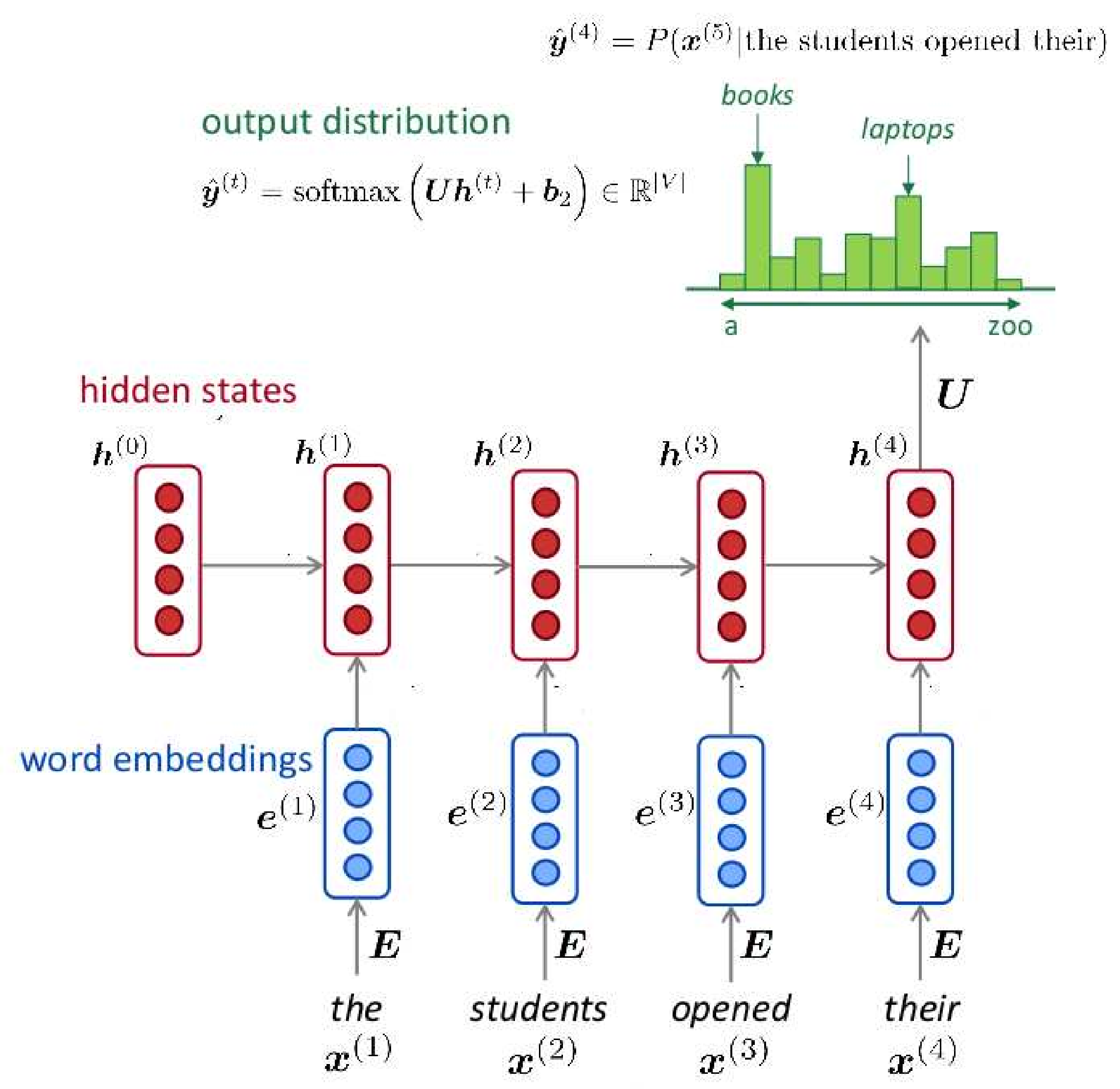
LSTM variants: Gated Recurrent Unit (GRU)

GRU is a simplified LSTM variant, it gets rid of the separate cell state and merges the forget and input gates:



Language modeling with RNNs

Language modeling with RNNs



Language modeling with RNNs cont.

The most notable features of the model are

- previous words ("left context") are processed step-by-step, one word at a time step;
- the first layer is a static word embedding;
- the h_t RNN direct output (hidden state) gets transformed to a continuation probability distribution over the vocabulary by an affine transformation and the $\mathrm{softmax}$ nonlinearity.

Sequence elements

Although traditionally RNN language models were word based, i.e., the sequence elements were words, there are two important alternatives:

- *character-level* language models treat characters as the sequence elements, and predict the next character based on the previous ones.
- *subword-level* language models are based on subword tokenization (e.g., BPE) and predict the next *subword* in the vocabulary.

Both types of model can utilize corresponding – character- and subword- – embeddings.

Training

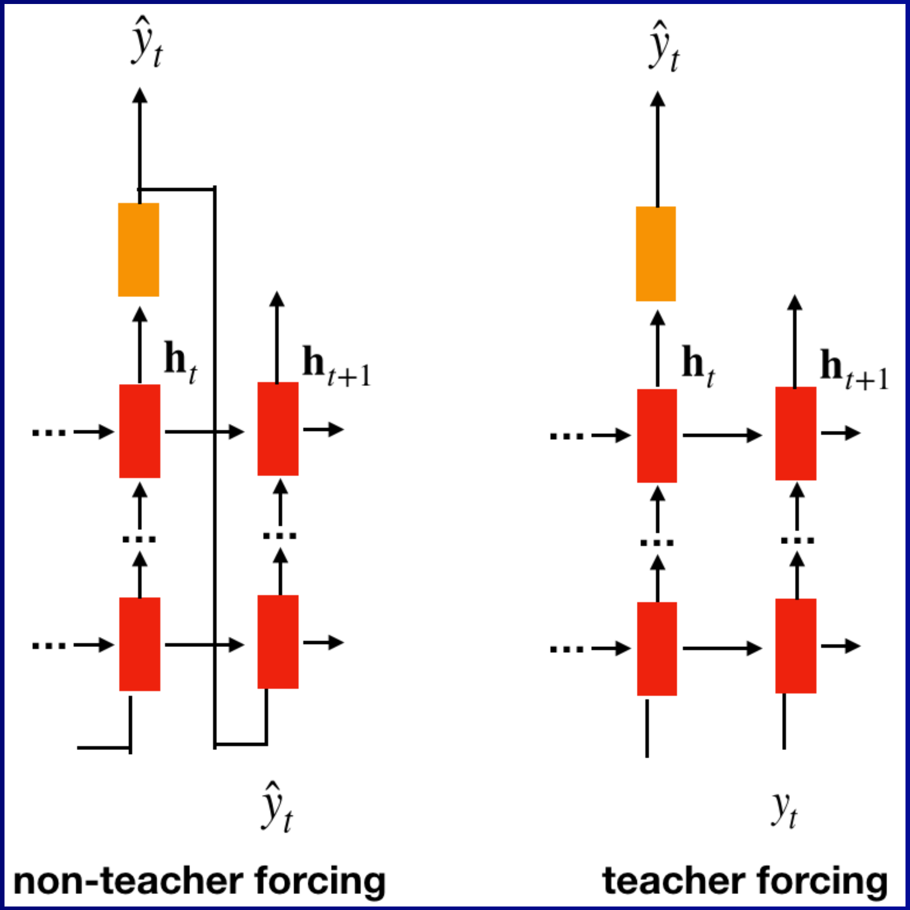
RNN-based language models, as all parametric language models, are trained using the usual negative log-likelihood loss: if the training sequence is $\langle w_1, \dots, w_n \rangle$ and $\hat{P}_i$ is the model's output distribution for the ith continuation probability, then the loss is

$$-\sum_{i=1}^{n} \log \hat{P}_i(w_i).$$

But what should the *input* of the RNN be at each time step during training? Should it come from the training data, or from the RNN's previous prediction?

Training cont.

RNN language models are typically trained using the training data as input. This is called *teacher forcing*.



Exposure bias

Although teacher forcing is by far the most used training method, it has a major problem, the phenomenon called *exposure bias*:

- Language models trained with teacher forcing are only exposed to situations in which the entirety of their input comes from the training corpus.
- During *inference*, in contrast, they have to produce continuations for texts not in the training data, most importantly, during text generation they have to continue *their own output*.

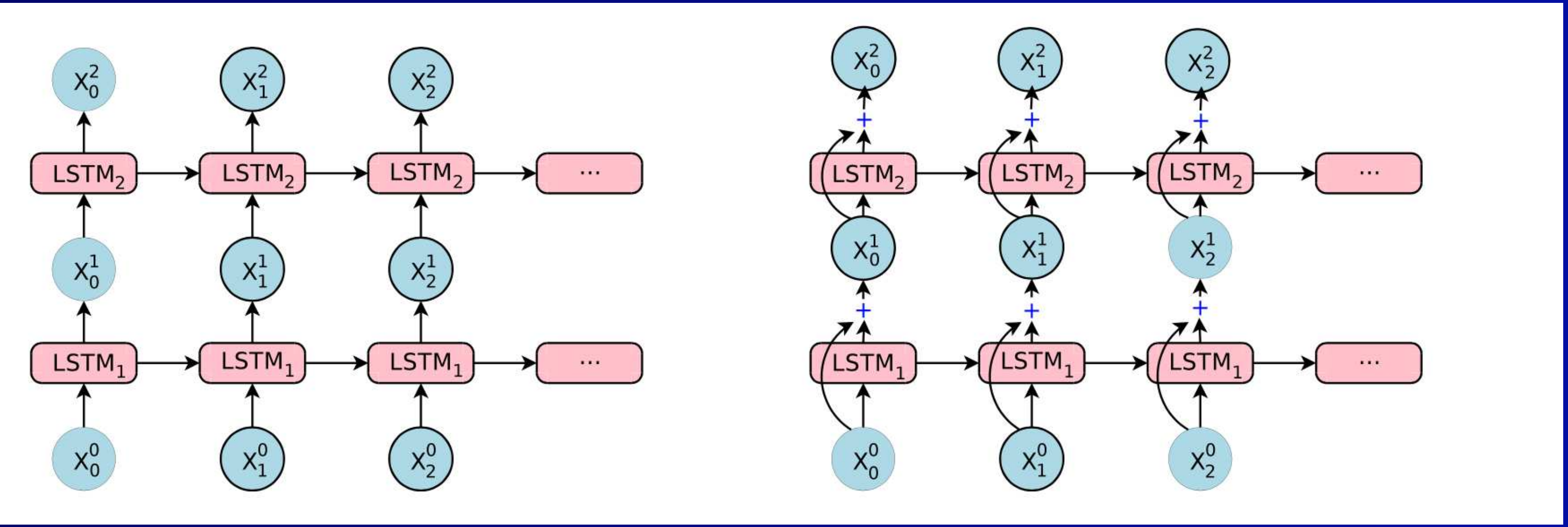
Exposure bias: solutions

- *Scheduled sampling*:[2] randomly choose at each time step between using the training data as input or sampling from the model's prediction. The probability of choosing from the training set starts from 1.0 and is slowly decreased during training.
- *Differentiable sampling*: In original scheduled sampling the error was not backpropagated through the used sampling operation, because it was undifferentiable. In response, alternative sampling solutions have been developed that are differentiable, the most important is using the so-called Gumbel softmax reparametrization (Jang et al. [2016]).

[2] Bengio et al. [2015].

Multiple RNN layers

Modern RNN-based architectures frequently stack multiple RNN cells on top of each other as layers, analogously to multi-layer feedforward networks:



(The architecture on the right uses skip connections.)

Performance

Before the appearance of transformers, LSTM-based language models performed consistently better than other architectures, and they are pretty competitive even now.

On 5 of the 9 language modeling datasets tracked by NLP-progress, models based on an LSTM-variant, the so-called Mogrifier LSTM have the best performance, and LSTM-based models are very close to the (transformer produced) state-of-the-art on 3 of the 4 remaining datasets.

References

References

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28:1171–1179, 2015. URL https://bit.ly/3rHrYCO.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3: 1137–1155, 2003. URL https://bit.ly/3rkKa4M.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990. URL https://crl.ucsd.edu/~elman/Papers/fsit.pdf.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. URL https://bit.ly/3hoXoJx.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Christopher Olah. Understanding LSTM networks. 2015. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/.