Lecture 4 Sequence to Sequence Learning

Shiwei Lan[1]

[1]School of Mathematical and Statistical Sciences
Arizona State University

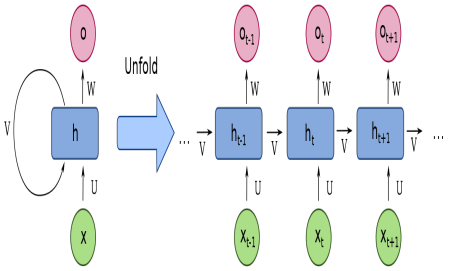
STP598 Advanced Deep Learning Models
Spring 2026

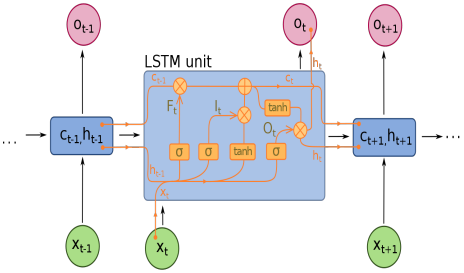
Table of Contents

Recurrent Neural Networks

- Recurrent Neural Networks (RNN) are a family of neural networks for processing sequential data, e.g. time-series, a sequence of images, etc.
- CNN is specialized for processing a grid of values (e.g. images); RNN is specialized for processing a sequence of values (e.g. time series).
- The idea going from MLP to RNN dates back to 1980s: *sharing parameters across different parts of a model*.
- What about CNN across a 1-D temporal sequence? Shallow.
- RNNs designed to retain long time memory, e.g. LSTM, GRU, are particularly useful and successful.

Recurrent Neural Networks

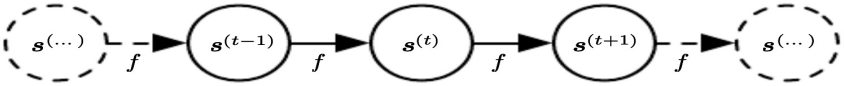




Recurrence

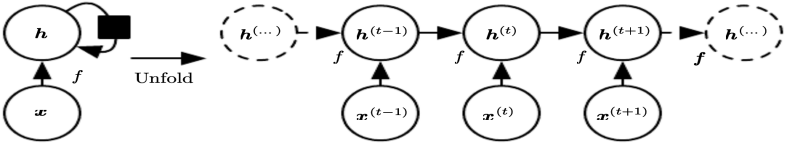
- Consider the classical form of a dynamical system:

$$s^{(t)} = f(s^{(t-1)}; \theta) \tag{1}$$



- Sometimes, the dynamics may be driven by external signal $x^{(t)}$

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta) \tag{2}$$



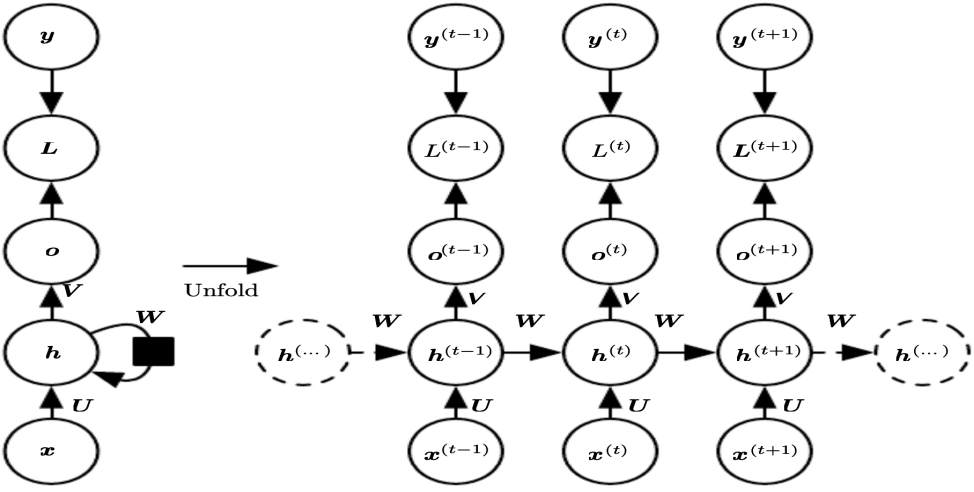
Unfolding

- We can represent the unfolded recurrence after t steps with a function $g^{(t)}$:

$$h^{(t)} = g^{(t)}(x^{(t)}, x^{(t-1)}, \cdots, x^{(1)}) = f(h^{(t-1)}, x^{(t)}; \theta) \tag{3}$$

- The unfolding process thus introduces two major advantages:
  1. Regardless of the sequence length, the learned model always has the same input size, because it is specified in terms of transition from one state to another state, rather than specified in terms of a variable-length history of states.
  2. It is possible to use the *same* transition function f with the same parameters at every time step.

Recurrent Neural Networks



Recurrent Neural Networks

- Assume the output is discrete, as if the RNN is used to predict words or characters.
- Let $\hat{y}$ be a vector of normalized probabilities over the output.
- Forward propagation begins with a specification of the initial state $h^{(0)}$. Then from $t = 1$ to $t = \tau$, we have

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)} \tag{4}$$

$$h^{(t)} = \tanh(a^{(t)}) \tag{5}$$

$$o^{(t)} = c + Vh^{(t)} \tag{6}$$

$$\hat{y}^{(t)} = \mathrm{softmax}(o^{(t)}) \tag{7}$$

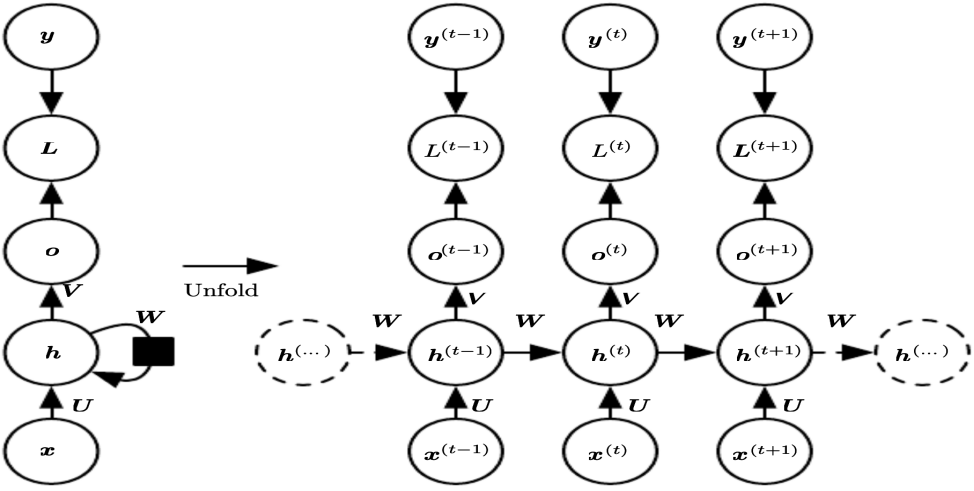
where the parameters are the bias vectors b and c along with the weight matrices U, V and W, respectively for input-to-hidden, hidden-to-output and hidden-to- hidden connections.

- The total loss for a given sequence of x values paired with a sequence of y values would be the sum of the losses over all the time steps.
- Let $L^{(t)}$ be the negative log-likelihood of $y^{(t)}$ given $x^{(1)}, \cdots, x^{(t)}$.

$$\begin{aligned} L(\{x^{(1)}, \cdots, x^{(\tau)}\}, \{y^{(1)}, \cdots, y^{(\tau)}\}) &= \sum_t L^{(t)} \\ &= -\sum_t \log p_{\text{model}}(y^{(t)} | \{x^{(1)}, \cdots, x^{(t)}\}) \end{aligned} \tag{8}$$

  where $p_{\text{model}}(y^{(t)} | \{x^{(1)}, \cdots, x^{(t)}\})$ is given by reading the entry for $y^{(t)}$ from the model's output vector $\hat{y}^{(t)}$, i.e. $L^{(t)} = -o^{(t)} + \log \sum_i \exp(o_i^{(t)})$.
- The runtime is $\mathcal{O}(\tau)$; the memory cost is also $\mathcal{O}(\tau)$.
- The back-propagation algorithm applied to the unrolled graph is called **back-propagation through time** (BPTT).

BPTT in RNN


y
L
o
V
h
W
U
x
Unfold
y(t−1)
y(t)
y(t+1)
L(t−1)
L(t)
L(t+1)
o(t−1)
o(t)
o(t+1)
h(···)
h(t−1)
h(t)
h(t+1)
x(t−1)
x(t)
x(t+1)

BPTT in RNN

- We compute the gradients in RNN:

$$\frac{\partial L}{\partial L^{(t)}} = 1 \tag{9}$$

$$(\nabla_{o^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}} \tag{10}$$

$$\nabla_{h^{(\tau)}} L = V^T \nabla_{o^{(\tau)}} L \tag{11}$$

$$\nabla_{h^{(t)}} L = \left(\frac{\partial h^{(t+1)}}{h^{(t)}} \right)^T (\nabla_{h^{(t+1)}} L) + \left(\frac{\partial o^{(t)}}{h^{(t)}} \right)^T (\nabla_{o^{(t)}} L) \tag{12}$$

$$= W^T (\nabla_{h^{(t+1)}} L) \mathrm{diag} \left(1 - (h^{(t+1)})^2 \right) + V^T (\nabla_{o^{(t)}} L) \tag{13}$$

BPTT in RNN

$$\nabla_{\boldsymbol{c}} L = \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}} \right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L \tag{10.22}$$

$$\nabla_{\boldsymbol{b}} L = \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}} \right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L \tag{10.23}$$

$$\nabla_{\boldsymbol{V}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}} \right) \nabla_{\boldsymbol{V}} o_i^{(t)} = \sum_t \left(\nabla_{\boldsymbol{o}^{(t)}} L\right) \boldsymbol{h}^{(t)\top} \tag{10.24}$$

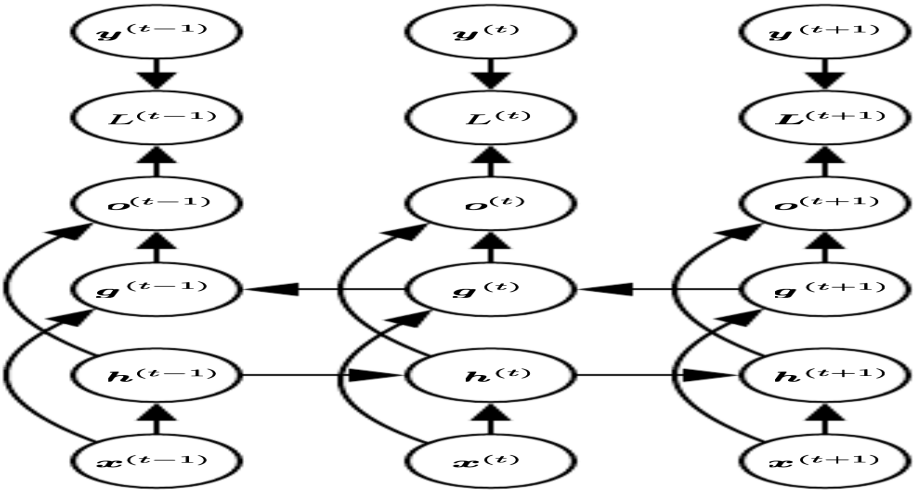
$$\nabla_{\boldsymbol{W}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \tag{10.25}$$

$$= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{h}^{(t-1)\top} \tag{10.26}$$

$$\nabla_{\boldsymbol{U}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \tag{10.27}$$

$$= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{x}^{(t)\top} \tag{10.28}$$

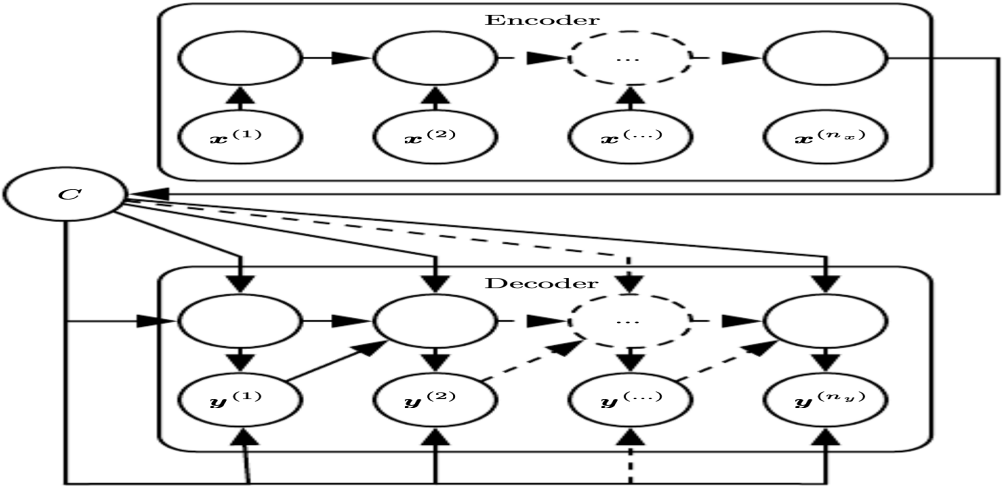
Variant: Bidirectional RNNs



Bidirectional RNNs

- In many applications we want to output a prediction of $y^{(t)}$ which may depend on *the whole input sequence*.
- Bidirectional RNNs (Schuster and Paliwal, 1997) have been extremely successful (Graves, 2012) in applications such as handwriting recognition (Graves et al., 2008; Graves and Schmidhuber, 2009), speech recognition (Graves and Schmidhuber, 2005; Graves et al., 2013).
- Bidirectional RNNs compute a representation that depends on *both the past (through $g^{(t)}$) and the future (through $h^{(t)}$)* without having to specify a fixed-size window around t.
- This idea can be naturally extended to 2-dimensional input (images) by having four RNNs in 4 directions: up, down, left, right.

What if input and output sequences have different length?



Variants: Deep RNNs and Recursive NNs

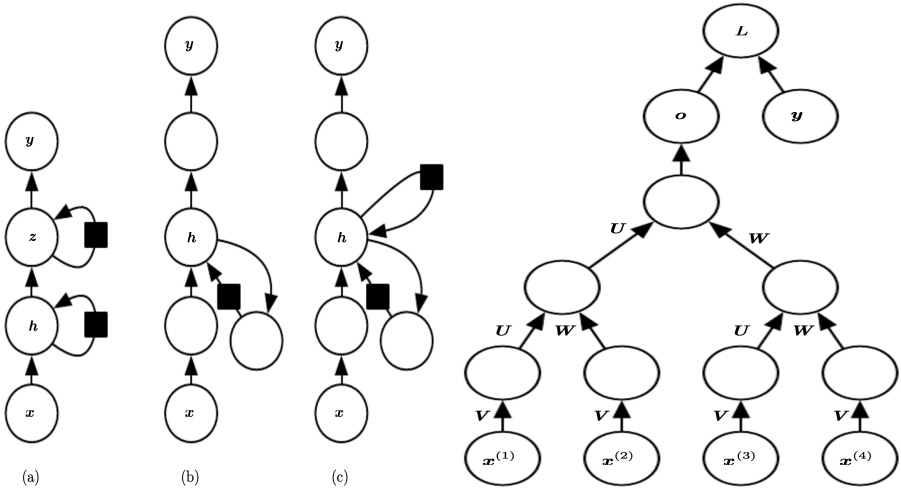
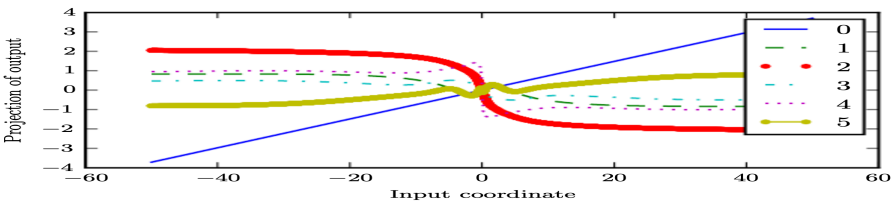


Table of Contents

The challenge of long-term memory

- The basic problem is that gradients propagated over many stages tend to either vanish (most of the time) or explode (rarely, but with much damage to the optimization).
- The difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions (involving the multiplication of many Jacobians) compared to short-term ones.



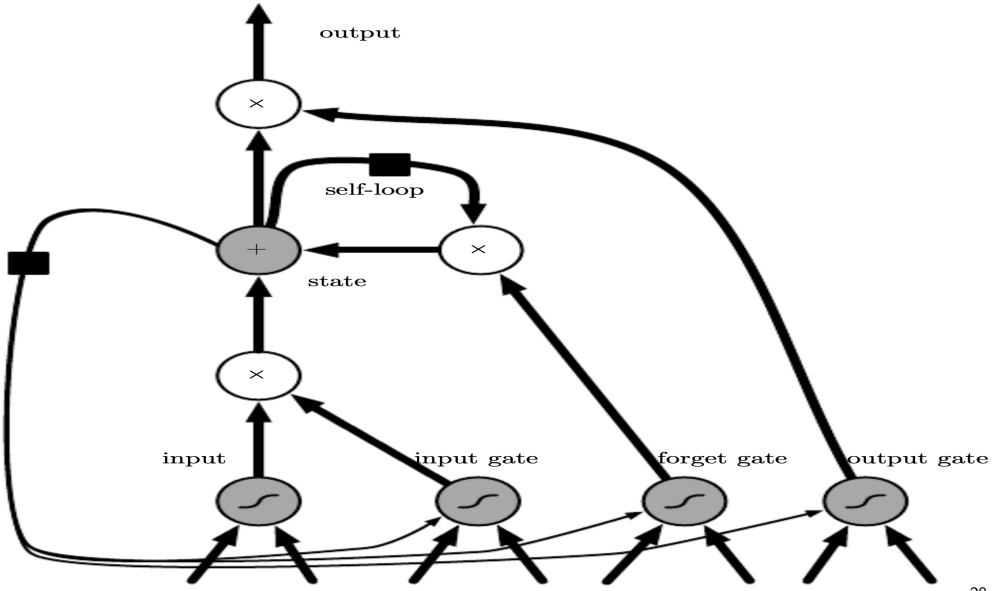
- A heuristic explanation:

$$h^{(t)} = W^T h^{(t-1)} = Q \Lambda Q^T h^{(t-1)} = \cdots = Q \Lambda^t Q^T h^{(0)} \tag{14}$$

Long Short-Term Memory (LSTM)

- The clever idea of introducing self-loops to produce paths where the gradient can flow for long durations is a core contribution of the initial long short-term memory (LSTM) model (Hochreiter and Schmidhuber, 1997)
- A crucial addition has been to make the weight on this self-loop conditioned on the context, rather than fixed (Gers et al., 2000).
- By making the weight of this self-loop gated (controlled by another hidden unit), the time scale of integration can be changed dynamically.
- The LSTM has been found extremely successful in many applications, such as unconstrained handwriting recognition (Graves et al., 2009), speech recognition (Graves et al., 2013; Graves and Jaitly, 2014), handwriting generation (Graves, 2013), machine translation (Sutskever et al., 2014), image captioning (Kiros et al., 2014b; Vinyals et al., 2014b; Xu et al., 2015) and parsing (Vinyals et al., 2014a).

LSTM


output
self-loop
state
input
input gate
forget gate
output gate

- The self-loop weight (or the associated time constant) is controlled by a **forget gate** unit $f_i^{(t)}$ (for time step t and cell i)

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right) \tag{15}$$

- The LSTM cell internal state is updated as follows

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right) \tag{16}$$

LSTM

- The **external input gate** unit $g_i^{(t)}$ is computed similarly

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right) \tag{17}$$

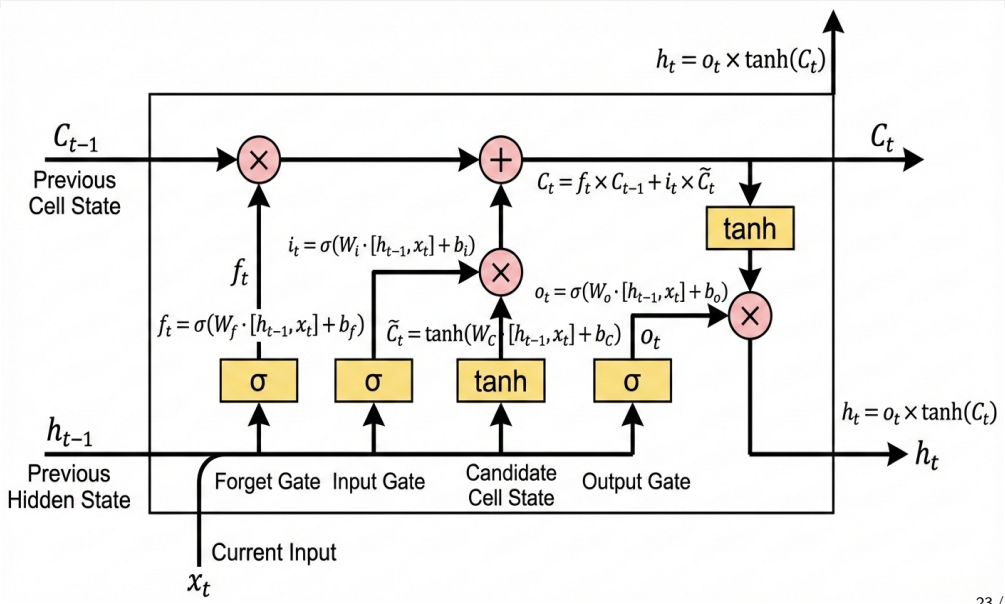
- The output $h_i^{(t)}$ of the LSTM cell can also be shut off, via the **output gate** $q_i^{(t)}$

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)} \tag{18}$$

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right) \tag{19}$$

- A simplified version is called gated recurrent units or GRUs (Cho et al., 2014b; Chung et al., 2014, 2015a; Jozefowicz et al., 2015; Chrupala et al., 2015).

LSTM Recap


$h_t = o_t \times \tanh(C_t)$
C_{t-1}
Previous Cell State
C_t
$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$
tanh
$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
f_t
$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
$\tilde{C}_t = \tanh(W_C [h_{t-1}, x_t] + b_C)$
o_t
σ
σ
tanh
σ
$h_t = o_t \times \tanh(C_t)$
h_{t-1}
Previous Hidden State
h_t
Forget Gate
Input Gate
Candidate Cell State
Output Gate
Current Input
x_t

PyTorch Implementation

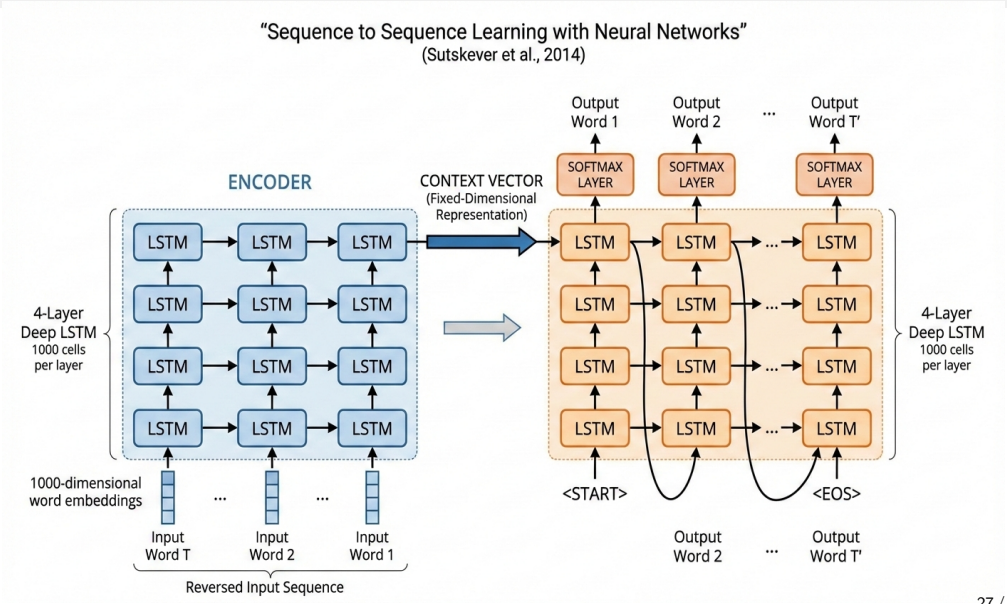
- In PyTorch, we build RNN with torch.nn.RNN, torch.nn.LSTM or torch.nn.GRU.
- torch.nn.RNN, torch.nn.LSTM and torch.nn.GRU share some common arguments:
  - input_size: the number of expected features in the input x
  - hidden_size: the number of features in the hidden state h
  - num_layers: number of recurrent layers
  - nonlinearity: non-linearity to use, 'tanh' or 'relu'.
  - bias: whether to use bias weights $b_i h$ and $b_h h$
  - bidirectional: whether to becomes a bidirectional RNN

Table of Contents

Sequence to Sequence (Seq2Seq) Neural Networks

- Standard deep neural networks (DNNs) work well in learning labeled data with fixed sizes, not mapping sequences to sequences of variable lengths.
- Sutskever et al (2014) proposed Seq2Seq by using a multilayered LSTM to map input sequence into a fixed latent space, then another deep LSTM to decode the target sequence from the latent code.
- One important contribution is the *reverse order of input sequence* to feed into encoding LSTM, which shortens the distance of early parts of both sequences, creating short term memory and making training easier.
- Seq2Seq makes minimal assumption on the sequence structure, but can learn long sequences via LSTM.
- It achieved BLEU score of 36.5 in WMT-14 English-French translation, close to the previous SOTA.

Seq2Seq Architecture



Seq2Seq Model

- Seq2Seq uses LSTM, a special RNN that maps input sequence $(x_1, \cdots, x_T)$ to output sequence $(y_1, \cdots, y_T)$

$$h_t = \sigma(W^{\mathrm{hx}} x_t + W^{\mathrm{hh}} h_{t-1}), \quad y_t = W^{\mathrm{yh}} h_t$$

- LSTM estimates the following conditional probability

$$p(y_1, \cdots, y_{T'} | x_1, \cdots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \cdots, y_{t-1})$$

  where v is the fixed-dimensional latent representation of input sequence $(x_1, \cdots, x_T)$ (output of encoder LSTM), and each $p(y_t | v, y_1, \cdots, y_{t-1})$ is represented by a `softmax` over all the words in the vocabulary.
- We train the model by maximizing $1/|\mathcal{S}| \sum_{T,S \in \mathcal{S}} \log p(T|S)$ with input S and output T in the training set $\mathcal{S}$; then predict the most probable result by $\hat{T} = \arg\max_T p(T|S)$ over outputs by decoder LSTM.

Reverse the Source

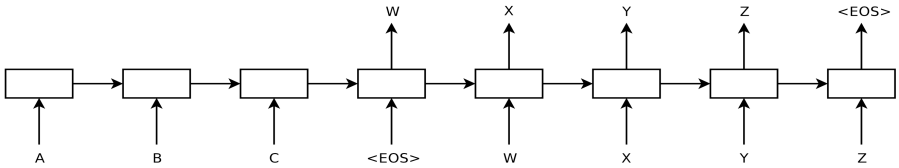


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

- Seq2Seq computes the representation of "A", "B", "C", "¡EOS¿" and then uses this representation to compute the probability of "W", "X", "Y", "Z", "¡EOS¿".
- With the reverse order of words in the source sentence but not the target sentences, SGD could train easier and LSTM did not suffer on very long sentences.

Numerical Results

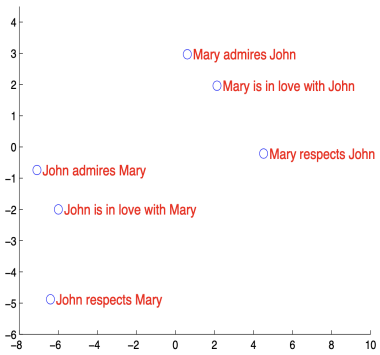
Method	test BLEU score (ntst14)
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	**34.81**

Table 1: The performance of the LSTM on WMT'14 English to French test set (ntst14). Note that an ensemble of 5 LSTMs with a beam of size 2 is cheaper than of a single LSTM with a beam of size 12.

Method	test BLEU score (ntst14)
Baseline System [29]	33.30
Cho et al. [5]	34.54
State of the art [9]	**37.0**
Rescoring the baseline 1000-best with a single forward LSTM	35.61
Rescoring the baseline 1000-best with a single reversed LSTM	35.85
Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs	**36.5**
Oracle Rescoring of the Baseline 1000-best lists	∼45

Table 2: Methods that use neural networks together with an SMT system on the WMT'14 English to French test set (ntst14).

Numerical Results



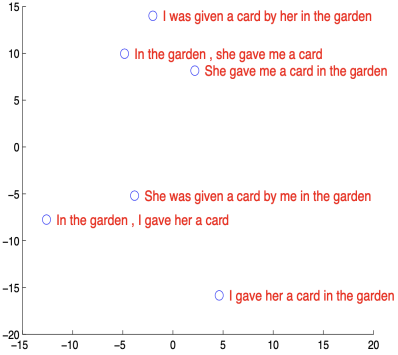


Figure 2: The figure shows a 2-dimensional PCA projection of the LSTM hidden states that are obtained after processing the phrases in the figures. The phrases are clustered by meaning, which in these examples is primarily a function of word order, which would be difficult to capture with a bag-of-words model. Notice that both clusters have similar internal structure.

Implementation of Seq2Seq

- In PyTorch, there is a good tutorial on building Seq2Seq to translate from French to English.
- Here is another Seq2Seq for time series forecasting https://discuss.pytorch.org/t/seq2seq-model-with-attention-for-time-series-forecasting/80463.
- This is a good Github repo for a PyTorch implementation of Seq2Seq https://github.com/bentrevett/pytorch-seq2seq.
- Yet another good tutorial on Seq2Seq for machine translation https://machinelearningmastery.com/building-a-plain-seq2seq-model-for-language-translation/.
- We will demo a Seq2Seq on sol.