Lecture 9 Recurrent Neural Networks

Shiwei Lan[1]

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

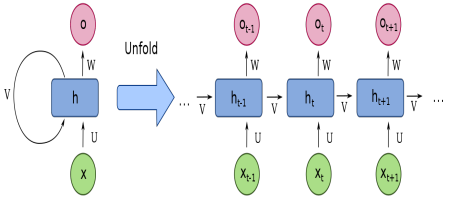
STP598 Machine Learning and Deep Learning
Fall 2021

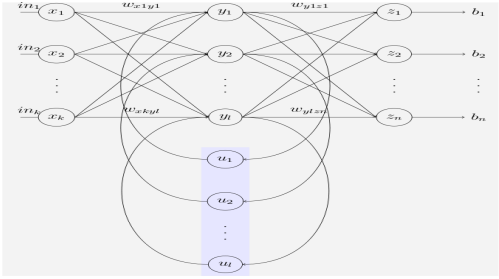
Table of Contents

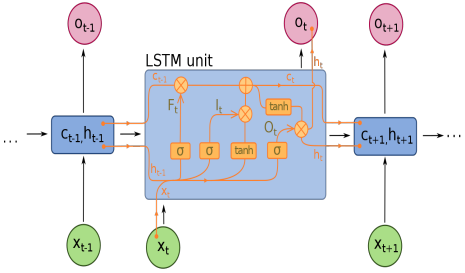
Recurrent Neural Networks

- 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







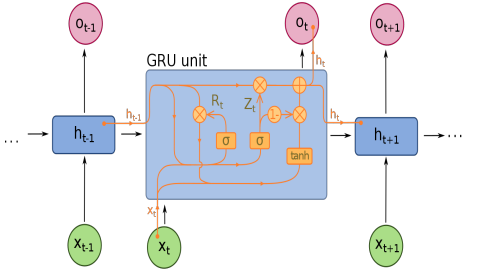
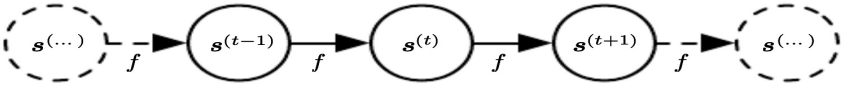


Table of Contents

Recurrence

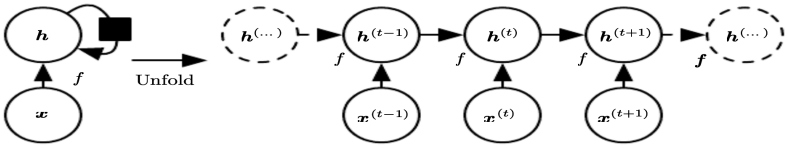
- Consider the classical form of a dynamical system:

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



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

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



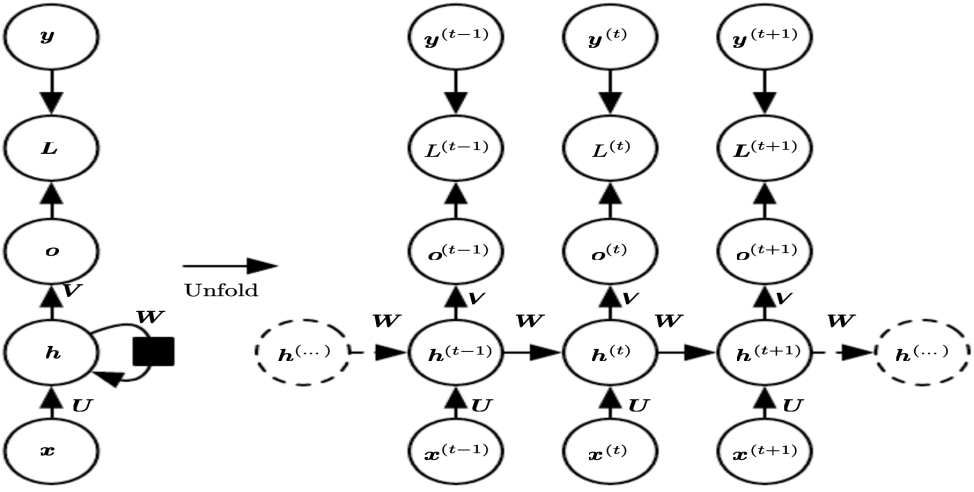
Unfolding

- 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.

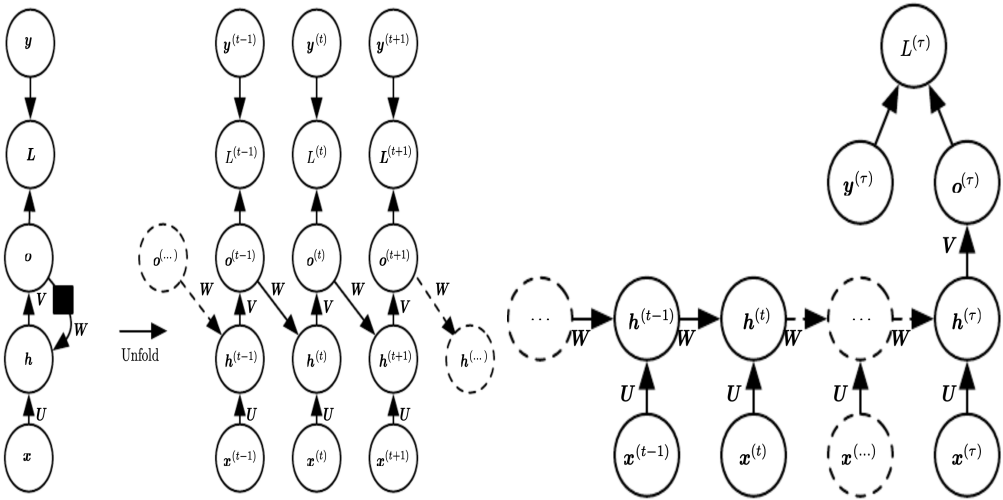
Recurrent Neural Networks

- 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 $x^{(1)}, \cdots, x^{(t)}$.

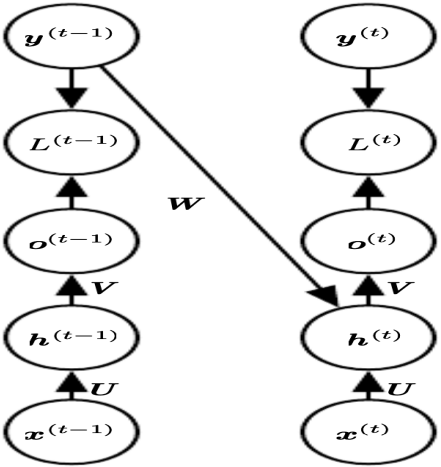
$$\begin{aligned} L(\{x^{(1)}, \cdots, x^{(\tau)}\}, \{y^{(1)}, \cdots, y^{(\tau)}\}) &= \sum_t L^{(t)} \\ &= -\sum_t \log p_{\text{model}}(y^{(t)} | \{x^{(1)}, \cdots, x^{(t)}\}) \end{aligned} \tag{8}$$

  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 $\hat{y}^{(t)}$.
- 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).

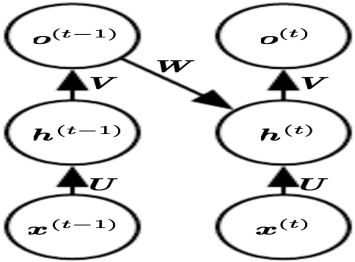
Alternative RNNs



Teacher Forcing

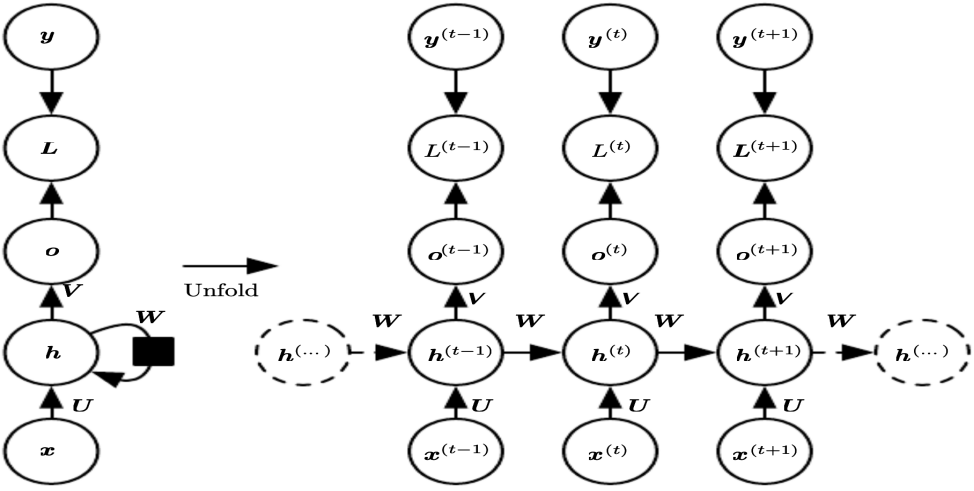


Train time



Test time

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:

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

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

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

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

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

BPTT in RNN

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

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

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

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

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

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

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

What if input is a fixed-size vector?

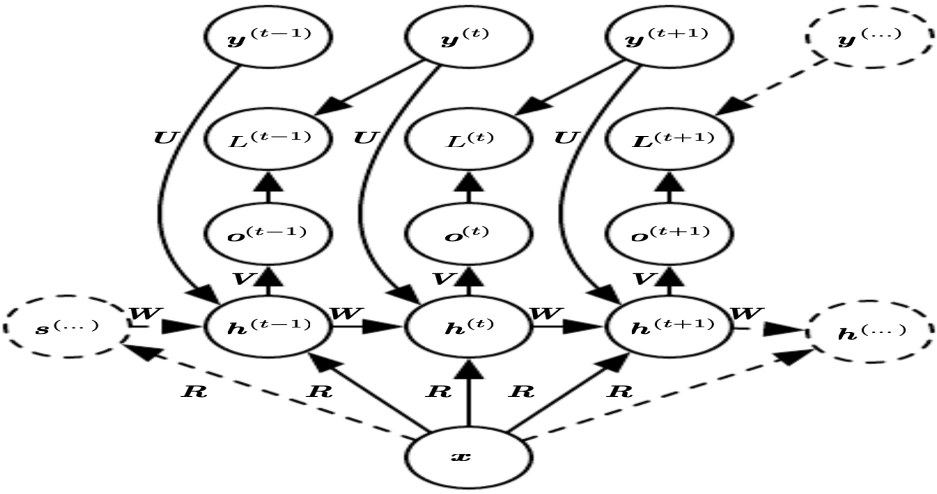
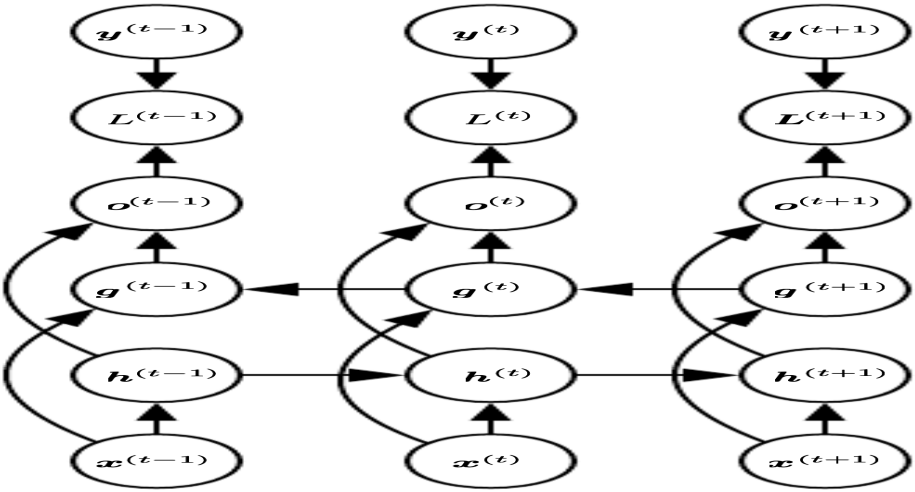


Table of Contents

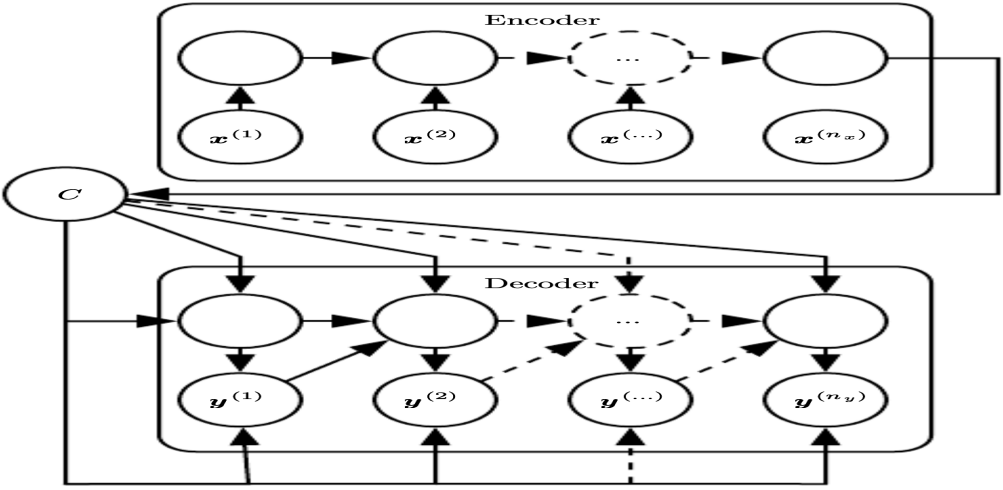
Bidirectional RNNs



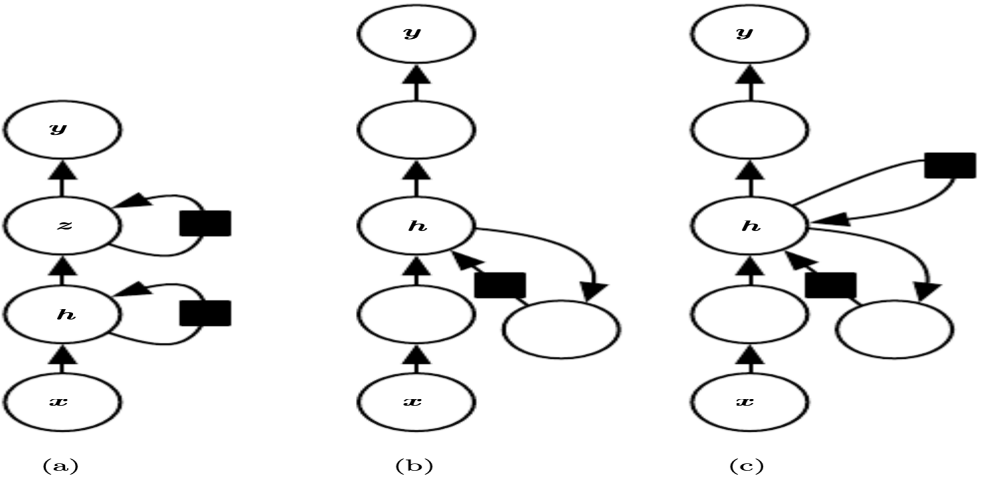
Bidirectional RNNs

- 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 having four RNNs in 4 directions: up, down, left, right.

What if input and output sequences have different length?



Deep RNNs



Recursive NNs

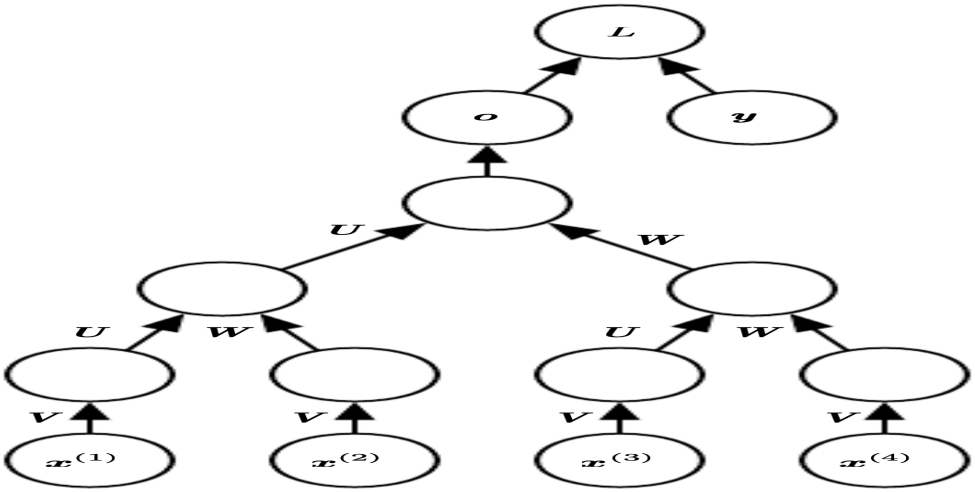
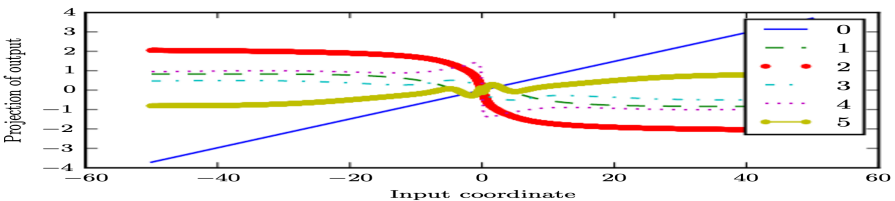


Table of Contents

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:

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

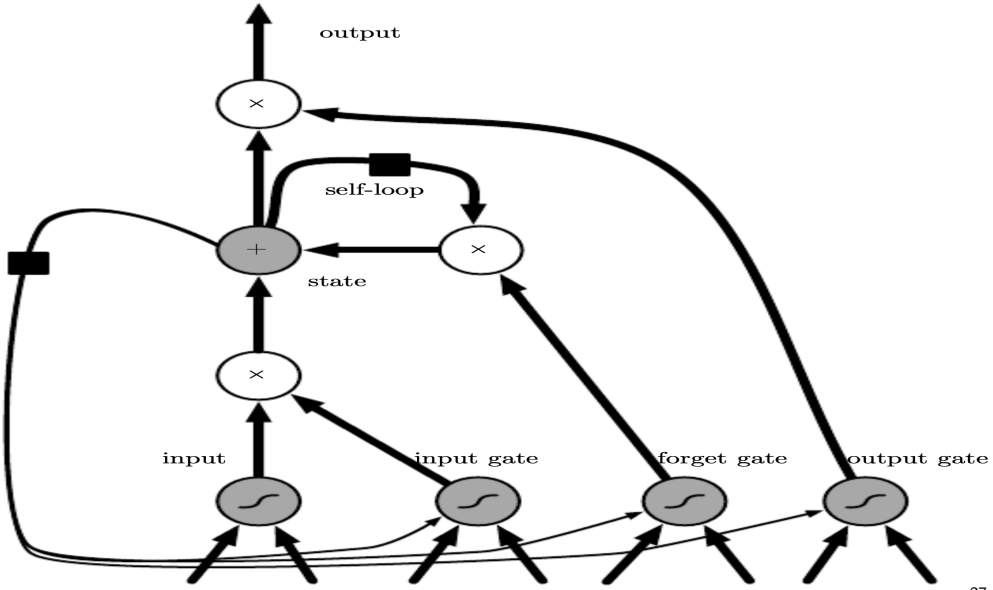
Strategies

- **Echo State Networks**(ESNs, Jaeger and Haas, 2004; Jaeger, 2007b): set the recurrent weights and *earn only the output weights*.
- **Leaky Units**(Mozer, 1992; El Hihi and Bengio, 1996): update running average $\mu^{(t)} = \alpha\mu^{(t-1)} + (1-\alpha)v^{(t)}$. Hidden units with linear self-connections behave similarly.
- **Skip/remove connections**(Lin et al., 1996; El Hihi and Bengio, 1996): longer delays, multiple time-scales.

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

LSTM

- 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

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

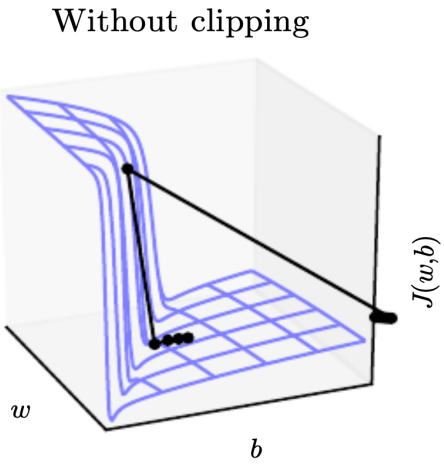
- The output $h_i^{(t)}$ of the LSTM cell can also be shut off, via the **output gate** $q_i^{(t)}$

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

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

- 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).

Clipping Gradients



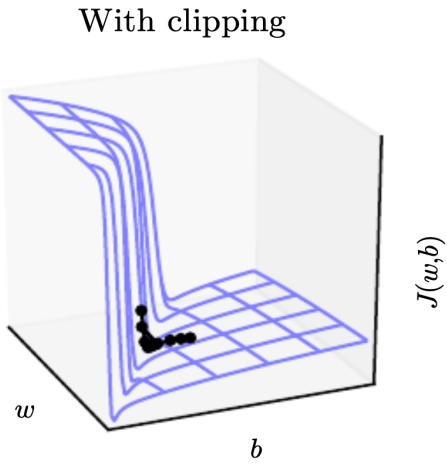


Table of Contents

TensorFlow

- In TensorFlow, we build RNN with tf.keras.layers.SimpleRNN, tf.keras.layers.LSTM or tf.keras.layers.GRU.
- tf.keras.layers.SimpleRNN, tf.keras.layers.LSTM and tf.keras.layers.GRU share some common arguments:
  - units: the dimensionality of the output space.
  - activation: activation function to use
  - use_bias: whether the layer uses a bias vector
  - go_backwards: whether to process the input sequence backwards and return the reversed sequence.
  - unroll: If True, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences.

PyTorch

- 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

More Reading

- TensorFlow
  - https://www.guru99.com/rnn-tutorial.html
  - https://www.kdnuggets.com/2017/04/build-recurrent-neural-network-tensorflow.html
  - https://www.tensorflow.org/guide/keras/rnn
- PyTorch
  - https://www.deeplearningwizard.com/deep_learning/practical_pytorch/pytorch_recu
  - https://blog.floydhub.com/a-beginners-guide-on-recurrent-neural-networks-with-pytorch/
  - https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutoriall