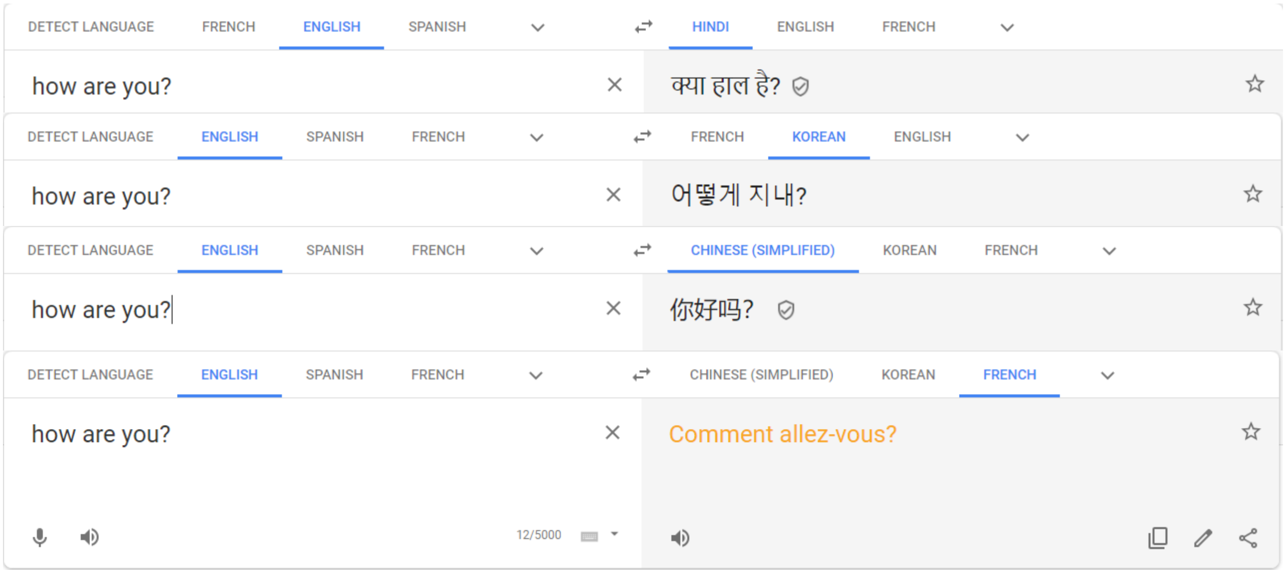
Sequence Modeling: Recurrent and Recursive Nets

ADVANCED ARTIFICIAL INTELLIGENCE

JUCHEOL MOON

How are you?



ENGLISH → HINDI: how are you? → क्या हाल है?

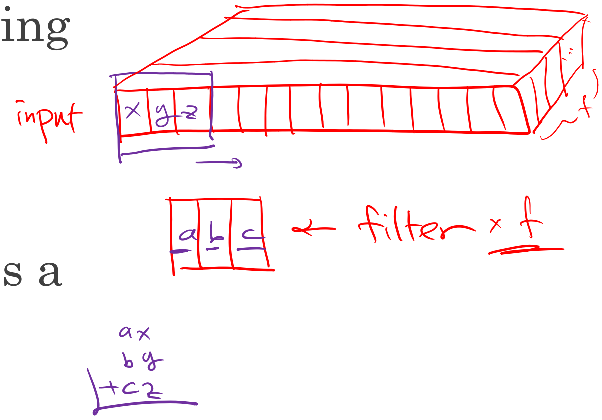
ENGLISH → KOREAN: how are you? → 어떻게 지내?

ENGLISH → CHINESE (SIMPLIFIED): how are you? → 你好吗?

ENGLISH → FRENCH: how are you? → Comment allez-vous?

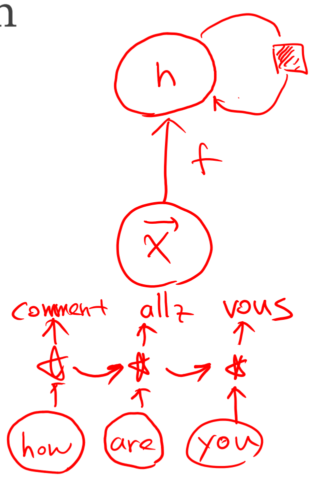
Recurrent neural networks (RNN)

- Neural networks for processing sequential data
- Sharing parameters across different parts of a model
- The use of convolution across a 1-D temporal sequence
  - The convolution operation allows a network to share parameters across time



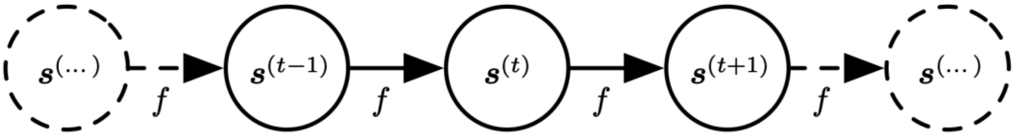
Recurrent neural networks (RNN)

- Recurrent networks share parameters in a different way.
  - Each member of the output is a function of the previous members of the output.
  - Each member of the output is produced using the same update rule applied to the previous outputs.
- RNNs as operating on a sequence that contains vectors $\vec{x}^{(t)}$ with the time step index t ranging from 1 to τ.



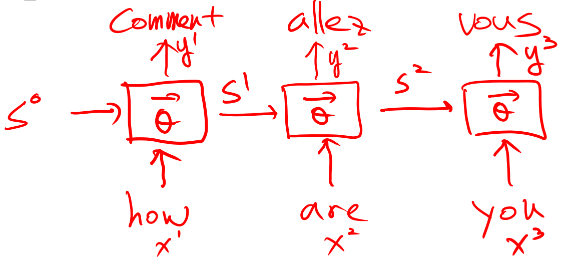
Unfolding Computational Graphs

- The classical form of a dynamical system
  - $\vec{s}^{(t)} = f(\vec{s}^{(t-1)}; \vec{\theta})$
  - where s^t is called the state of the system
- For a finite number of time steps τ, the graph can be unfolded by applying the definition τ − 1 times.
  - $\vec{s}^{(3)} = f(\vec{s}^{(2)}; \vec{\theta}) = f(f(\vec{s}^{(1)}; \vec{\theta}); \vec{\theta})$



Unfolding Computational Graphs

- Many recurrent neural networks use equation
  - $\vec{h}^{(t)} = f(\vec{h}^{(t-1)}, \vec{x}^{(t)}; \vec{\theta})$
  - where the state is the hidden units
- The network typically learns to use $h(t)$ as a kind of lossy summary of the task-relevant aspects of the past sequence of inputs up to t.

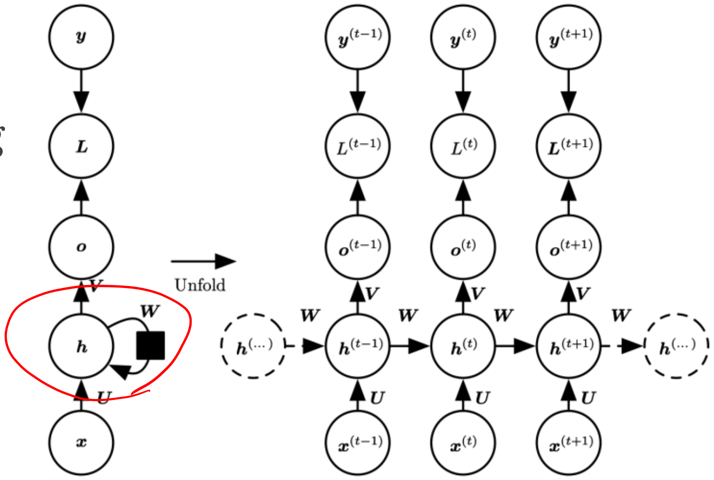


Unfolding Computational Graphs

- The unfolding process thus introduces two major advantages:
  - 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
  - It is possible to use the same transition function f with the same parameters at every time step
- These two factors make it possible to learn a single model f that operates on all time steps and all sequence lengths

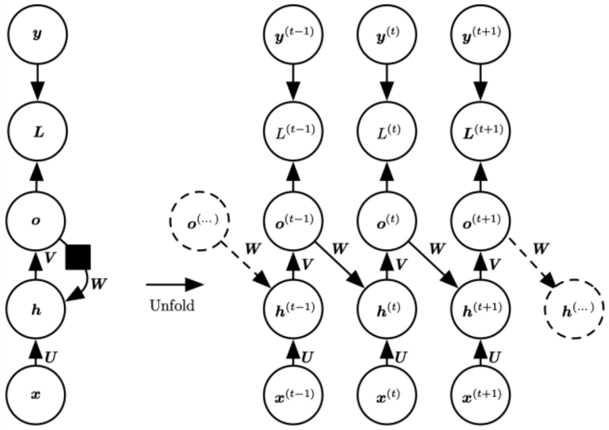
Recurrent Neural Networks

- Recurrent Hidden Units
  - A loss L measures how far each $\vec{o}$ is from the corresponding training target $\vec{y}$.
  - The figure does not specify the choice of activation function for the hidden units.



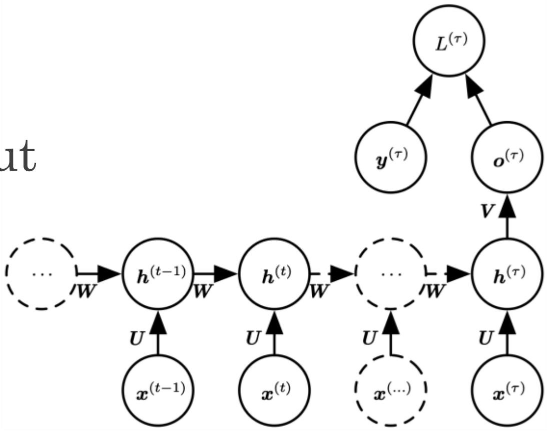
Recurrent Neural Networks

- Recurrence through only the Output
  - An RNN whose recurrence is the feedback connection from the output to the hidden layer.



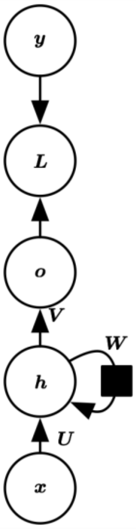
Recurrent Neural Networks

- Sequence Input, Single Output
  - network can be used to summarize a sequence and produce a fixed-size representation used as input for further processing



Forward propagation

- Assumptions
  - The hyperbolic tangent activation function
  - The output $\vec{o}$ as giving the unnormalized log probabilities of each possible value of the discrete variable
  - Applying the softmax operation as a post-processing step to obtain a vector $\hat{\vec{y}}$ of normalized probabilities over the output



$\underset{\vec{x}^{(1)}}{\underline{\text{how}}} \quad \underset{\vec{x}^{(2)}}{\underline{\text{are}}} \quad \underset{\vec{x}^{(3)}}{\underline{\text{you}}} \quad \xrightarrow{\text{one-hot-encode}}$

$\vec{x}^{(1)} = [0\ 0\ 0 \cdots 1 \cdots 0\ 0]$

$\vec{x}^{(2)} = [0\ 0 \cdots 1 \cdots\cdots 0\ 0]$

$\vec{x}^{(3)} = [0\ 0 \cdots\cdots 1 \cdots 0\ 0]$

Forward propagation

- Forward propagation begins with a specification of the initial state $\vec{h}^{(0)}$

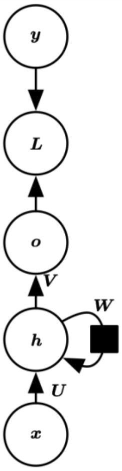
$$\vec{a}^{(t)} = \underline{\vec{b}} + \boxed{\vec{W}}\,\vec{h}^{(t-1)} + \boxed{\vec{U}}\,\vec{x}^{(t)}$$

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

$$\vec{o}^{(t)} = \underline{\vec{c}} + \boxed{\vec{V}}\,\vec{h}^{(t)}$$

$$\hat{\vec{y}}^{(t)} = \text{softmax}(\vec{o}^{(t)})$$

$$\left\{ \begin{array}{l} \vec{b} \ \& \ \vec{c} \quad \text{bias vectors} \\ \vec{W}, \vec{U}, \vec{V} \quad \text{weigh matrices} \end{array} \right.$$



Forward propagation

- An example of a recurrent network that maps an input sequence to an output sequence of the same length
- The total loss for a given sequence of $\vec{x}$ values paired with a sequence of $\vec{y}$ values would then be just the sum of the losses over all the time steps

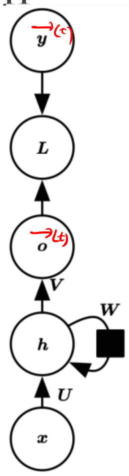
$$\vec{y}=[\overset{\leftarrow 0}{y_1},\overset{\leftarrow 0}{\underline{y_2}}\cdots,y_n]$$

- $$L=\sum_t L^{(t)}=\sum_t L(\vec{y}^{(t)},\hat{\vec{y}}^{(t)})$$
$$=\sum_t\sum_k y_k^{(t)}\log \underset{\sim\sim}{\hat{y}_k^{(t)}}=-\sum_t\sum_k \underbrace{y_k^{(t)}}_{y_i^{(t)}=1}\log\frac{e^{o_k^{(t)}}}{\sum_j e^{o_j^{(t)}}}$$
$$=\sum_t\left(-o_i^{(t)}+\log\sum_j e^{o_j^{(t)}}\right)$$

(i is the target)

Computing the Gradient

- Computing the gradient of this loss function with respect to the parameters is an expensive operation
  - The back-propagation algorithm applied to the unrolled graph with $O(\tau)$ cost is called back-propagation through time (BPTT)



$$L=\sum_t L^{(t)} \qquad \frac{\partial L}{\partial L^{(t)}}=1$$
$$=\cancel{L^{(1)}}+\cancel{L^{(2)}}+\cdots+\boxed{L^{(t)}}+\cdots+L^{(\tau)}$$

$$(\nabla_{\vec{o}^{(t)}}L)_i=\frac{\partial L}{\partial o_i^{(t)}}=\underset{\sim\sim}{\frac{\partial L}{\partial L^{(t)}}}\cdot\frac{\partial L^{(t)}}{\partial o_i^{(t)}}=\frac{\partial L^{(t)}}{\partial o_i^{(t)}}$$

Computing the Gradient

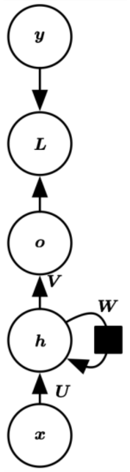
$$\frac{d \log f(x)}{dx} = \frac{f'(x)}{f(x)}$$

- $L^{(t)} = -o_i^{(t)} + \log \sum_j e^{o_j^{(t)}}$

$$\frac{\partial L^{(t)}}{\partial o_i^{(t)}} = -1 + \frac{\frac{\partial}{\partial o_i^{(t)}}\left(e^{o_1^{(t)}} + e^{o_2^{(t)}} + \cdots + e^{o_i^{(t)}} + \cdots\right)}{\sum_j e^{o_j^{(t)}}}$$

$$= -1 + \underbrace{\frac{e^{o_i^{(t)}}}{\sum_j e^{o_j^{(t)}}}}_{\hat{y}_i^{(t)}}$$

$$= -1 + \hat{y}_i^{(t)}$$



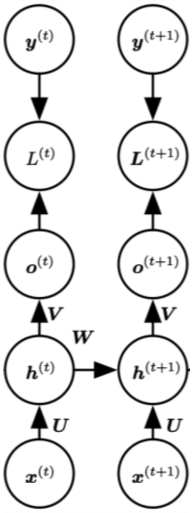
Computing the Gradient

- At the final time (τ) step, $\vec{h}^{(t)}$ only has $\vec{o}^{(t)}$ as a descendent
  - $\left(\nabla_{\vec{h}^{(\tau)}} L\right)_i = \frac{\partial L}{\partial h_i^{(\tau)}} = \frac{\partial L}{\partial L^{(\tau)}} \sum_j \frac{\partial L^{(\tau)}}{\partial o_j^{(\tau)}} \frac{\partial o_j^{(\tau)}}{\partial h_i^{(\tau)}}$

$$\vec{o}^{(t)} = \cancelto{0}{\vec{c}} + \vec{V}\vec{h}^{(t)} \qquad \frac{\partial L^{(\tau)}}{\partial o_j^{(\tau)}} = -1 + \hat{y}_j^{(\tau)}$$

$$\frac{\partial \vec{o}^{(\tau)}}{\partial h_i^{(\tau)}} = \frac{\partial\left(\cancel{V_{j1} h_1^{(\tau)}} + \cancel{V_{j2} h_2^{(\tau)}} + \cdots + \cancel{V_{jn} h_n^{(\tau)}}\right)}{\partial h_i^{(\tau)}} = V_{ji}$$

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

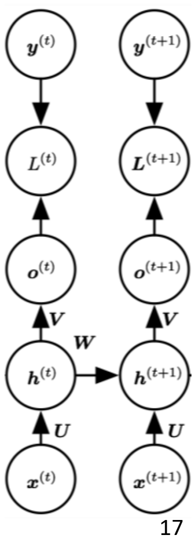


Computing the Gradient

$\tanh' = 1 - \tanh^2$

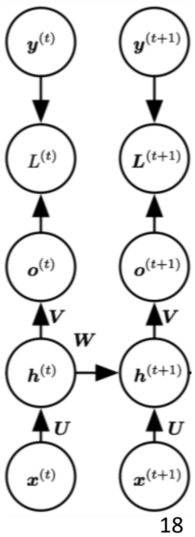
- From $t = \tau - 1$ down to $t = 1$
  - $\left(\nabla_{\vec{h}^{(t)}} L\right)_i = \frac{\partial L}{\partial h_i^{(t)}} = \sum_j \frac{\partial h_j^{(t+1)}}{\partial h_i^{(t)}} \cdot \frac{\partial L}{\partial h_j^{(t+1)}} + \sum_j \frac{\partial o_j^{(t)}}{\partial h_i^{(t)}} \cdot \frac{\partial L}{\partial o_j^{(t)}}$
  - $\vec{h}^{(t+1)} = \tanh\left(\vec{b} + \underline{\vec{W}\vec{h}^{(t)}} + \vec{U}\vec{x}^{(t+1)}\right)$
  - $h_j^{(t+1)} = \tanh\left(\cancel{b_j} + \underbrace{\sum_k W_{jk} h_k^{(t)}} + \cancelto{0}{(\vec{U}\vec{x}^{(t-1)})_j}\right)$

    $\frac{\partial(W_{j1} + W_{j2} + \cdots + \boxed{W_{ji}} + \cdots)}{\partial h_i}$
  - $\frac{\partial h_j^{(t+1)}}{\partial h_{\boxed{i}}^{(t)}} = \left(1 - \left(h_j^{(t+1)}\right)^2\right)(W_{ji})$



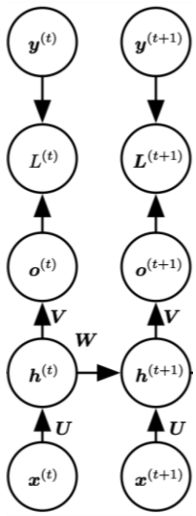
Computing the Gradient

- From $t = \tau - 1$ down to $t = 1$
  - $\left(\nabla_{\vec{h}^{(t)}} L\right)_i = \sum_j \frac{\partial h_j^{(t+1)}}{\partial h_j^{(t)}} \frac{\partial L}{\partial h_j^{(t+1)}} + \sum_j \frac{\partial o_j^{(t)}}{\partial h_i^{(t)}} \cdot \frac{\partial L}{\partial o_j^{(t)}}$
  - $\vec{o}^{(t)} = \underline{\vec{c} + \vec{V}\vec{h}^{(t)}}$ $\quad (\text{—})(|) = (\cdot)$
  - $o_j^{(t)} = c_j + \sum_k V_{jk} \vec{h}_k^{(t)}$
  - $\frac{\partial o_j^{(t)}}{\partial h_i^{(t)}} = V_{ji}$



Computing the Gradient

- From $t = \tau - 1$ down to $t = 1$
- $$\frac{\partial \vec{h}^{(t+1)}}{\partial \vec{h}^{(t)}} = \begin{bmatrix} \frac{\partial h_1^{(t+1)}}{\partial h_1^{(t)}} & \cdots & \frac{\partial h_1^{(t+1)}}{\partial h_n^{(t)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_n^{(t+1)}}{\partial h_1^{(t)}} & \cdots & \frac{\partial h_n^{(t+1)}}{\partial h_1^{(t)}} \end{bmatrix}$$
- $$\nabla_{\vec{h}^{(t)}} L = \left(\frac{\partial \vec{h}^{(t+1)}}{\partial \vec{h}^{(t)}}\right)^T \left(D_{\vec{h}^{(t+1)}} L\right) + \left(\frac{\partial \vec{o}^{(t)}}{\partial \vec{h}^{(t)}}\right)^T \left(D_{\vec{o}^{(t)}} L\right)$$
$$= \vec{W}^T \left(D_{\vec{h}^{(t+1)}} L\right) \text{diag}\left(1 - \left(\vec{h}^{(t+1)}\right)^2\right) + \vec{V}^T \left(D_{\vec{o}^{(t)}} L\right)$$

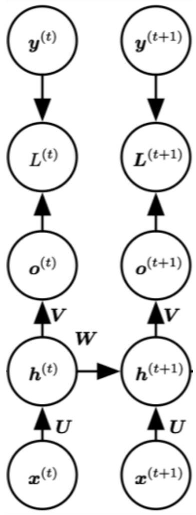


19

Computing the Gradient

- $\vec{a}^{(t)} = \vec{b} + \vec{W}\vec{h}^{(t-1)} + \vec{U}\vec{x}^{(t)}$
- $\vec{h}^{(t)} = \tanh(\vec{a}^{(t)})$
- $\vec{o}^{(t)} = \vec{c} + \vec{V}\vec{h}^{(t)}$

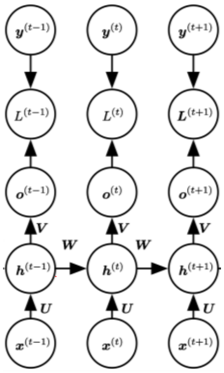
- From $\nabla_{\vec{o}^{(t)}} L$ and $\nabla_{\vec{h}^{(t)}} L$
- $\nabla_{\vec{c}} L = \sum_t \nabla_{\vec{o}^{(t)}} L$
- $\nabla_{\vec{b}} L = \sum_t \text{diag}\left(1 - \left(\vec{h}^{(t)}\right)^2\right) \nabla_{\vec{h}^{(t)}} L$
- $\nabla_{\vec{V}} L = \sum_t \left(\nabla_{\vec{o}^{(t)}} L\right) \vec{h}^{(t)^T}$
- $\nabla_{\vec{W}} L = \sum_t \text{diag}\left(1 - \left(\vec{h}^{(t)}\right)^2\right) (\nabla_{\vec{h}^{(t)}} L) \vec{h}^{(t-1)^T}$
- $\nabla_{\vec{U}} L = \sum_t \text{diag}\left(1 - \left(\vec{h}^{(t)}\right)^2\right) (\nabla_{\vec{h}^{(t)}} L) \vec{x}^{(t)^T}$

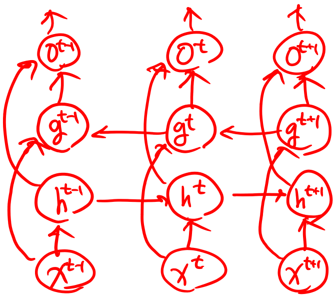


20

Bidirectional RNNs

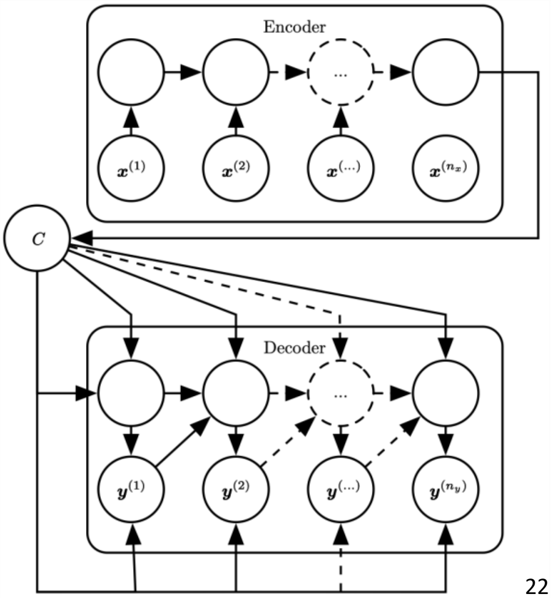
- All of the recurrent networks we have considered up to now have a "causal" structure.
  - The state at time t only captures information from the past, $\underline{x^{(1)}, \dots, x^{(t-1)}}$, and the present input $\boxed{x^{(t)}}$.





Sequence-to-sequence

- How are you?
  - ↔ Comment allez vous?
  - ↔ ¿cómo estás?
- RNN can be trained to map an input sequence to an output sequence which is not necessarily of the same length
- Context variable C represents a semantic summary of the input sequence and is given as input to the decoder.

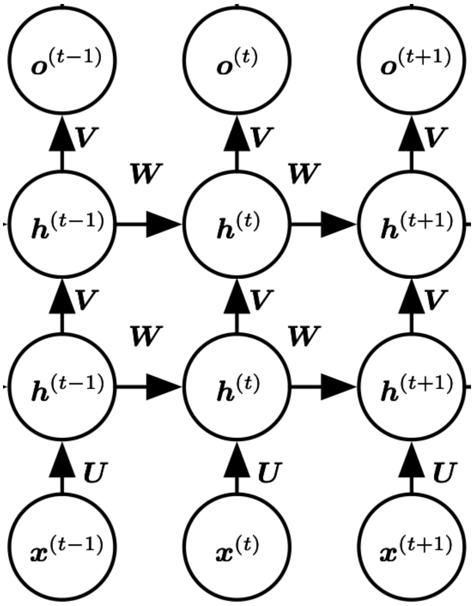


Encoder-Decoder

- The idea
  - (1) an encoder or reader or input RNN processes the input sequence. The encoder emits the context C, usually as a simple function of its final hidden state.
  - (2) a decoder or writer or output RNN is conditioned on that fixed-length vector to generate the output sequence
- One clear limitation of this architecture is when the context C output by the encoder RNN has a dimension that is too small to properly summarize a long sequence
  - make C a variable-length sequence

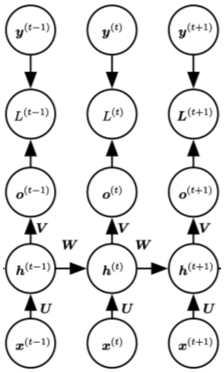
Deep Recurrent Networks

- Three components of RNN
  - from the input to the hidden state
  - from the previous hidden state to the next hidden state
  - from the hidden state to the output



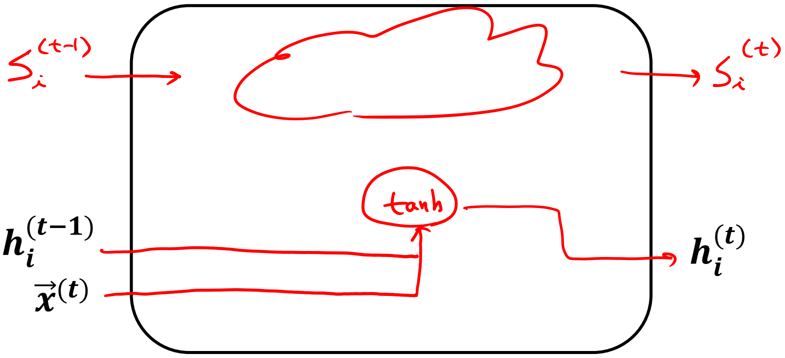
The Challenge of Long-Term Dependencies

- The basic problem is that gradients propagated over many stages tend to either vanish or explode.
  - Simple case w/o activation
  - $\vec{h}^{(t)} = \vec{w}^T \vec{h}^{(t-1)} = (\vec{w}^t)^T \underbrace{(\vec{h}^{(0)})} = \underbrace{\vec{O}^T \vec{\Lambda}^t \vec{O}} \vec{h}^{(0)}$
  - The eigenvalues are raised to the power of t causing eigenvalues with magnitude less than one to decay to zero and eigenvalues with magnitude greater than one to explode



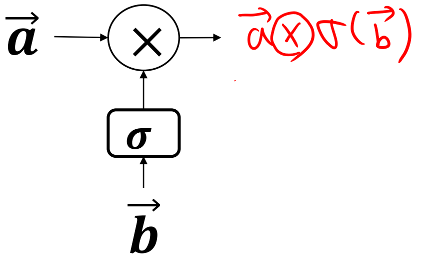
Long Short-Term Memory (LSTM)

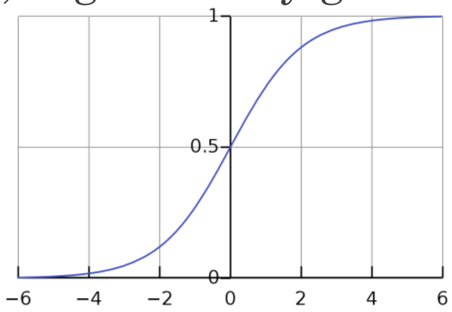
- Self-loops to produce paths where the gradient can flow for long durations
  - $\vec{h}^{(t)} = \tanh(\vec{b} + \vec{w}\,\vec{h}^{(t-1)} + \vec{U}\vec{x}^{(t)})$



Long Short-Term Memory (LSTM)

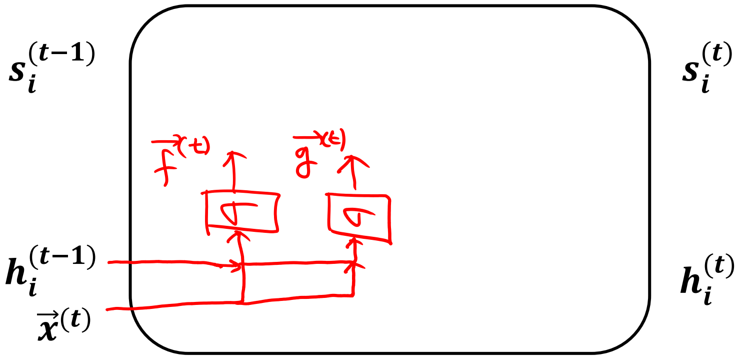
- The key to LSTM is the cell (internal) state $\vec{s}^{(t)}$
- The cell state runs straight down the entire chain, with only some minor linear interactions
- The LSTM does have the ability to remove or add information to the cell state, regulated by gates





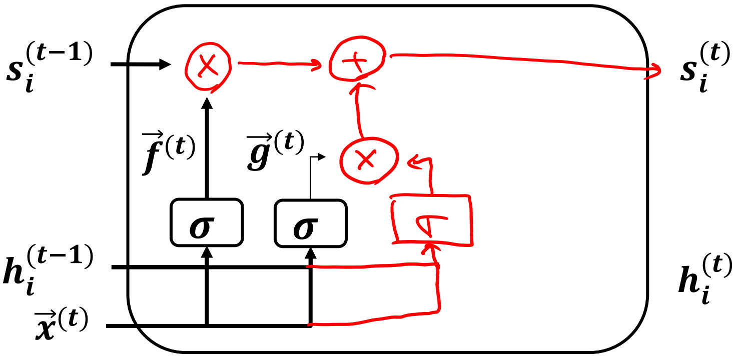
Forget/input gate

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



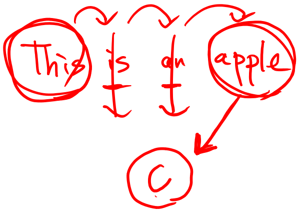
Cell state

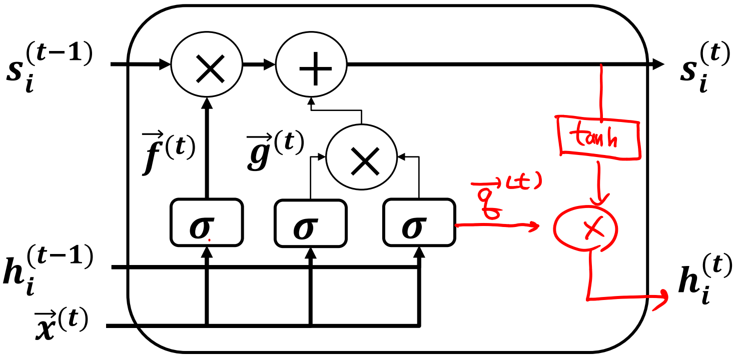
- $s_i^{(t)} = \underline{f_i^{(t)} s_i^{(t-1)}} + g_i^{(t)} \underline{\sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right)}$



Output gate

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



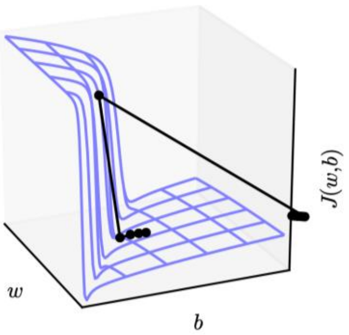


Gradient Clipping

- Recurrent net over many time steps tend to have derivatives that can be either very large in magnitude.
- A simple type of solution has been in use by practitioners for many years

if $\|\vec{g}\| > v$ ← hyper-param.

$$\vec{g} \leftarrow v \cdot \frac{\vec{g}}{\|\vec{g}\|}$$



Regularization

- Naïve dropout
  - Use different masks at different time steps
  - No dropout on the recurrent layers
- Variational dropout
  - Same mask at each time step and recurrent layers