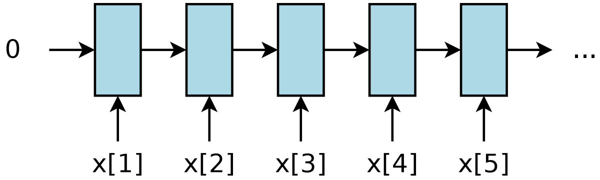
11-695: Competitive Engineering
Implementing Recurrent Neural Networks

Spring 2018

Outline

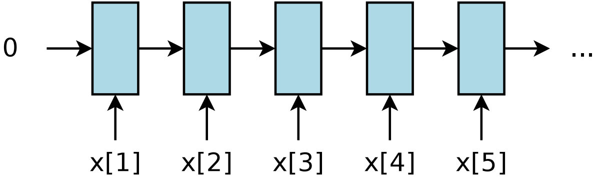
Recurrent Neural Networks



- Processes a sequence of signals
  - $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T \in \mathbb{R}^D$
- ... in a qequential order
  - $\mathbf{h}_0 = \mathbf{0}_H$
  - $\mathbf{h}_t = \mathbf{f}(\mathbf{x}_t, \mathbf{h}_{t-1})$
- Designing an RNN means designing $\mathbf{f} : \mathbb{R}^D \times \mathbb{R}^H \to \mathbb{R}^H$.

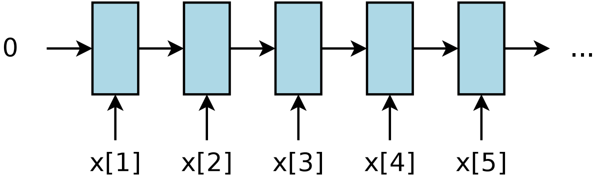
Example 1: A Dummy RNN

CarnegieMellon



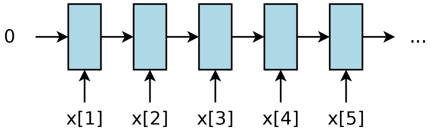
- With the function $\mathbf{f}(\mathbf{x}, \mathbf{h}) = \mathbf{x} + \mathbf{h}$
  - $\mathbf{h}_0 = \mathbf{0}_H$
  - $\mathbf{h}_t = \mathbf{h}_{t-1} + \mathbf{x}_t$
- This network requires $D = H$, and is really dumb...

Example 2: Vanilla RNN



- With the function $\mathbf{f}(\mathbf{x}, \mathbf{h}) = g(\mathbf{x} \cdot \mathbf{W}_x + \mathbf{h} \cdot \mathbf{W}_h)$
  - g is an activation function, e.g. tanh, ReLU, etc.
  - $\mathbf{W}_x \in \mathbb{R}^{D \times H}$, $\mathbf{W}_x \in \mathbb{R}^{H \times H}$ are the *shared* parameters.
- Much less dumb. Invented in 1990. Drove people crazy in 2011...

Example 3: Long Short-Term Memory (LSTM)



- The function $\mathbf{f}$ goes wild

$$\begin{bmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{bmatrix}^{\top} = \begin{bmatrix} \text{sigmoid} \\ \tanh \\ \text{sigmoid} \\ \text{sigmoid} \end{bmatrix} \mathbf{W}_{H \times (D+H)} \cdot \begin{bmatrix} \mathbf{x}_t^{\top} \\ \mathbf{h}_t^{\top} \end{bmatrix} \tag{1}$$

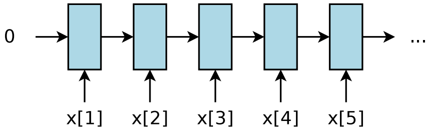
$$\mathbf{c}_t = \mathbf{i} \otimes \mathbf{g} + \mathbf{f} \cdot \mathbf{c}_{t-1}$$

$$\mathbf{h}_t = \mathbf{o} \otimes \tanh \mathbf{c}_{t-1}$$

- Finally looks smart. Invented in 1997. Drove people crazy in 2014...

Example 4: Gated Recurrent Units (GRU)

Carnegie Mellon

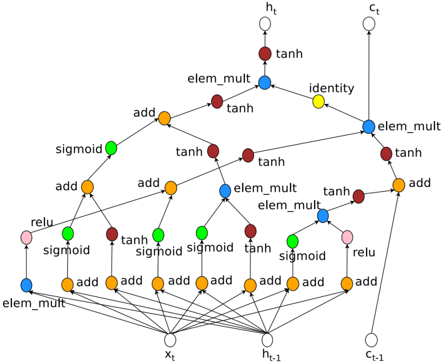


- Someone doesn't like LSTM and wants to be creative with $\mathbf{f}$

$$\begin{aligned}
\mathbf{z} &= \text{sigmoid}(\mathbf{x}_t \cdot \mathbf{W}_{xz} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hz}) \\
\mathbf{r} &= \text{sigmoid}(\mathbf{x}_t \cdot \mathbf{W}_{xr} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hr}) \\
\tilde{\mathbf{h}} &= \text{sigmoid}(\mathbf{x}_t \cdot \tilde{\mathbf{W}}_{x} + (\mathbf{r} \cdot \mathbf{h}_{t-1}) \cdot \tilde{\mathbf{W}}_{h}) \\
\mathbf{h}_t &= (1 - \mathbf{z}) \otimes \mathbf{h}_{t-1} + \mathbf{z} \otimes \tilde{\mathbf{h}}
\end{aligned} \tag{2}$$

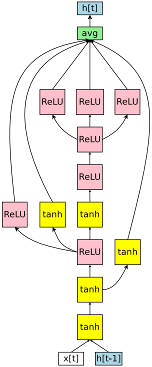
- Sure. We're so tired with different formulas for $\mathbf{f}$

Example 5: Neural Architecture Search



- You can also use a computer to generate good formulas for **f**
- My advisor's work :D

Example 6: Efficient Neural Architecture Search



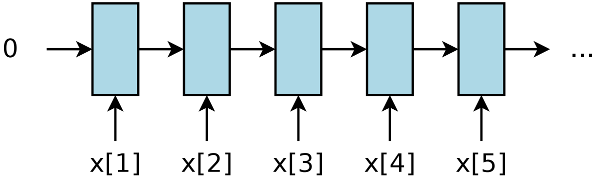
- Yet another one, also generated by a computer
- My work :D
  - looks significantly like a pokemon

Outline

CarnegieMellon

1. Recurrent Neural Networks as a Composite of Functions
2. Using Recurrent Neural Networks as Models
3. Flexible Inputs
4. Flexible Outputs
5. Training Recurrent Neural Networks
6. Test Time Usage
7. Regularization

Back to the big picture

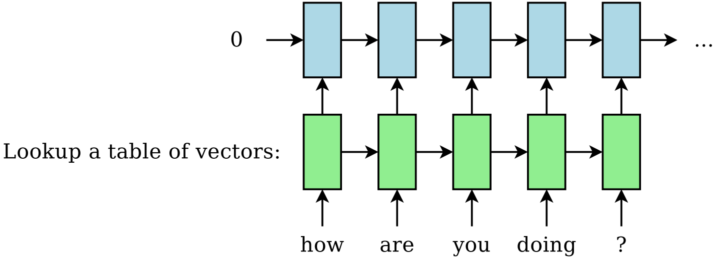


- You can be *very* creative about RNNs:
  - How to choose the input sequence $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T$
  - What to do with the "hidden" sequence $\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_T$

Outline

Word embeddings

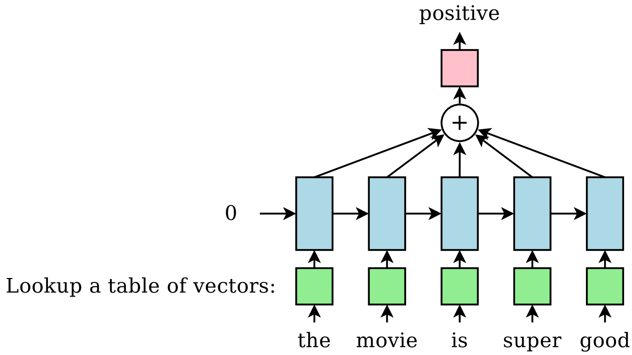
Carnegie Mellon



- To process a sequence of words
  - Store a dictionary that maps words to vectors in $\mathbf{R}^D$
  - Use these $\mathbb{R}^D$ vectors as inputs to an RNN.
- Work by: Yoshua Bengio et al (2003). Drove people crazy in 2013.

Outline

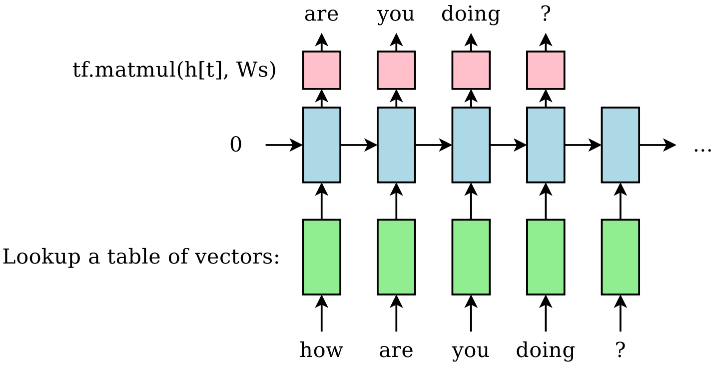
Adding a Softmax head



- You can sum over the $\mathbf{h}_t$ and hook up a softmax head to make a prediction about your sequence
  - This example: sentiment analysis
- My undergraduate advisor's work...

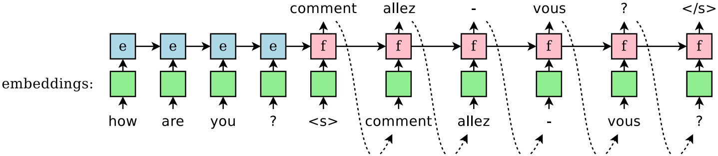
Multiple softmax heads

CarnegieMellon



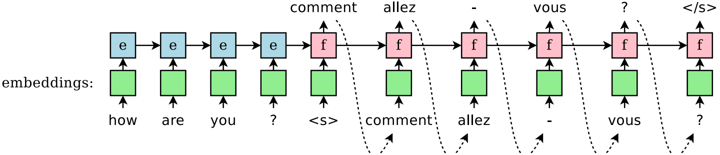
- You can use the $\mathbf{h}_t$ as to predict the next word in your sequence
  - It's called *language model*
  - Because it can model $p(w_t|w_{<t})$

Sequence to Sequence models



- Only use the softmax heads where you want to generate a translated word
  - It's called *neural machine translation*
  - Because it can model $p(\mathbf{t}_t | \mathbf{t}_{<t}, \mathbf{s})$
- Another of my advisor's work...

Attention



- Yet another way to manipulate your $\mathbf{h}_t$ states.
- $\mathbf{e}_i$, $\mathbf{f}_j$ are your blue and red states

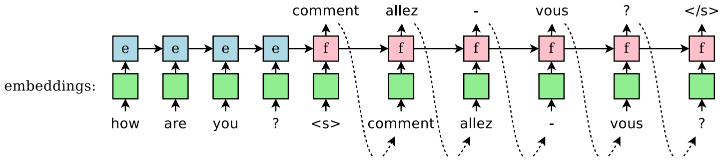
$$
\begin{aligned}
\alpha_{j,i} &= g(\mathbf{f}_j, \mathbf{e}_i) \\
a_{j,i} &= \text{Softmax}(\alpha_{j,1}, \alpha_{j,2}, ..., \alpha_{j,|\mathbf{s}|}) \\
c_j &= \sum_{i=1}^{|\mathbf{s}|} a_{j,i}\mathbf{e}_i
\end{aligned}
\tag{3}
$$

Outline

Carnegie Mellon

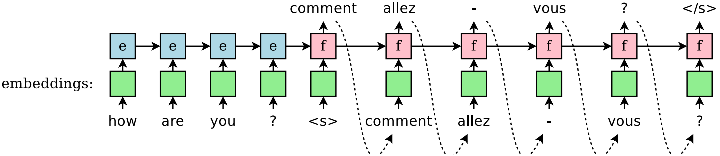
The Picture So Far

Carnegie Mellon



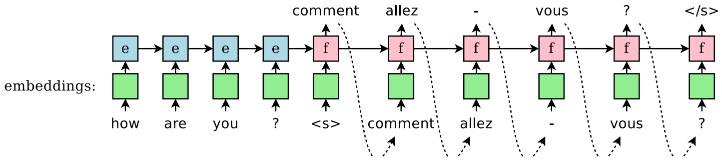
- We have a sequence of hidden vectors
  - In general: $\mathbf{h}_i \in \mathbf{R}^H$ for any input sequences
  - In this case: $\mathbf{e}_i, \mathbf{f}_j \in \mathbf{R}^H$ are the blue and red states
- Can hook up softmax heads to these $\mathbf{h}_i$, $\mathbf{e}_i$, $\mathbf{f}_j$ to make predictions.
- How can we train the RNN to make such predictions?

The Computational Pipeline: Hidden States



- **Inputs:** the words. You need *both* English and French words.
  - `how`, `are`, `you`, `?`, $\langle$`s`$\rangle$, `comment`, `allez`, `-`, `vous`, `?`, $\langle$`s`$\rangle$
- **Word embeddings:** look up the words in a saved dictionary
  - $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4, \mathbf{y}_5, \mathbf{y}_6 \in \mathbb{R}^D$
- **Recurrent Computations:** f is your chosen RNN function
  - Encoder: $\mathbf{e}_0 = 0; \mathbf{e}_t = f(\mathbf{x}_t, \mathbf{e}_{t-1})$
  - Decoder: $\mathbf{f}_0 = \mathbf{e}_4; \mathbf{f}_t = f(\mathbf{y}_t, \mathbf{f}_{t-1})$

The Computational Pipeline: Loss Function



- **Predictions:** Let $\mathbf{W}_{\text{soft}} \in \mathbb{R}^{D \times \text{vocab\_size}}$ be trainable parameters

$$p(y_t | y_{<t}, \mathbf{x}) = \text{Softmax}(\mathbf{f}_{t-1} \cdot \mathbf{W}_{\text{soft}}), \text{ for } t = 2, 3, ..., |\mathbf{y}| \tag{4}$$

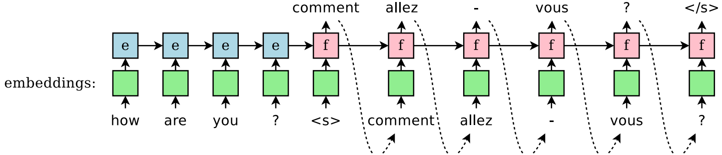
$$p(\mathbf{y}|\mathbf{x}) = \prod_{t=2}^{|\mathbf{y}|} p(y_t | y_{<t}, \mathbf{x}) \tag{5}$$

- **Loss function:** The canonical cross-entropy loss

$$\mathcal{L} = -\log p(\mathbf{y}|\mathbf{x}) = -\sum_{t=2}^{|\mathbf{y}|} \log p(y_t | y_{<t}, \mathbf{x}) \tag{6}$$

The Computational Pipeline: Training

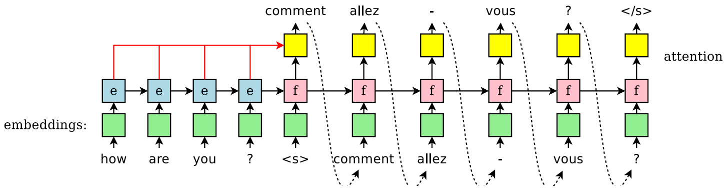
Carnegie Mellon



- We have defined *a computational graph*
  - which is a composite of *many* functions
- Thus we can use back-propagation to compute the gradients
  - which is just the chain rule
- Model parameters consist of:
  - Relevant word embeddings
  - $\mathbf{W}_{\text{soft}}$
  - Any parameters of the recurrent function f

The Computational Pipeline: Attention

Carnegie Mellon



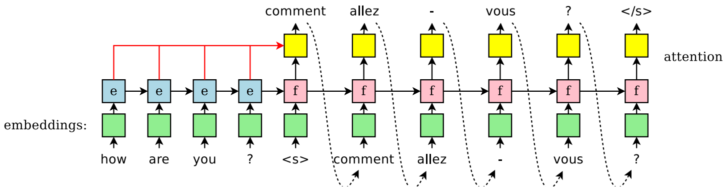
- **Recurrent Computations:** f is your chosen RNN function
  - Encoder: $\mathbf{e}_0 = 0; \mathbf{e}_t = f(\mathbf{x}_t, \mathbf{e}_{t-1})$; Decoder: $\mathbf{f}_0 = \mathbf{e}_4; \mathbf{f}_t = f(\mathbf{y}_t, \mathbf{f}_{t-1})$
- **Predictions:** previously *without attention*

$$p(y_t | y_{<t}, \mathbf{x}) = \text{Softmax}(\mathbf{f}_{t-1} \cdot \mathbf{W}_{\text{soft}}), \text{ for } t = 2, 3, ..., |\mathbf{y}| \qquad (7)$$

- **Predictions:** now *with attention*

$$p(y_t | y_{<t}, \mathbf{x}) = \text{Softmax}(\mathbf{a}(\mathbf{f}_{t-1}, \mathbf{e}_{1\cdots|\mathbf{x}|}) \cdot \mathbf{W}_{\text{soft}}) \qquad (8)$$

The Computational Pipeline: Attention



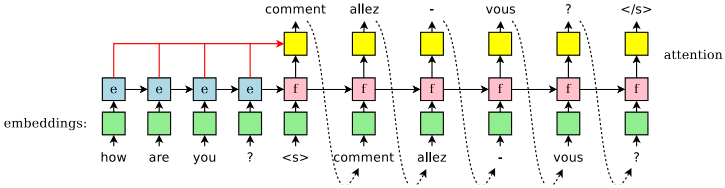
- **Predictions:** now *with attention*

$$p(y_t|y_{<t}, \mathbf{x}) = \mathrm{Softmax}(\mathbf{a}(\mathbf{f}_{t-1}, \mathbf{e}_{1\ldots|\mathbf{x}|}) \cdot \mathbf{W}_{\mathrm{soft}}) \quad (9)$$

- **Attention:** how is $\mathbf{a}(\mathbf{f}, \mathbf{e}_{1\ldots|\mathbf{x}|})$ computed?

$$\alpha_i = g(\mathbf{f}, \mathbf{e}_i);\ a_i = \mathrm{Softmax}(\alpha_{1\ldots|\mathbf{x}|});\ \mathbf{a}(\mathbf{f}, \mathbf{e}_{1\ldots|\mathbf{x}|}) = \sum_{i=1}^{|\mathbf{x}|} a_i \mathbf{e}_i \quad (10)$$

The Computational Pipeline: Attention

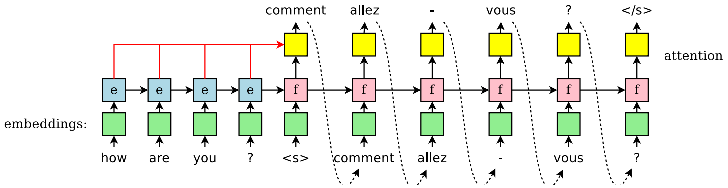


- **Attention:** how is $\mathbf{a}(\mathbf{f}, \mathbf{e}_{1\dots|\mathbf{x}|})$ computed?

$$\alpha_i = g(\mathbf{f}, \mathbf{e}_i);\ a_i = \text{Softmax}(\alpha_{1\dots|\mathbf{x}|});\ \mathbf{a}(\mathbf{f}, \mathbf{e}_{1\dots|\mathbf{x}|}) = \sum_{i=1}^{|\mathbf{x}|} a_i \mathbf{e}_i \quad (11)$$

- Choices of g:
  - Bahdanau attention: $g(\mathbf{f}, \mathbf{e}_i) = \tanh\left(\mathbf{f} \cdot \mathbf{w}_f + \mathbf{e}_i \cdot \mathbf{w}_e\right) \cdot \mathbf{v}$, where $\mathbf{w}_f, \mathbf{w}_e \in \mathbf{R}^{H \times H}$ and $\mathbf{v} \in \mathbb{R}^{H \times 1}$ are trainable parameters
  - Luong attention: $g(\mathbf{f}, \mathbf{e}_i) = \mathbf{f} \cdot \mathbf{e}_i^\top$

The Computational Pipeline: Attention

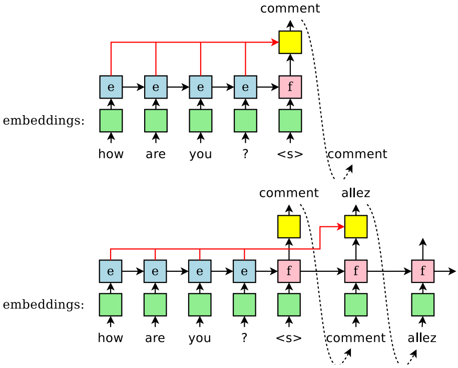


- Even with attention, the overall RNN is still a composite of functions
- The training procedure stays the same

Outline

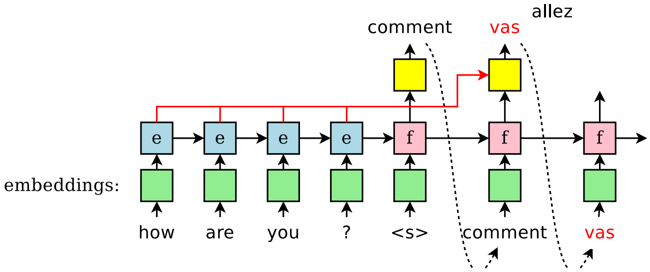
CarnegieMellon

How to Translate with a Trained RNN?



- Goes step-by-step, based on *your own predictions*

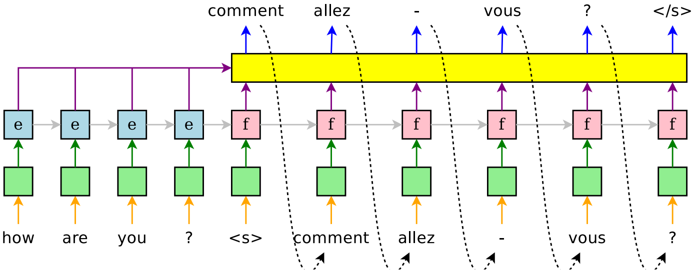
What If You Are Wrong?



- You live with your mistakes...
- Yes, it is bad. Therefore many people are finding a fix
  - Reinforcement Learning: Data as Demonstrator; MIXER, etc.
  - Reward-Augmented Maximum Likelihood

Outline

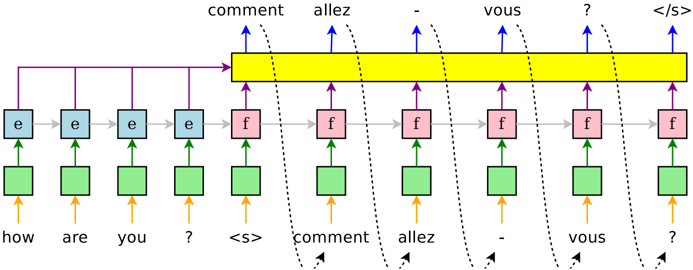
General Regularization Strategy: Dropout



- Each colored arrowed can be dropped using *the same mask*.
  - Word embeddings dropout mean to remove *the whole word*

Other Strategies: ℓ_p

Carnegie Mellon



- ℓ_2 norm of all or some parameters
- ℓ_2 norm of all or some hidden states: $\sum_i \|\mathbf{e}_i\|^2$, $\sum_j \|\mathbf{f}_j\|^2$
- ℓ_2 difference of all or some hidden states: $\sum_i \|\mathbf{e}_i - \mathbf{e}_{i-1}\|^2$, $\sum_j \|\mathbf{f}_j - \mathbf{f}_{j-1}\|^2$