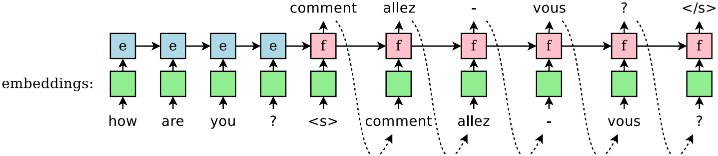
11-695: AI Engineering
Recurrent Neural Networks II

LTI/SCS

Spring 2020

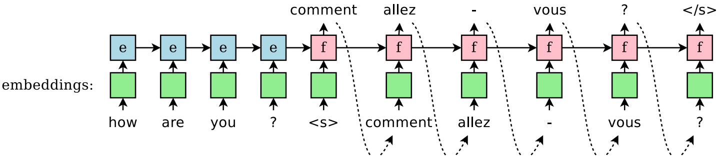
Outline

Take Seq2Seq as an exemplary example



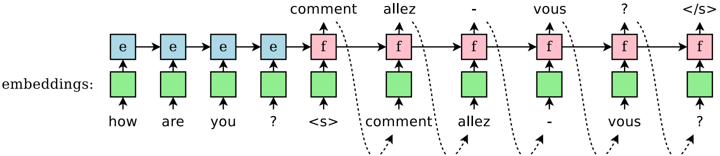
- We have a sequence of hidden vectors
  - In general: $\mathbf{h}_i \in \mathbf{R}^H$ for any input sequences
  - In this case: $\mathbf{e}_i, \mathbf{f}_j \in \mathbf{R}^H$ are the blue and red states
- Can hook up softmax heads to these $\mathbf{e}_i$ and $\mathbf{f}_j$ to make predictions.
- How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

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 saved dictionaries
  - $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4, \mathbf{y}_5, \mathbf{y}_6 \in \mathbb{R}^D$
- **Recurrent Computations:** f is your chosen RNN function such as LSTM or GRU. What are the shapes of each output?
  - Encoder: $\mathbf{e}_0 = 0; \mathbf{e}_t = f(\mathbf{x}_t, \mathbf{e}_{t-1})$
  - Decoder: $\mathbf{f}_0 = \mathbf{e}_{\text{last}}; \mathbf{f}_t = f(\mathbf{y}_t, \mathbf{f}_{t-1})$

Loss Function



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

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

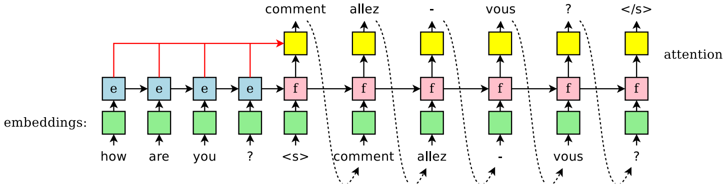
$$p(\mathbf{y}|\mathbf{x}) = \prod_{t=2}^{|\mathbf{y}|} p(y_t | y_{<t}, \mathbf{x}) \tag{2}$$

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

$$\mathcal{L} = -\mathbf{y} \log \hat{\mathbf{y}} = -\sum_{t=1}^{|\hat{\mathbf{y}}|} y_t \log \hat{y}_t \tag{3}$$

Note: *Loss of one sample (sentence) is the sum of all steps.*

Attention: Important Technique



- **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*

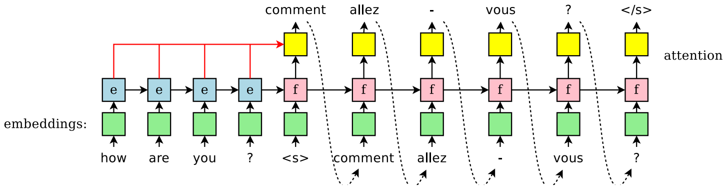
$$p(y_t | y_{<t}, \mathbf{x}) = \text{Softmax}(\mathbf{f}_{t-1} \cdot \mathbf{W}_{\text{soft}}), \text{ for } t = 2, 3, ..., |\mathbf{y}| \tag{4}$$

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

$$p(y_t | y_{<t}, \mathbf{x}) = \text{Softmax}(a(\mathbf{f}_{t-1}, \mathbf{e}_{1 \cdots |\mathbf{x}|}) \cdot \mathbf{W}_{\text{soft}}) \tag{5}$$

Attention

CarnegieMellon



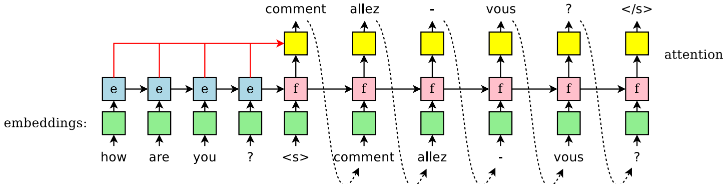
- **Predictions:** now *with attention*

$$p(y_t|y_{<t},\mathbf{x}) = \text{Softmax}(\mathbf{a}(\mathbf{f}_{t-1}, \mathbf{e}_{1\ldots|\mathbf{x}|}) \cdot \mathbf{W}_{\text{soft}}) \quad (6)$$

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

$$\alpha_i = g(\mathbf{f}, \mathbf{e}_i);\ s = \text{Softmax}(\alpha_{1\ldots|\mathbf{x}|});\ a(\mathbf{f}, \mathbf{e}_{1\ldots|\mathbf{x}|}) = \sum_{i=1}^{|\mathbf{x}|} s_i \mathbf{e}_i \quad (7)$$

Attention



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

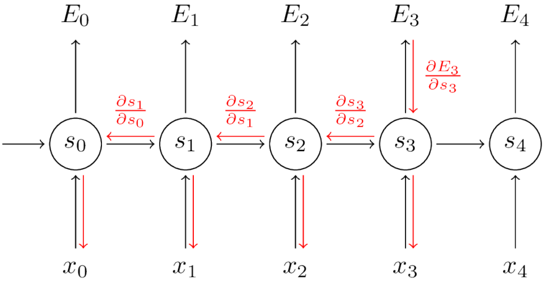
$$\alpha_i = g(\mathbf{f}, \mathbf{e}_i);\ \ s = \mathrm{Softmax}(\alpha_{1\ldots|\mathbf{x}|});\ \ a(\mathbf{f}, \mathbf{e}_{1\ldots|\mathbf{x}|}) = \sum_{i=1}^{|\mathbf{x}|} s_i \mathbf{e}_i$$

- Choices of g:
  - Bahdanau attention[1]: $g(\mathbf{f}, \mathbf{e}_i) = \tanh\left(\mathbf{f} \cdot \mathbf{w}_f + \mathbf{e}_{i-1} \cdot \mathbf{w}_e\right) \cdot \mathbf{v}$, 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[2]: $g(\mathbf{f}, \mathbf{e}_i) = \mathbf{f} \cdot \mathbf{e}_i^\top$ (dot type)

[1] https://arxiv.org/pdf/1409.0473.pdf

[2] https://arxiv.org/pdf/1508.04025.pdf

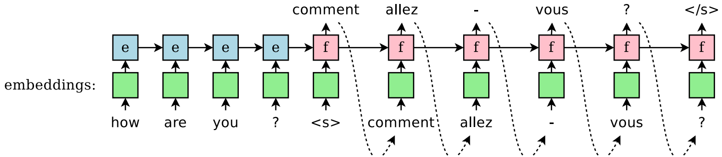
Backprop Through Time (BPTT)



- Even with attention, the overall RNN is fundamentally the same
- Training is called BPTT but it's basically Backprop with:
  - Gradient of a weight is a sum of all timesteps' gradients
  - It's the same if we follow the chain rule

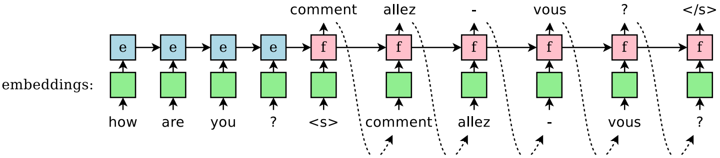
Image credit: Denny Britz

Training Summary



- We have defined *a computational graph*
  - which is a composition of RNN functions

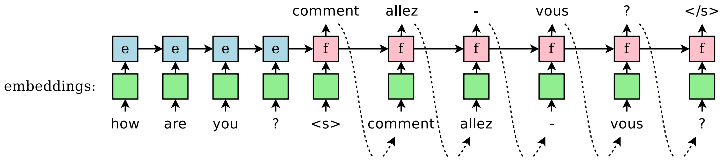
Training Summary



- We have defined *a computational graph*
  - which is a composition of RNN functions
- Thus we can use back-propagation to compute the gradients
  - which is just the chain rule (yet it's called BPTT)

Training Summary

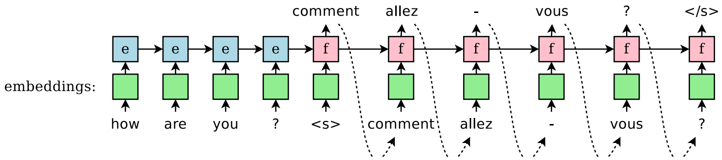
Carnegie Mellon



- We have defined *a computational graph*
  - which is a composition of RNN functions
- Thus we can use back-propagation to compute the gradients
  - which is just the chain rule (yet it's called BPTT)
- Model parameters consist of:
  - Word embedding params
  - $\mathbf{W}_{\text{soft}}$
  - Any parameters of the recurrent function $\mathbf{f}$

Training Summary

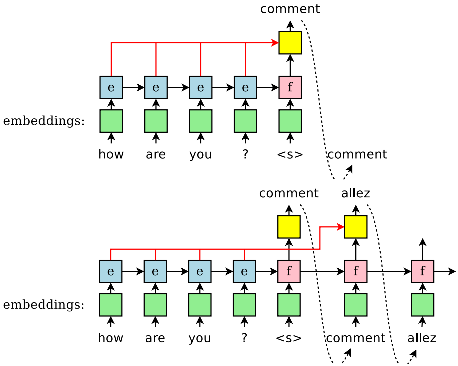
CarnegieMellon



- We have defined *a computational graph*
  - which is a composition of RNN functions
- Thus we can use back-propagation to compute the gradients
  - which is just the chain rule (yet it's called BPTT)
- Model parameters consist of:
  - Word embedding params
  - $\mathbf{W}_{\text{soft}}$
  - Any parameters of the recurrent function $\mathbf{f}$
- During training, we feed ground truth to guide (teacher-forcing)

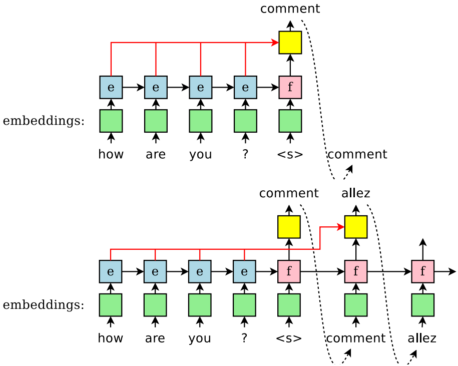
Outline

How to Translate with a Trained RNN?



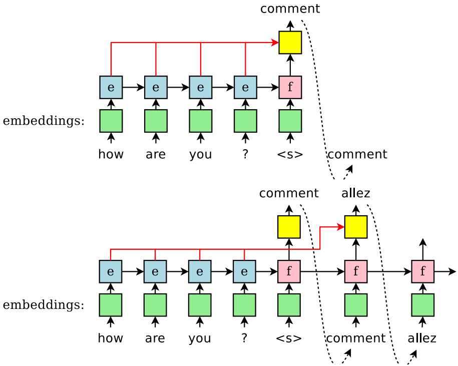
- Goes step-by-step, based on *your own predictions*

How to Translate with a Trained RNN?



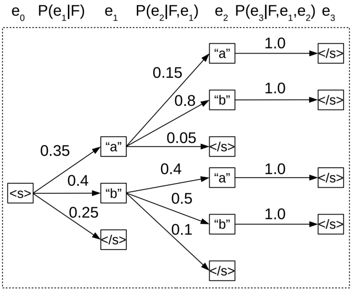
- Goes step-by-step, based on *your own predictions*
- Can we use bidirectional RNNs for encoder?

How to Translate with a Trained RNN?



- Goes step-by-step, based on *your own predictions*
- Can we use bidirectional RNNs for encoder?
- How about decoder?

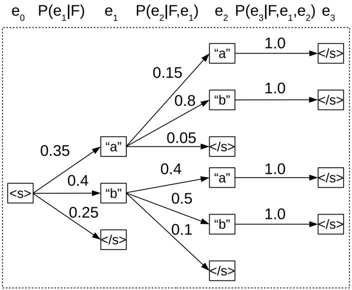
Decoding method: Greedy



- Always take top 1

Image credit: Graham Neubig

Decoding method: Greedy

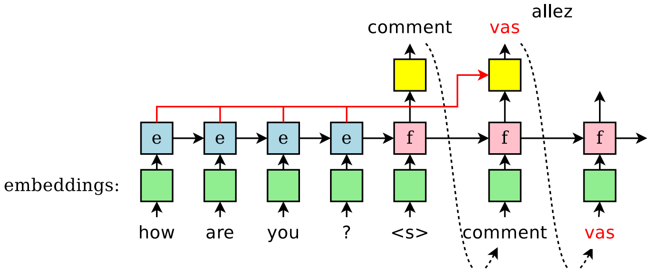


- Always take top 1
- But fast starter might be a straggler later

Image credit: Graham Neubig

What If You Are Wrong?

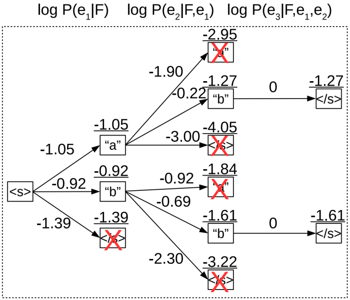
CarnegieMellon



- You live with your mistakes
- Or have other methods

Decoding method: Beam Search

Carnegie Mellon

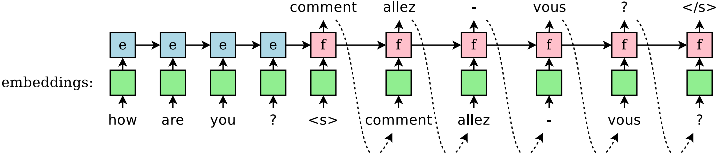


- Beam search: maintain multiple top paths
  - Canonical method in decoding
  - Keep `top_k` with $k > 1$ at every step
- Greedy is a special case where $k = 1$

Image credit: Graham Neubig

Summary: Training and Evaluation

Carnegie Mellon



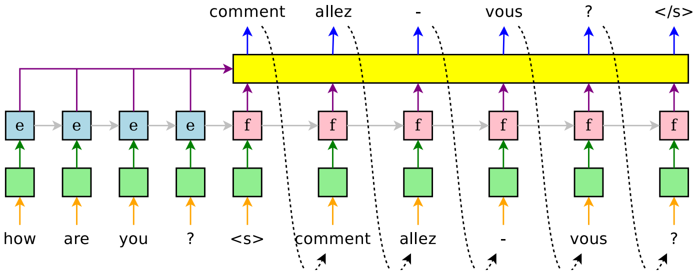
- Training with Teacher-Forcing
  - Encoder: directly use the encoder, simple!
  - Decoder: using "teacher" mode, for *every* token
- Evaluation
  - No teacher anymore
  - Collect attention weights: $|\mathbf{f}| \times |\mathbf{e}|$ and plot them if needed

Outline

1. Neural Machine Translation: Training
2. Neural Machine Translation: Testing
3. Regularization
4. Resources

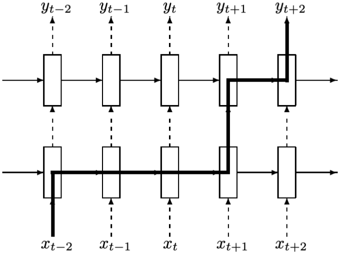
General Strategy: Dropout

Carnegie Mellon



- Each colored represents each possibility to be “dropped-out”
  - Word embeddings dropout mean to remove *the whole word*
- Input, output, embedding (vertical) or recurrent (horizontal)?

Non-recurrent Dropout[3]

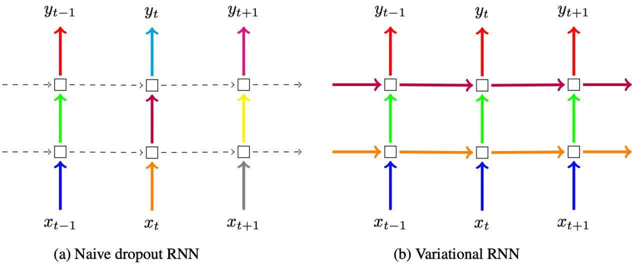


- Only apply to non-recurrent to leave RNN "memory" intact
- Works, but not so well ...

[3] https://arxiv.org/pdf/1409.2329.pdf

Image credit: Wojciech Zaremba *et al.*

Variational Dropout[4]



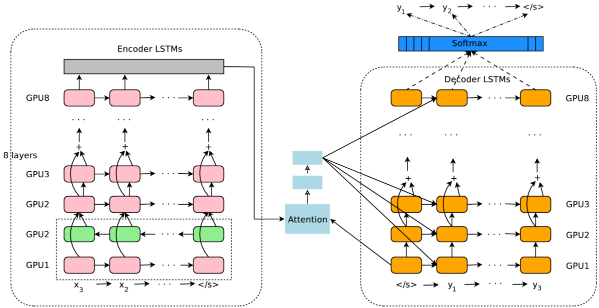
- To recurrent connections as well, shown better than naive way
- Same mask for input, output and recurrent connections at each time step
- This dropout is shown to be similar to variational appx (Bayesian interpretation)

[4] https://arxiv.org/pdf/1512.05287.pdf

Image credit: Wojciech Zaremba *et al.*

Outline

Example: Google Neural Machine Translation[5]



- Enormous, distributed, 1024 nodes for all RNN layers
- First 2 layers: bidirectional RNNs
- 4th - 8th layers: residual connections added
- Attention network: single hidden FC 1024

[5] https://arxiv.org/pdf/1609.08144v2.pdf

Resources

- API: tf.keras.layers.LSTM
- API: tf.keras.layers.GRU
- Tutorial: Build RNNs with tf.keras
- Tutorial: Time series forecasting with RNNs
- Tutorial: Text classification with RNNs
- **Tutorial**: NMT with Attention