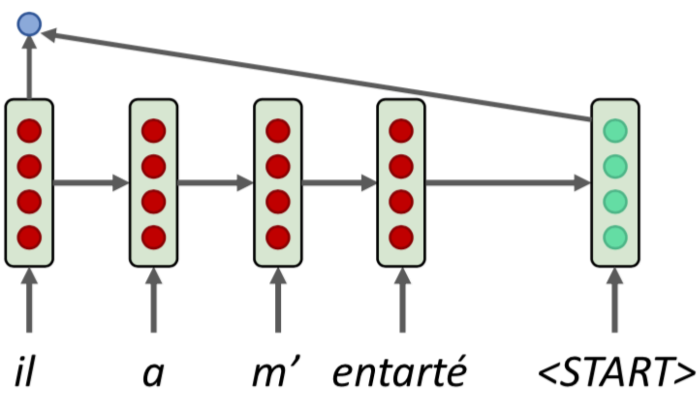
Deep Learning and Applications

Attention

- Compute attention scores given some function α
  - Encoder hidden state h_1
  - Decoder hidden state s_1
  - → Attention score $a_{11} = f(h_1, s_1)$

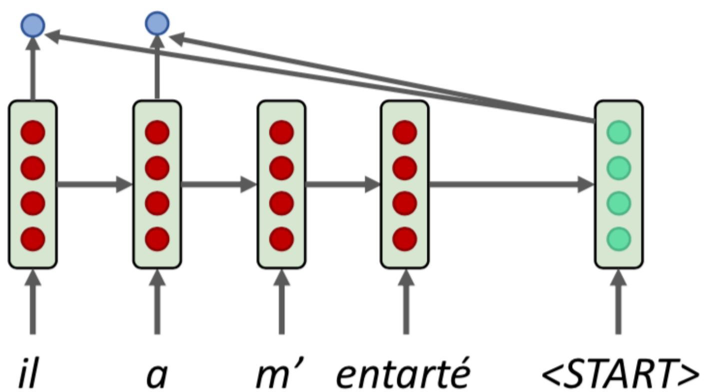


[1]Figure: Moontae Lee, UIC

Attention

- Compute attention scores given some function α
  - Encoder hidden state h_2
  - Decoder hidden state s_1

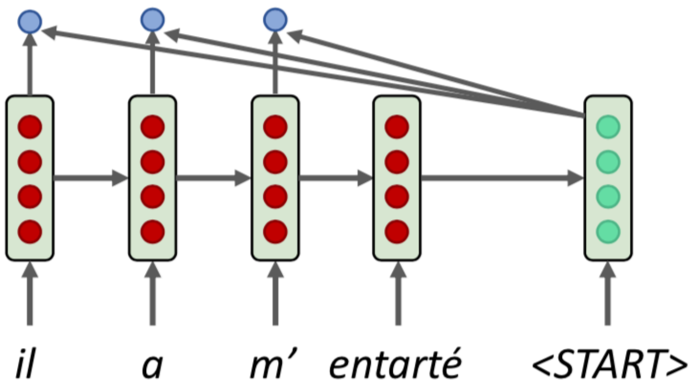
  → Attention score $a_{12} = f(h_2, s_1)$



[1]Figure: Moontae Lee, UIC

Attention

- Compute attention scores given some function α
  - Encoder hidden state h_3
  - Decoder hidden state s_1
  - → Attention score $a_{13} = f(h_3, s_1)$

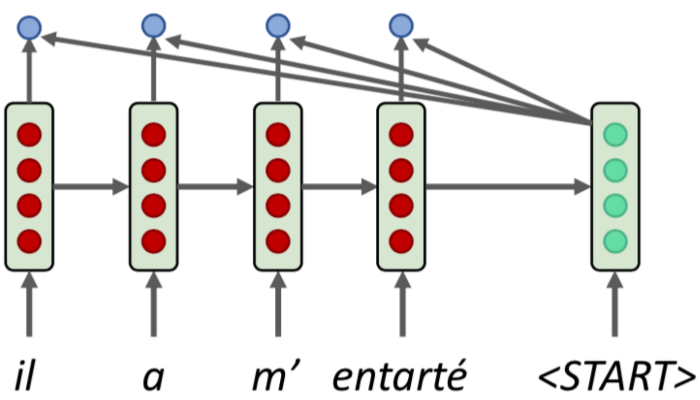


[1]Figure: Moontae Lee, UIC

Attention

- Compute attention scores given some function α
  - Encoder hidden state h_4
  - Decoder hidden state s_1

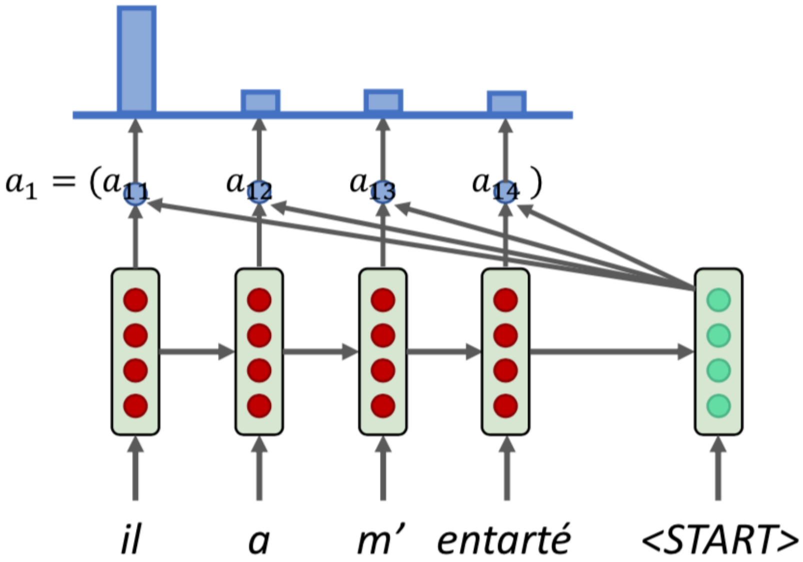
  → Attention score $a_{14} = f(h_4, s_1)$



[1]Figure: Moontae Lee, UIC

Attention

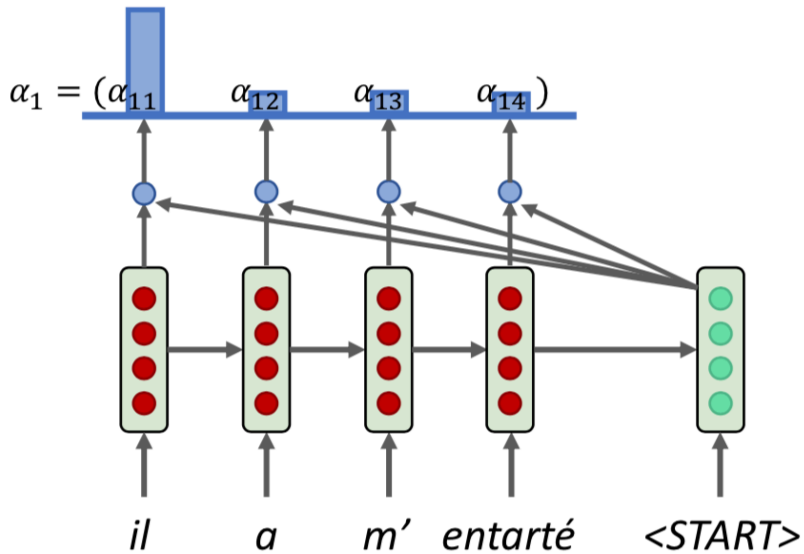
- Convert attention scores into a distribution by softmax.



[1]Figure: Moontae Lee, UIC

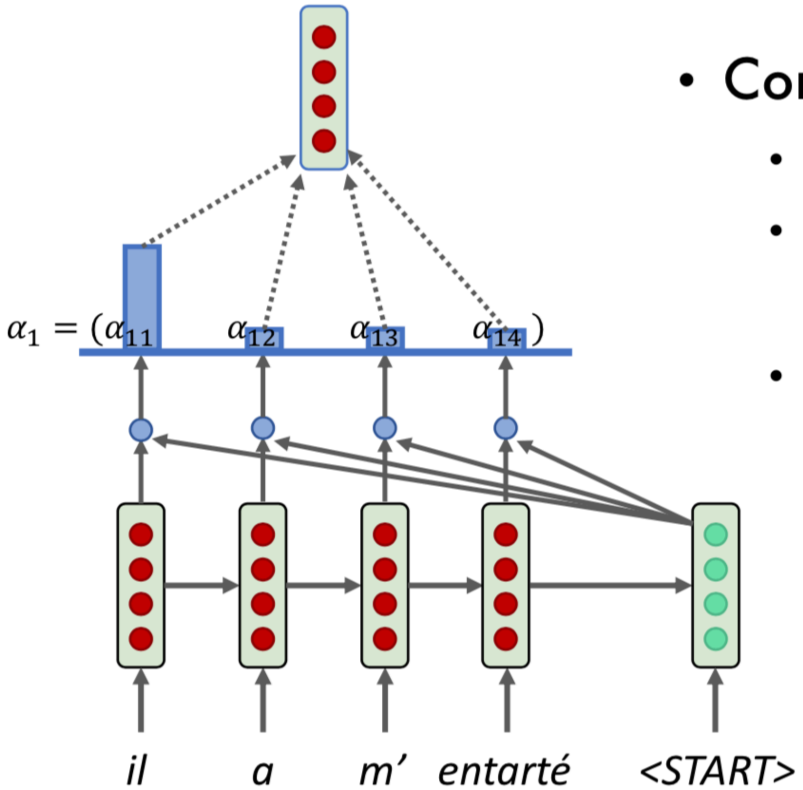
Attention

- Convert attention scores into a distribution by softmax.
  - $\alpha_1 = softmax(a_1)$
  - We are mostly focusing on the encoder's 1st hidden state h_1.



[1]Figure: Moontae Lee, UIC

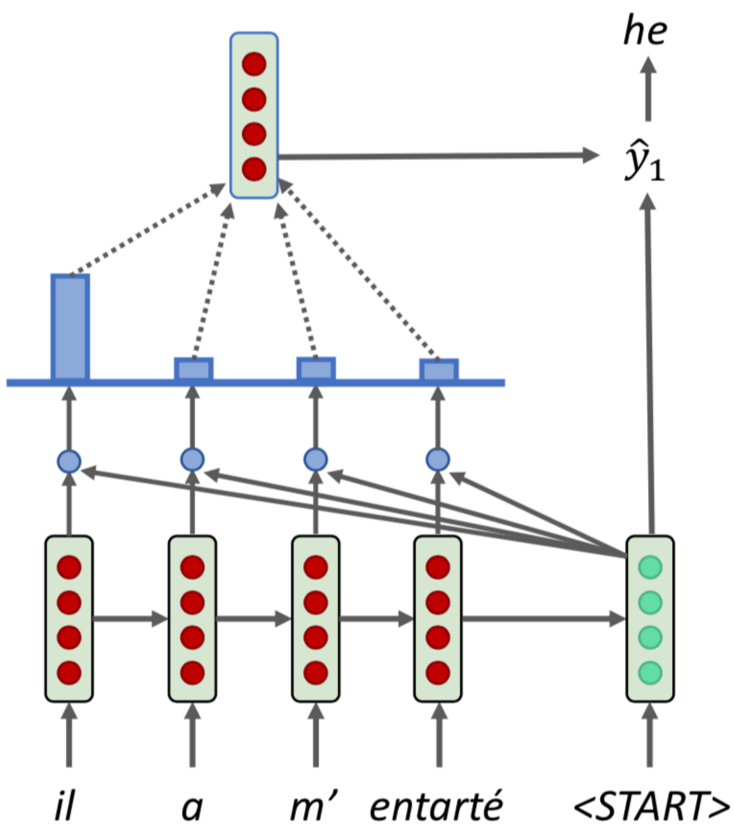
Attention



- Compute the attention output z_1.
  - Weighted sum of hidden states.
  - $z_1 = \sum_{t=1}^{4} \alpha_{1t} h_t$
  - Attention output contains information of every hidden state proportionally to attention distribution α.

[1]Figure: Moontae Lee, UIC

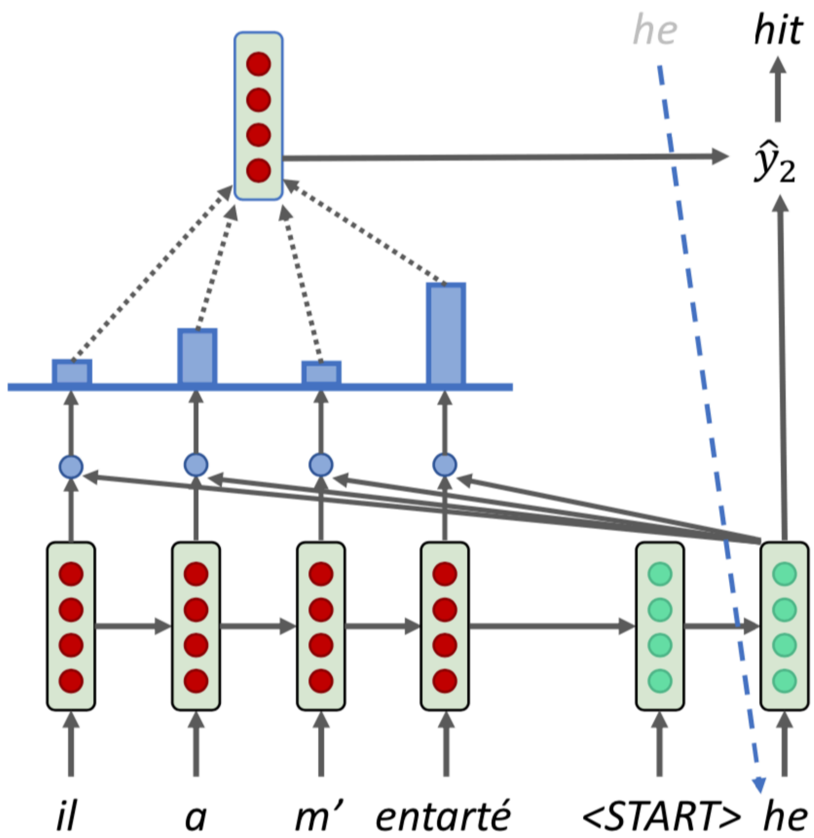
Attention



- Concatenate then generate
  - $[z_t; s_t] \in \mathbb{R}^{2d}$
  - $\hat{y}_t = softmax(g([z_t; s_t]))$

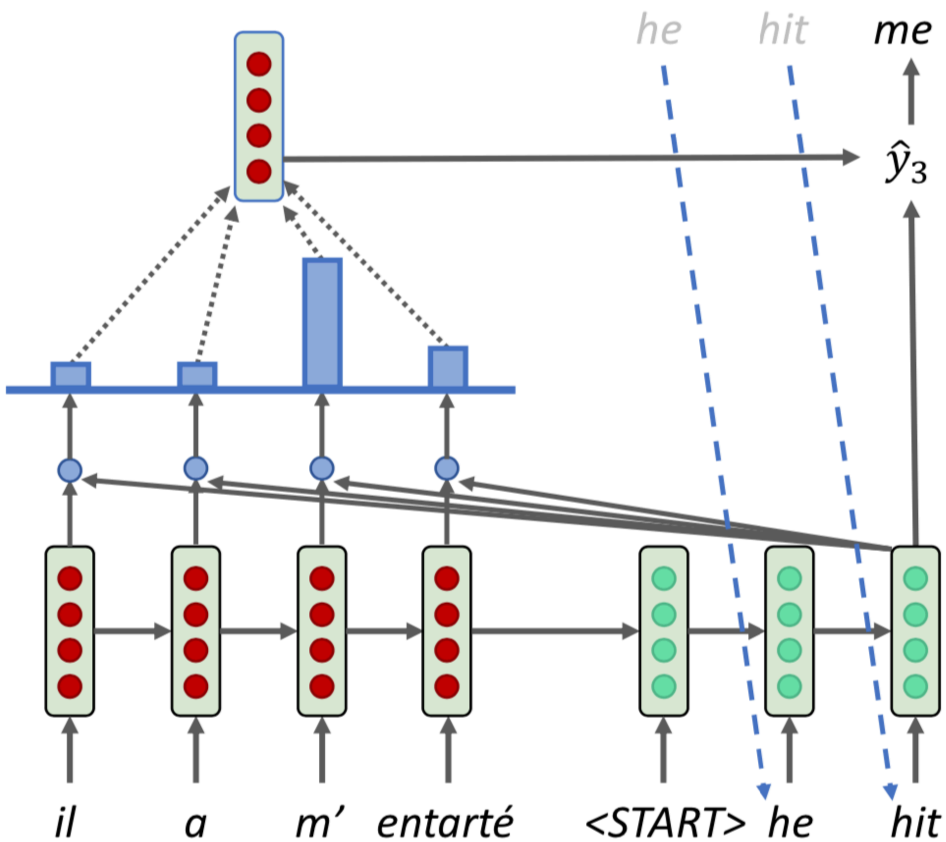
[1]Figure: Moontae Lee, UIC

Attention



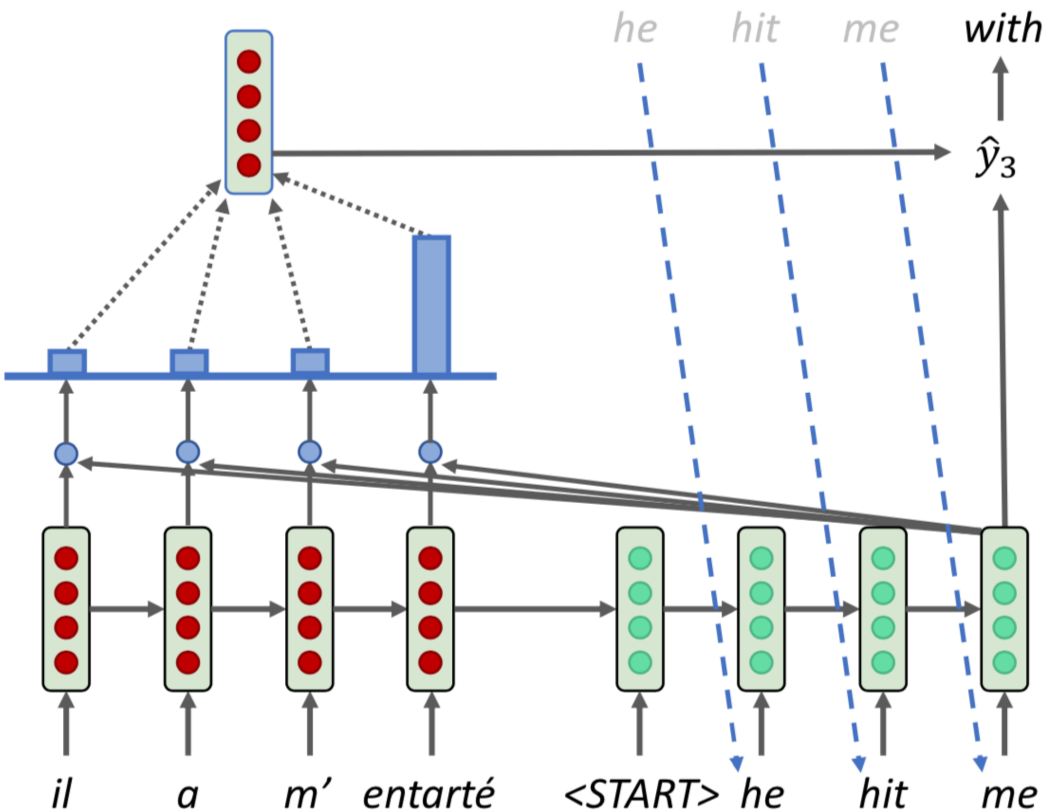
[1]Figure: Moontae Lee, UIC

Attention



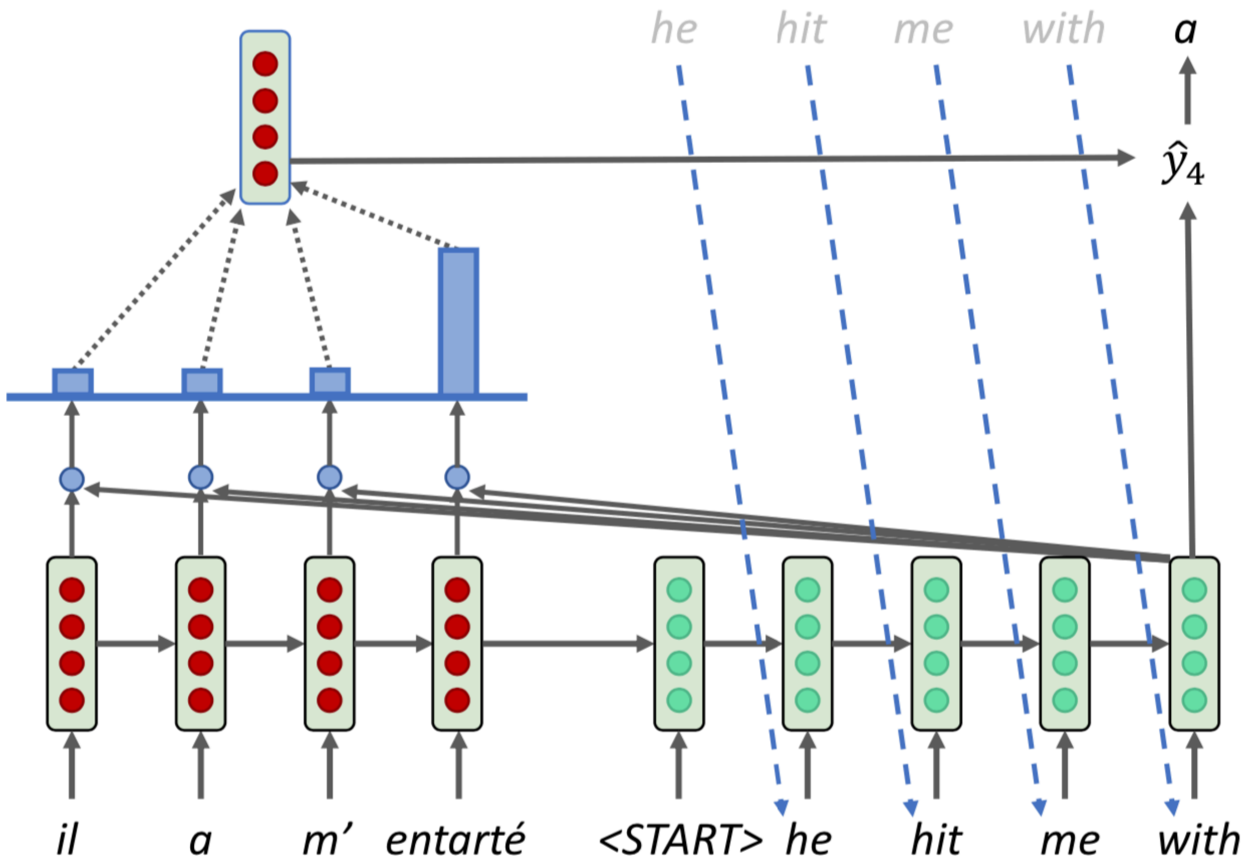
[1]Figure: Moontae Lee, UIC

Attention



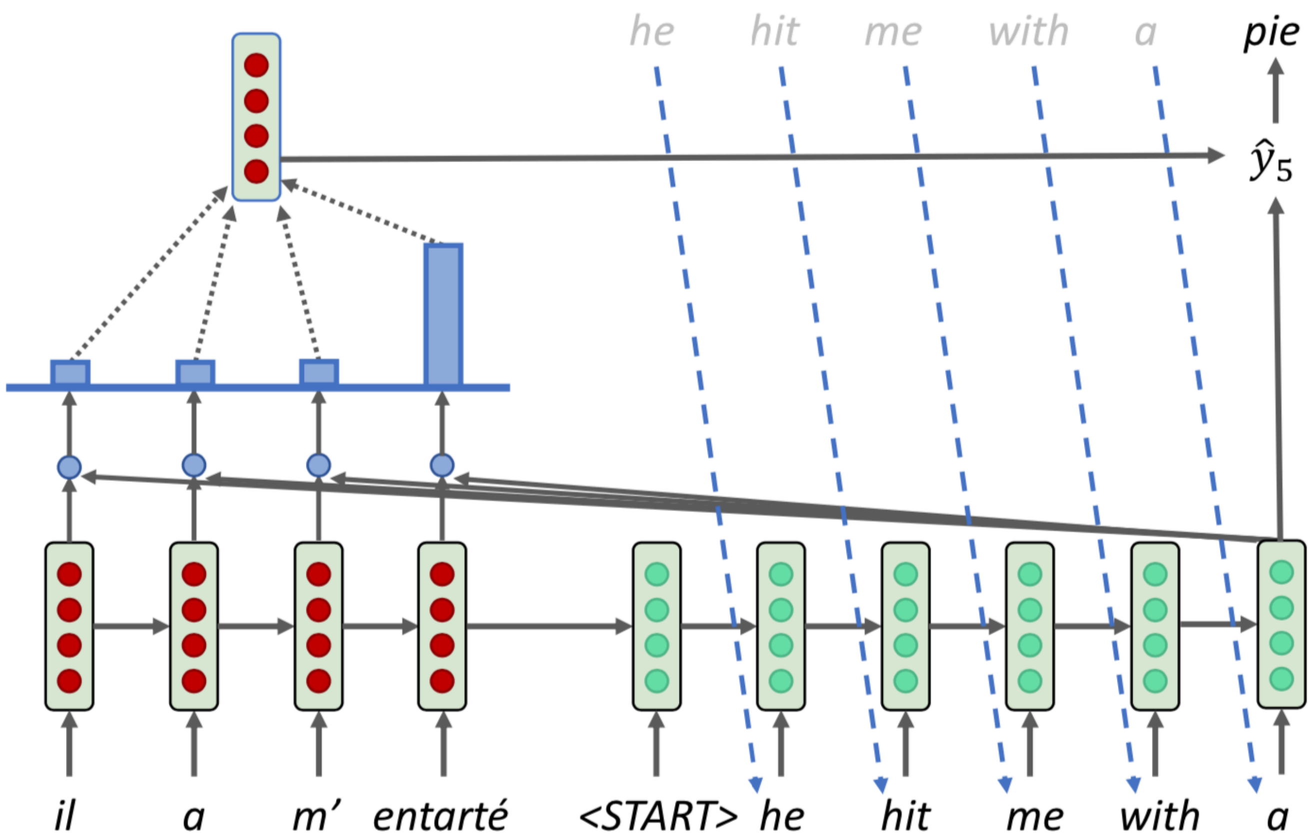
[1]Figure: Moontae Lee, UIC

Attention



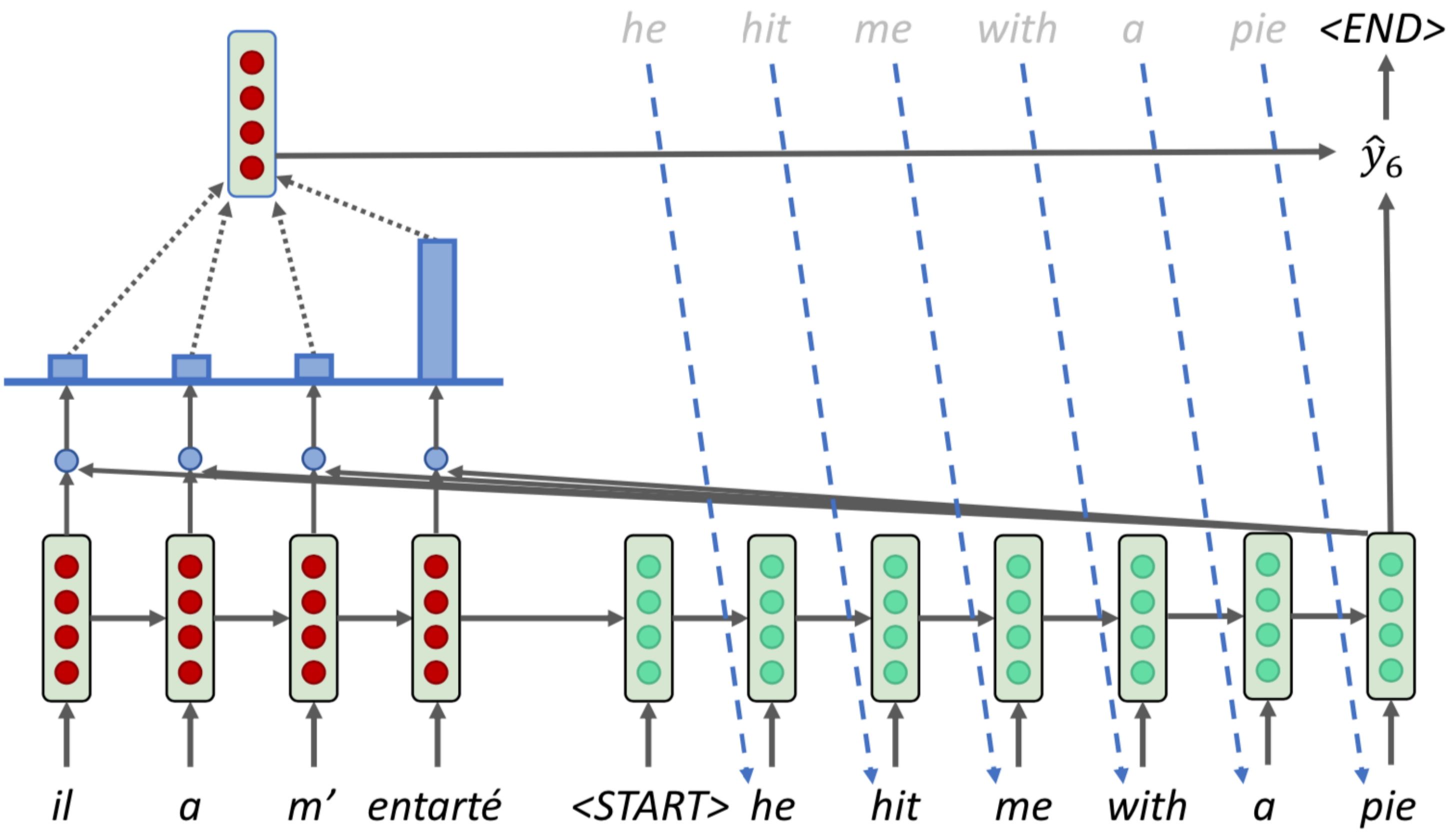
[1]Figure: Moontae Lee, UIC

Attention



[1]Figure: Moontae Lee, UIC

Attention



[1]Figure: Moontae Lee, UIC

Attention

- For each time t,
  - Decoder hidden state $s_t \in \mathbb{R}^d$.
  - For every encoder hidden state $h_1, \dots, h_T$,

[1]Figure: Moontae Lee, UIC

Attention

- For each time t,
  - Decoder hidden state $s_t \in \mathbb{R}^d$.
  - For every encoder hidden state $h_1, \dots, h_T$,
    - Compute attention scores $a_t = (a_{t1}, \dots, a_{tT})$ where $a_{tu} = f(h_u, s_t)$.

[1]Figure: Moontae Lee, UIC

Attention

- For each time t,
  - Decoder hidden state $s_t \in \mathbb{R}^d$.
  - For every encoder hidden state $h_1, \dots, h_T$,
    - Compute attention scores $a_t = (a_{t1}, \dots, a_{tT})$ where $a_{tu} = f(h_u, s_t)$.
    - Convert a_t to attention distribution $\alpha_t = (\alpha_{t1}, \dots, \alpha_{tT}) = softmax(a_t)$.
    - Get the weighted encoder state $z_t = \sum_{u=1}^{T} \alpha_{tu} h_u$

    - Vertically concatenate the states: $[z_t; s_t] \in \mathbb{R}^{2d}$
    - Predict an output (as a distribution): $\hat{y}_t = softmax(g([z_t; s_t])$

[1]Figure: Moontae Lee, UIC

Attention

- **Basic dot-product attention:** $\alpha_{tu} = f(h_u, s_t\ ) = s_t^T h_u$
  - Assume $\dim(s_t) = \dim(h_u)$.
  - Nothing to learn for f

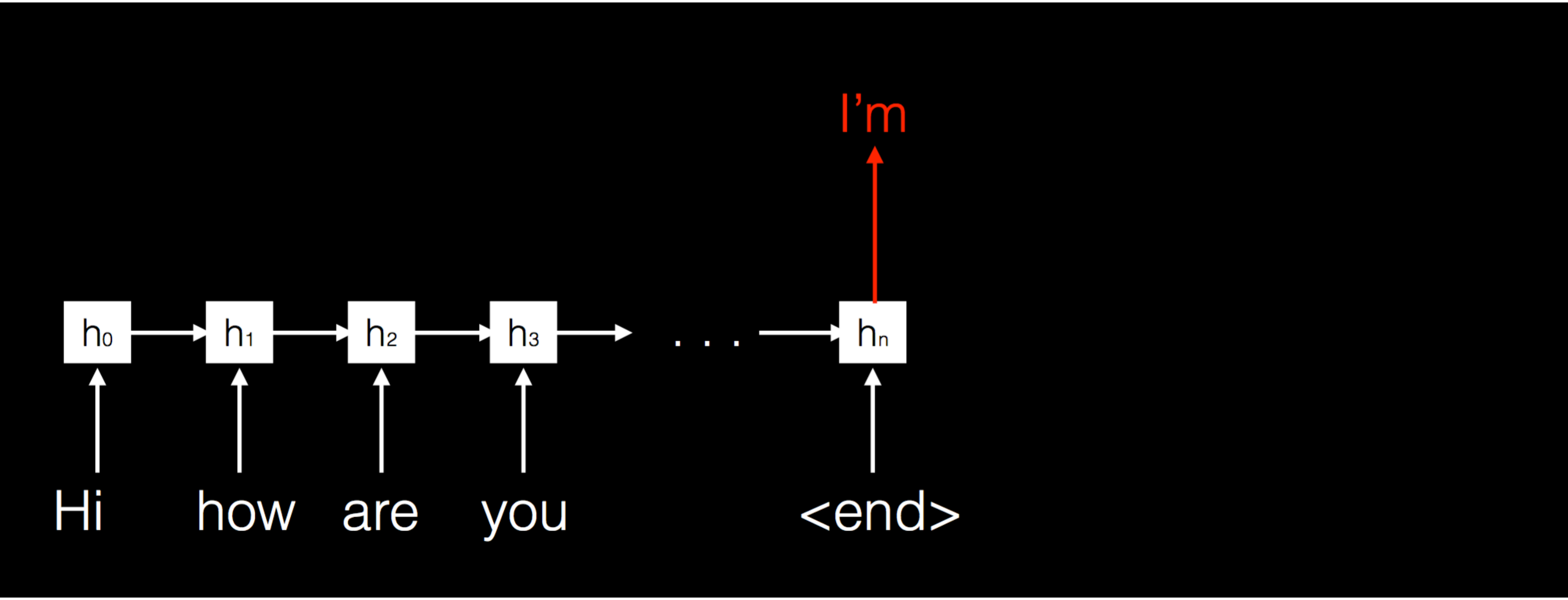
- **Multiplicative attention:** $\alpha_{tu} = f(h_u, s_t,\ ) = s_t^T W h_u$
  - Say $d_1 = \dim(h_u)\ , d_2 = \dim(s_t)$.
  - Then we should learn $W \in \mathbb{R}^{d_2 \times d_1}$ from the training data.

- **Additive attention:** $a_{tu} = f(h_u, s_t) = v^T \tanh(W_h h_u + W_s s_t)$.
  - Say $d_3 = \dim(v)$, which will be a new user hyper-parameter.
  - Then we should learn $W_h \in \mathbb{R}^{d_3 \times d_1}, W_s \in \mathbb{R}^{d_3 \times d_2}$, and $v \in \mathbb{R}^{d_3}$.

[1]Figure: Moontae Lee, UIC

Example III (Extension): Auto-Reply

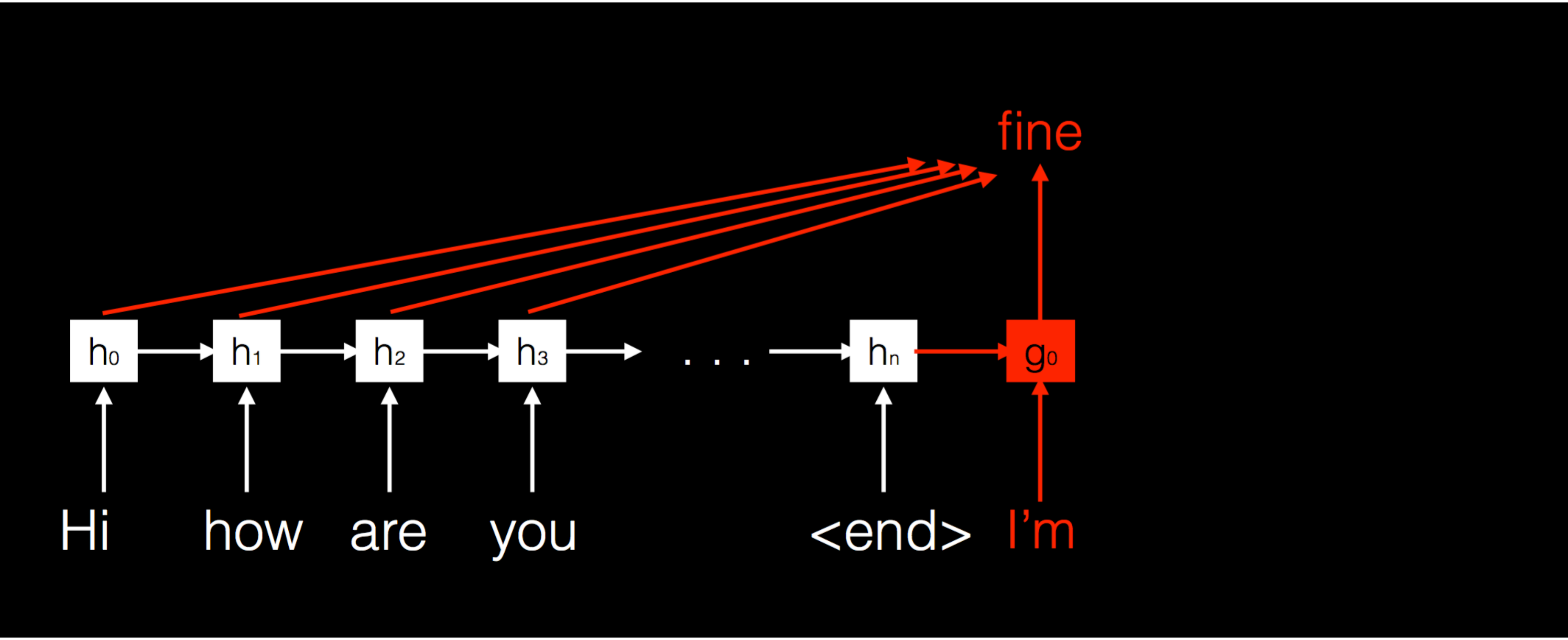
- Third version: Attention Mechanism
- Ideally output could consider ‘attention’ to parts of history



[1]Figure: Quoc Le, Google Brain

Example III (Extension): Auto-Reply

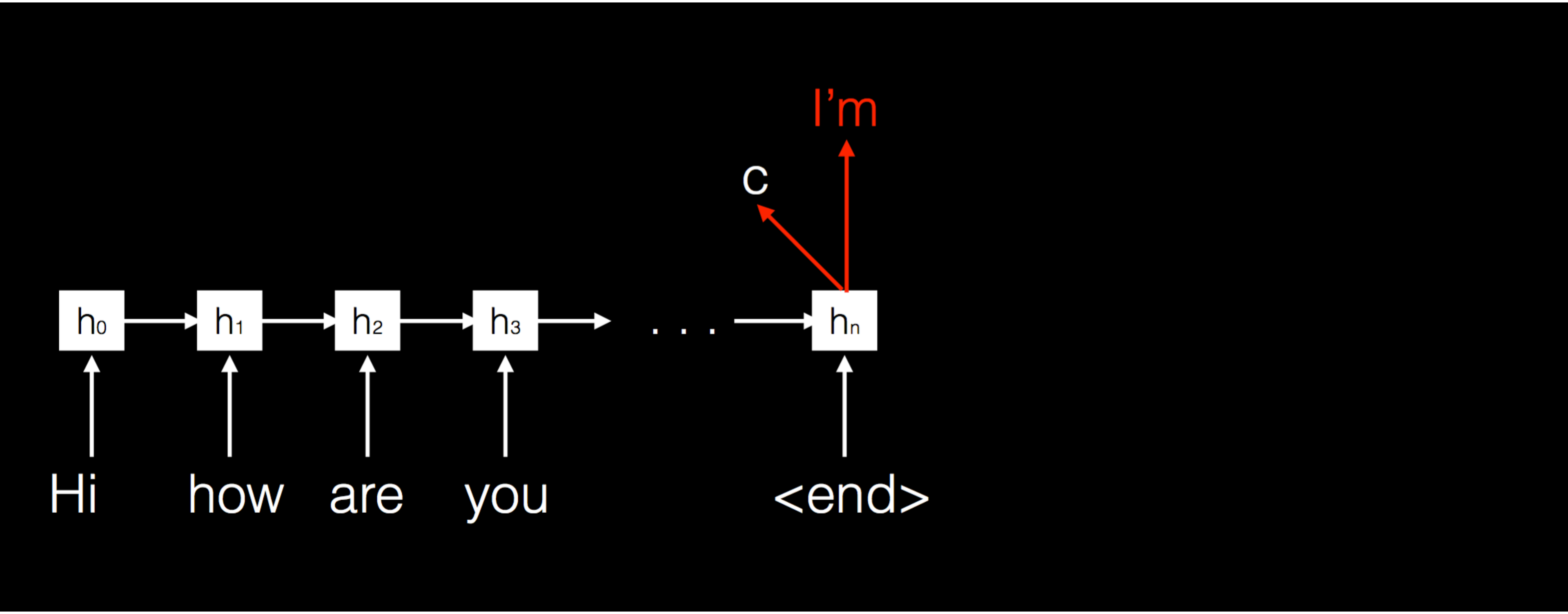
- Could look at every state in the past



[1]Figure: Quoc Le, Google Brain

Example III (Extension): Auto-Reply

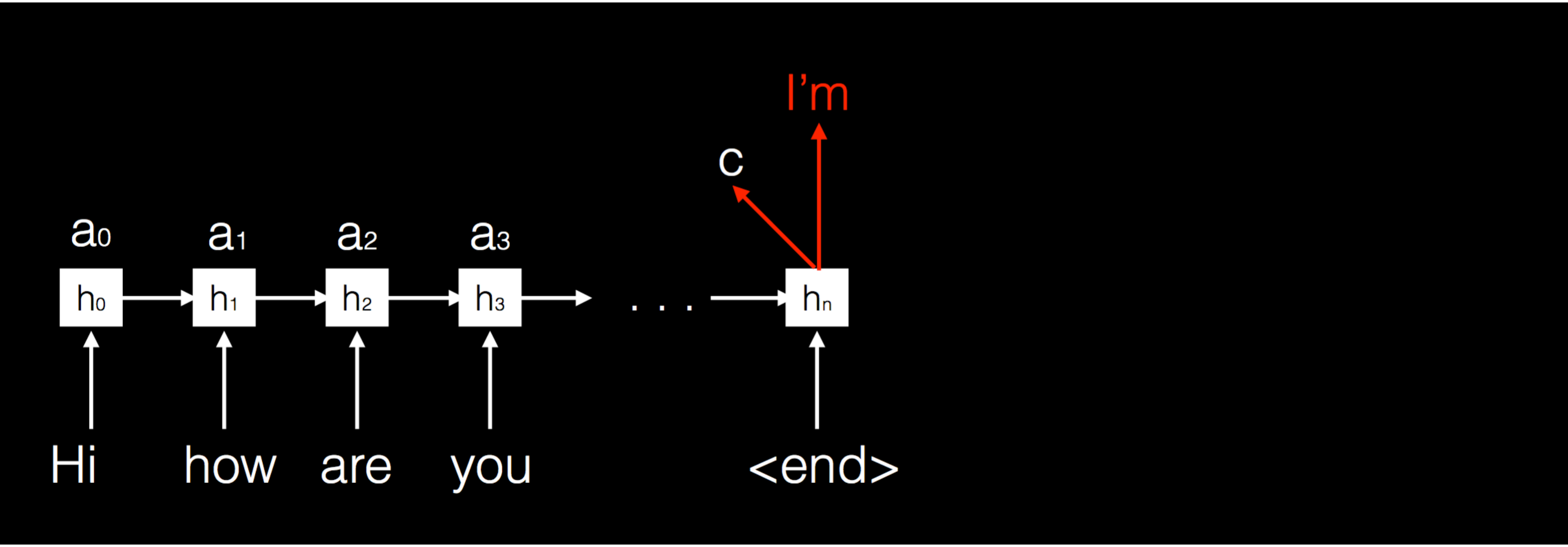
- So instead of returning a word, output the current state



[1]Figure: Quoc Le, Google Brain

Example III (Extension): Auto-Reply

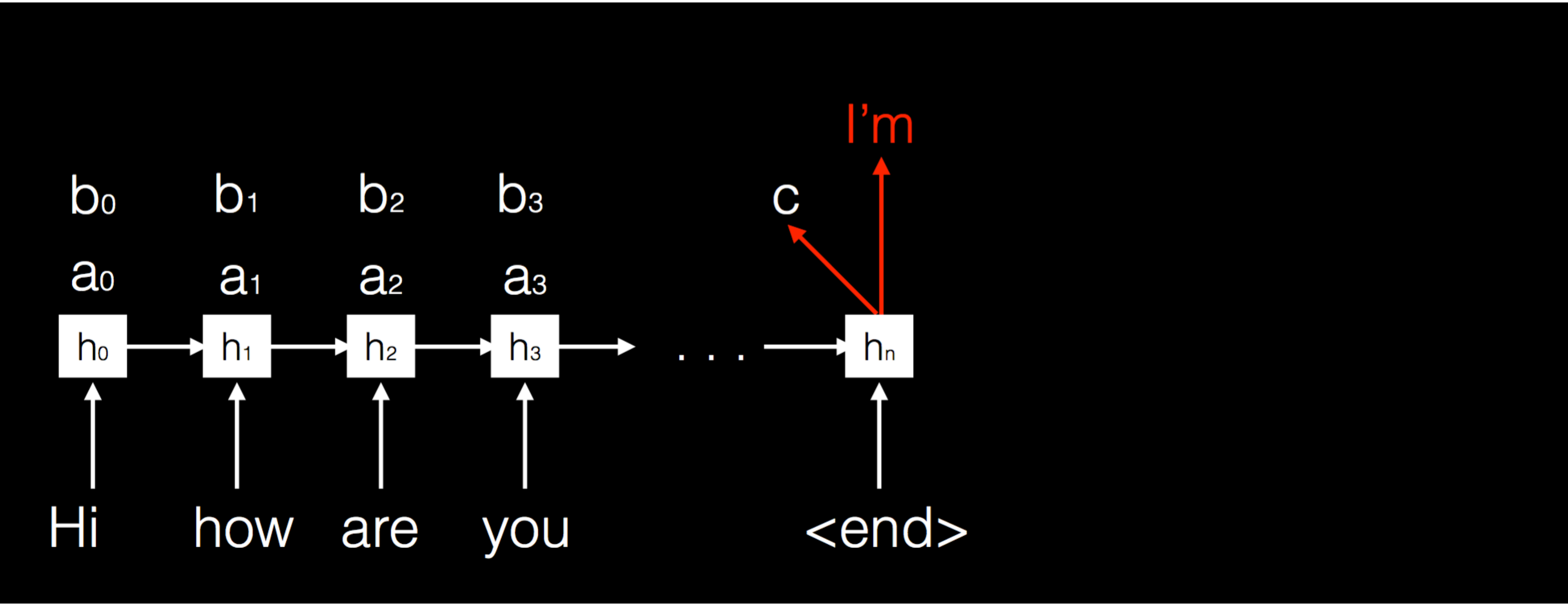
- Take inner products with previous states



[1]Figure: Quoc Le, Google Brain

Example III (Extension): Auto-Reply

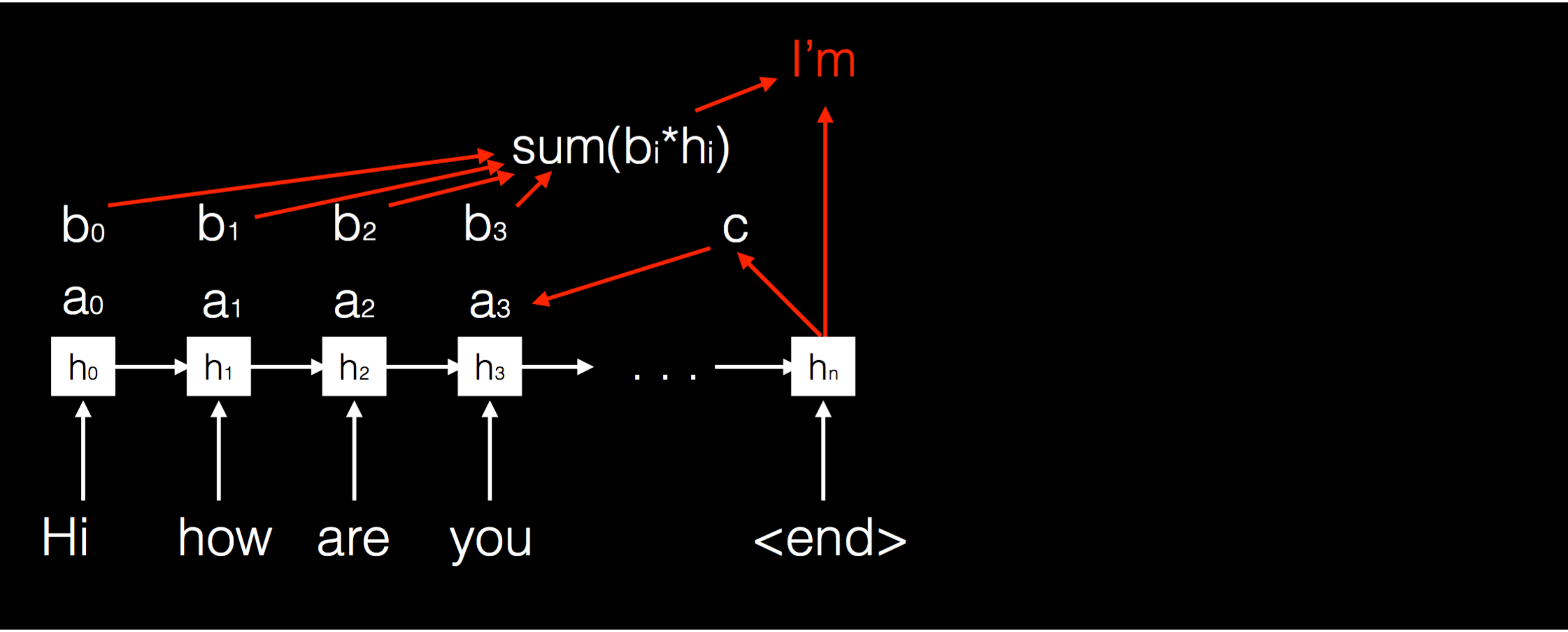
- Take inner products with previous states



[1]Figure: Quoc Le, Google Brain

Example III (Extension): Auto-Reply

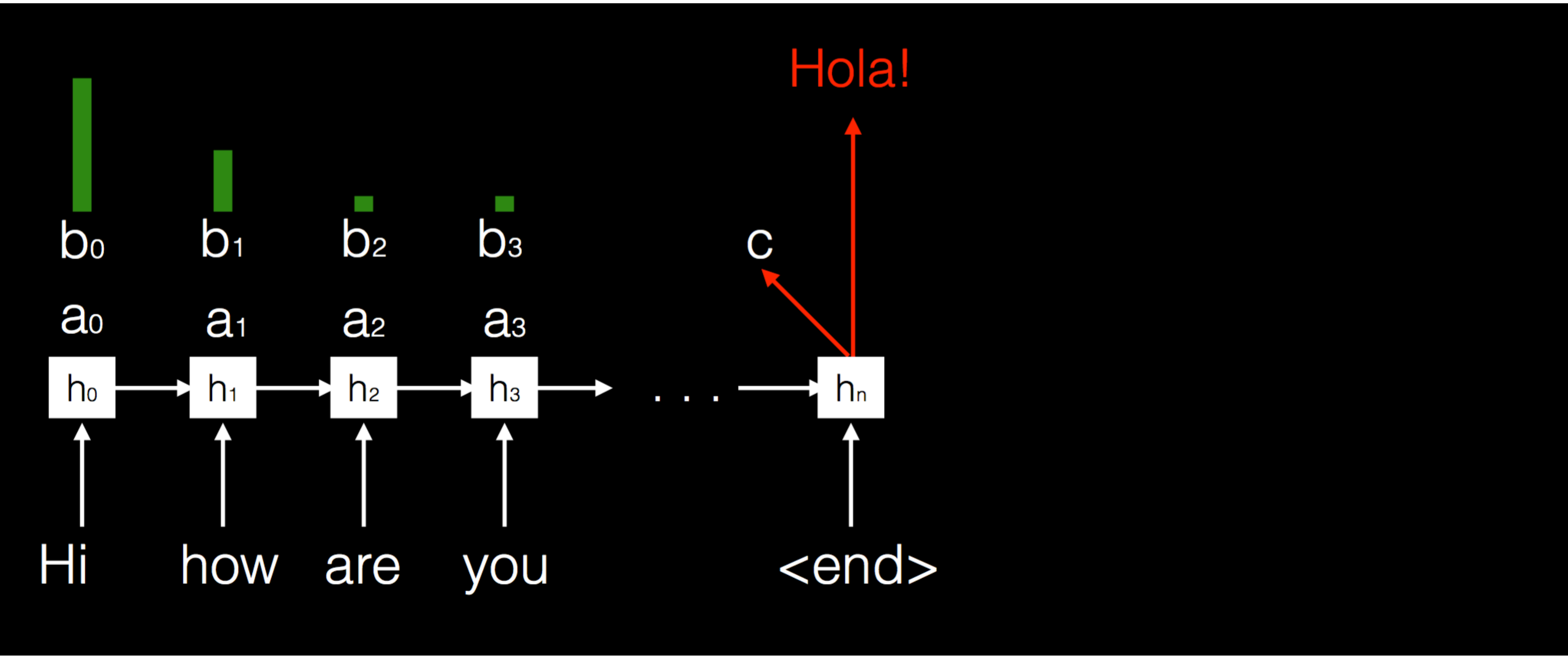
- Pass through a neural net layer to predict final word



[1]Figure: Quoc Le, Google Brain

Example III (Extension): Same with Translation!

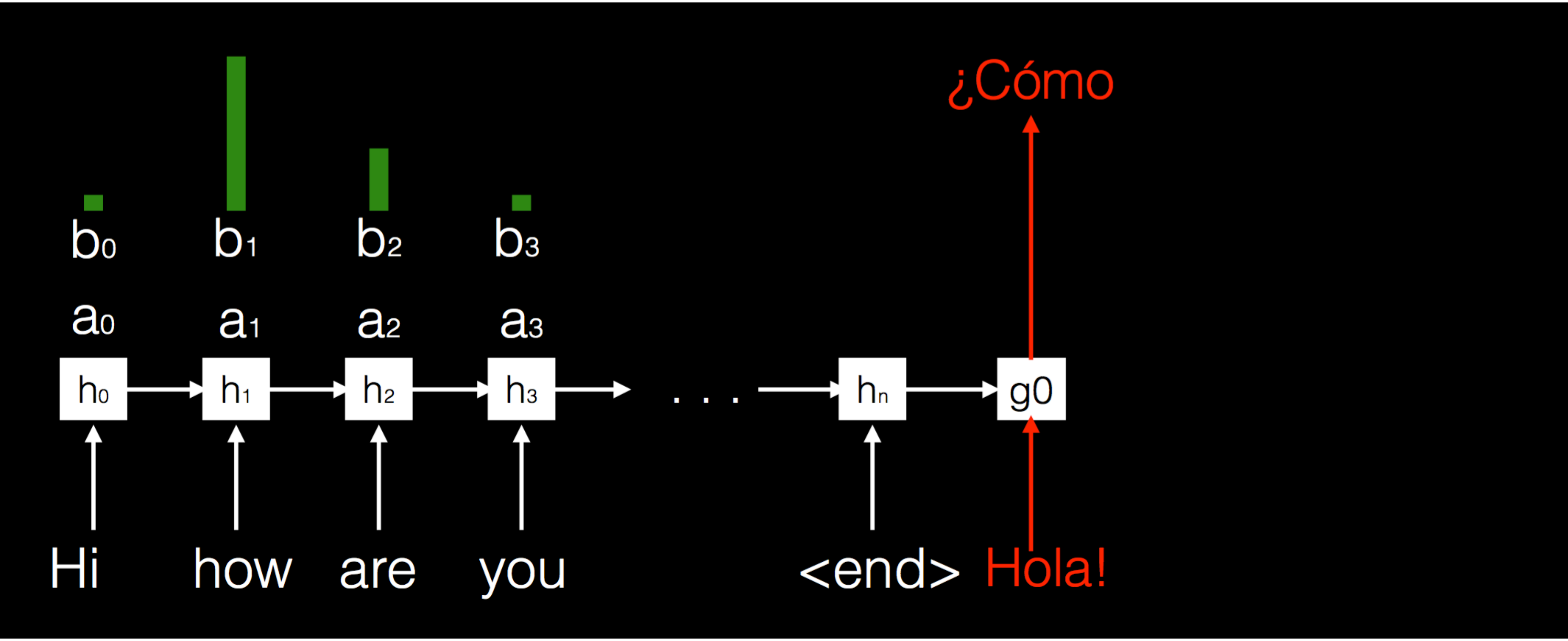
- Same principle also applies for translation. The first prediction learns to focus on certain part of the input



[1]Figure: Quoc Le, Google Brain

Example III (Extension): Auto-Reply

- The second prediction learns to focus on certain part of the input



[1]Figure: Quoc Le, Google Brain