ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Αλληλεπίδραση Ανθρώπου–Υπολογιστή

Β6. Επεξεργασία φυσικής γλώσσας με ανατροφοδοτούμενα νευρωνικά δίκτυα

(2023–24)

Ίων Ανδρουτσόπουλος

http://www.aueb.gr/users/ion/

Contents

- Recurrent neural networks (RNNs), GRUs/LSTMs.
- Applications in token classification (e.g., named entity recognition).
- RNN language models.
- Layer normalization.
- RNNs with self-attention and applications in text classification.
- Bidirectional and stacked RNNs.
- Sequence-to-sequence RNN models with attention, and applications in machine translation.

Extracting contract elements

THIS **AGREEMENT** is made the 15th day of October 2009 (The **"Effective Date"**) BETWEEN:

(1) **Sugar 13 Inc.**, a corporation whose office is at James House, 42-50 Bond Street, London, EW2H TL (**"Sugar"**);

(2) **E2 UK Limited**, a limited company whose registered office is at 260 Bathurst Road, Yorkshire, SL3 4SA (**"Provider"**).

RECITALS:

A. The Parties wish to enter into a framework agreement which will enable Sugar, from time to time, to [...]
B. [...]

NO THEREFORE IT IS AGREED AS FOLLOWS:

ARTICLE I - DEFINITIONS

"Sugar"	shall mean:	Sugar 13 Inc.
"Provider"	shall mean:	E2 UK Limited
"1933 Act"	shall mean:	Securities Act of 1933

ARTICLE II - TERMINATION

The Service Period will be for five (5) years from the Effective Date (The **"Initial Term"**). The agreement is considered to be terminated in October 16, 2014.

ARTICLE III - PAYMENT - FEES

During the service period monthly payments should occur. The estimated fees for the Initial Term are £154,800.

ARTICLE IV - GOVERNING LAW

This agreement shall be governed and construed in accordance with the Laws of England & Wales. Each party hereby irrevocably submits to the exclusive jurisdiction of the courts sitting in Northern London.

IN WITNESS WHEREOF, the parties have caused their respective duly authorized officers to execute this Agreement.

BY: George Fake
Authorized Officer
Sugar 13 Inc.

BY: Olivier Giroux
CEO
E2 UK LIMITED

Identify **start/end dates, duration, contractors, amount, legislations refs, jurisdiction** etc. Similar to **Named Entity Recognition** (NER).

I. Chalkidis, I. Androutsopoulos, A. Michos, "Extracting Contract Elements", ICAIL 2017, http://nlp.cs.aueb.gr/pubs/icail2017.pdf .

Window-based token classification

i-th word of the text being classified

3-word **window** (often larger)

yesterday language tech announced that…

1-hot vectors ($|V|\times 1$) of the **words** in the **window**. ($|V|$ is the **vocabulary** size).

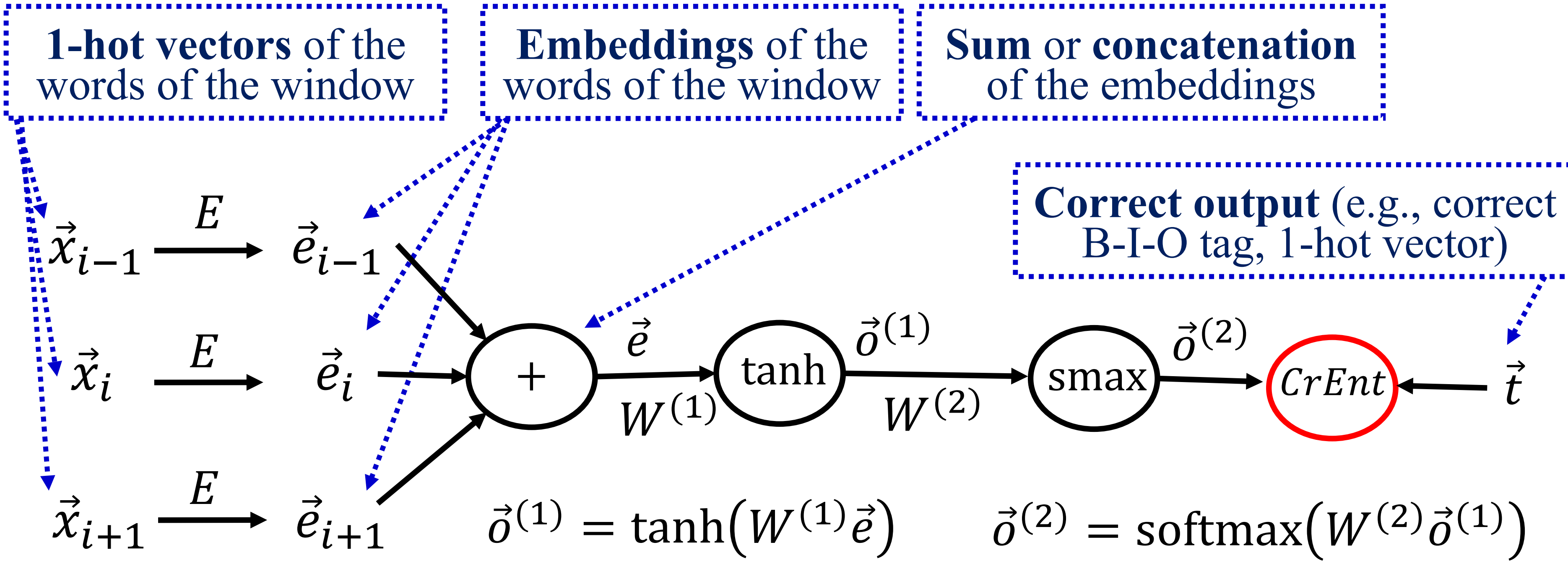
$$\vec{x}_{i-1} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \end{bmatrix} \quad \vec{x}_{i} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \dots \\ 0 \end{bmatrix} \quad \vec{x}_{i+1} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 1 \end{bmatrix}$$

Embeddings ($d\times 1$) of the **words** in the **window**. (d is the **dimensionality** of the word embeddings).

$$\vec{e}_{i-1} = \begin{bmatrix} 1.8 \\ 2.3 \\ -1.4 \\ 3.7 \\ \dots \\ -1.1 \end{bmatrix} \quad \vec{e}_{i} = \begin{bmatrix} 2.4 \\ -3 \\ 9.3 \\ 5.1 \\ \dots \\ 3.9 \end{bmatrix} \quad \vec{e}_{i+1} = \begin{bmatrix} 2.2 \\ 3.8 \\ 1.2 \\ -6.4 \\ \dots \\ 7.1 \end{bmatrix}$$

Let $\boldsymbol{E}$ be a matrix ($d\times|V|$) that **contains** all the **embeddings** of the **vocabulary** as **columns**. Then: $\vec{e}_{i-1} = E\vec{x}_{i-1}$, $\vec{e}_{i} = E\vec{x}_{i}$, …

Window-based token classification



We learn $\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}$ with **backpropagation**. We can also learn (or modify) the **word embeddings *E*** during **backpropagation**! But when we don't have large training datasets (e.g., corpus manually annotated with B-I-O tags), it may be better to use **pre-trained embeddings**, which can be obtained from large non-annotated corpora (e.g., via Word2Vec, GloVe).

We can use the same window-based approach for **POS-tagging**, **chunking**, …

Reminder: cross-entropy loss

Word being **classified**.

3-word window (often larger).

yesterday language tech announced that…

Probability estimates produced **by the classifier** for the class of the word "tech".

$$\vec{o} = \begin{bmatrix} P_m(C = c_1) \\ P_m(C = c_2) \\ P_m(C = c_3) \\ \dots \\ P_m(C = c_k) \end{bmatrix} = \begin{bmatrix} 0.05 \\ 0.12 \\ 0.08 \\ \dots \\ 0.14 \end{bmatrix}$$

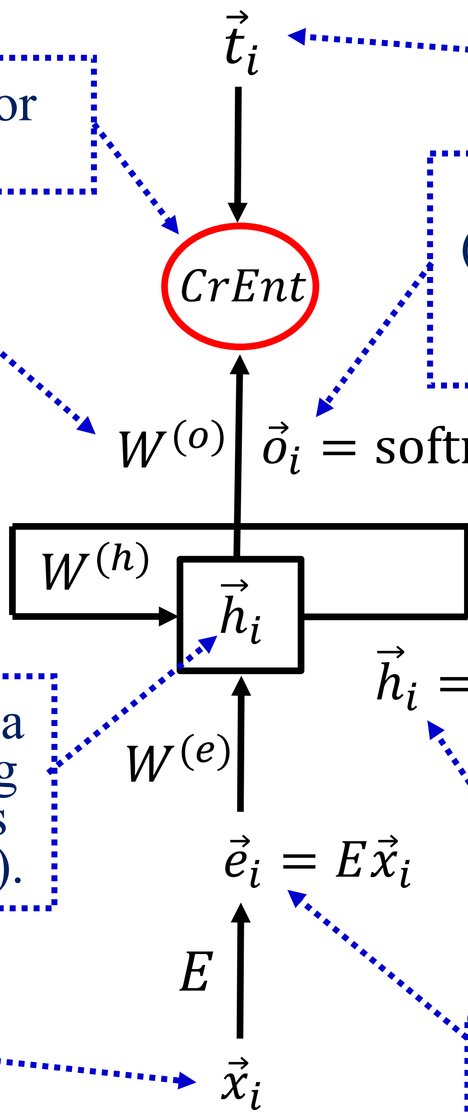
The **correct "probabilities"** for the class of "tech". A **1-hot vector**.

$$\vec{t} = \begin{bmatrix} P(C = c_1) \\ P(C = c_2) \\ P(C = c_3) \\ \dots \\ P(C = c_k) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

The **log-likelihood** of the **correct class** according to the **classifier** (with a minus sign).

$$H_{P_m}(C) = -\sum_{i=1}^{k} P(C = c_i) \log_2 P_m(C = c_i) = -\log_2 P_m(C = c_2)$$

RNN-based token classification



Correct class probabilities for the current word ($|C|\times 1$, 1-hot).

Cross-entropy loss for the current word.

Predicted probability distribution ($|C|\times 1$) over the classes (e.g., B-Person, I-Person, B-Location, Other) in Named Entity Recognition.

We can think of $\boldsymbol{W}^{(o)}$ as containing **class embeddings**.

$$\vec{o}_i = \mathrm{softmax}\left(W^{(o)}\vec{h}_i\right)$$

$$\vec{h}_i = \tanh\left(W^{(h)}\vec{h}_{i-1} + W^{(e)}\vec{e}_i\right)$$

State: vector acting as a **memory** (remembering things about the words of the input seen so far).

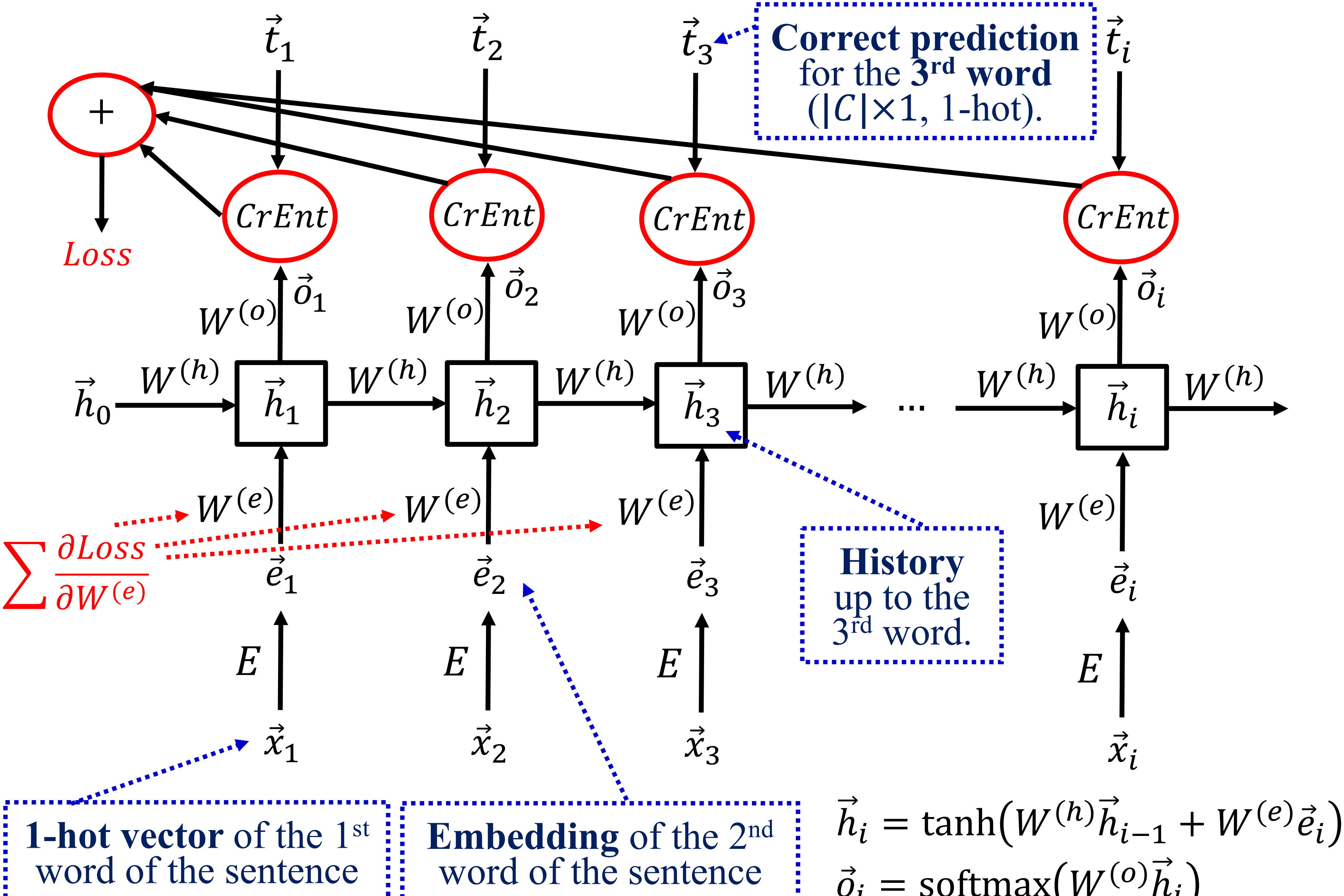
The new state (memory) is a combination of the previous one and the new word embedding.

$$\vec{e}_i = E\vec{x}_i$$

1-hot vector of the current word.

Embedding of the current word.

Unrolled RNN



$$\vec{h}_i = \tanh\left(W^{(h)}\vec{h}_{i-1} + W^{(e)}\vec{e}_i\right)$$

$$\vec{o}_i = \operatorname{softmax}\left(W^{(o)}\vec{h}_i\right)$$

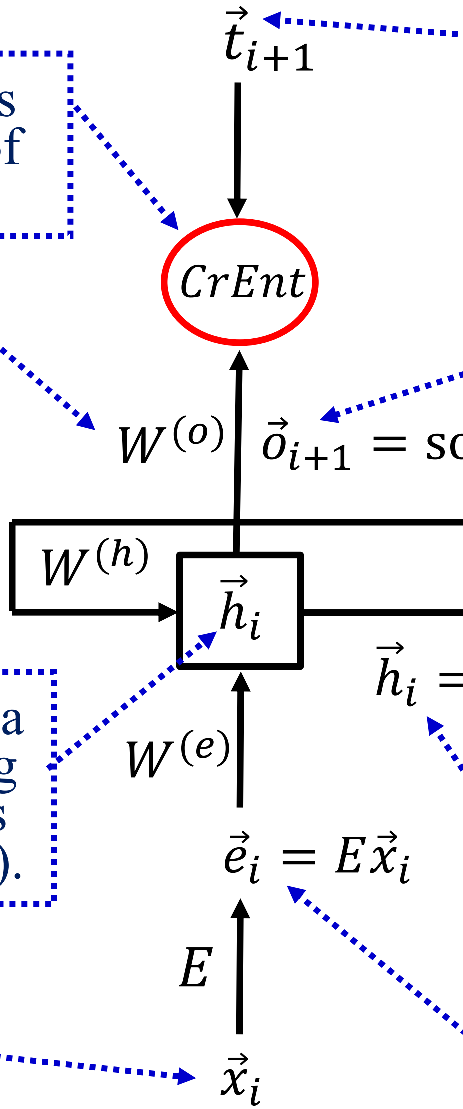
RNN language model

Correct prediction for **next word** ($|V|\times 1$, 1-hot).

Cross-entropy loss for the prediction of the next word.

$W^{(o)}$ contains **alternative** (output) **word embeddings**. Some RNN LMs use E^T as $W^{(o)}$.

Probability distribution ($|V|\times 1$) over the **vocabulary**. Shows which **words** the LM expects to see **next**.



$$\vec{o}_{i+1} = \mathrm{softmax}\left(W^{(o)}\vec{h}_i\right)$$

$$\vec{h}_i = \tanh\left(W^{(h)}\vec{h}_{i-1} + W^{(e)}\vec{e}_i\right)$$

$$\vec{e}_i = E\vec{x}_i$$

State: vector acting as a **memory** (remembering things about the input seen so far).

The new state (memory) is a combination of the previous one and the new word embedding.

1-hot vector of the current word.

Embedding of the current word.

Reminder: LMs as next word predictors

- **Sequence probability** using a bigram LM:

$$P(w_1^k) = P(w_1, \ldots, w_k) = P(w_1) \cdot P(w_2 \mid w_1) \cdot$$

$$P(w_3 \mid w_1, w_2) \cdot P(w_4 \mid w_1^3) \cdots P(w_k \mid w_1^{k-1}) \simeq$$

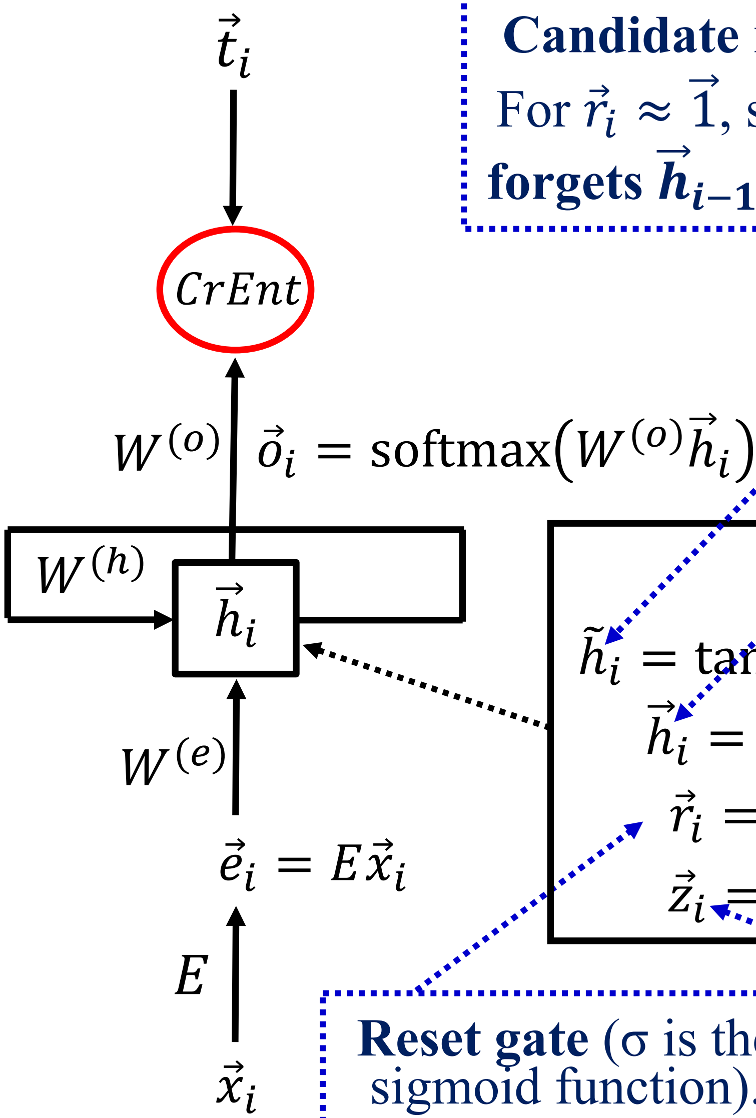
$$\boxed{P(w_1 \mid start) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_2) \cdots P(w_k \mid w_{k-1})}$$

- We can think of the **LM** as a system that **provides the probabilities $\boldsymbol{P(w_i|w_{i-1})}$**, which we then multiply.
  - Or the probabilities $\boldsymbol{P(w_i|w_{i-2}, w_{i-1})}$ for a **trigram LM**.
  - Or the probabilities $\boldsymbol{P(w_i|h)}$ for an LM that considers all the **"history" (previous words) $\boldsymbol{h}$**, e.g., in an **RNN LM**.

- An **LM** typically provides a **distribution $\boldsymbol{P(w|h)}$** showing how probable it is for **every word** $w \in V$ to be the next one.

RNN LM with GRU cells

Optional study



Candidate new history ($\circ$ denotes pairwise multiplication). For $\vec{r}_i \approx \vec{1}$, same as the $\vec{h}_i$ of a simple RNN cell. For $\boldsymbol{\vec{r}_i} \approx \boldsymbol{\vec{0}}$, **forgets** $\boldsymbol{\vec{h}_{i-1}}$ and considers only the current word embedding.

New history. For $\vec{z}_i \approx \vec{0}$, same as $\tilde{h}_i$. For $\boldsymbol{\vec{z}_i} \approx \boldsymbol{\vec{1}}$, ignores $\tilde{h}_i$ and **maintains** $\boldsymbol{\vec{h}_{i-1}}$ as $\vec{h}_i$.

GRU cell:

$$\tilde{h}_i = \tanh\left(\vec{r}_i \circ W^{(h)}\vec{h}_{i-1} + W^{(e)}\vec{e}_i\right)$$
$$\vec{h}_i = \vec{z}_i \circ \vec{h}_{i-1} + \left(\vec{1} - \vec{z}_i\right) \circ \tilde{h}_i$$
$$\vec{r}_i = \sigma\left(W^{(r)}\vec{h}_{i-1} + U^{(r)}\vec{e}_i\right)$$
$$\vec{z}_i = \sigma\left(W^{(z)}\vec{h}_{i-1} + U^{(z)}\vec{e}_i\right)$$

Reset gate (σ is the sigmoid function).

Update gate.

LSTM cells are similar, but with **more gates**. See http://colah.github.io/posts/2015-08-Understanding-LSTMs/

More about RNNs

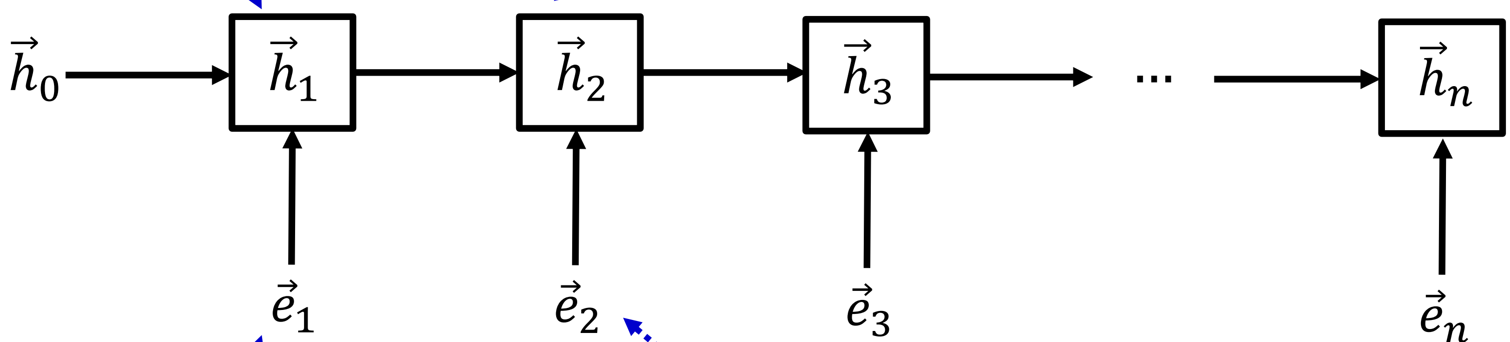
- Trained by **backpropagation** (with **unrolled** view).
  - For **each sentence** (**or window**), **feed** it to the **unrolled RNN**, compute the **loss** and **backpropagate**, **adding gradients** obtained for the **same matrix** (e.g., same $W^{(h)}$ at each cell).
  - **GRU** or **LSTM** cells help avoid **vanishing gradients**.
  - The norms of the **gradients** can be **clipped** (when larger than a max value) to avoid **exploding gradients**.
  - Use **layer normalization**, not batch normalization in RNNs.
- We can also **learn** the **word embeddings** (E) with an RNN LM. Billions of **free training examples**!
  - We can then use the **word embeddings** in **other NLP tasks**.
  - With a **large vocabulary**, **softmax** is too **slow** (alternatives: small vocabulary, hierarchical softmax, negative sampling).

What about the right-context of each token?

Revised embedding of the 1st word. Knows we are at the beginning of a sentence.

Revised embedding of the 2nd word. Knows the left-context.

We can also treat the $\vec{h}_i$ vectors as the **memory** of the RNN, but in recent NLP work, it's easier to think of them as **revised word embeddings**.



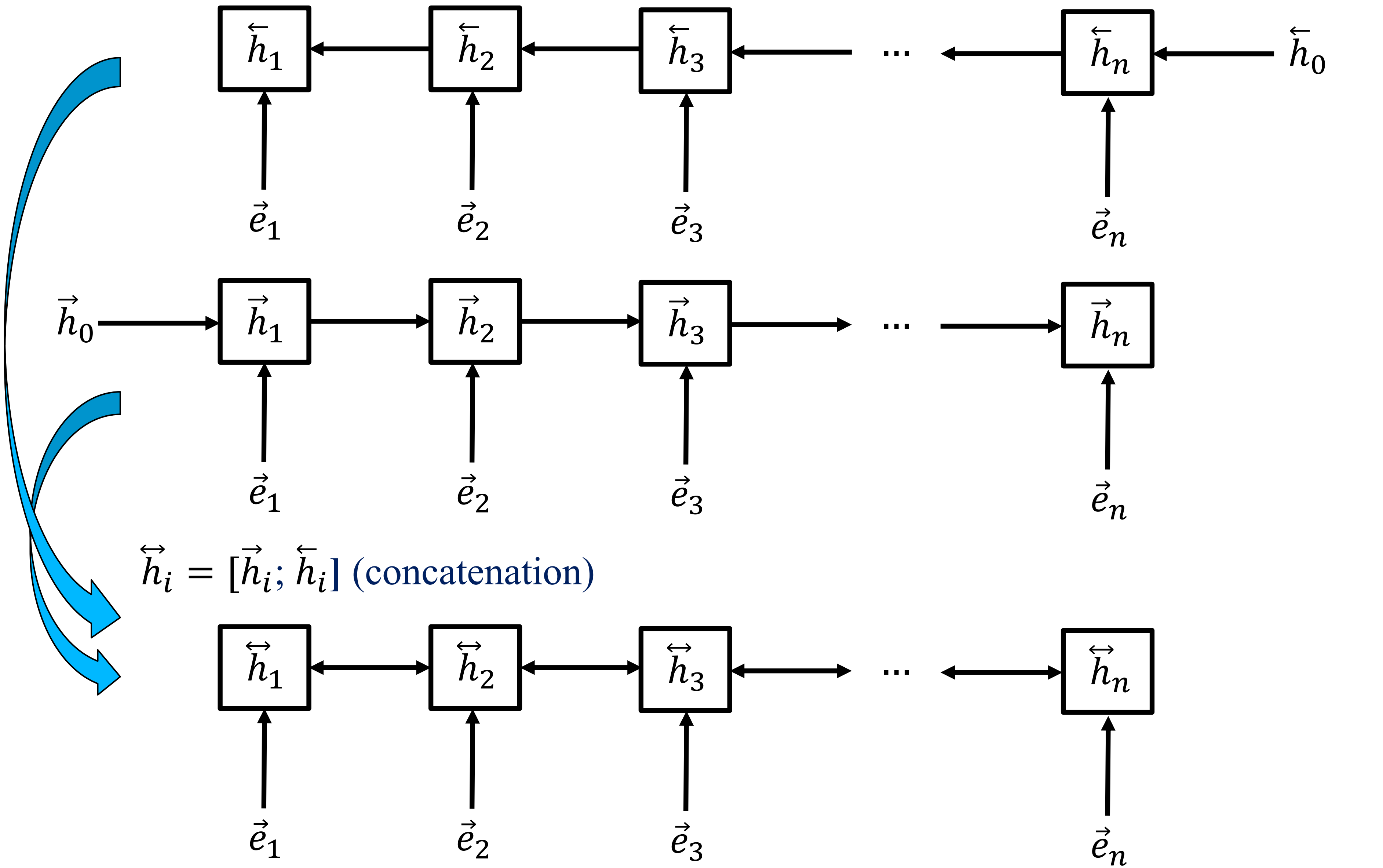
Embedding of the 1st word of the sentence

Embedding of the 2nd word of the sentence

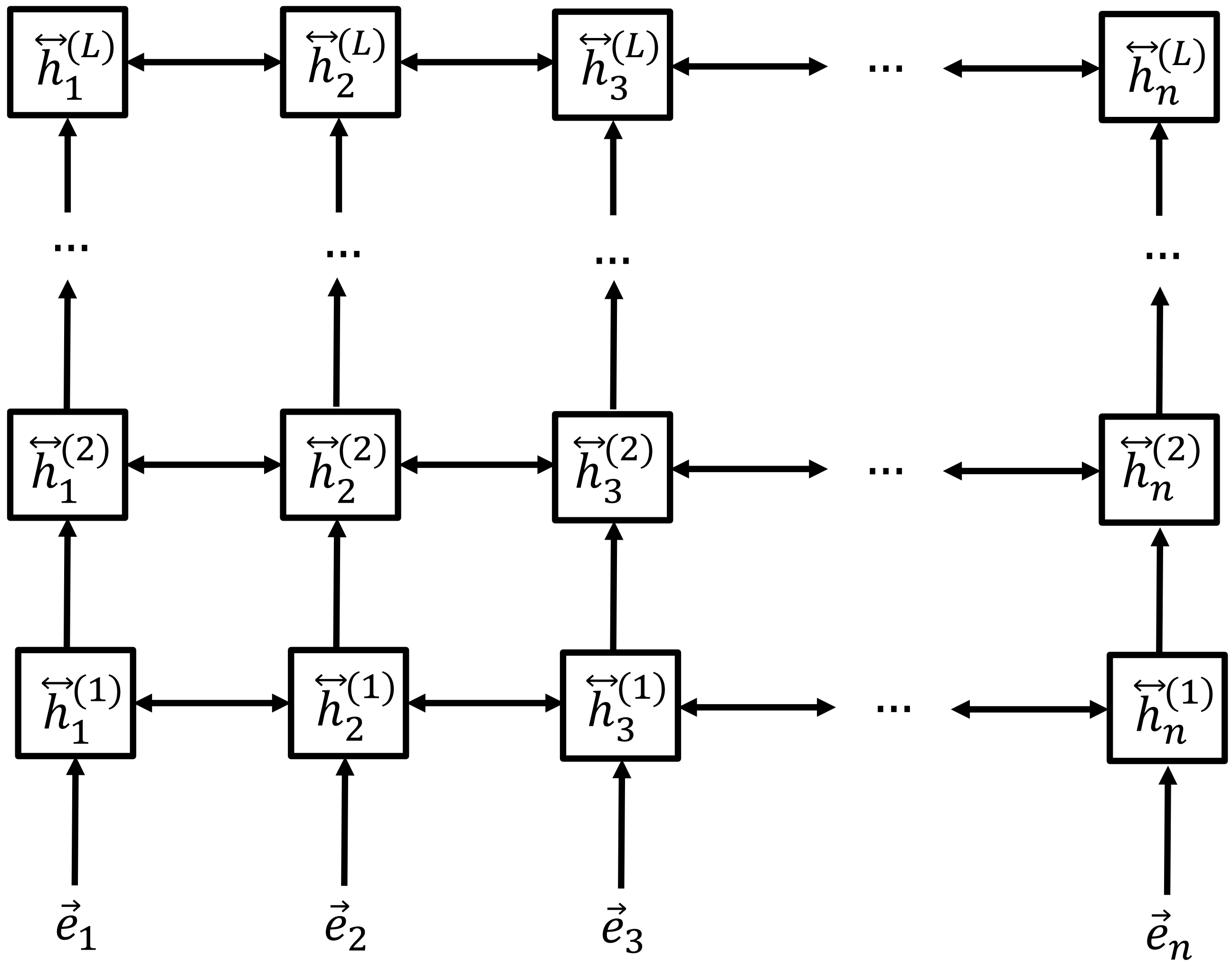
g is an **activation function** (e.g., sigmoid). More complex update mechanisms in practice: **LSTM** or **GRU** cells.

$$\vec{h}_i = g\left(W^{(h)}\vec{h}_{i-1} + W^{(e)}\vec{e}_i + \vec{b}^{(h)}\right)$$

Bidirectional RNN (biRNN)



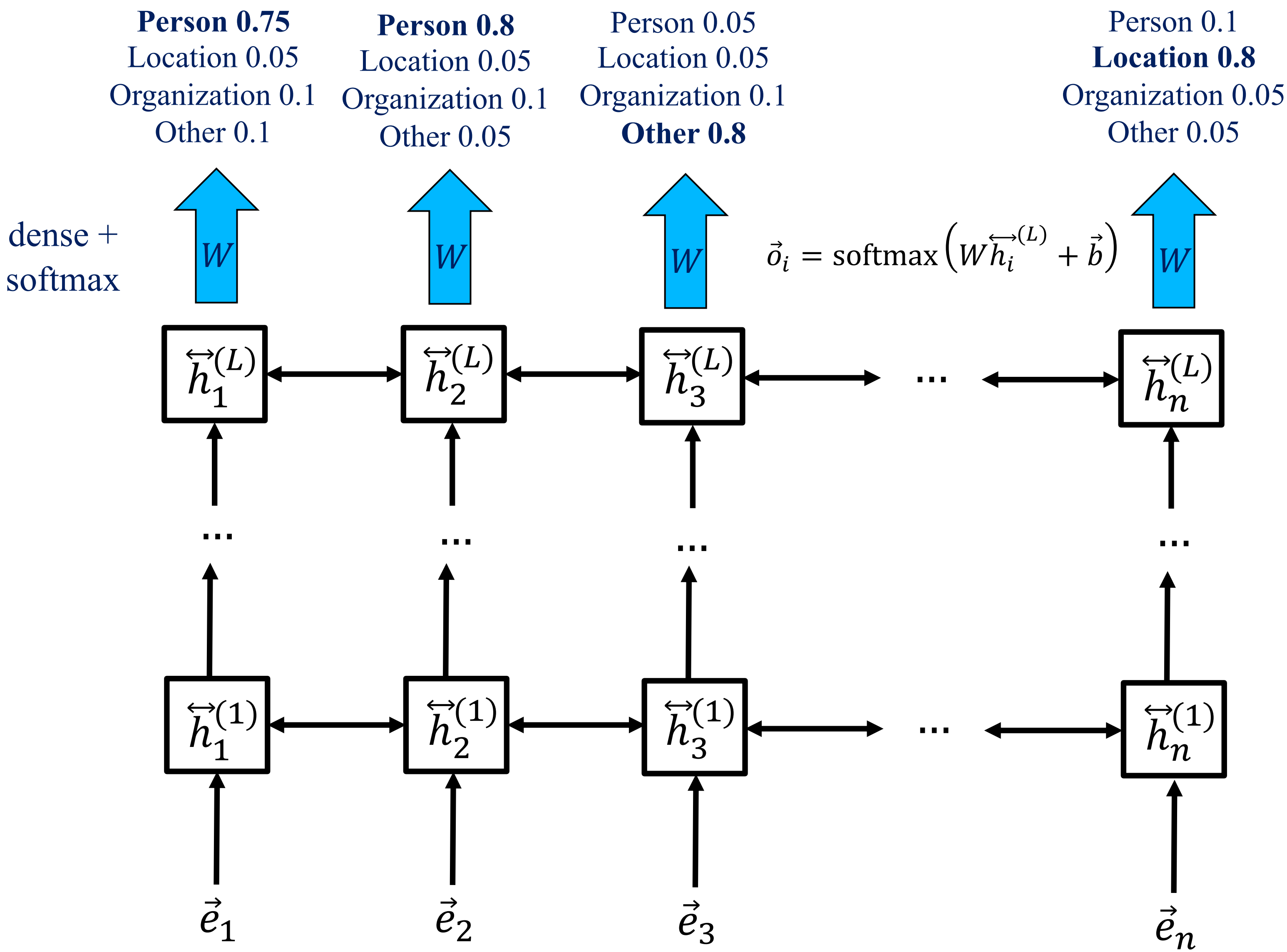
Stacked bidirectional RNN



Each layer revises the word embeddings of the previous (lower) layer. The **embeddings** become **increasingly more context-aware** and also **increasingly more appropriate** for the **particular task** we address…

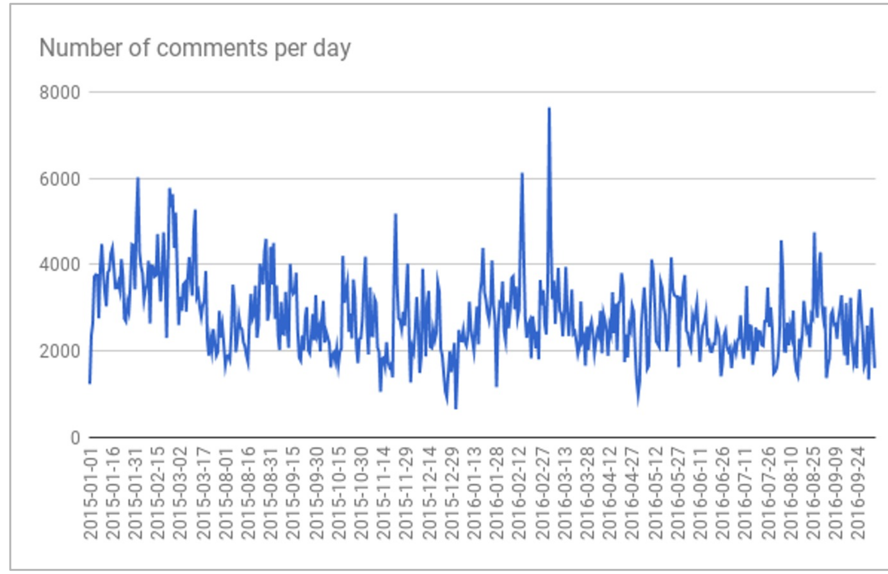
Token classification with a stacked biRNN

Compare to the correct predictions (**sum** the **cross-entropy loss** for **all token positions**) and **backpropagate** to **adjust all the weights**, including the weights of the stacked biRNN.

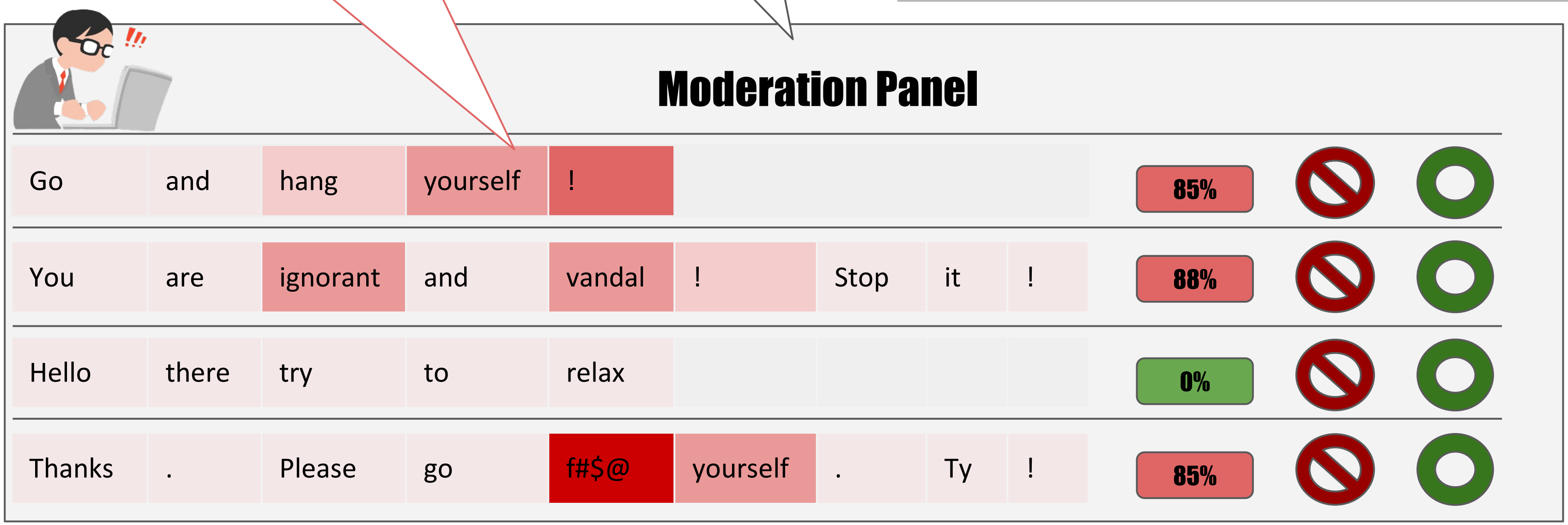


User comment moderation

A **moderation panel assists the moderators** to **detect abusive comments**, and leads to **quicker publication** of non-abusive comments.



Highlighting suspicious words using an **RNN** with **self-attention**.

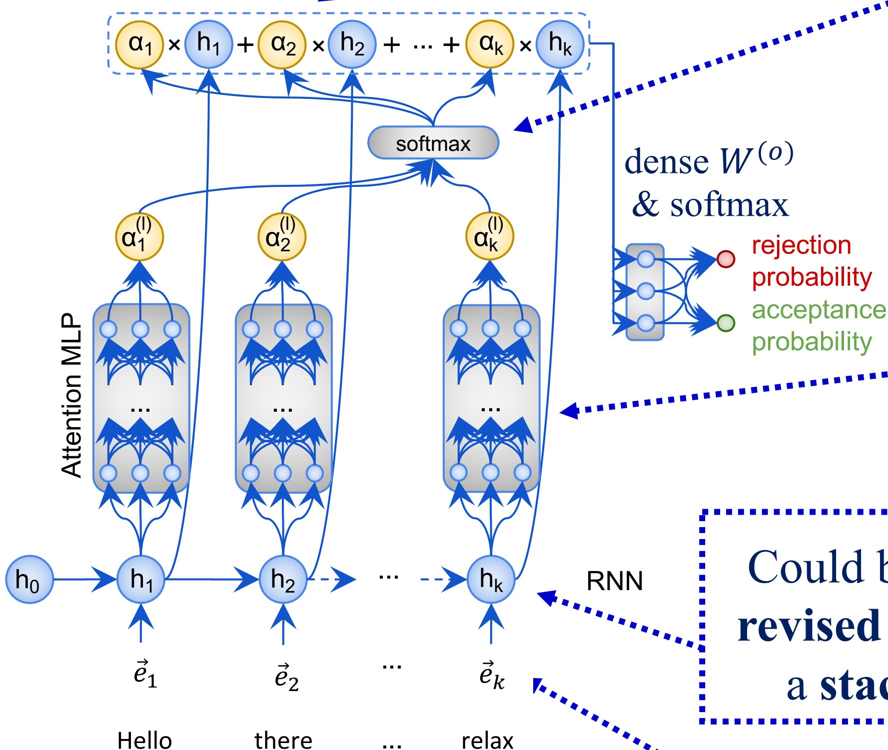


RNN with deep self-attention

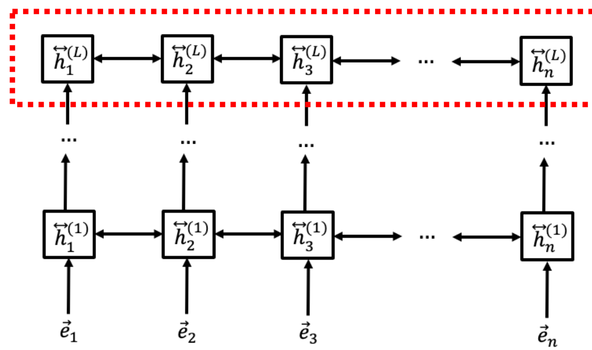
The **entire input text** is now represented by the **weighted (by a_i scores) sum** of the **revised embeddings** of its words.

The **softmax** ensures all the a_i scores are between 0 and 1, and that they sum to 1.

We use an **MLP** (the **same at all time-steps**) to obtain an **attention score** (importance) a_i **for each word from its revised embedding** h_i. We could also use a **single dense layer**: $a_i = W^{(a)} h_i$.



Could be the **top-level revised embeddings** of a **stacked biRNN**.



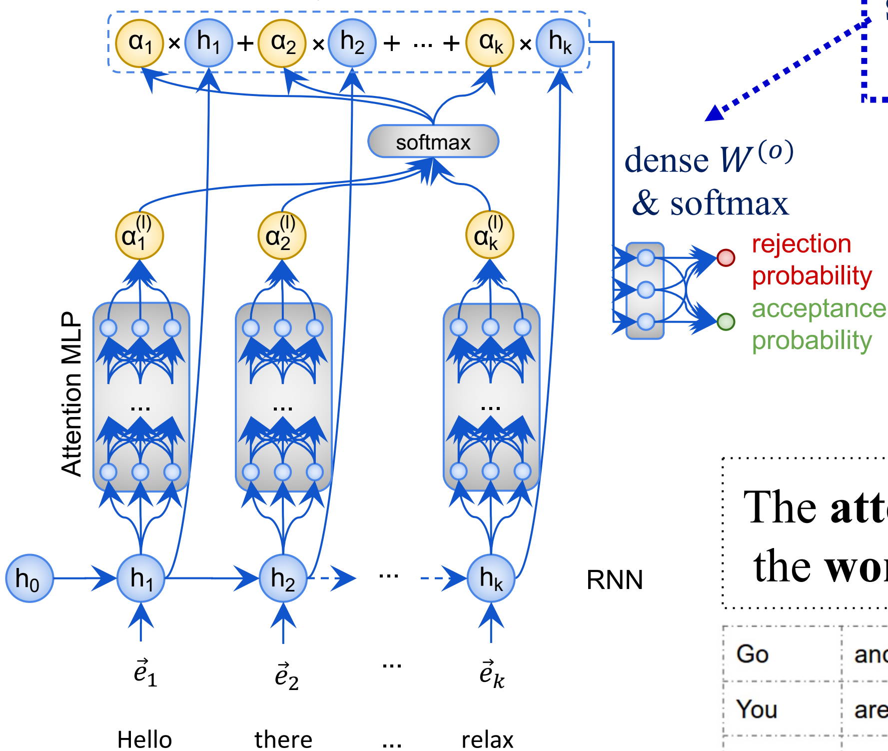
Initial word embeddings (e.g., via **Word2Vec**).

J. Pavlopoulos, P. Malakasiotis and I. Androutsopoulos, "Deeper Attention to Abusive User Content Moderation", EMNLP 2017, http://nlp.cs.aueb.gr/pubs/emnlp2017.pdf.

RNN with deep self-attention

The **entire input text** is now represented by the **weighted (by a_i scores) sum** of the **revised embeddings** of its words.

We pass the **weighted sum vector** (point) through another **dense layer and softmax** to obtain a **probability** score for **each class** (here accept, reject).



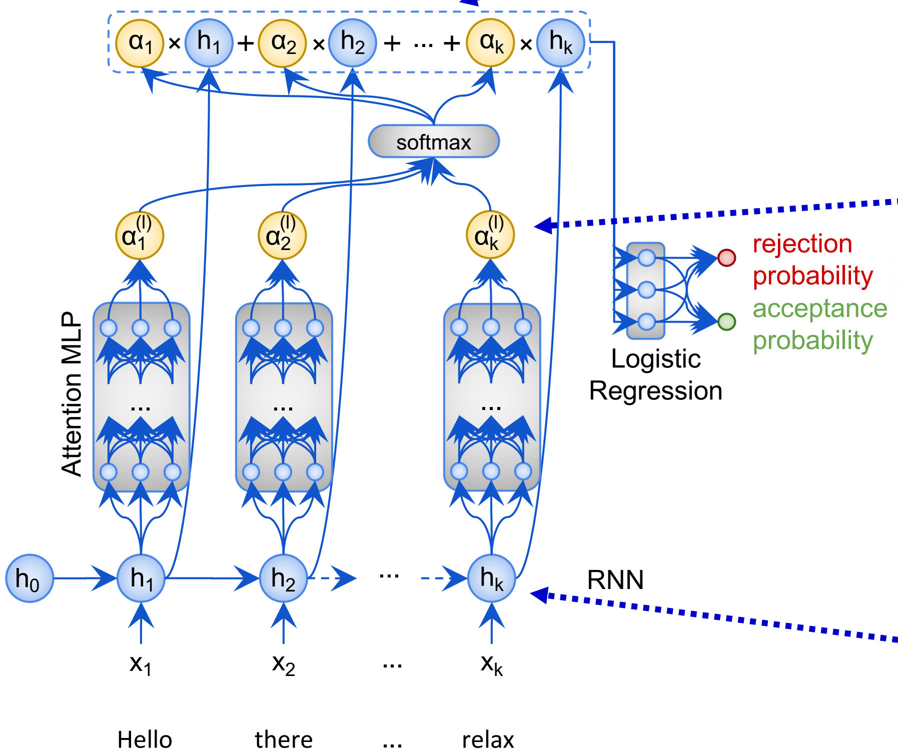
Compare to the correct predictions with a **cross-entropy loss** and **backpropagate** to **adjust the weights** of the **entire neural net**, including the MLP and RNN(s).

The **attention scores** a_i can also be used to **highlight** the **words** that influence the system's decision most.

Go	and	hang	yourself	!				
You	are	ignorant	and	vandal	!	Stop	it	!
Thanks	.	Please	go	[redacted]	yourself	.	ty	!

J. Pavlopoulos, P. Malakasiotis and I. Androutsopoulos, "Deeper Attention to Abusive User Content Moderation", EMNLP 2017, http://nlp.cs.aueb.gr/pubs/emnlp2017.pdf.

RNN with deep self-attention



$$h_{sum} = \sum_{t=1}^{k} a_t h_t$$
$$P_{a-\text{RNN}}(reject|c) = \sigma(W_p h_{sum} + b_p)$$

$$a_t^{(1)} = \text{RELU}(W^{(1)} h_t + b^{(1)})$$
$$\cdots$$
$$a_t^{(l-1)} = \text{RELU}(W^{(l-1)} a_t^{(l-2)} + b^{(l-1)})$$
$$a_t^{(l)} = W^{(l)} a_t^{(l-1)} + b^{(l)}$$
$$a_t = \texttt{softmax}(a_t^{(l)}; a_1^{(l)}, \ldots, a_k^{(l)})$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$
$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

J. Pavlopoulos, P. Malakasiotis and I. Androutsopoulos, "Deeper Attention to Abusive User Content Moderation", EMNLP 2017, http://nlp.cs.aueb.gr/pubs/emnlp2017.pdf.

RNNs for Machine Translation

From the slides of R. Socher's course "Deep Learning for NLP", 2015. http://cs224d.stanford.edu/

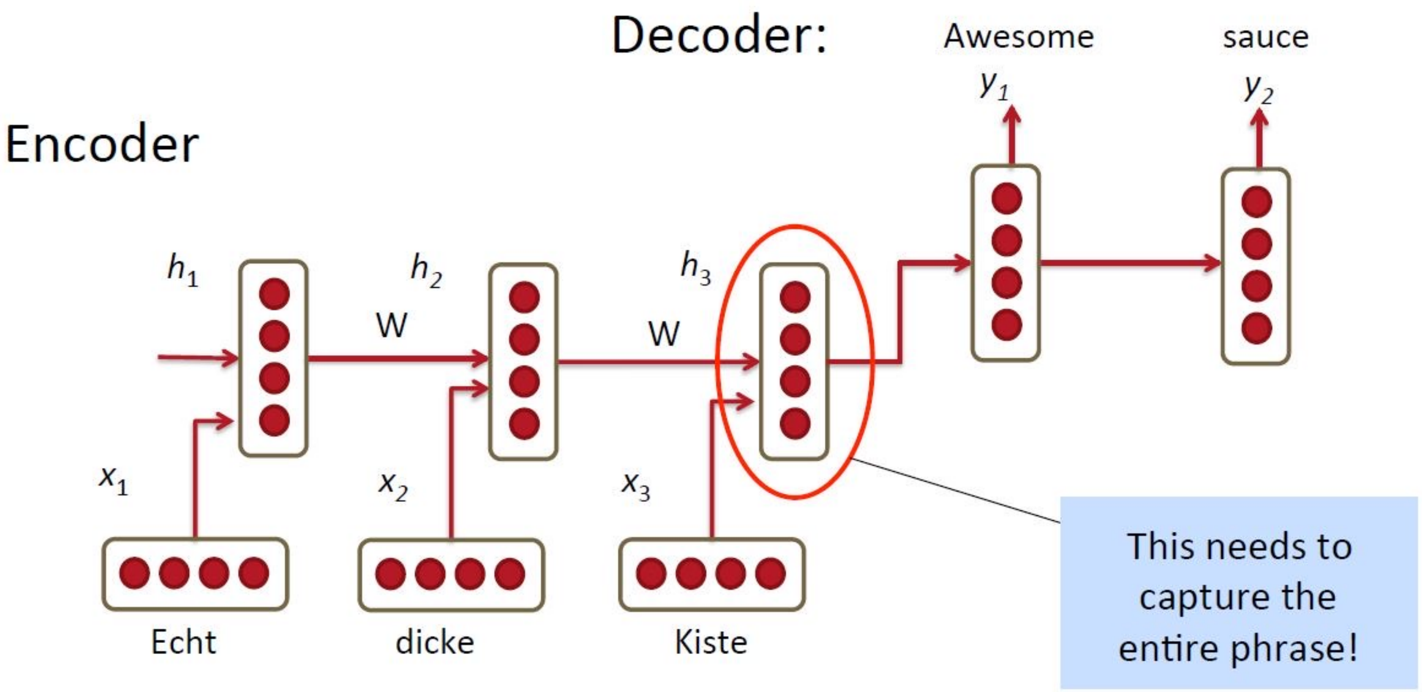
Encoder: $h_t = \phi(h_{t-1}, x_t) = f\left(W^{(hh)} h_{t-1} + W^{(hx)} x_t\right)$

Decoder: $h_t = \phi(h_{t-1}) = f\left(W^{(hh)} h_{t-1}\right)$

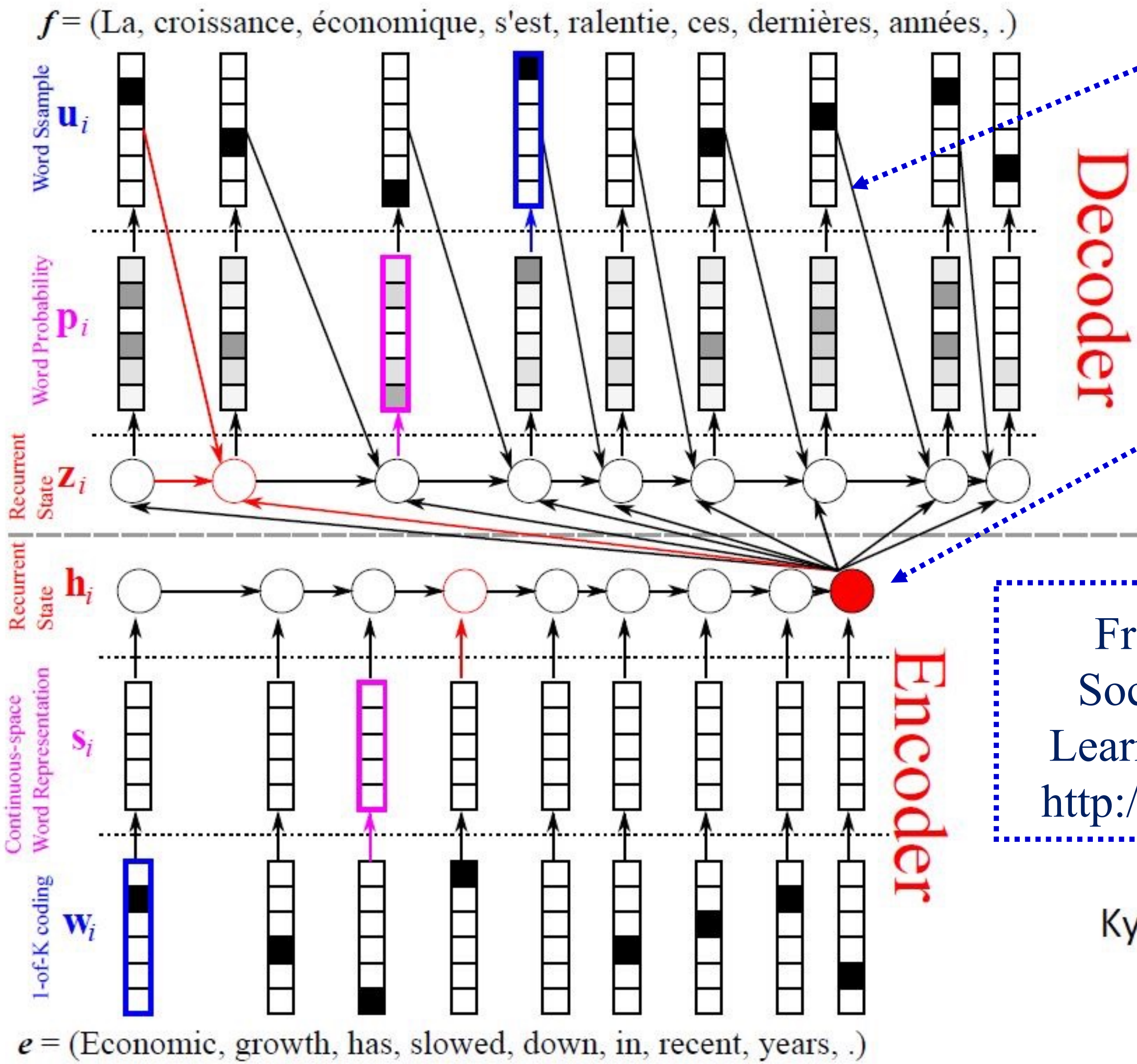
$y_t = softmax\left(W^{(S)} h_t\right)$

Minimize cross entropy error for all target words conditioned on source words

$$\max_\theta \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(y^{(n)} | x^{(n)})$$



Different picture, same idea



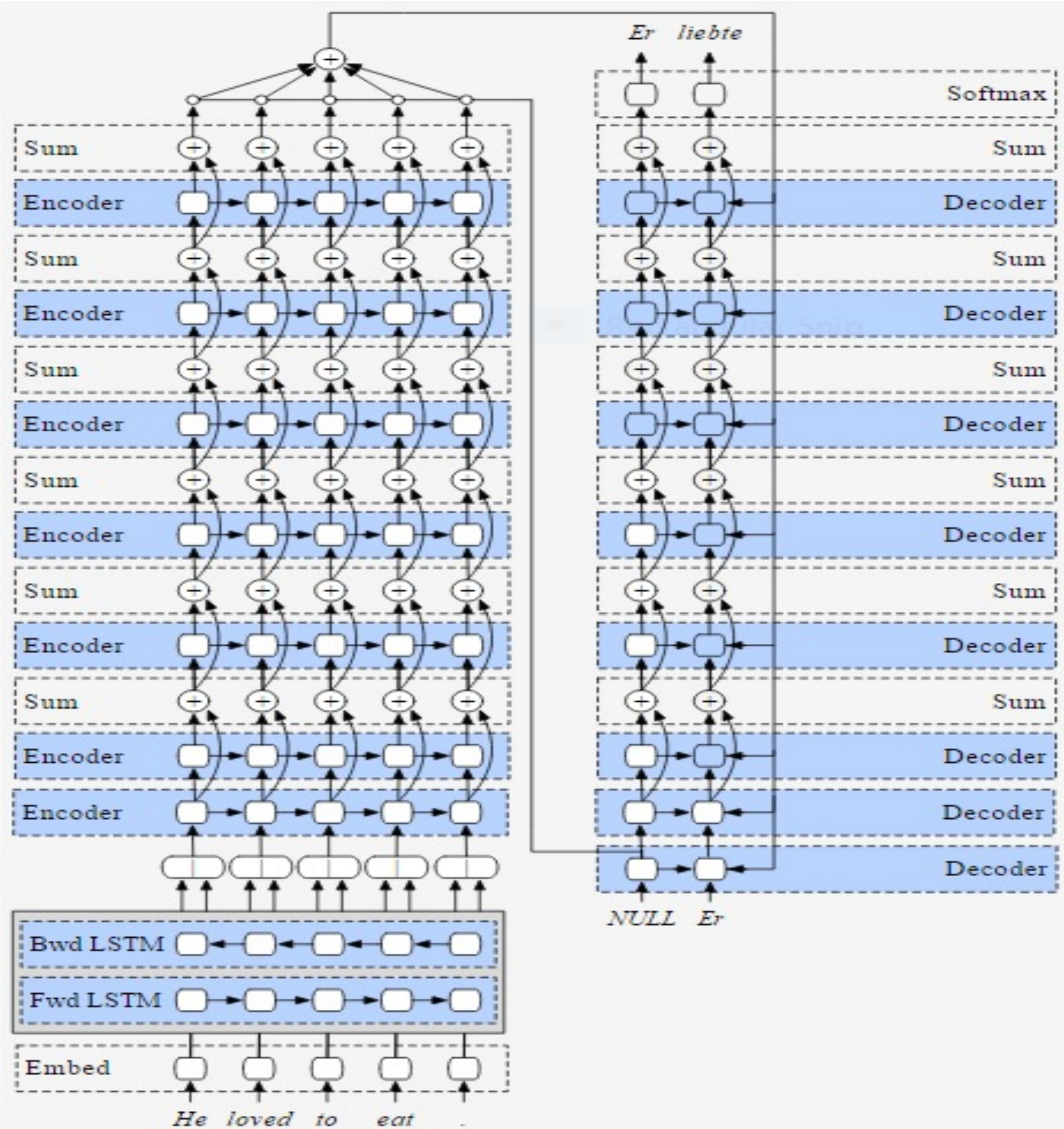
Embedding of the **previously generated word**.

Last $\overrightarrow{h_i}$ of the **encoder RNN**. Treated as **embedding** of the entire **input sentence**.

From the slides of R. Socher's course "Deep Learning for NLP", 2015. http://cs224d.stanford.edu/

Kyunghyun Cho et al. 2014

RNN-based Machine Translation

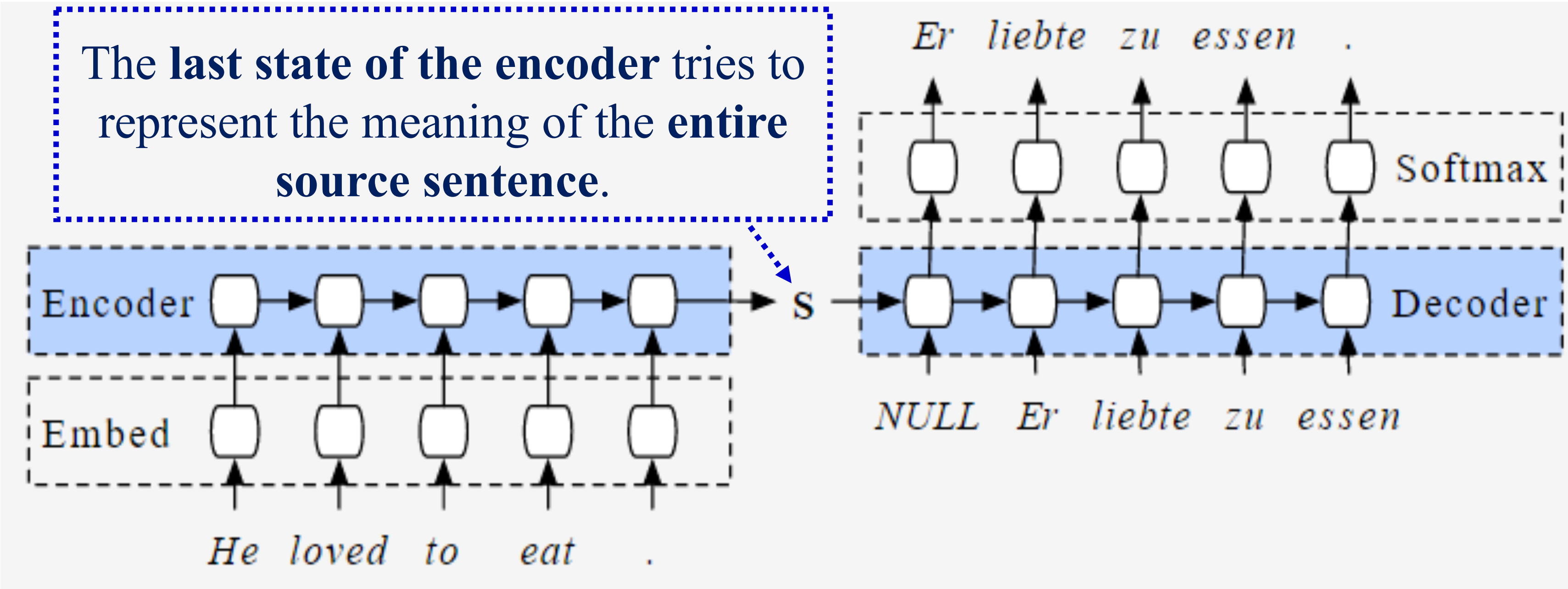


Google's paper:
https://arxiv.org/abs/1609.08144

Images from Stephen Merity's
http://smerity.com/articles/2016/google_nmt_arch.html

Easier to explain step by step…

Basic Encoder-Decoder NMT



During training, at each time-step of the **decoder**, we can use the **correct previous word** of the human translation (**teacher forcing)**, or we can **randomly use the correct or the predicted** previous word (**scheduled sampling**).

During testing (inference), we always use the **predicted previous word**; and we either **greedily select the most probable next word,** or we use **beam search** to find the translation y_1^m of x_1^n with the highest probability:

$$p(y_1|x_1^n)\; p(y_2|y_1, x_1^n)\; p(y_3|y_1^2, x_1^n) \ldots p(y_m|y_1^{m-1}, x_1^n)$$

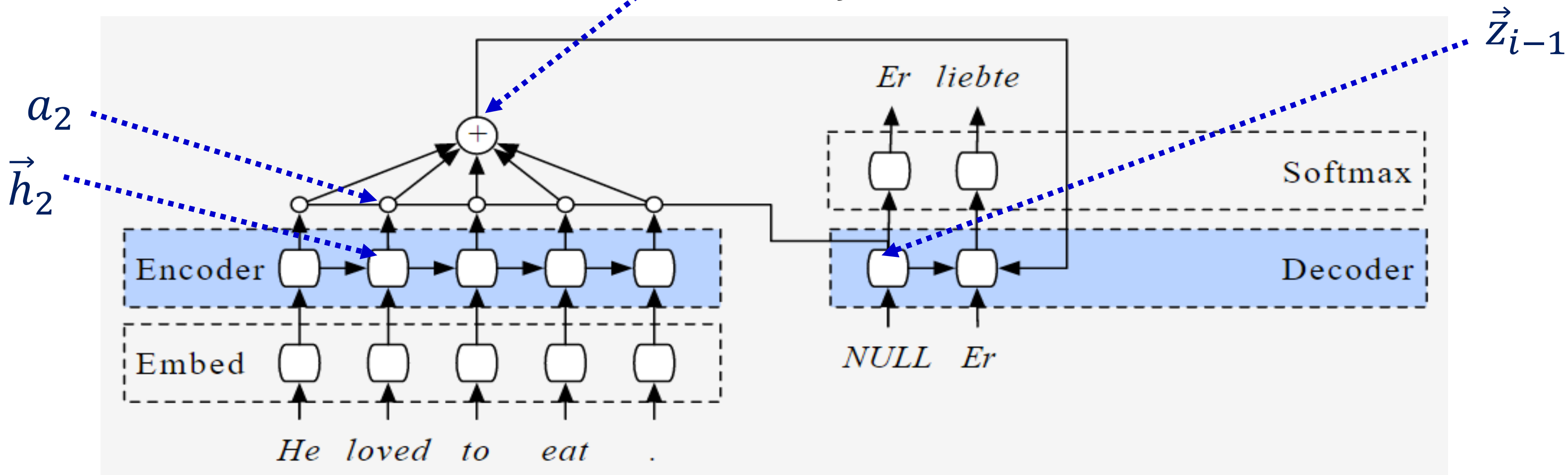
Google's paper: https://arxiv.org/abs/1609.08144

Images from Stephen Merity's http://smerity.com/articles/2016/google_nmt_arch.html

Encoder-Decoder with attention

The **source sentence** is now represented by the **weighted sum** of the **encoder states**:

$$\vec{h}_{sum} = \sum_j a_j \vec{h}_j$$



For each German word, the **attention scores** over the English words **change**!

Each "**attention**" **weight** $\boldsymbol{a_j}$ is a **function** of the **corresponding encoder state** $\vec{\boldsymbol{h}}_{\boldsymbol{j}}$ and the **previous state** $\vec{\boldsymbol{z}}_{\boldsymbol{i-1}}$ **of the decoder** (memory of translation so far), e.g.:

$$\tilde{a}_j = v^T \cdot f\left(W^{(h)}\vec{h}_j + W^{(z)}\vec{z}_{i-1}\right) = v^T \cdot f\left(W\left[\vec{h}_j; \vec{z}_{i-1}\right]\right), \qquad a_j = softmax(\tilde{a}_j)$$

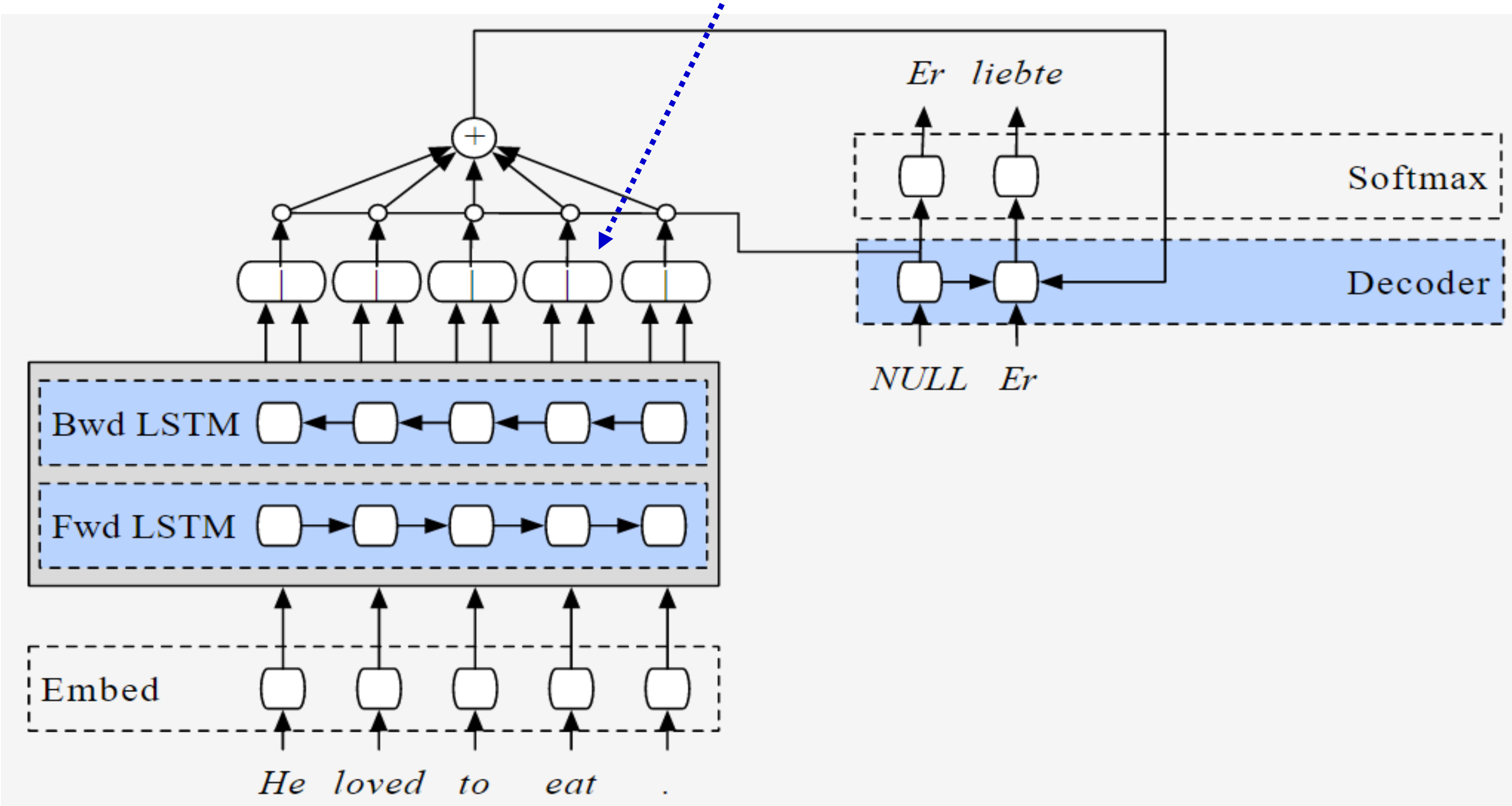
with a **softmax** to make the $\boldsymbol{a_j}$ weights sum to 1.

Google's paper: https://arxiv.org/abs/1609.08144

Images from Stephen Merity's http://smerity.com/articles/2016/google_nmt_arch.html

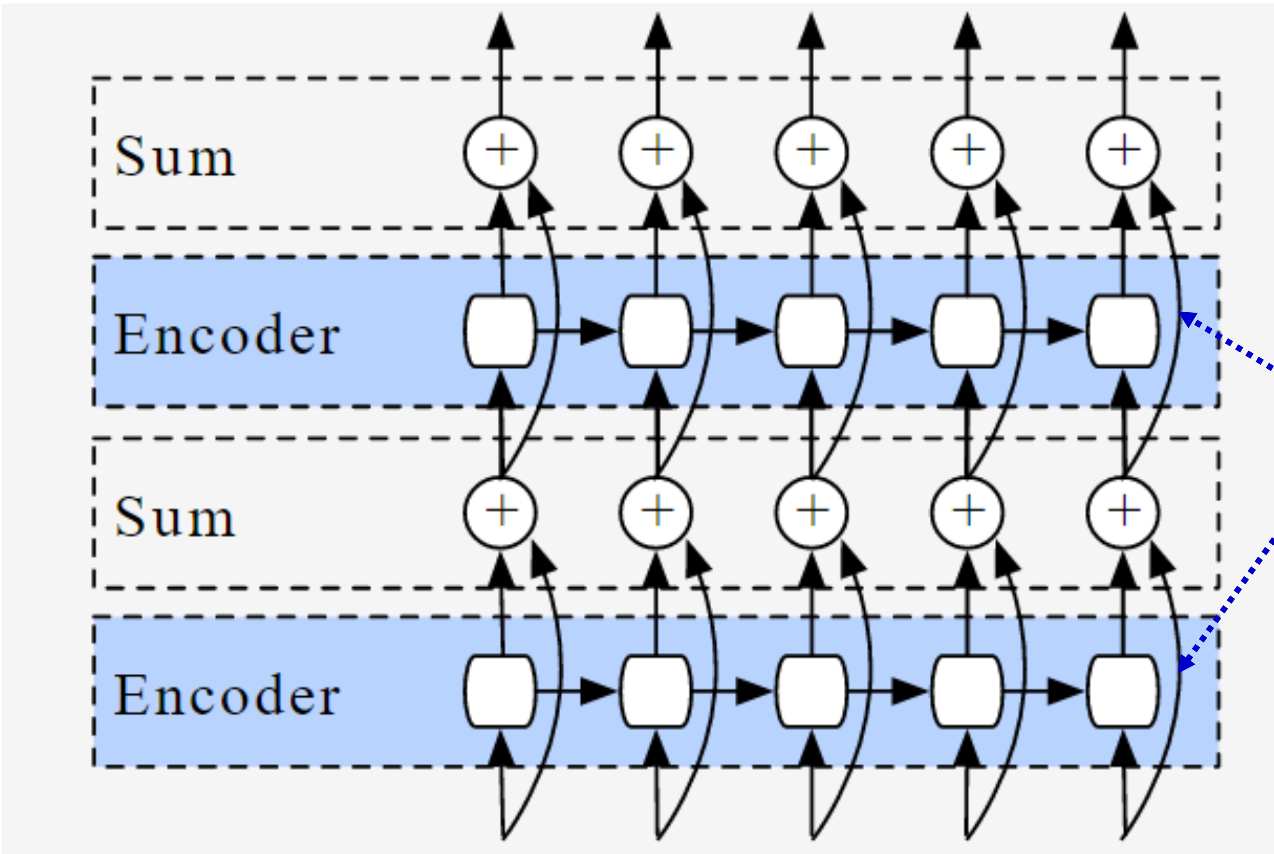
Bidirectional LSTM encoder

The encoder is now a **bidirectional LSTM**. The **encoder state** for the ***j*-th word of the source** sentence is the **concatenation** of the **corresponding states** of the **forward** and **backward** LSTM.



Google's paper: https://arxiv.org/abs/1609.08144
Images from Stephen Merity's http://smerity.com/articles/2016/google_nmt_arch.html

Stacking RNNs and residuals

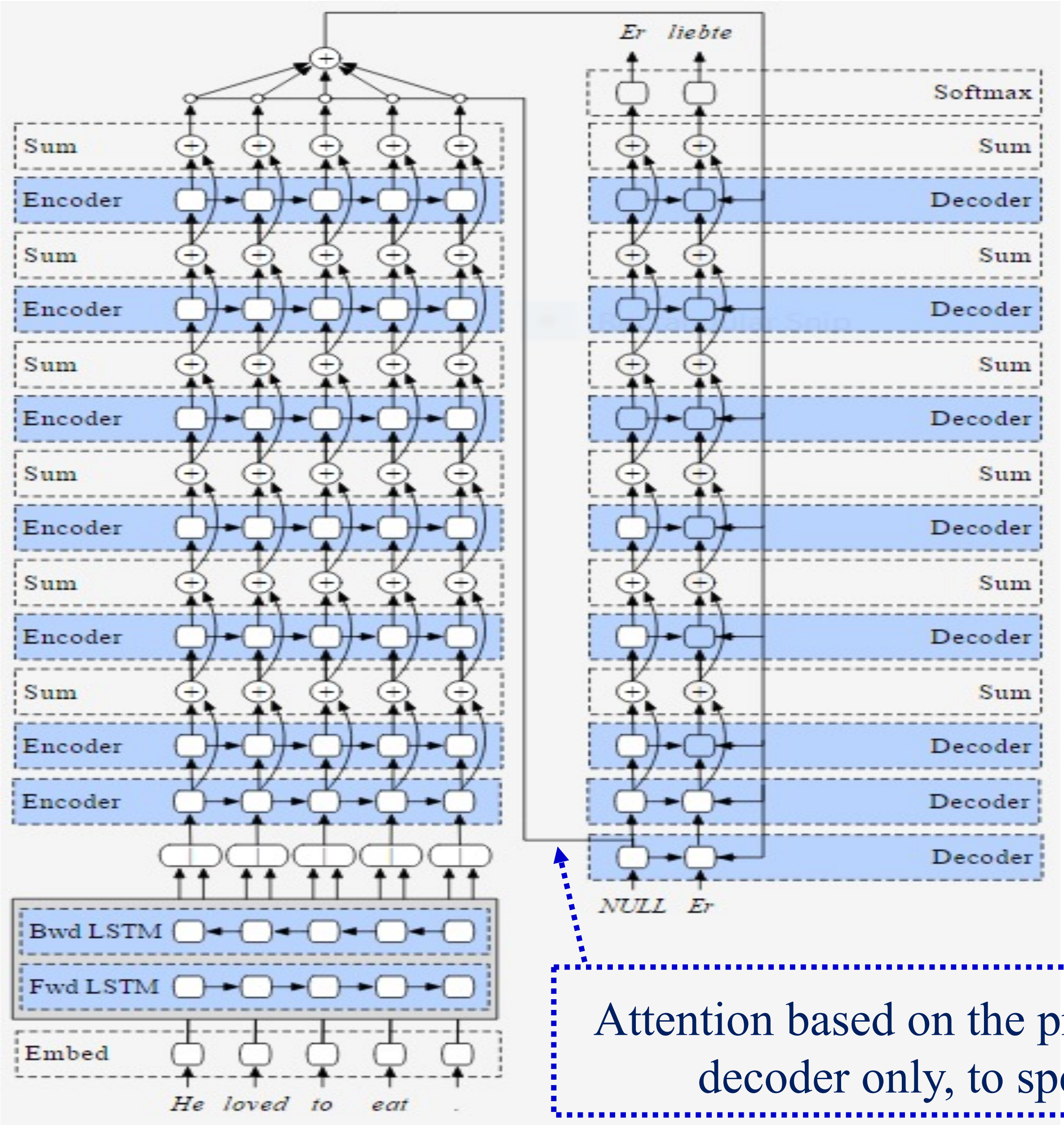


“**Residual**” connections (a kind of skip-connections) **helps fight vanishing gradients** in backpropagation (sum-nodes copy the gradients to their inputs). Also allows upper layers to **learn only modifications** (differences) from representations of lower layers.

Google’s paper: https://arxiv.org/abs/1609.08144

Images from Stephen Merity’s http://smerity.com/articles/2016/google_nmt_arch.html

RNN-based Machine Translation



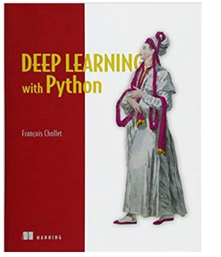
Google's paper:
https://arxiv.org/abs/1609.08144

Images from Stephen Merity's
http://smerity.com/articles/2016/google_nmt_arch.html

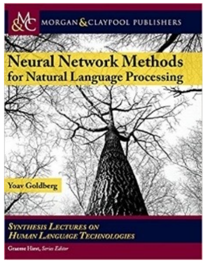
Attention based on the previous state of the bottom decoder only, to speed up computations.

Recommended reading

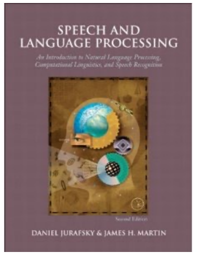
- F. Chollet, *Deep Learning in Python*, 1st edition, Manning Publications, 2017.
  - 1st edition freely available (and sufficient for this course): https://www.manning.com/books/deep-learning-with-python
  - See mostly sections 6.1–6.3, section 8.1.
  - 2nd edition available, recommended, requires payment.



- Y. Goldberg, *Neural Network Models for Natural Language Processing*, Morgan & Claypool Publishers, 2017.
  - Mostly chapters 14–17.



- Jurafsky and Martin's, *Speech and Language Processing* is being revised (3rd edition) to include DL methods.
  - http://web.stanford.edu/~jurafsky/slp3/



- See also the recommended reading and resources of the previous part (B5) of this course.

Βιβλιογραφία – συνέχεια

- Αν έχετε από το μάθημα της ΤΝ το βιβλίο των Russel & Norvig «Τεχνητή Νοημοσύνη – Μια σύγχρονη προσέγγιση», 4η έκδοση, Κλειδάριθμος, 2021, μπορείτε να συμβουλευτείτε τα κεφάλαια 21 και 24.
  - Κυρίως τις ενότητες 21.6, 21.8.2, 24.1, 24.2, 24.3.
  - Άλλες ενότητες αυτών των κεφαλαίων θα καλυφθούν σε επόμενες διαλέξεις.