Language Models

Philipp Koehn

23 April 2024

Language models

- **Language models** answer the question:

  How likely is a string of English words good English?

- Help with word order

$$p_{\text{LM}}(\text{the house is small}) > p_{\text{LM}}(\text{small the is house})$$

- Help with word choice

$$p_{\text{LM}}(\text{I am going home}) > p_{\text{LM}}(\text{I am going house})$$

N-Gram Language Models

- Given: a string of English words $W = w_1, w_2, w_3, ..., w_n$
- Question: what is $p(W)$?
- Sparse data: Many good English sentences will not have been seen before

$\rightarrow$ Decomposing $p(W)$ using the chain rule:

$$p(w_1, w_2, w_3, ..., w_n) = p(w_1)\ p(w_2|w_1)\ p(w_3|w_1, w_2)...p(w_n|w_1, w_2, ...w_{n-1})$$

(not much gained yet, $p(w_n|w_1, w_2, ...w_{n-1})$ is equally sparse)

Markov Chain

- **Markov assumption**:
  - only previous history matters
  - limited memory: only last k words are included in history (older words less relevant)

  $\rightarrow$ k**th order Markov model**

- For instance 2-gram language model:

$$p(w_1, w_2, w_3, ..., w_n) \simeq p(w_1)\ p(w_2|w_1)\ p(w_3|w_2)...p(w_n|w_{n-1})$$

- What is conditioned on, here w_{i-1} is called the **history**

Estimating N-Gram Probabilities

- Maximum likelihood estimation

$$p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)}$$

- Collect counts over a large text corpus

- Millions to billions of words are easy to get
  (trillions of English words available on the web)

Example: 3-Gram

- Counts for trigrams and estimated word probabilities

the green (total: 1748)

word	c.	prob.
paper	801	0.458
group	640	0.367
light	110	0.063
party	27	0.015
ecu	21	0.012

the red (total: 225)

word	c.	prob.
cross	123	0.547
tape	31	0.138
army	9	0.040
card	7	0.031
,	5	0.022

the blue (total: 54)

word	c.	prob.
box	16	0.296
.	6	0.111
flag	6	0.111
,	3	0.056
angel	3	0.056

  - 225 trigrams in the Europarl corpus start with the red
  - 123 of them end with cross

→ maximum likelihood probability is $\frac{123}{225} = 0.547$.

How good is the LM?

- A good model assigns a text of real English W a high probability
- This can be also measured with cross entropy:

$$H(W) = -\frac{1}{n} \log_2 p(W_1^n)$$

- Or, **perplexity**

$$\text{perplexity}(W) = 2^{H(W)}$$

Example: 3-Gram

prediction	p_{LM}	-$\log_2 p_{\text{LM}}$
p_{LM}(i|</s><s>)	0.109	3.197
p_{LM}(would|<s>i)	0.144	2.791
p_{LM}(like|i would)	0.489	1.031
p_{LM}(to|would like)	0.905	0.144
p_{LM}(commend|like to)	0.002	8.794
p_{LM}(the|to commend)	0.472	1.084
p_{LM}(rapporteur|commend the)	0.147	2.763
p_{LM}(on|the rapporteur)	0.056	4.150
p_{LM}(his|rapporteur on)	0.194	2.367
p_{LM}(work|on his)	0.089	3.498
p_{LM}(.|his work)	0.290	1.785
p_{LM}(</s>|work .)	0.99999	0.000014
	average	2.634

Comparison 1–4-Gram

word	unigram	bigram	trigram	4-gram
i	6.684	3.197	3.197	3.197
would	8.342	2.884	2.791	2.791
like	9.129	2.026	1.031	1.290
to	5.081	0.402	0.144	0.113
commend	15.487	12.335	8.794	8.633
the	3.885	1.402	1.084	0.880
rapporteur	10.840	7.319	2.763	2.350
on	6.765	4.140	4.150	1.862
his	10.678	7.316	2.367	1.978
work	9.993	4.816	3.498	2.394
.	4.896	3.020	1.785	1.510
</s>	4.828	0.005	0.000	0.000
average	8.051	4.072	2.634	2.251
perplexity	265.136	16.817	6.206	4.758

N-Gram Backoff Language Model

- Previously, we approximated

$$p(W) = p(w_1, w_2, ..., w_n)$$

- ... by applying the chain rule

$$p(W) = \sum_i p(w_i|w_1, ..., w_{i-1})$$

- ... and limiting the history (Markov order)

$$p(w_i|w_1, ..., w_{i-1}) \simeq p(w_i|w_{i-4}, w_{i-3}, w_{i-2}, w_{i-1})$$

- Each $p(w_i|w_{i-4}, w_{i-3}, w_{i-2}, w_{i-1})$ may not have enough statistics to estimate

  $\rightarrow$ we back off to $p(w_i|w_{i-3}, w_{i-2}, w_{i-1})$, $p(w_i|w_{i-2}, w_{i-1})$, etc., all the way to $p(w_i)$

  – exact details of backing off get complicated — "interpolated Kneser-Ney"

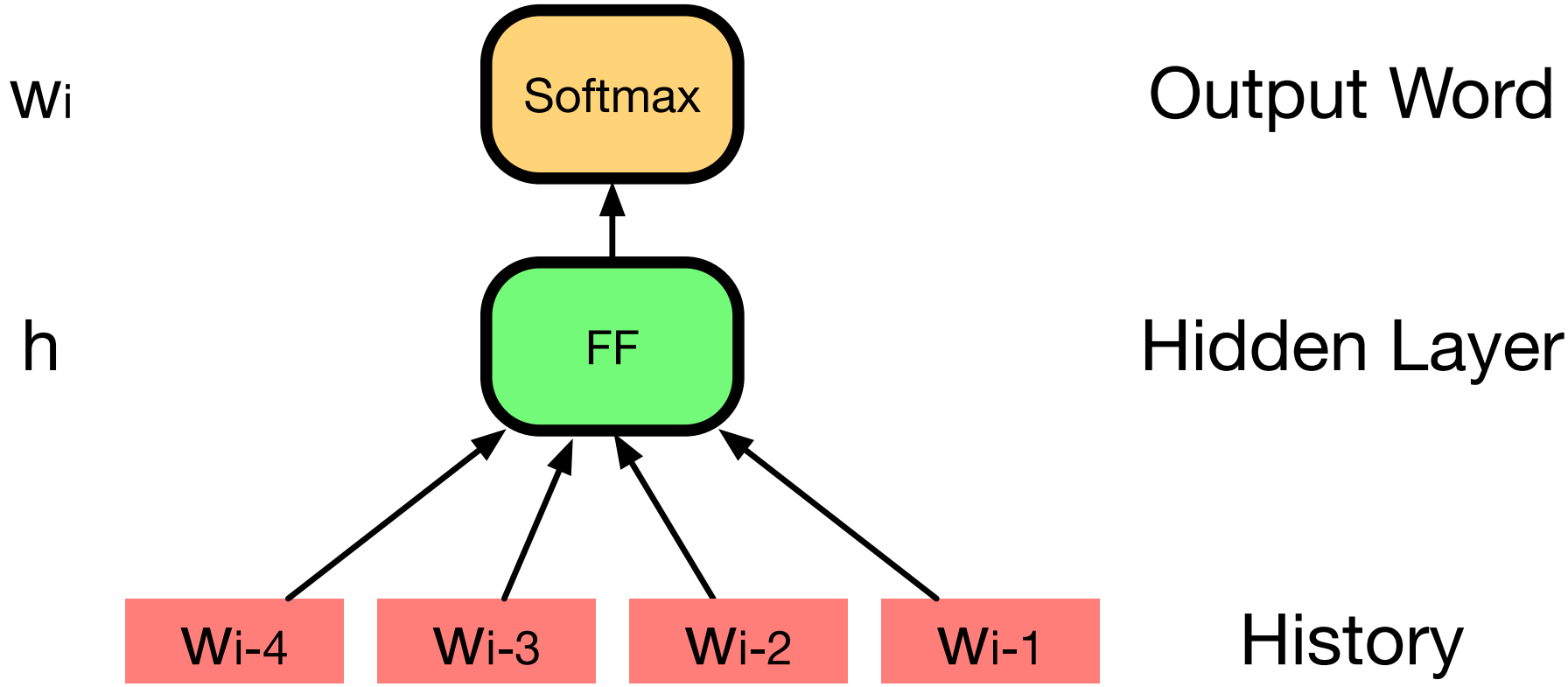
Refinements

- A whole family of back-off schemes
- Skip-n gram models that may back off to $p(w_i|w_{i-2})$
- Class-based models $p(C(w_i)|C(w_{i-4}), C(w_{i-3}), C(w_{i-2}), C(w_{i-1}))$

$\Rightarrow$ We are wrestling here with

- using as much relevant evidence as possible
- pooling evidence between words

First Sketch

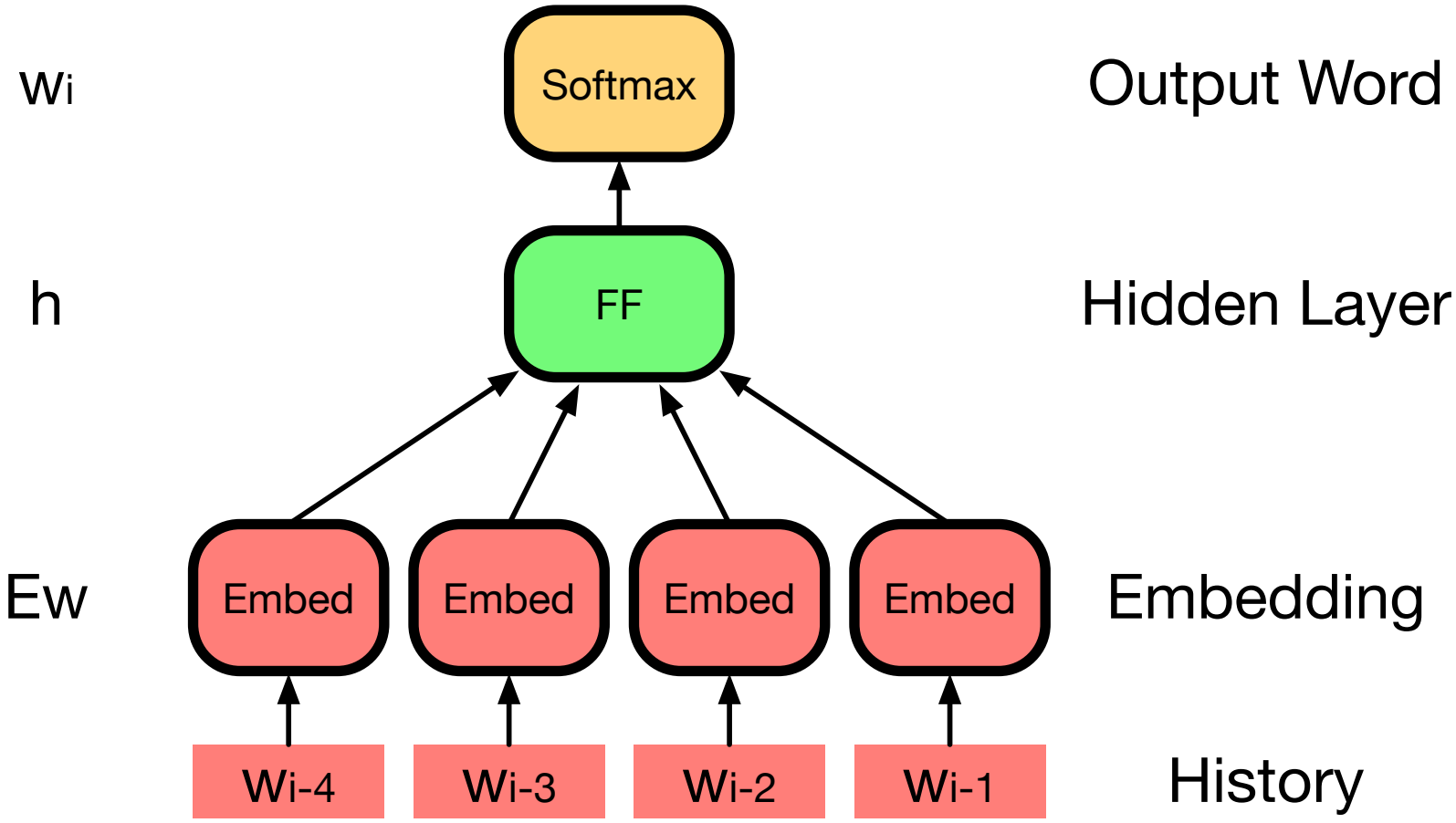


Representing Words

- Words are represented with a one-hot vector, e.g.,
  - dog = (0,0,0,0,1,0,0,0,0,....)
  - cat = (0,0,0,0,0,0,0,1,0,....)
  - eat = (0,1,0,0,0,0,0,0,0,....)
- That's a large vector!
- Remedies
  - limit to, say, 20,000 most frequent words, rest are OTHER
  - splitting rare words into subwords
  - character-based models

word embeddings

Add an Embedding Layer

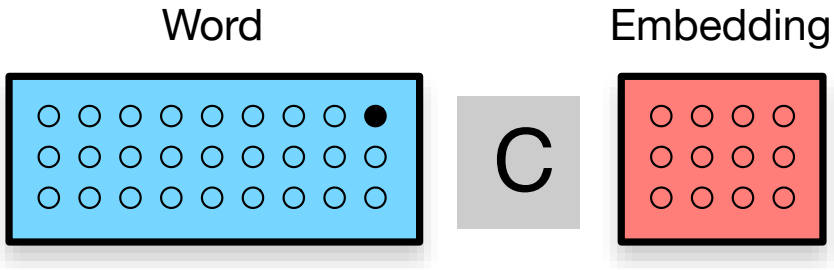


- Map each word first into a lower-dimensional real-valued space
- Shared weight matrix E

Details (Bengio et al., 2003)

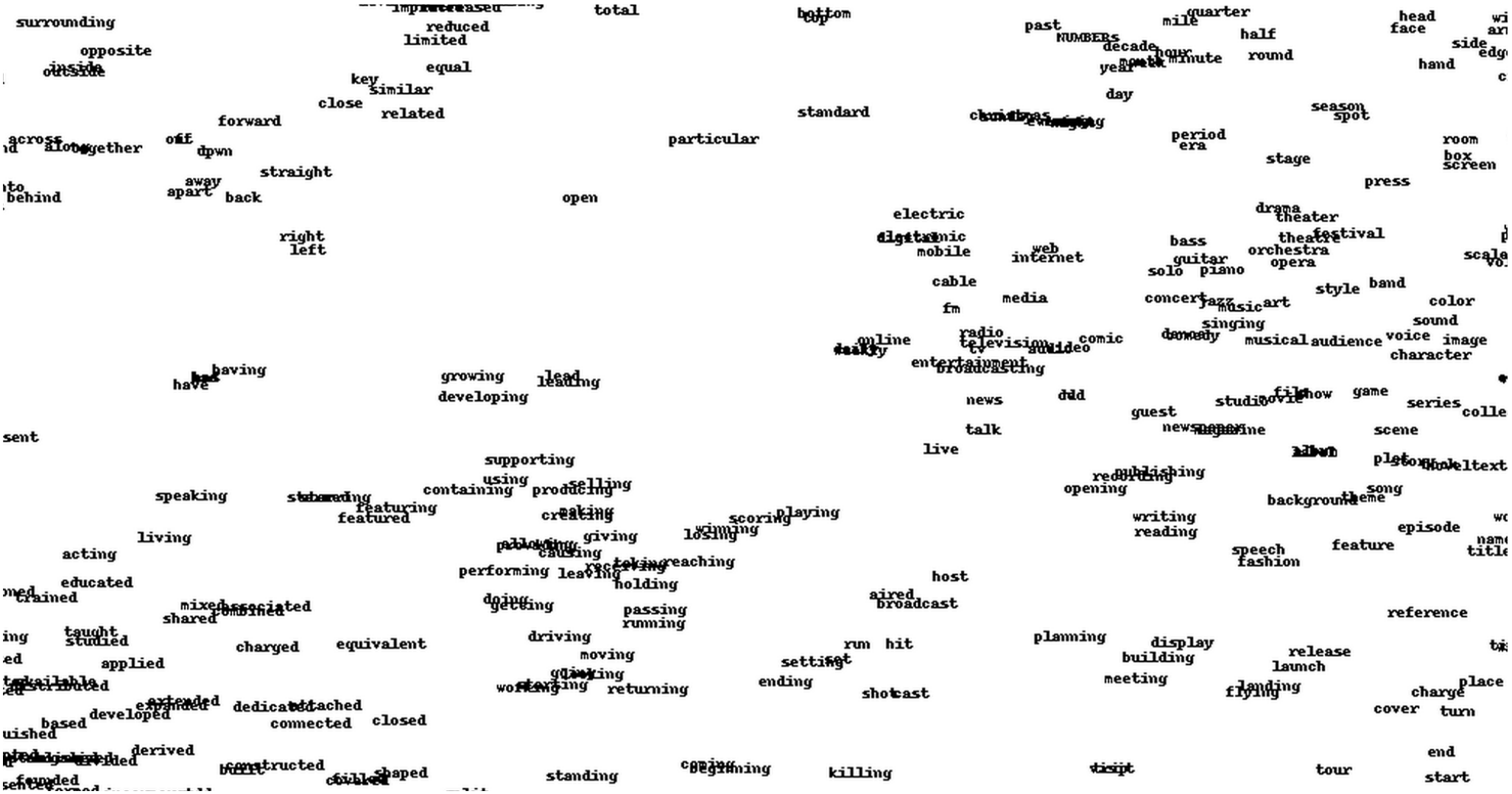
- Add direct connections from embedding layer to output layer
- Activation functions
  - input→embedding: none
  - embedding→hidden: tanh
  - hidden→output: softmax
- Training
  - loop through the entire corpus
  - update between predicted probabilities and 1-hot vector for output word

Word Embeddings

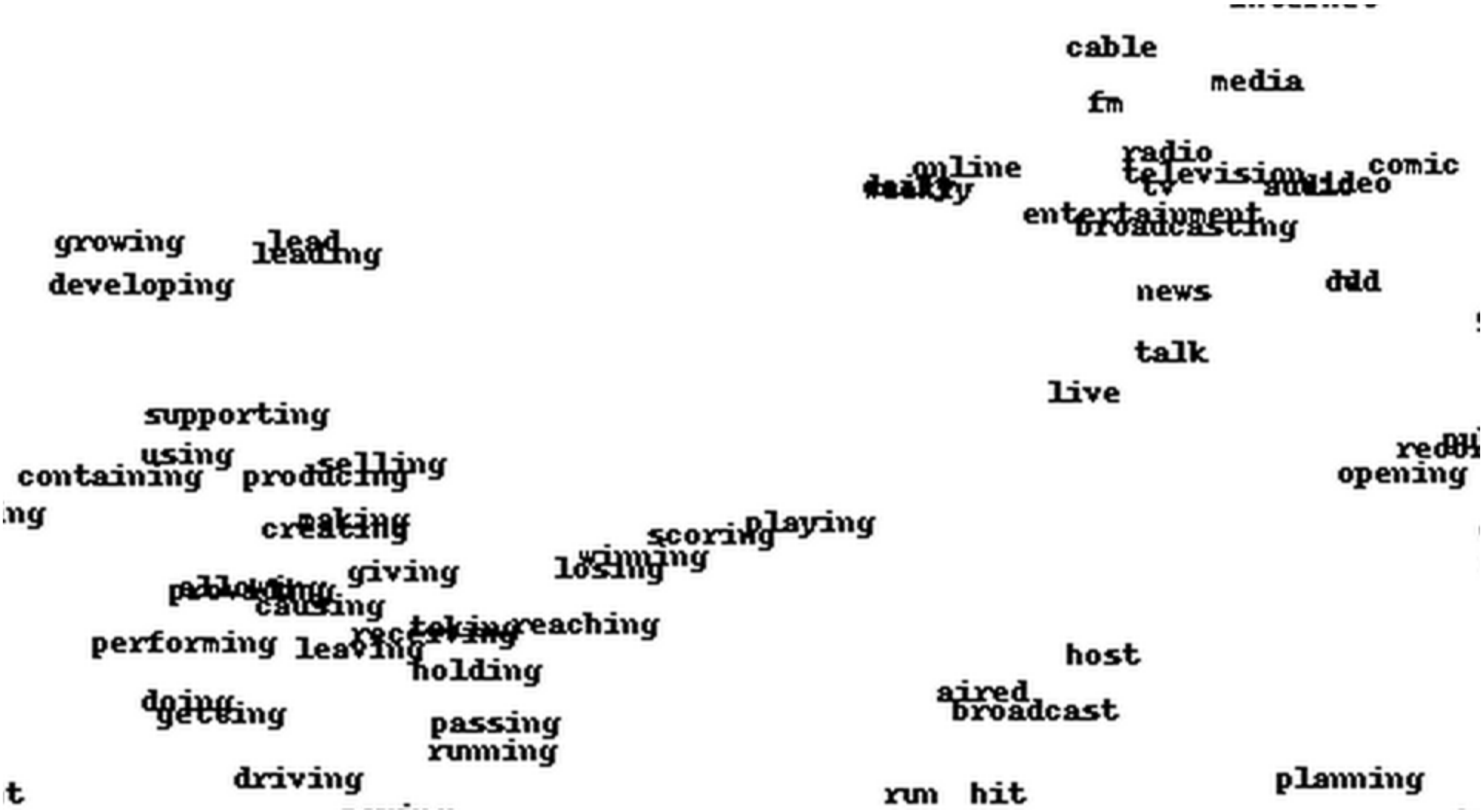


- By-product: embedding of word into continuous space
- Similar contexts $\rightarrow$ similar embedding
- Recall: distributional semantics

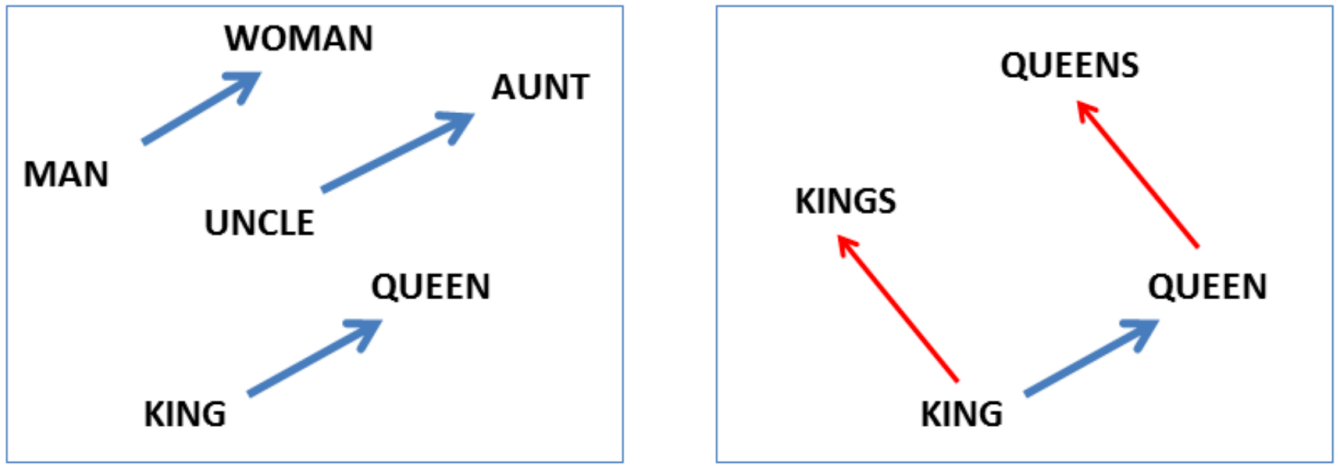
Word Embeddings



Word Embeddings



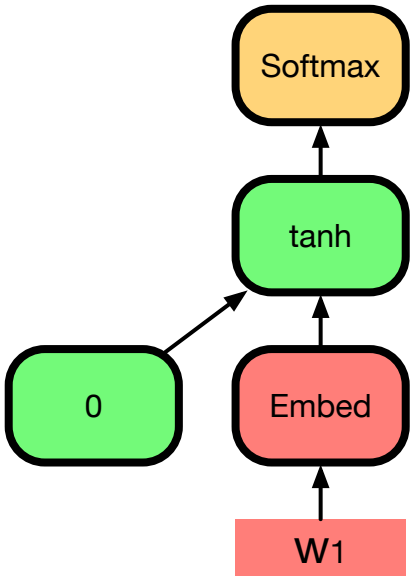
Are Word Embeddings Magic?



- Morphosyntactic regularities (Mikolov et al., 2013)
  - adjectives base form vs. comparative, e.g., good, better
  - nouns singular vs. plural, e.g., year, years
  - verbs present tense vs. past tense, e.g., see, saw
- Semantic regularities
  - clothing is to shirt as dish is to bowl
  - evaluated on human judgment data of semantic similarities

recurrent neural networks

Recurrent Neural Networks



Output Word

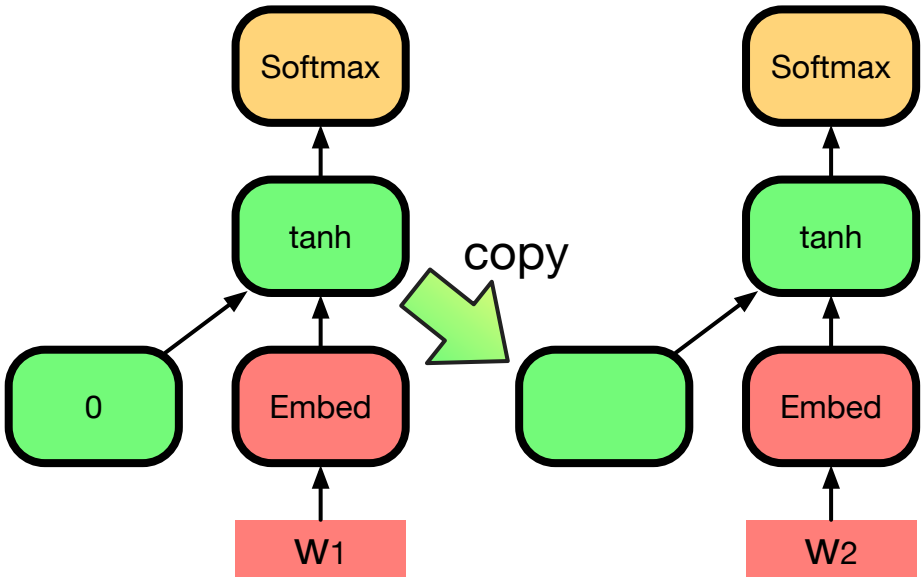
Hidden Layer

Embedding

History

- Start: predict second word from first
- Mystery layer with nodes all with value 1

Recurrent Neural Networks



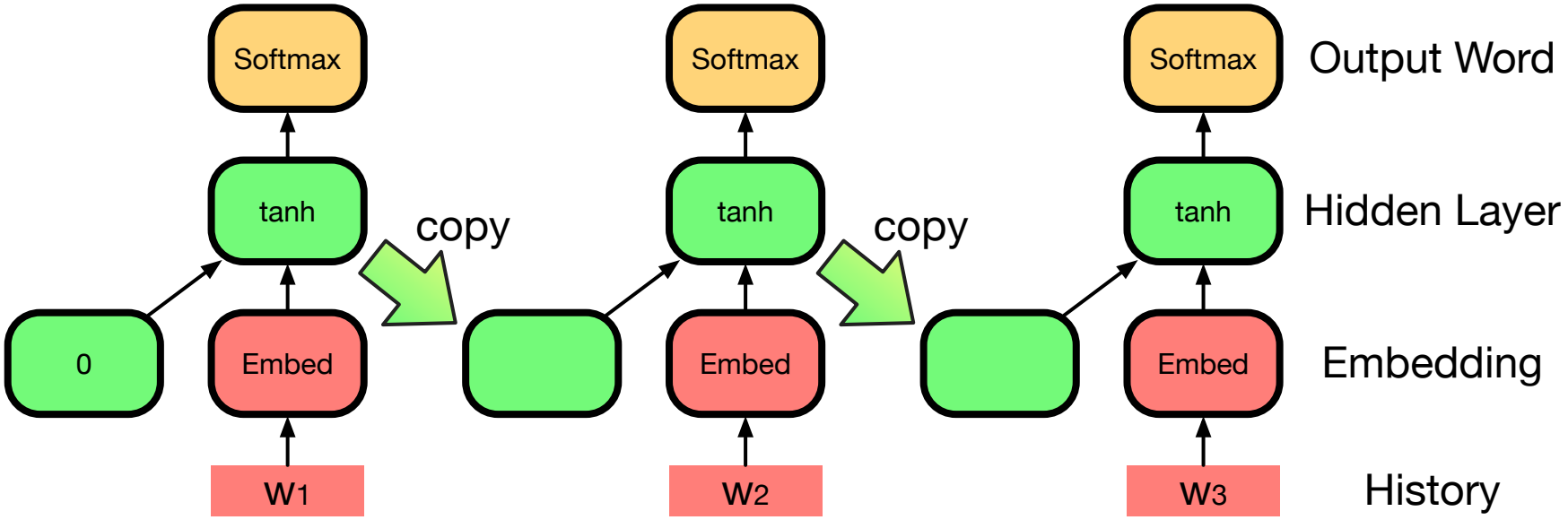
Output Word

Hidden Layer

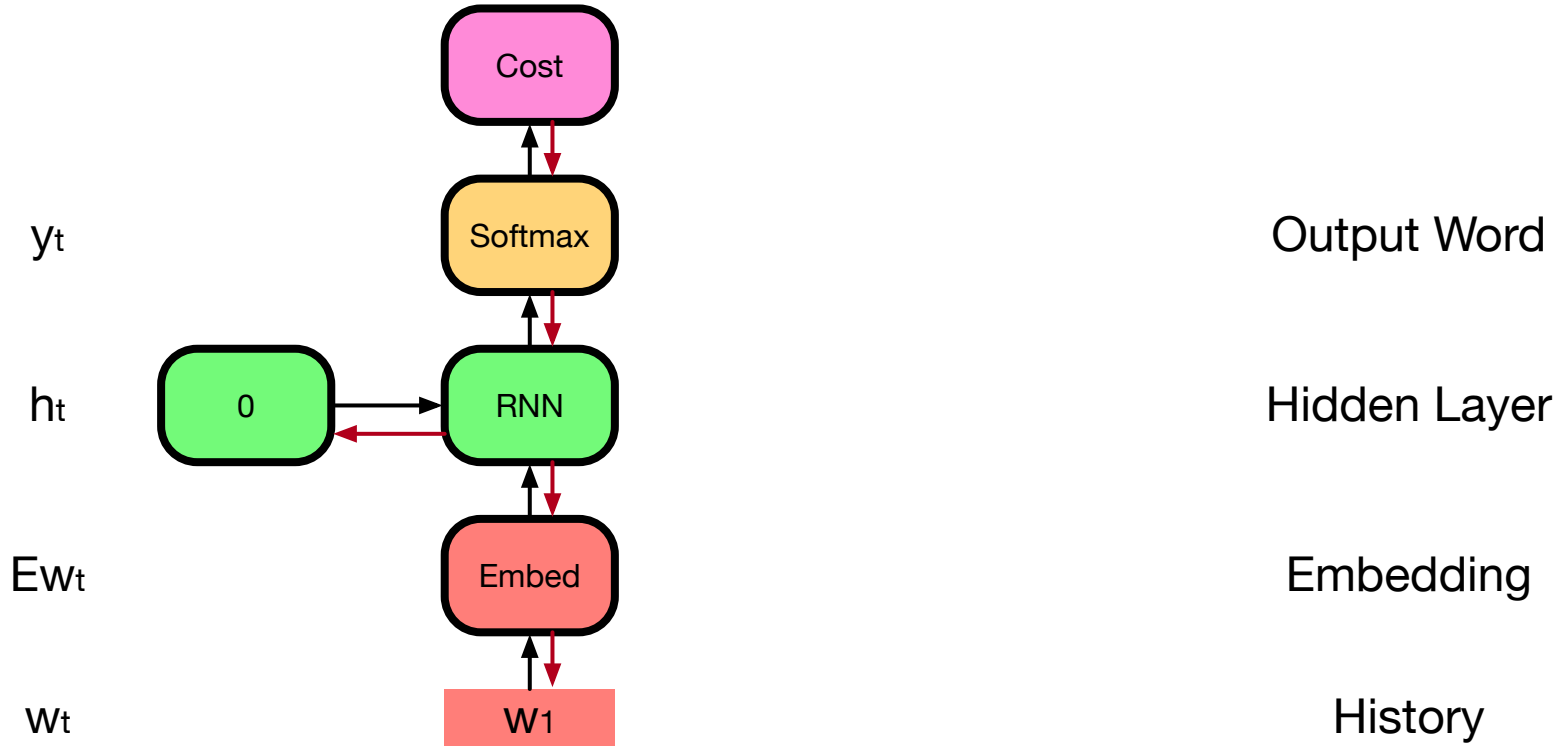
Embedding

History

Recurrent Neural Networks

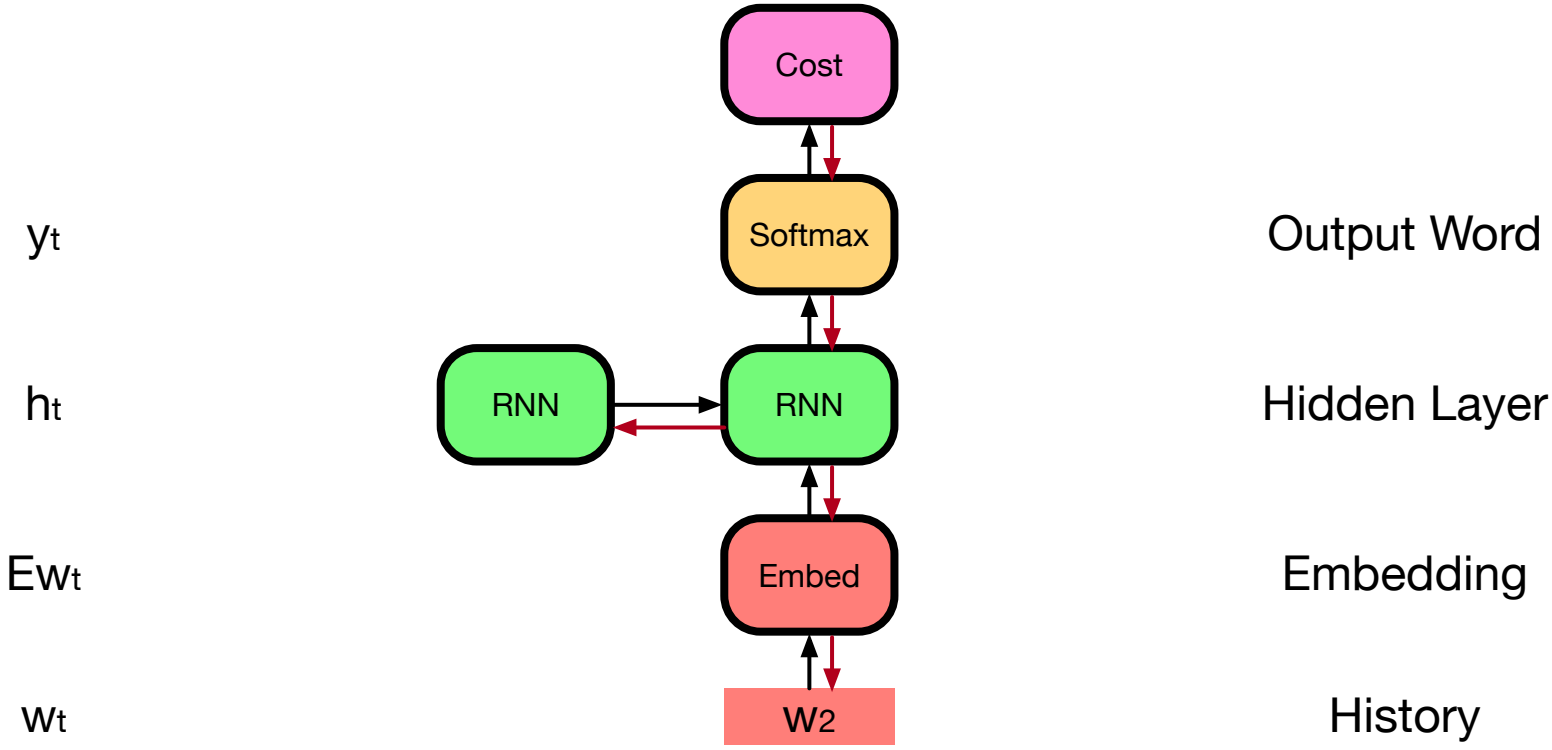


Training



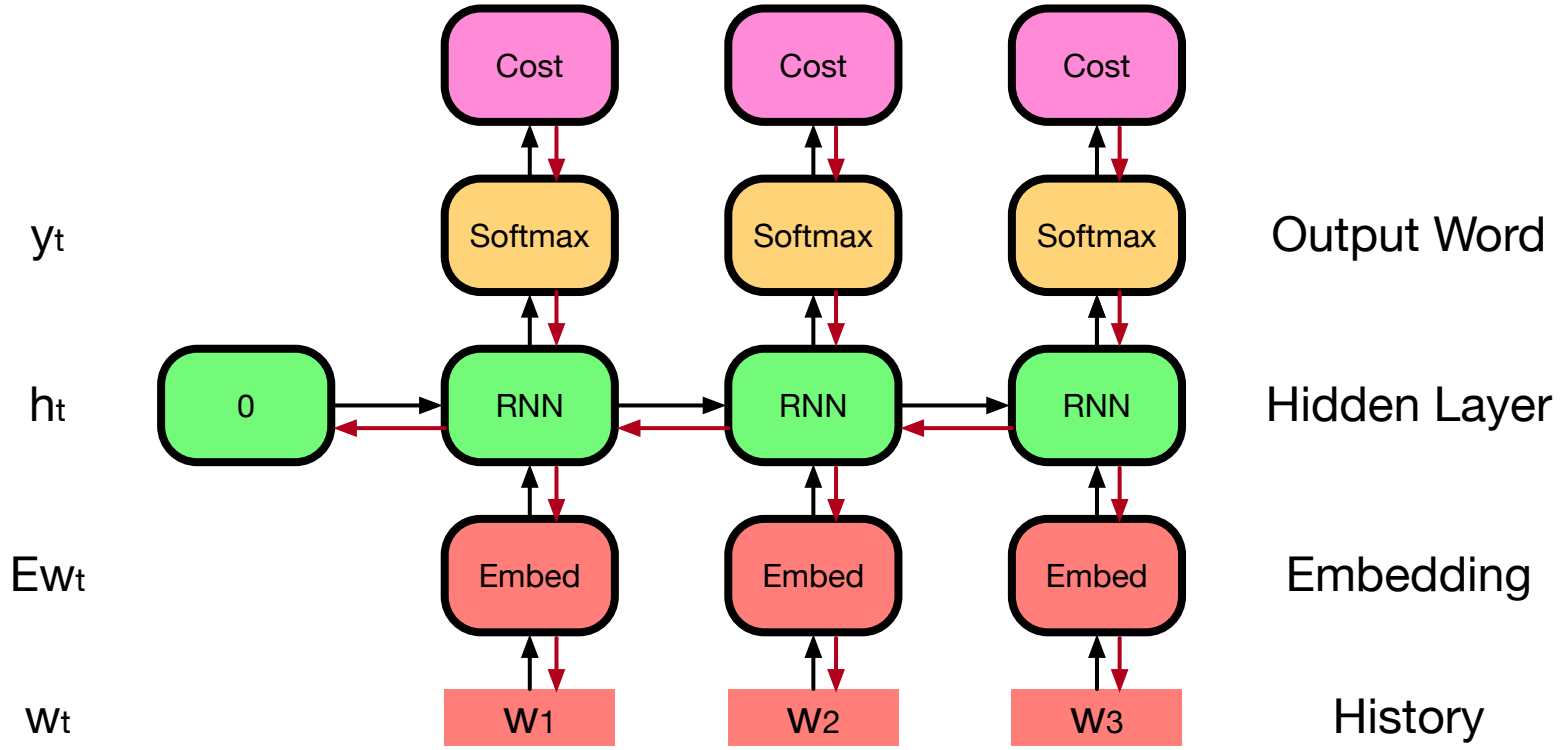
- Process first training example
- Update weights with back-propagation

Training



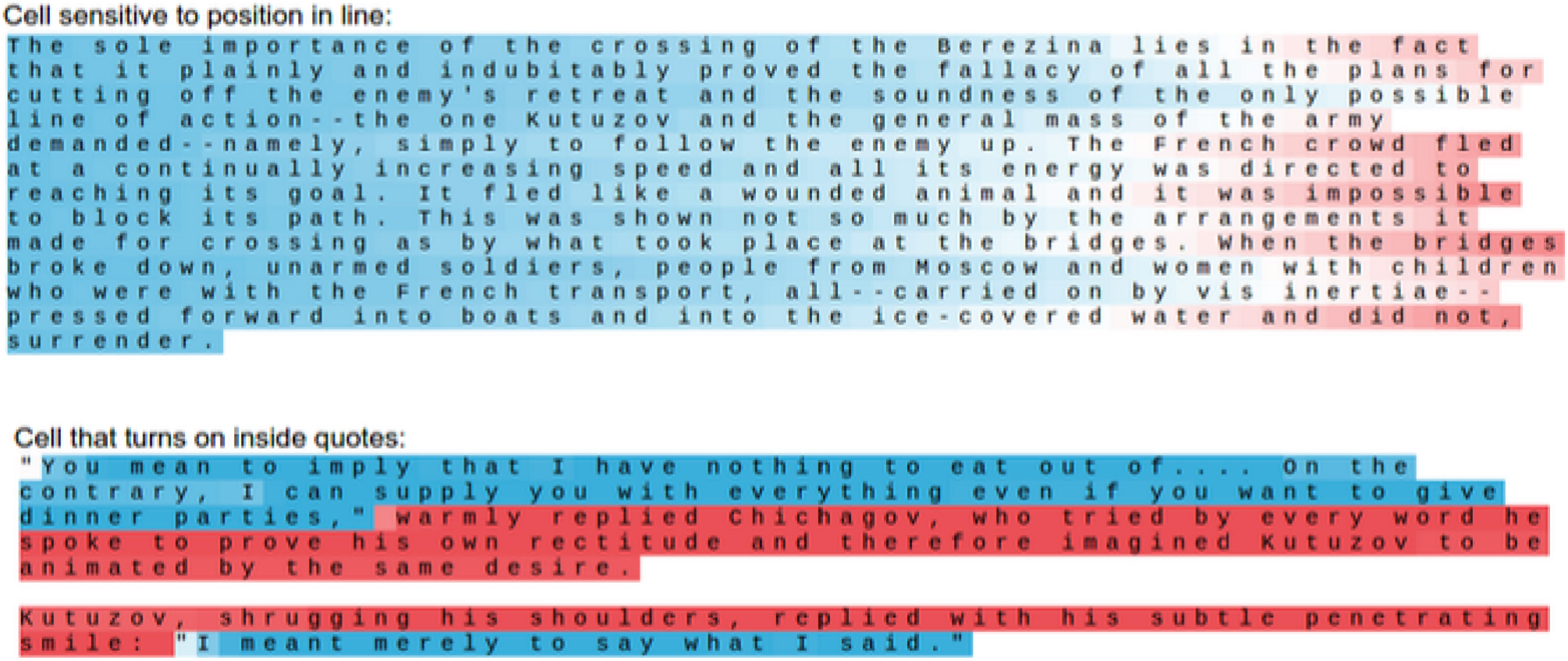
- Process second training example
- Update weights with back-propagation
- And so on...
- But: no feedback to previous history

Back-Propagation Through Time



- After processing a few training examples, update through the unfolded recurrent neural network

Visualizing Individual Cells



Karpathy et al. (2015): ”Visualizing and Understanding Recurrent Networks”

Visualizing Individual Cells

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

A large portion of cells are not easily interpretable. Here is a typical example:

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /* Of the currently implemented string fields, PATH_MAX
   * defines the longest valid length.
   */
```

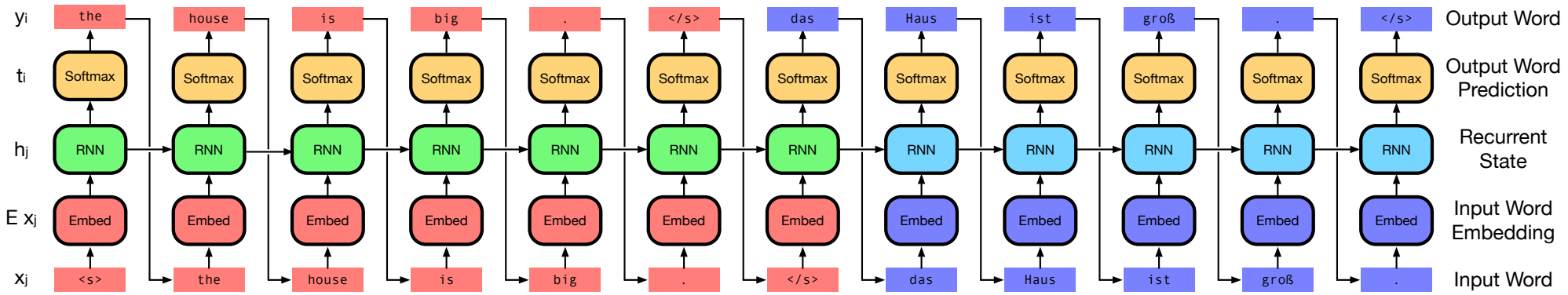
encoder-decoder models

Recurrent Neural Translation Model

- We predicted the words of a sentence

- Why not also predict their translations?

Encoder-Decoder Model



- Obviously madness
- Proposed by Google (Sutskever et al. 2014)

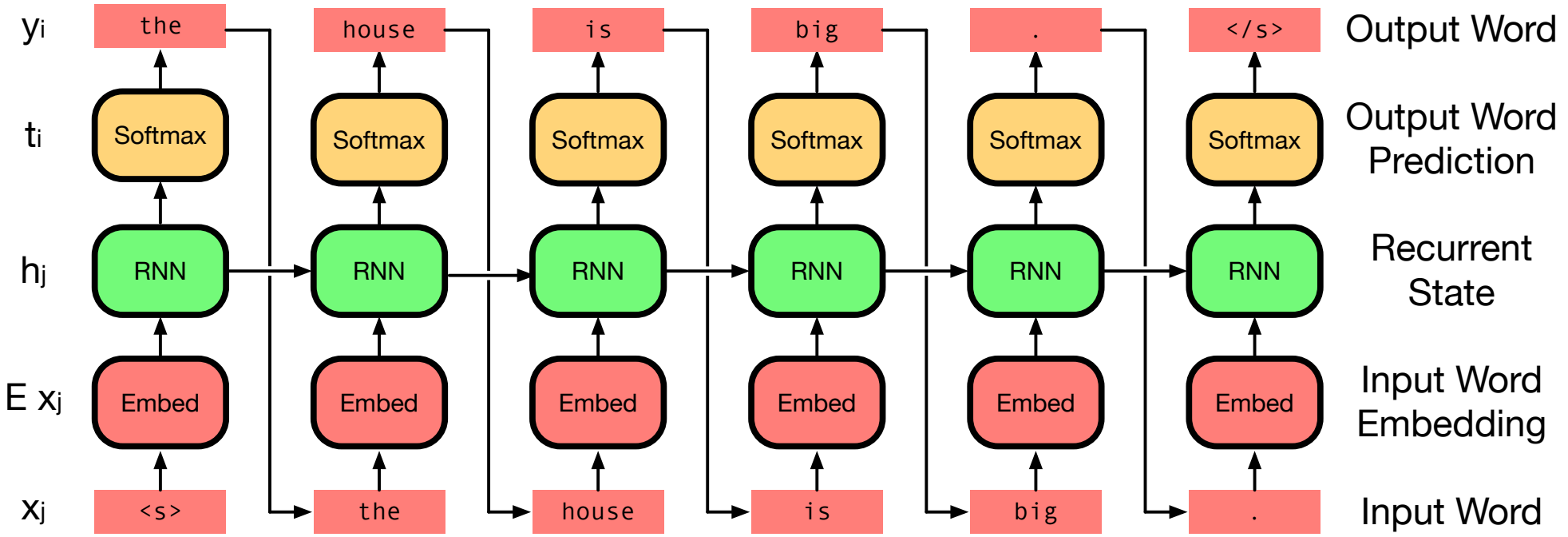
What is Missing?

- Alignment of input words to output words

$\Rightarrow$ Solution: attention mechanism

neural translation model with attention

Input Encoding



- Inspiration: recurrent neural network language model on the input side

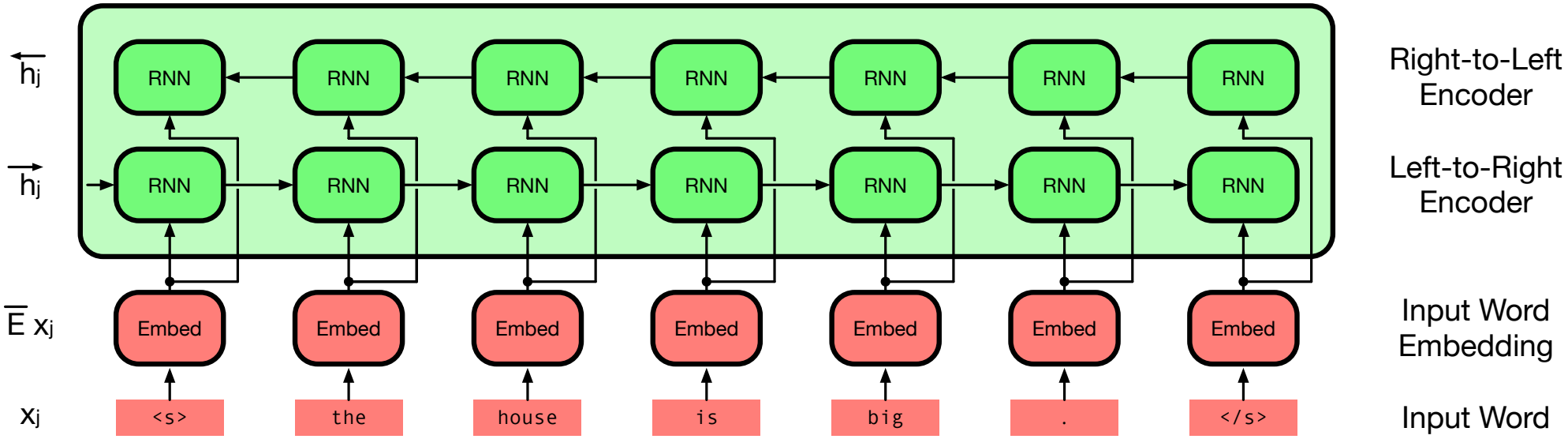
Hidden Language Model States

- This gives us the hidden states

- These encode left context for each word

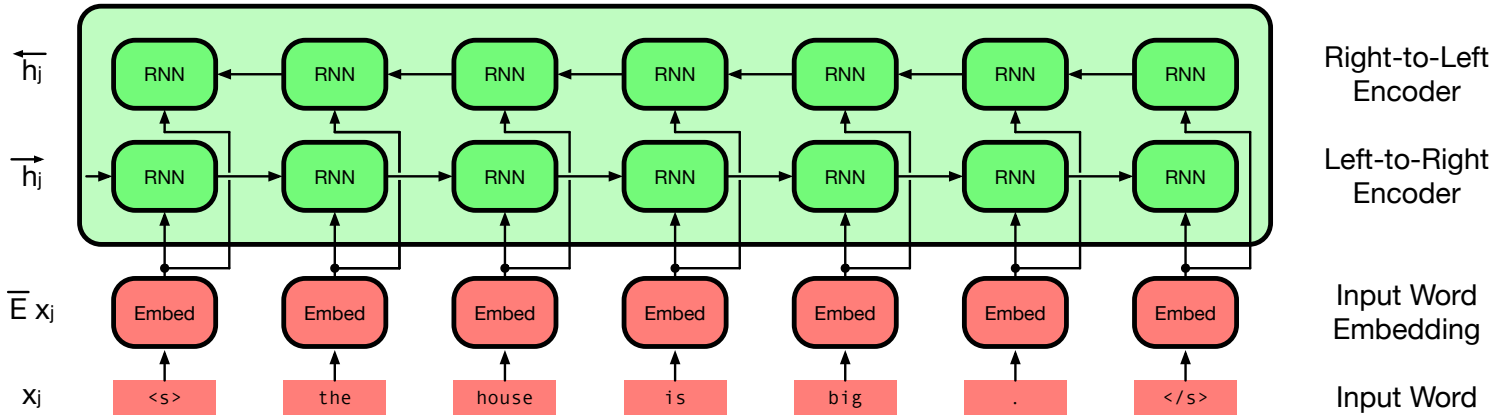
- Same process in reverse: right context for each word

Input Encoder



- Input encoder: concatenate bidirectional RNN states
- Each word representation includes full left and right sentence context

Encoder: Math



- Input is sequence of words x_j, mapped into embedding space $\bar{E}\ x_j$

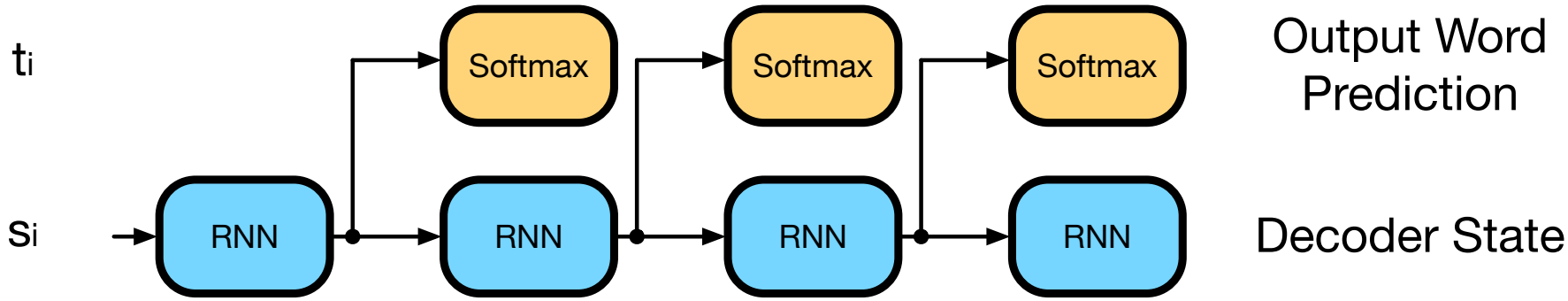
- Bidirectional recurrent neural networks

$$\overleftarrow{h_j} = f(\overleftarrow{h_{j+1}}, \bar{E}\ x_j)$$
$$\overrightarrow{h_j} = f(\overrightarrow{h_{j-1}}, \bar{E}\ x_j)$$

- Various choices for the function $f()$: feed-forward layer, GRU, LSTM, ...

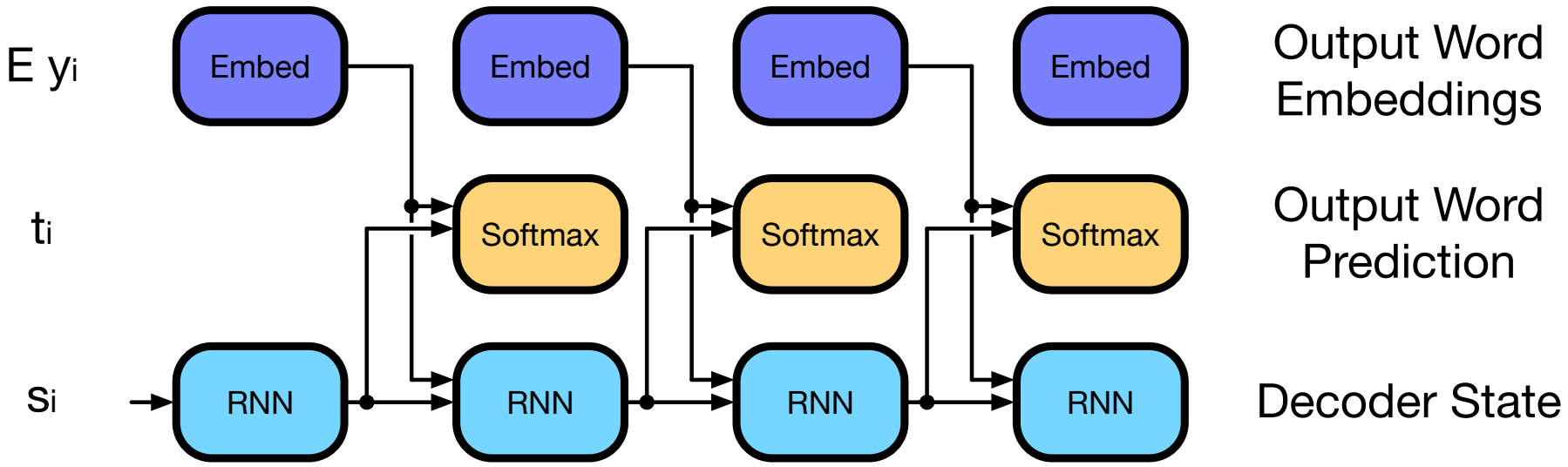
Decoder

- We want to have a recurrent neural network predicting output words



Decoder

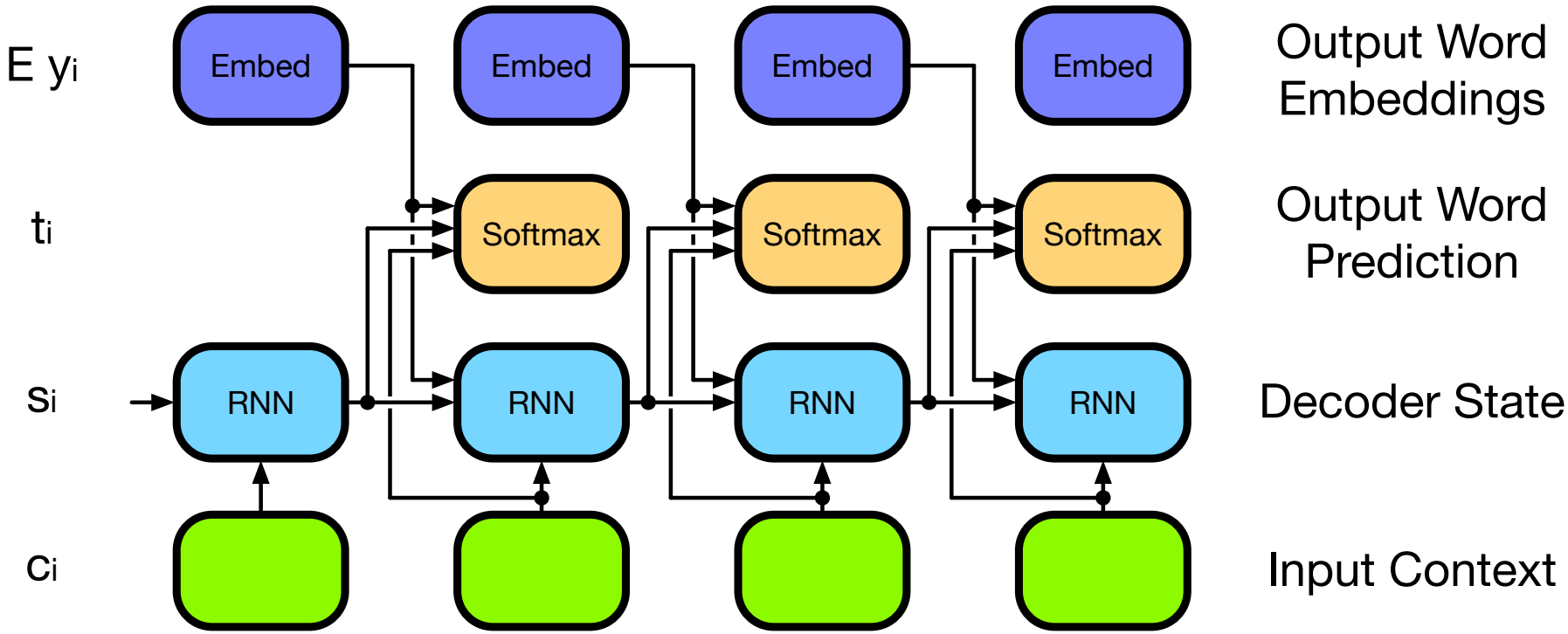
- We want to have a recurrent neural network predicting output words



- We feed decisions on output words back into the decoder state

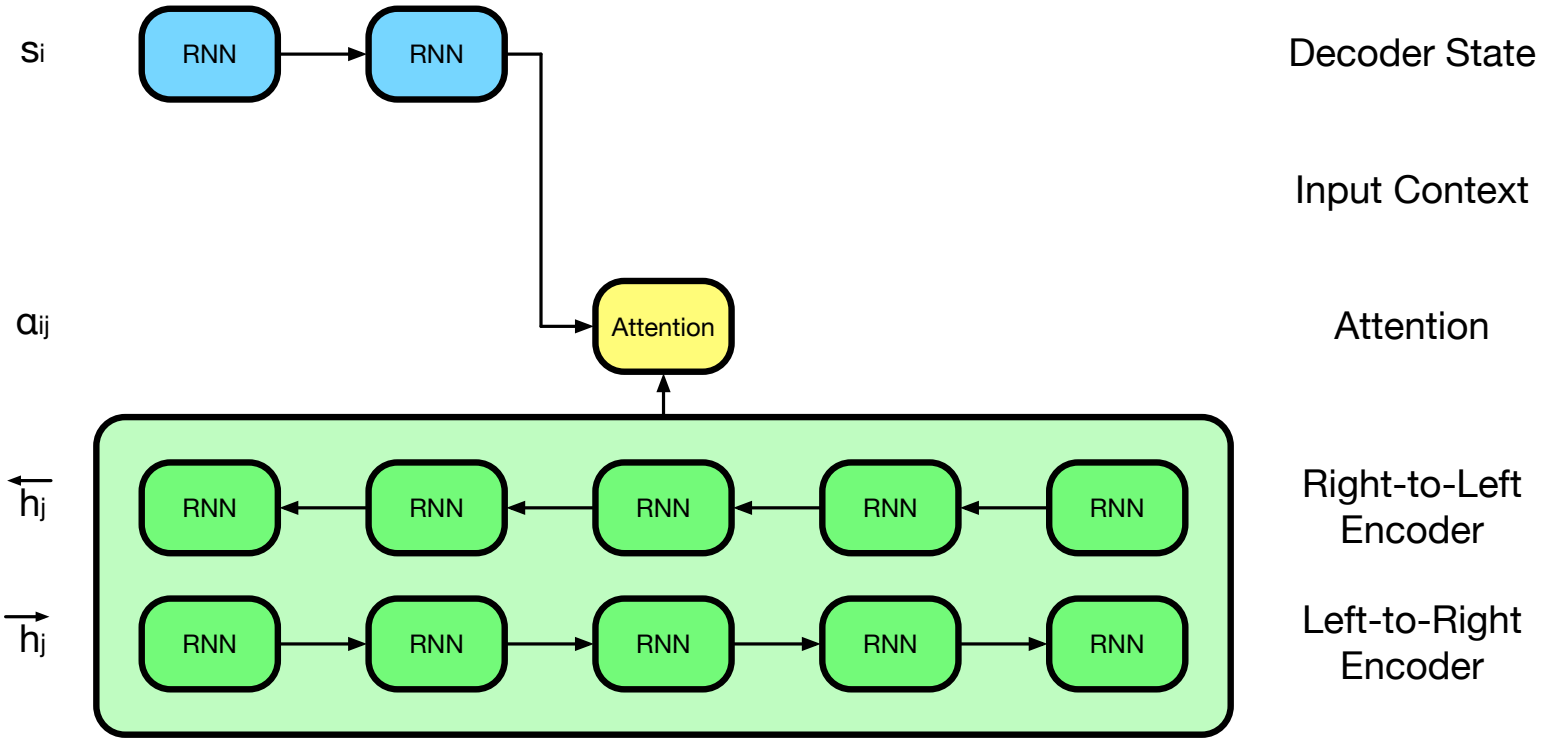
Decoder

- We want to have a recurrent neural network predicting output words



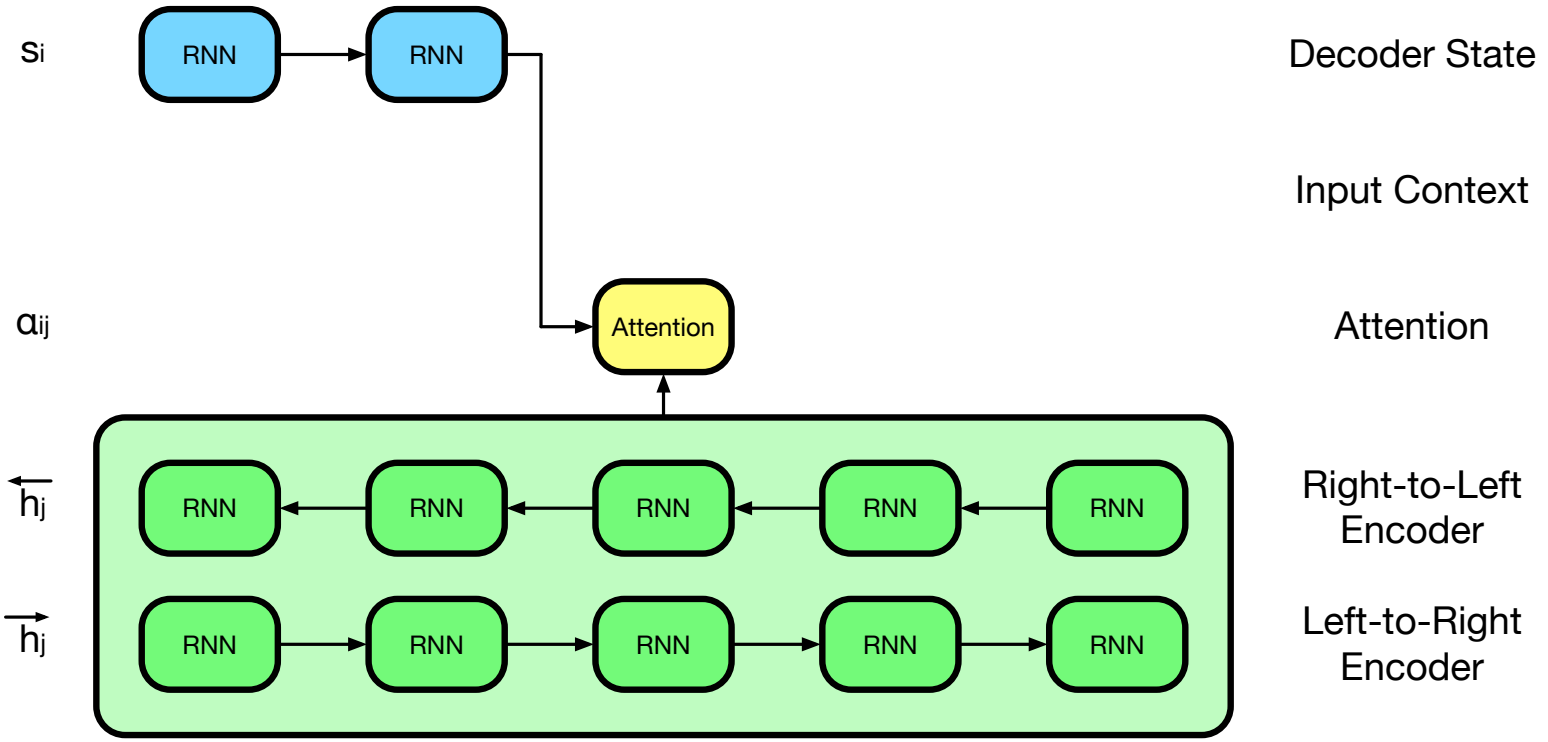
- We feed decisions on output words back into the decoder state
- Decoder state is also informed by the input context

Attention



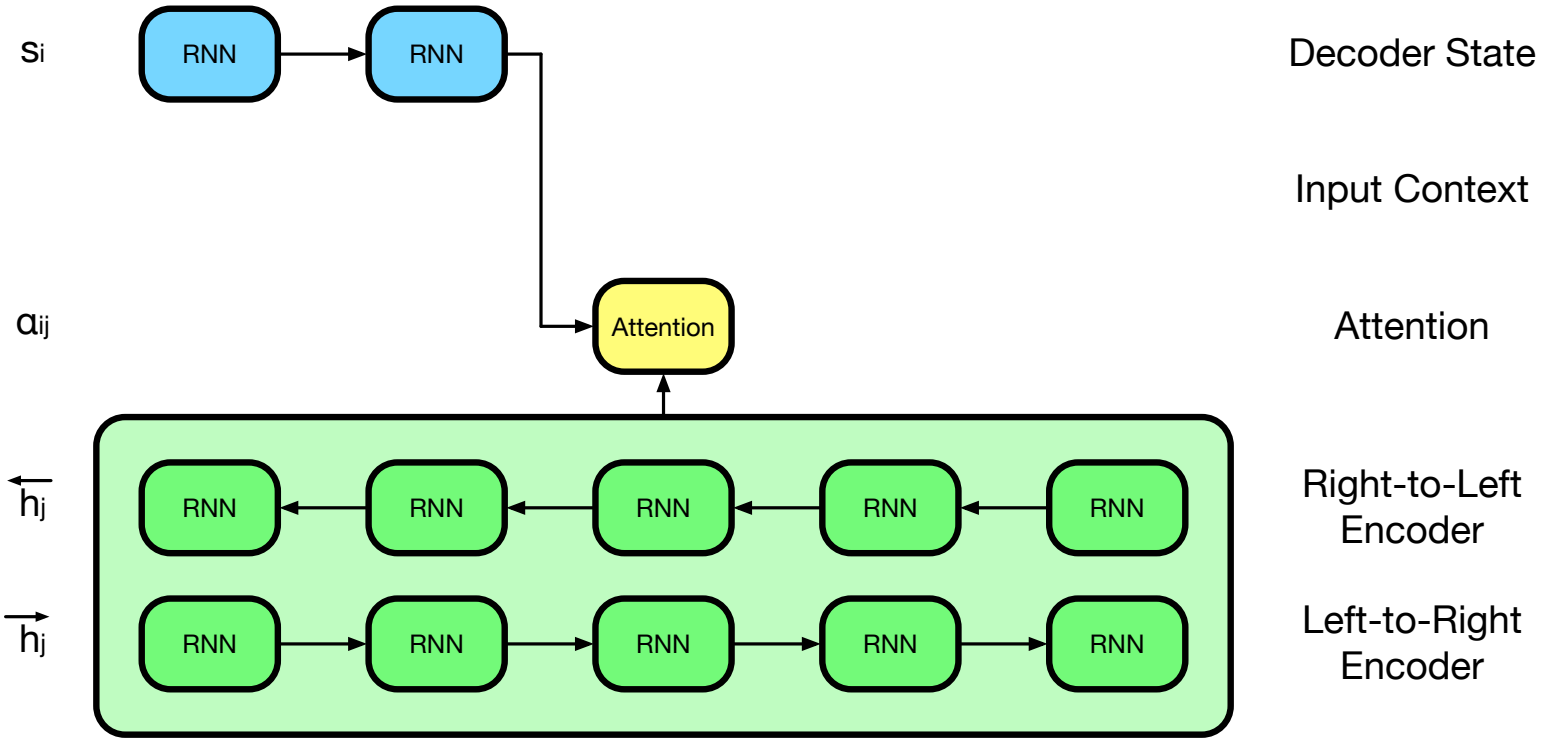
- Given what we have generated so far (decoder hidden state)
- ... which words in the input should we pay attention to (encoder states)?

Attention



- Given: – the previous hidden state of the decoder s_{i-1}
  – the representation of input words $h_j = (\overleftarrow{h_j}, \overrightarrow{h_j})$
- Predict an alignment probability $a(s_{i-1}, h_j)$ to each input word j (modeled with with a feed-forward neural network layer)

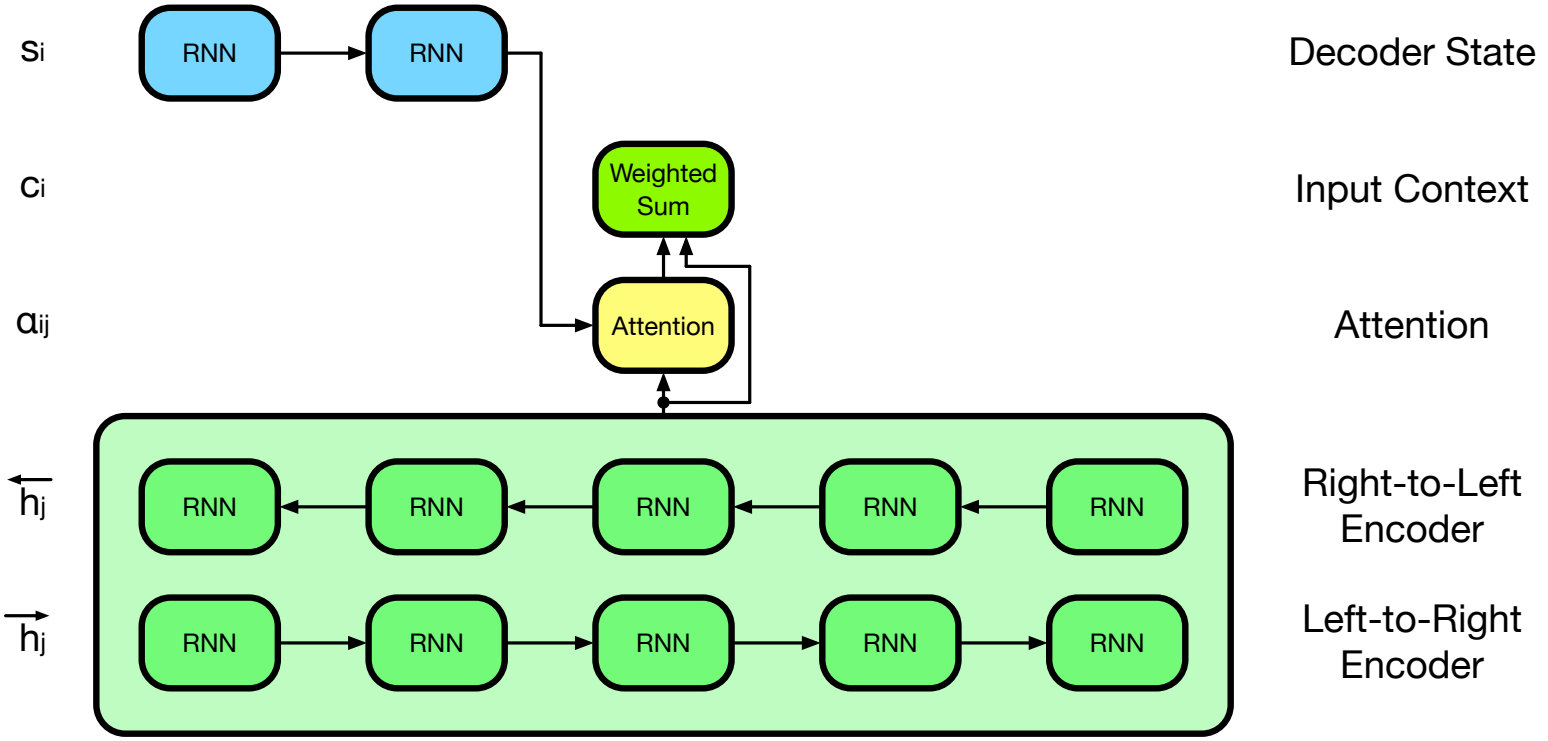
Attention



- Normalize attention (softmax)

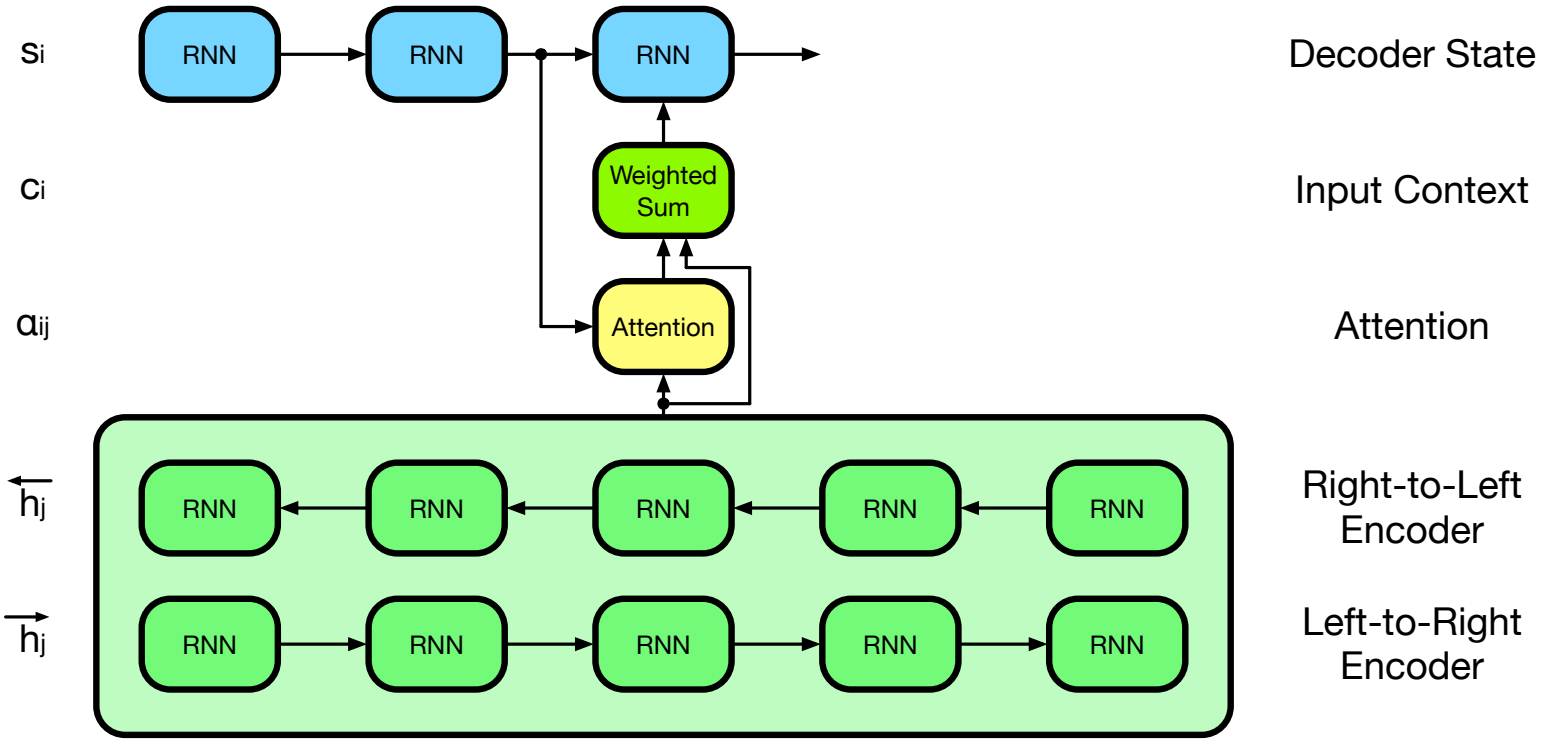
$$\alpha_{ij} = \frac{\exp(a(s_{i-1}, h_j))}{\sum_k \exp(a(s_{i-1}, h_k))}$$

Attention



- Relevant input context: weigh input words according to attention: $c_i = \sum_j \alpha_{ij} h_j$

Attention



- Use context to predict next hidden state and output word

attention

Attention

- Machine translation is a structured prediction task
  - output is not a single label
  - output structure needs to be built, word by word
- Relevant information for each word prediction varies
- Human translators pay attention to different parts of the input sentence when translating

$\Rightarrow$ Attention mechanism

Computing Attention

- Attention mechanism in neural translation model (Bahdanau et al., 2015)
  - previous hidden state s_{i-1}
  - input word embedding h_j
  - trainable parameters b, W_a, U_a, v_a

$$a(s_{i-1}, h_j) = v_a^T \tanh(W_a s_{i-1} + U_a h_j + b)$$

- Other ways to compute attention
  - Dot product: $a(s_{i-1}, h_j) = s_{i-1}^T h_j$
  - Scaled dot product: $a(s_{i-1}, h_j) = \frac{1}{\sqrt{|h_j|}} s_{i-1}^T h_j$
  - General: $a(s_{i-1}, h_j) = s_{i-1}^T W_a h_j$
  - Local: $a(s_{i-1}) = W_a s_{i-1}$

Attention of Luong et al. (2015)

- Luong et al. (2015) demonstrate good results with the dot product

$$a(s_{i-1}, h_j) = s_{i-1}^T h_j$$

- No trainable parameters
- Additional changes
- Currently more popular

General View of Dot-Product Attention

- Three element

  Query : decoder state
  Key : encoder state
  Value : encoder state

- Intuition

  - given a query (the decoder state)
  - we check how well it matches keys in the database (the encoder states)
  - and then use the matching score to scale the retrieved value (also the encoder state)

- Computation

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Scaled Dot-Product Attention

- Refinement of query, key, and value
- Scale it down to lower-dimensional vectors (e.g., 512 from 4096)
- Using a weight matrix for each: QW^Q, KW^K, VW^V

Multi-Head Attention

- Add redundancy
  - say, 16 attention weights
  - each based on its own parameters W
  - matrix W also reduces the dimensionality

- Formally:

$$\begin{aligned} \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \\ \text{MultiHead}(Q,K,V) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \end{aligned}$$

- Multi-head attention is a form of ensembling

Self Attention

- Finally, a very different take at attention
- Motivation so far: need for alignment between input words and output words
- Now: refine representation of input words in the encoder
  - representation of an input word mostly depends on itself
  - but also informed by the surrounding context
  - previously: recurrent neural networks (considers left or right context)
  - now: attention mechanism
- Self attention:

  Which of the surrounding words is most relevant to refine representation?

Self Attention

- Formal definition (based on sequence of vectors h_j, packed into matrix H

$$\text{self-attention}(H) = \text{Attention}(HW_i^Q, HW_i^K, HW_i^V)$$

- Association between every word representation h_j any other context word h_k
- Resulting vector of normalized association values used to weigh context words

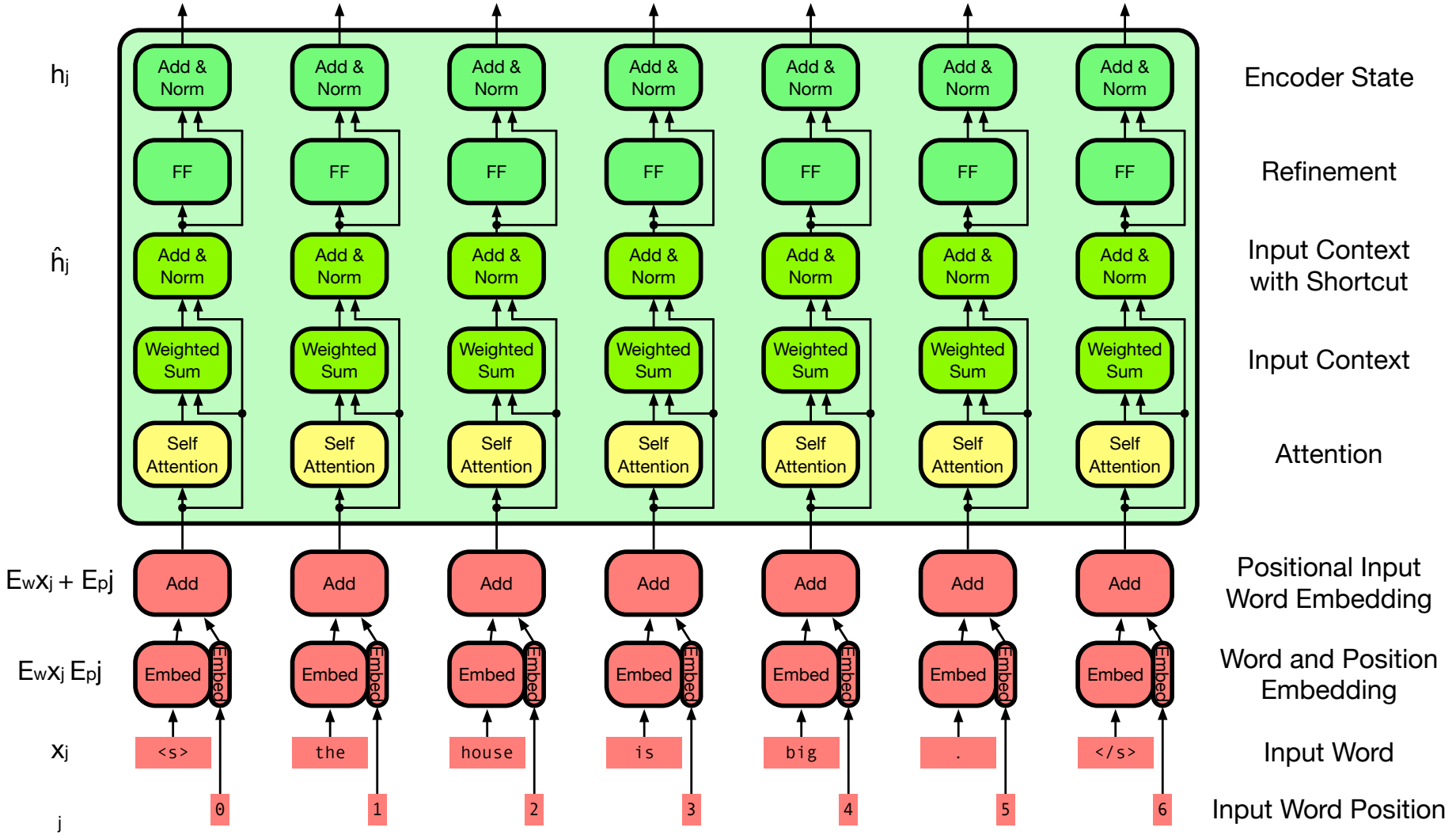
transformer

Self Attention: Transformer

- Self-attention in encoder
  - refine word representation based on relevant context words
  - relevance determined by self attention
- Self-attention in decoder
  - refine output word predictions based on relevant previous output words
  - relevance determined by self attention
- Also regular attention to encoder states in decoder
- Currently most successful model

  (maybe only with self attention in decoder, but regular recurrent decoder)

Encoder



Sequence of self-attention layers

Self Attention Layer

- Given: input word representations h_j, packed into a matrix $H = (h_1, ..., h_j)$
- Self attention

$$\text{self-attention}(H) = \text{MultiHead}(H, H, H)$$

- Shortcut connection

$$\text{self-attention}(h_j) + h_j$$

- Layer normalization

$$\hat{h}_j = \text{layer-normalization}(\text{self-attention}(h_j) + h_j)$$

- Feed-forward step with ReLU activation function

$$\text{relu}(W\hat{h}_j + b)$$

- Again, shortcut connection and layer normalization

$$\text{layer-normalization}(\text{relu}(W\hat{h}_j + b) + \hat{h}_j)$$

Stacked Self Attention Layers

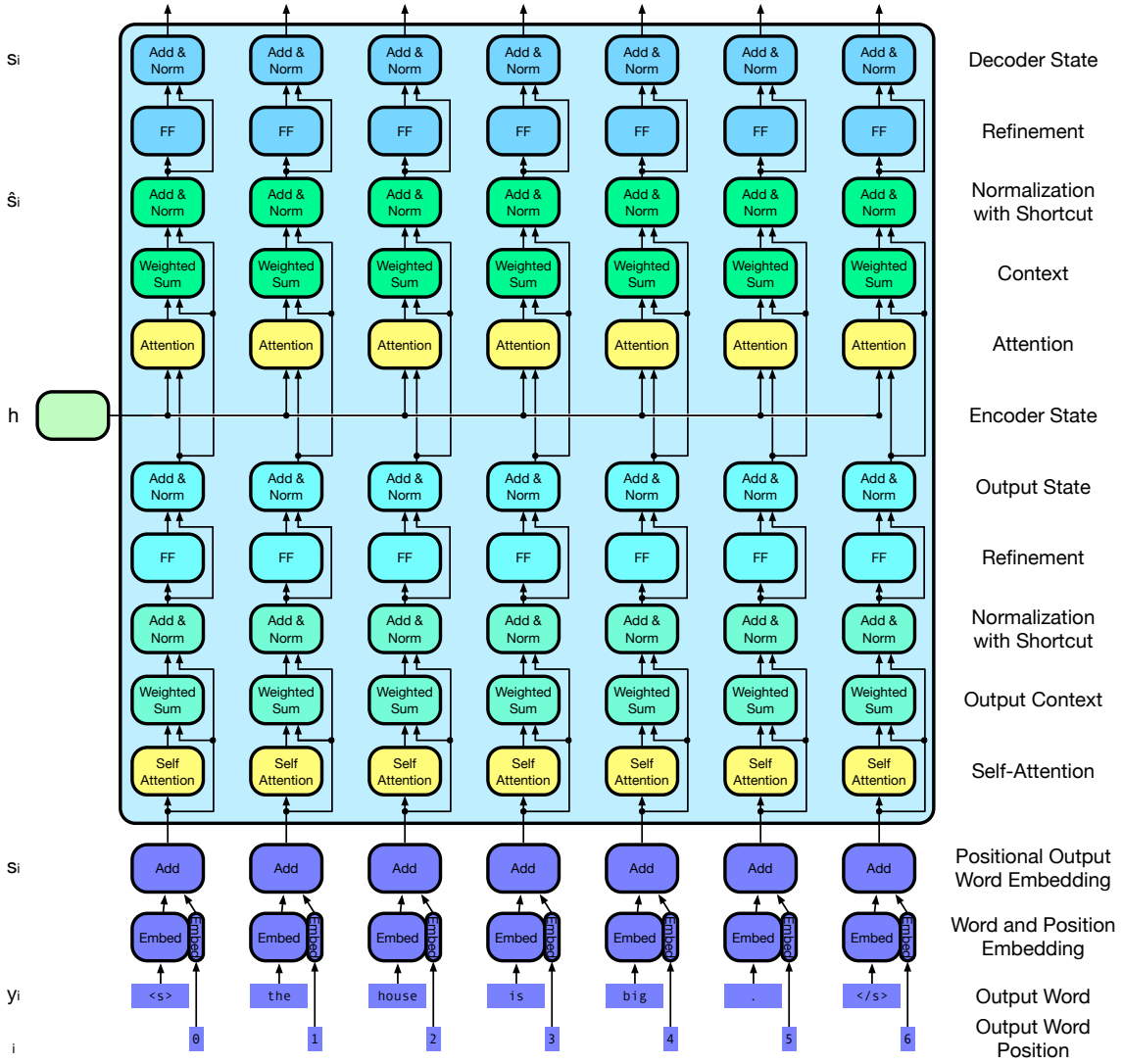
- Stack several such layers (say, $D = 6$)
- Start with input word embedding

$$h_{0,j} = Ex_j$$

- Stacked layers

$$h_{d,j} = \text{self-attention-layer}(h_{d-1,j})$$

Decoder



Decoder computes attention-based representations of the output in several layers, initialized with the embeddings of the previous output words

Self-Attention in the Decoder

- Same idea as in the encoder
- Output words are initially encoded by word embeddings $s_i = Ey_i$.
- Self attention is computed over previous output words
  - association of a word s_i is limited to words s_k ($k \leq i$)
  - resulting representation $\tilde{s}_i$

$$\text{self-attention}(\tilde{S}) = \text{MultiHead}(\tilde{S}, \tilde{S}, \tilde{S})$$

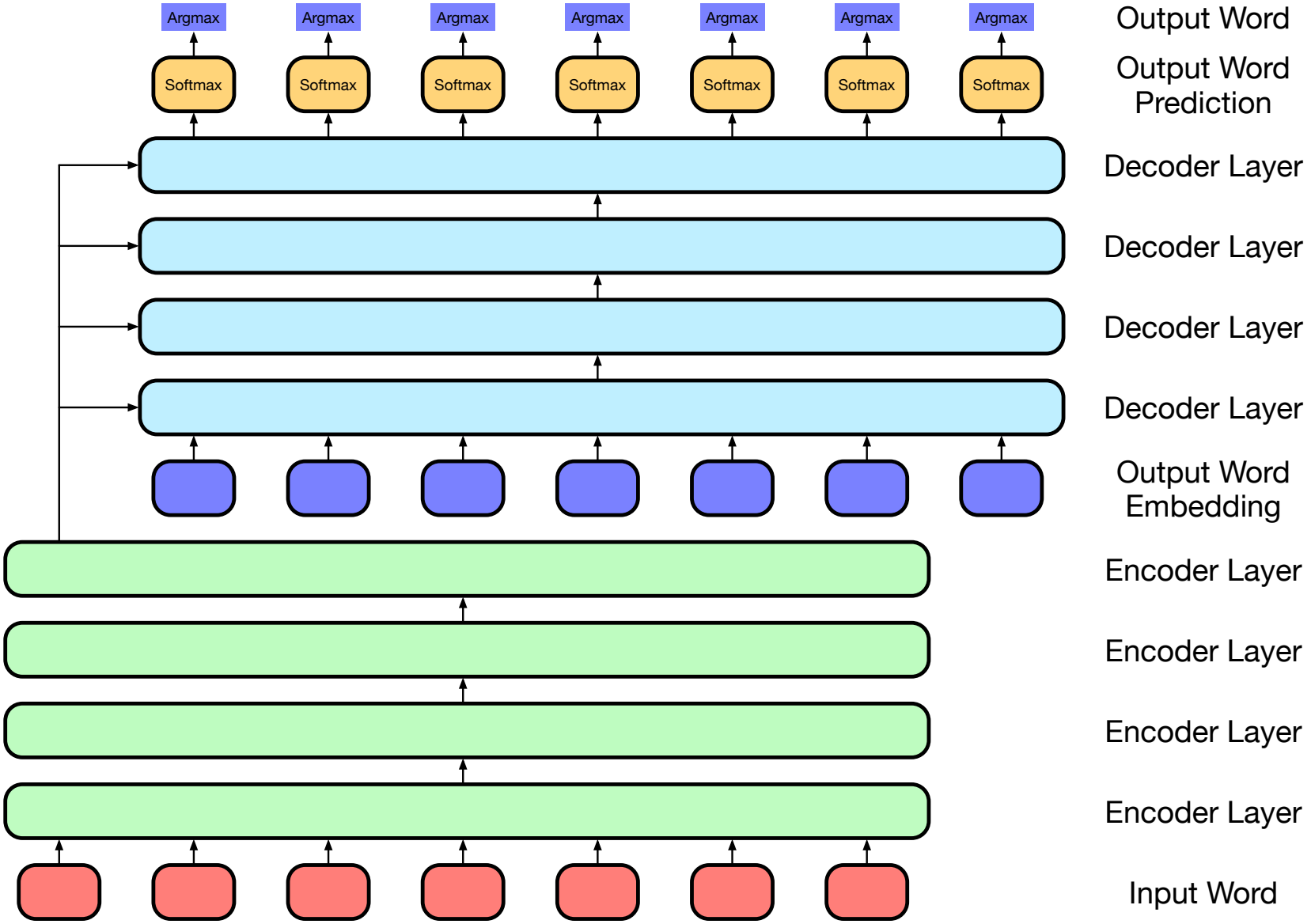
Attention in the Decoder

- Original intuition of attention mechanism: focus on relevant input words
- Compute attention between the decoder states $\tilde{S}$ and the final encoder states H

$$\text{attention}(\tilde{S}, H) = \text{MultiHead}(\tilde{S}, H, H)$$

- Note: attention mechanism formally mirrors self-attention

Full Decoder



Full Decoder

- Self-attention

$$\text{self-attention}(\tilde{S}) = \text{MultiHead}(\tilde{S}, \tilde{S}, \tilde{S})$$

  - shortcut connections
  - layer normalization
  - feed-forward layer

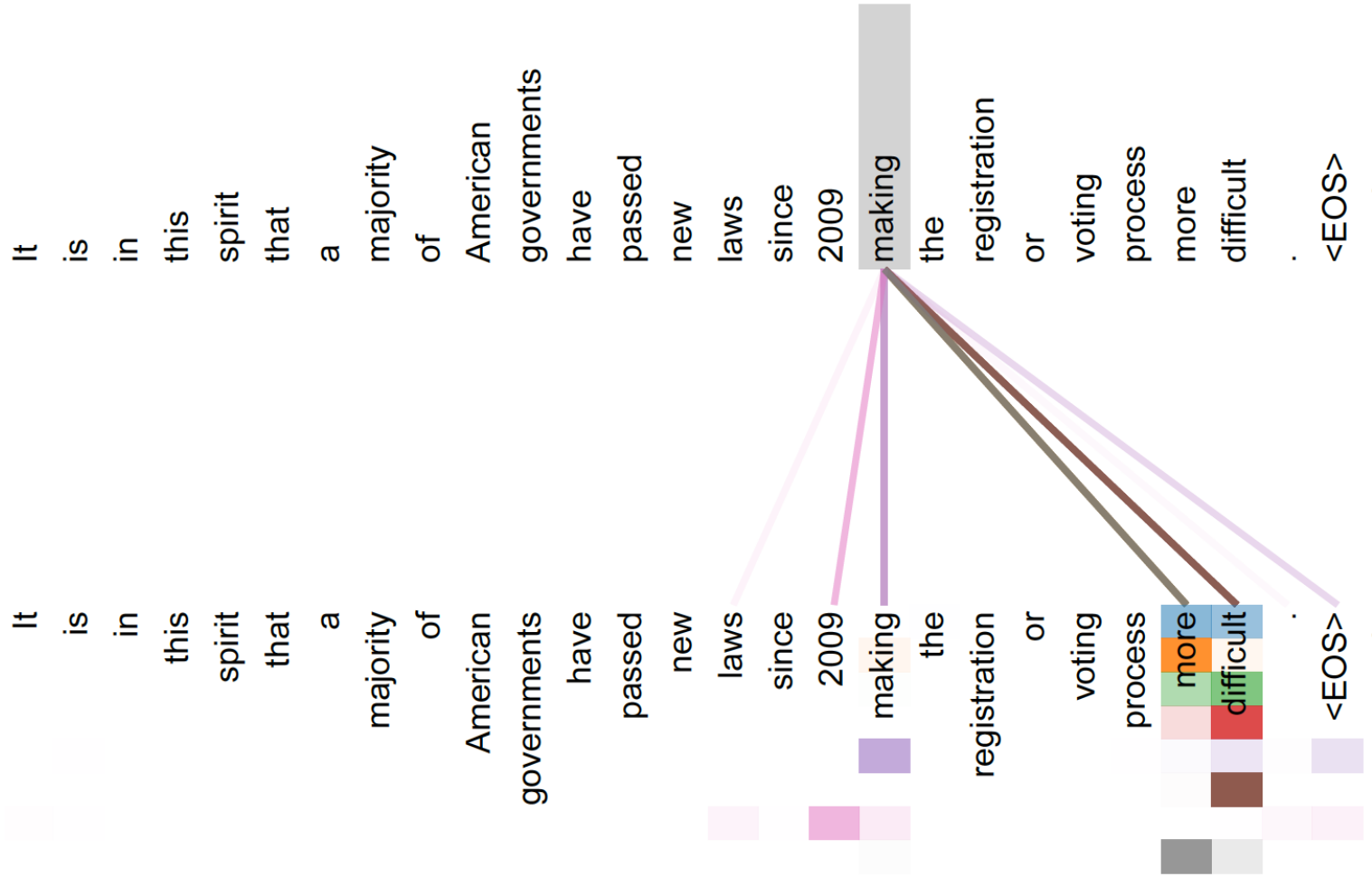
- Attention

$$\text{attention}(\tilde{S}, H) = \text{softmaxMultiHead}(\tilde{S}, H, H)$$

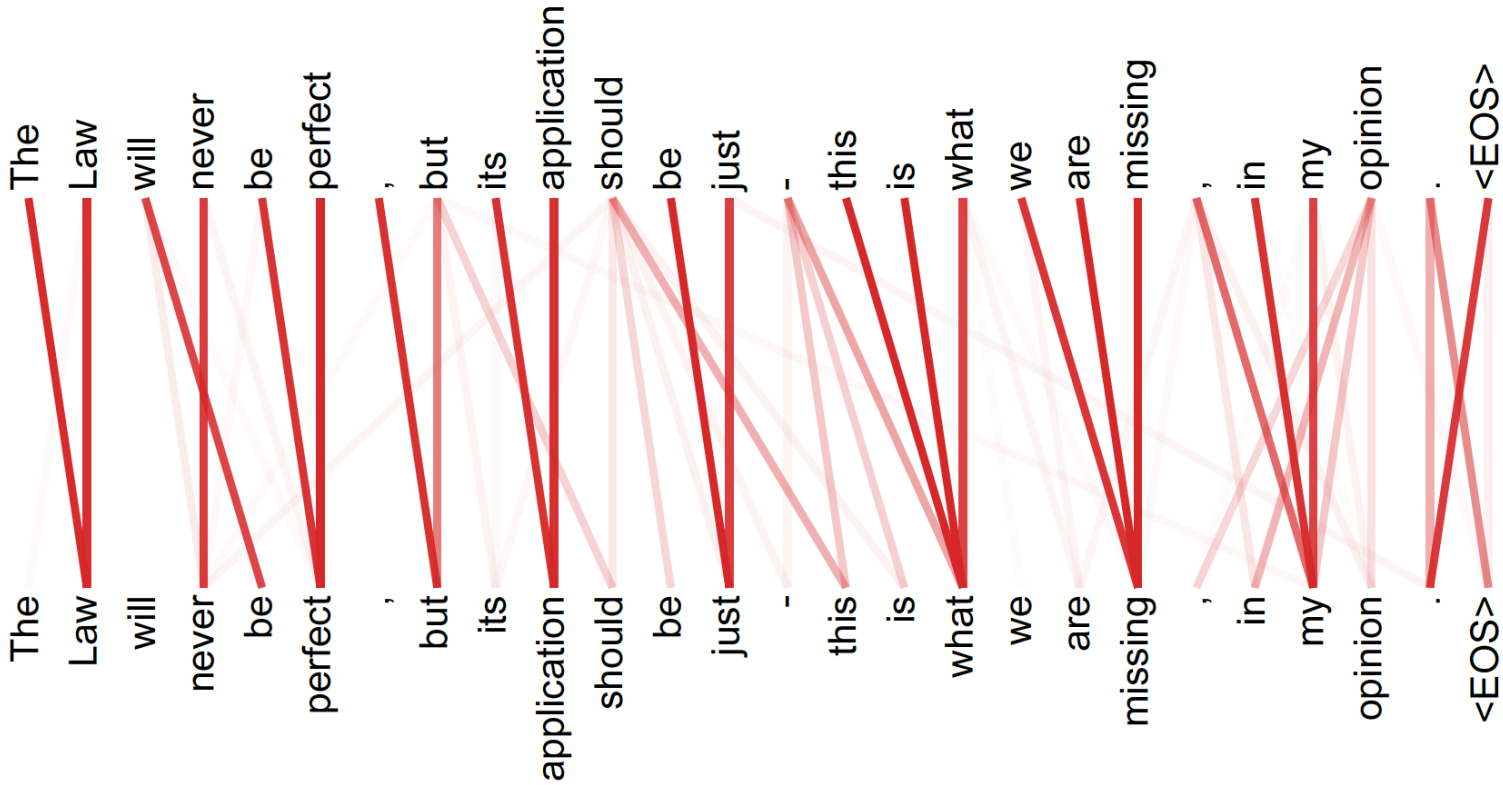
  - shortcut connections
  - layer normalization
  - feed-forward layer

- Multiple stacked layers

Multi-Head Attention



Multi-Head Attention



"Many of the attention heads exhibit behaviour that seems related to the structure of the sentence."

large language models

Masked Language Model Training

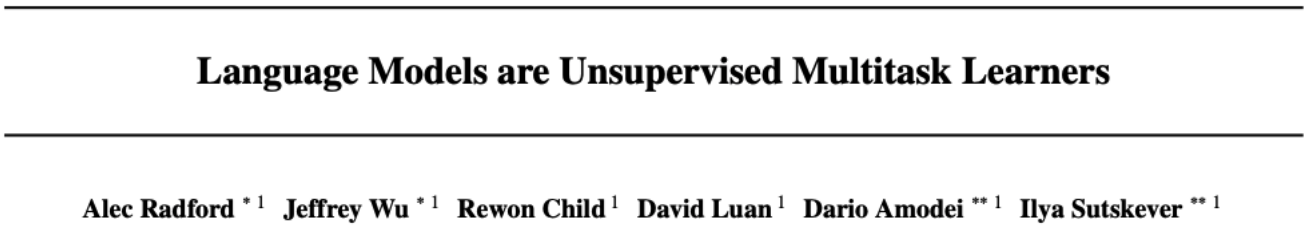
- Transformer expect an input and an output sequence

- Masked training
  - output sequence: one sentence of text
  - input sequence: same sentence, with some words masked out
    This [MASK] an [MASK] → *This is an example*

- Next sentence prediction
  - input sequence: one sentence
  - output sequence: next sentence in document

- Masked training with multiple shuffled sentences
  - same idea as masked training
  - multiple sentences (say, 3)
  - also reorder order of sentence in input sequence

LMs as Unsupervised Learners (2018)



Language Models are Unsupervised Multitask Learners

Alec Radford [*1] Jeffrey Wu [*1] Rewon Child [1] David Luan [1] Dario Amodei [**1] Ilya Sutskever [**1]

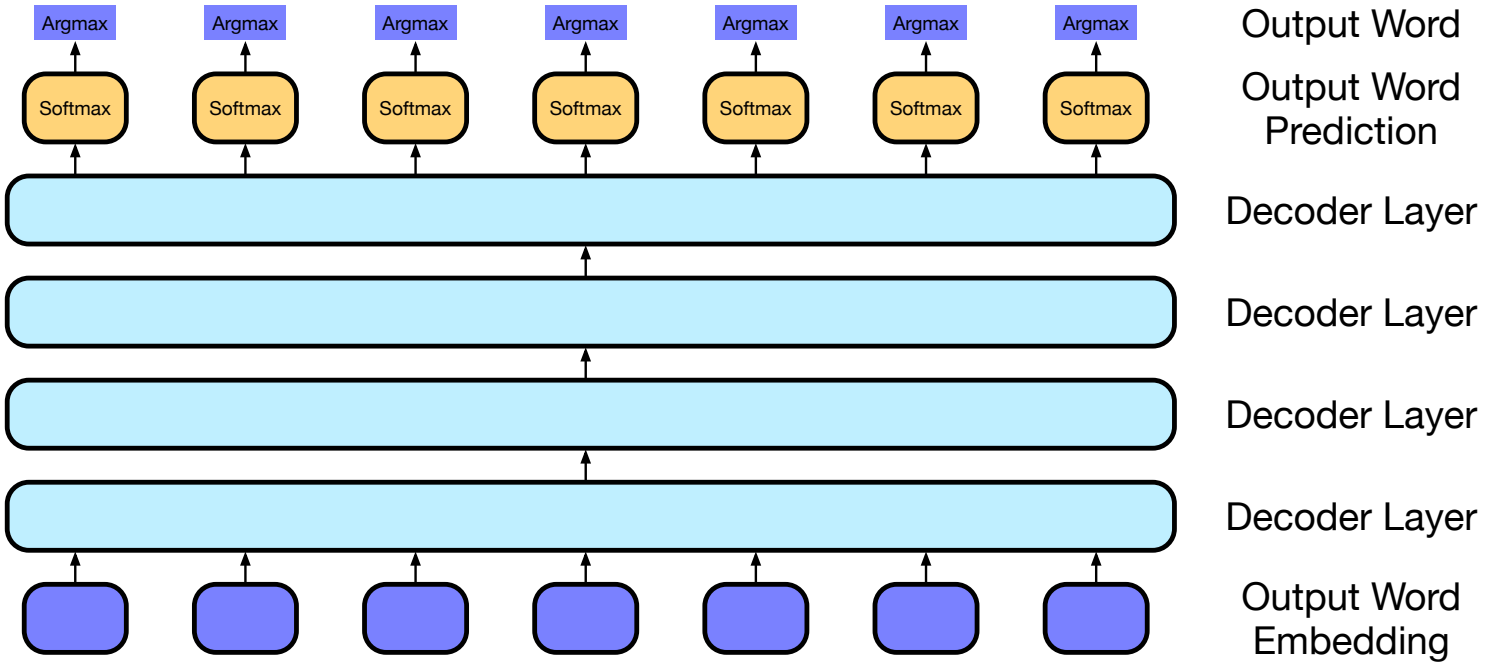
- Train language models on relatively clean text data (GPT-2)
- Convert any NLP problem into a text continuation problem
  - goes into some detail of how each task is converted
  - impressive performance on many tasks

Decoder-Only Models

- Alternative architecture: Just decoder of Transformer model

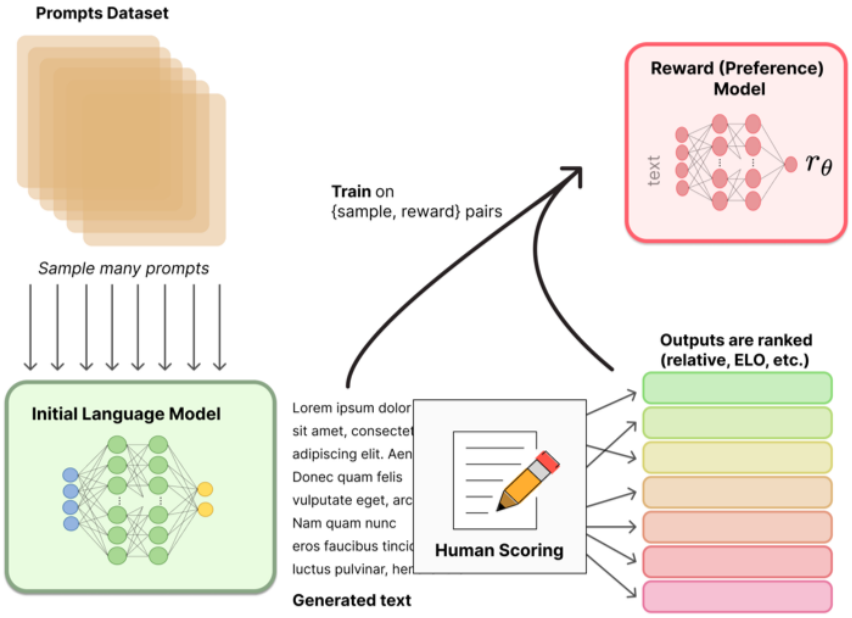
$\Rightarrow$ no input, only self-attention

- Trained with next-word prediction



Three Stages of Training Large Language Model

- Stage 1: Train on massive amounts of text (up to a trillion words)
- Stage 2: Instruction training
  - Examples of requests / responses constructed by human annotators
  - *"Summarize the following: ..."*
  - *"Give me ten examples of ..."*
  - *"Translate from French into English: ..."*
- Stage 3: Reinforcement learning from human feedback
  - Machine generates multiple responses to a prompt
  - Human annotators rank them
  - Train a reward model from
  - Fine-tune model with reward model



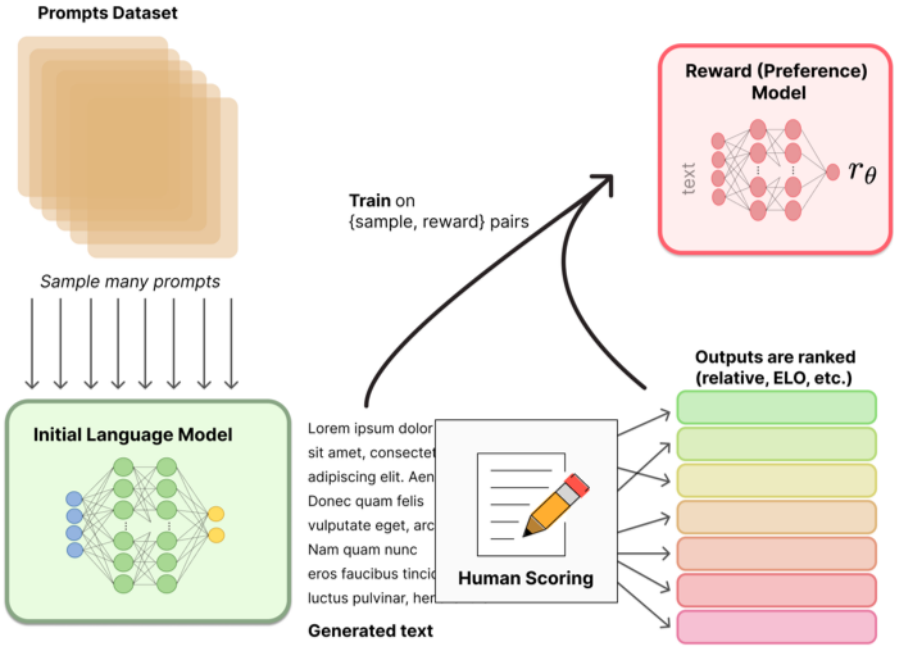
Instruction Training

- Examples of requests and responses constructed by human annotators
- May be collected from actual user requests and edited by experts
- May be generated from existing data sets

Question Answering	**Summarization**	**Translation**
What is the highest mountain in the world? The highest mountain in the world is Mount Everest.	*Summarize the following paragraph into one sentence.* The Federal Reserve paused its campaign of interest rate increases for the first time in more than a year. But officials suggested that rates would rise more in 2023, as inflation remains "well above" the central bank's target. Summary: No interest rate rise for now but maybe later in the year.	*Translate from English to German.* English: My name is Ozymandias, King of Kings; Look on my Works, ye Mighty, and despair! German: Mein Name ist Ozymandias, König der Könige; Schau auf meine Werke, du Mächtiger, und verzweifle!

Reinforcement Learning from Human Feedback

- Machine generates multiple responses to a prompt
- Human annotators rank them
- Train a reward model
  - predict human annotation
  - can be applied to any text generated by model
  - normalized as reward function
- Fine-tune model with reward model
  - model generates response to prompt
  - reward function assesses response
  - if low reward, model needs to change



Prompt Engineering

- How a task is presented to the language model matters
- Black art: often unclear what is in the training data
- "Think step by step."
  - language model has very limited working memory
  - complex reasoning may require several inference steps
  - by allowing the model to produce intermediate steps, working memory is created
- "Imagine you are..."
  - creates a style or sets a context for generation

In-Context Learning

- Provide examples of the task in the prompt

I want you to assess the sentiment of statements as either positive or negative.

Here are some examples:
It's a beautiful day. Assessment: positive
We did not play well in the game. Assessment: negative
That's not bad. Assessment: positive

Now assess the following statement:

Yeah, right. Assessment:

Retrieval Augmented Generation

- Question answering task
  - may be aided by specific documents
  - these documents may not even be part of LLM training
- Step 1: Retrieve relevant document
- Step 2: Add document to the prompt
- Step 3: Call LLM

Decomposition of Tasks

Write an outline for an argument that LLMs are great.

Outline:
1. LLMs have emerged as general problem solvers
2. LLMs have become publicly available
3. Many people find LLMs useful 4. ...

Write a paragraph about "LLMs have emerged as general problem solvers".

Language model (LLMs) have rapidly emerged as versatile general problem solvers across various domains. Leveraging vast datasets and sophisticated algorithms, LLMs exhibit remarkable capabilities in understanding, generating,...

Tool Use by LLMs

- LLM generates call to APIs
- Executes call
- Receives result back
- Uses result to continue text generation

The New England Journal of Medicine is a registered trademark of [QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society] the MMS.

Out of 1400 participants, 400 (or [Calculator(400 / 1400) → 0.29] 29%) passed the test.

The name derives from "la tortuga", the Spanish word for [MT("tortuga") → turtle] turtle.

The Brown Act is California's law [WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.] that requires legislative bodies, like city councils, to hold their meetings open to the public.

(Toolformer, Schick et al., 2023)

Programmatic Calls to LLMs

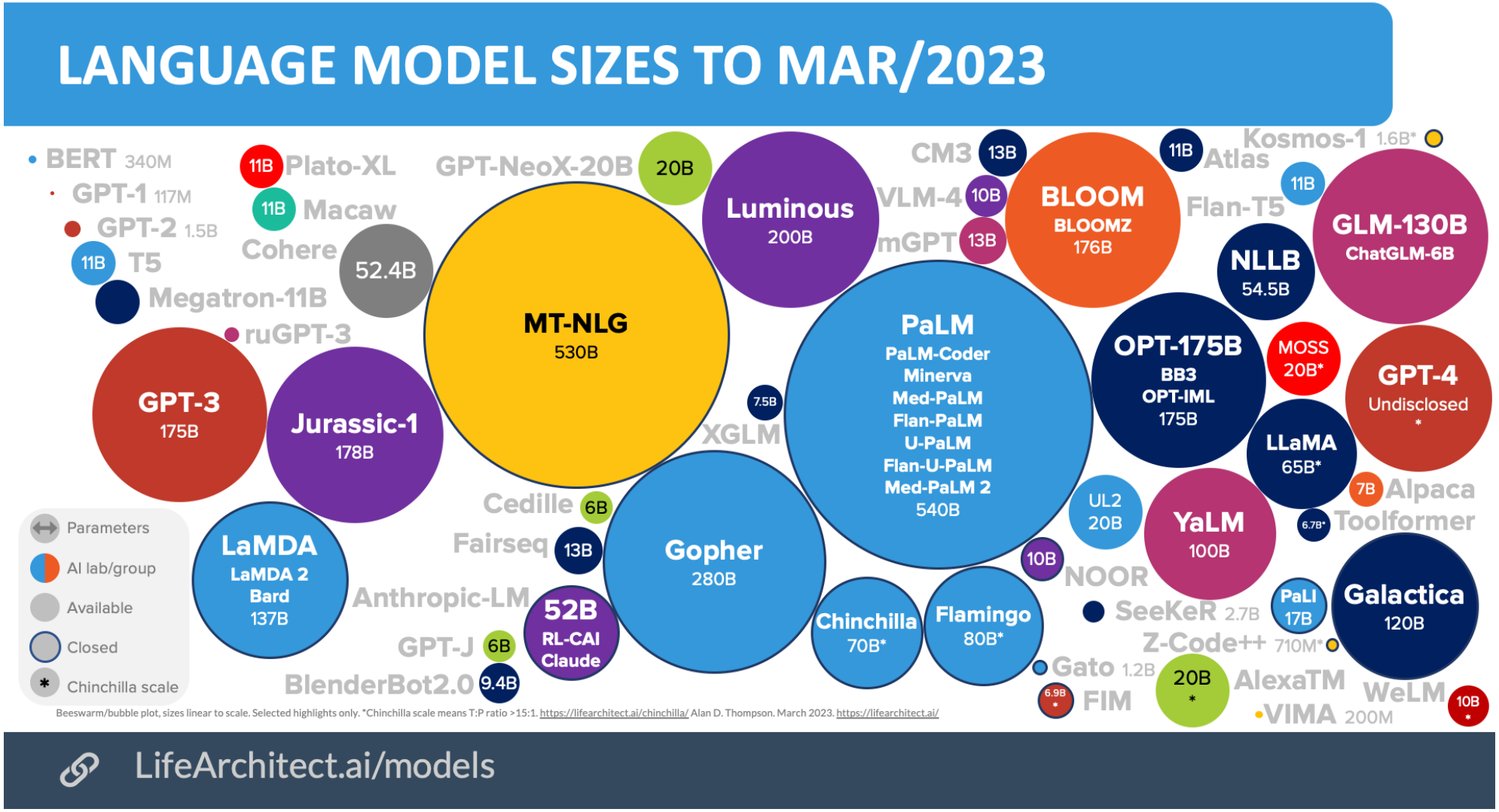
- If LLMs are universal problem solvers, use them for any task

```
a = 1.6
b_text = "What is the distance to the moon in miles?"
c_text = LLM(b_text)
c = process_response(c_text)
print("Distance to moon in km:", a * c)
```

LLMs for Everything

- Create training data ... with a language model
- Solve problem ... with a language model
- Evaluate results ... with a language model

Size of Large Language Models



Large Language Models for Mere Mortals

- Considering the size of language models
  - parameters are typically stored as 16-bit floats
  - during training also gradients and optimizer states need to be stored

  ⇒ 6 bytes per parameter
  - Also need to store the state of training examples (depends on sequence length and batch size)

- Size of GPUs
  - A100: 40-80GB RAM ($15,000)
  - RTX2080ti: 11GB RAM ($800)

- Only a few billion parameters models fit on single GPU

Adapting Compressed Models

QLoRA: Efficient Finetuning of Quantized LLMs

Tim Dettmers* **Artidoro Pagnoni*** **Ari Holtzman**

Luke Zettlemoyer

University of Washington
{dettmers,artidoro,ahai,lsz}@cs.washington.edu

- Low-rank adaptation parameter matrices
- Quantize all values into 4-bit floats
- Double quantization: also quantize the quantization factors
- Paged optimizers: NVIDIA unified memory that pages into CPU RAM
- Integrated into Huggingface, may adapt any model

Grand Challenges

- Hallucinations
- Interpretability
- HHH: Helpful, Honest, Harmless
- Preserving privacy of training data

questions?