Sequence modeling

Armand Joulin

Google DeepMind
ajoulin@google.com

Why?

- Example of temporal sequences:
  - videos
  - robot moving in an environment
  - video games...

...but first an introduction to language modeling

What is language modeling

- **Language modeling** assigning probability to a text
- A text is a sequence of tokens
- tokens can be words, characters or group of characters.
- For example:

$$\{\text{a cat}\} \quad = \quad \{\text{a}, \text{cat}\},$$

What is language modeling

- **Language modeling** assigning probability to a text
- A text is a sequence of tokens
- tokens can be words, characters or group of characters.
- For example:

$$\begin{aligned}\{\text{a cat}\} &= \{\text{a}, \text{cat}\}, \\ &= \{\text{a}, \ , \text{c}, \text{a}, \text{t}\},\end{aligned}$$

What is language modeling

- **Language modeling** assigning probability to a text
- A text is a sequence of tokens
- tokens can be words, characters or group of characters.
- For example:

$$\begin{aligned} \{\text{a cat}\} &= \{\text{a}, \text{cat}\}, \\ &= \{\text{a}, \ , \text{c}, \text{a}, \text{t}\}, \\ &= \{\text{a}, \ , \text{ca}, \text{t}\}. \end{aligned}$$

What is language modeling

- **Language modeling** assigning probability to a text
- A text is a sequence of tokens
- tokens can be words, characters or group of characters.
- For example:

$$
\begin{aligned}
\{\text{a cat}\} &= \{\text{a}, \text{cat}\}, \\
&= \{\text{a}, \ , \text{c}, \text{a}, \text{t}\}, \\
&= \{\text{a}, \ , \text{ca}, \text{t}\}.
\end{aligned}
$$

- For most of this lecture, we assume that tokens are words

What is language modeling

- Given a sequence $\{w_1, \ldots, w_T\}$ of tokens, a language model estimates its probability:

$$P(w_1, \ldots, w_T)$$

- P depends on a **vocabulary**, i.e., the set of unique tokens.

- P can be conditioned on an external variable, i.e., $P(.) = P(. \mid C)$

Applications of language modeling

Language models are applied in several fields:

- Speech recognition:

$$P(\text{"Vanilla, I scream"}) < P(\text{"Vanilla ice cream"}).$$

- Machine translation:

$$P(\text{"Déçu en bien"} \mid \text{"Pleasantly surprised"}) \quad < \\ P(\text{"Agréablement surpris"} \mid \text{"Pleasantly surprised"})$$

- Optical Character Recognition:

$$P(\text{"m0ve fast"}) < P(\text{"move fast"})$$

Applications of language modeling

- Language models are just models of sequences
- they can apply to any sequence, like video or audio

Probabilistic language model

- Sequence probability as a product of token probabilities:

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_{t-1}, \ldots, w_1)$$

Probabilistic language model

- Sequence probability as a product of token probabilities:

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_{t-1}, \ldots, w_1)$$

- Indeed we have:

$$P(a, b) = P(a)P(b \mid a)$$

Probabilistic language model

- Sequence probability as a product of token probabilities:

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_{t-1}, \ldots, w_1)$$

- Indeed we have:

$$P(a, b) = P(a)P(b \mid a)$$

- Recursively applied to a sequence:

$$\begin{aligned} P(w_1, w_2, w_3) &= P(w_1)P(w_2, w_3 \mid w_1) \\ &= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_2, w_1). \end{aligned}$$

Probabilistic language model

- Sequence probability as a product of token probabilities:

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_{t-1}, \ldots, w_1)$$

- Indeed we have:

$$P(a, b) = P(a)P(b \mid a)$$

- Recursively applied to a sequence:

$$\begin{aligned} P(w_1, w_2, w_3) &= P(w_1)P(w_2, w_3 \mid w_1) \\ &= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_2, w_1). \end{aligned}$$

- Language models estimate probability of upcoming token given past:

$$P(w_t \mid w_{t-1}, \ldots, w_1).$$

Preliminaries: words as vectors

- We assume a fixed vocabulary of V words
- we represent the i-th word by a V dimensional vector $\mathbf{w}_i$:

$$\mathbf{w}_i[j] = \begin{cases} 1 & \text{if } j = i, \\ 0 & \text{otherwise} \end{cases}$$

- These word vectors are:
    - independent: $\mathbf{w}_i^T \mathbf{w}_j = 0$ if $i \neq j$
    - normalized: $\mathbf{w}_i^T \mathbf{w}_i = 1$
- We call this representation "one-hot vectors"
- For now on, the notation $\mathbf{w}_t$ represents the one-hot vector of the word at the t-th position in the sentence

A linear model for bigrams

- The input is the 1-hot vector of the previous word: $\mathbf{x}_t = \mathbf{w}_{t-1}$
- The output is the 1-hot vector of the upcoming word: $y_t = \mathbf{w}_t$

- **Linear model $\mathbf{z} = \mathbf{A}\mathbf{x}$**
- Build a probability over all possible words:

$$f(\mathbf{y},\ \mathbf{z})[k] = \frac{\exp(\mathbf{z}[k])}{\sum_{i=1}^{V} \exp(\mathbf{z}[i])}$$

- A cross-entropy loss: $\ell(\mathbf{q}, \mathbf{p}) = -\mathbf{q}^T \log(\mathbf{p})$
- Learning a linear bigram model is equivalent to:

$$\min_{\mathbf{A} \in \mathbb{R}^{V \times V}} \ \frac{1}{T} \sum_{t=1}^{T} \ell(\mathbf{y}_t, f(\mathbf{A}\mathbf{x}_t))$$

Limitations of linear models

$$\min_{\mathbf{A}\in\mathbb{R}^{V\times V}} \frac{1}{T}\sum_{t=1}^{T}\ell(\mathbf{y}_t, \mathbf{A}\mathbf{x}_t)$$

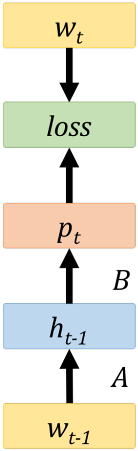
- The matrix **A** is $O(V^2)$
- Example: $V = 10\text{k} \rightarrow 100,000,000$ parameters
- Difficult and slow to scale to longer *n*-grams

Neural bigram model

- feedforward network:

$$\begin{aligned}\mathbf{h}_{t-1} &= \sigma(\mathbf{A}\mathbf{w}_{t-1})\\ \mathbf{p}_t &= f(\mathbf{B}\mathbf{h}_{t-1})\end{aligned}$$

$\sigma(x) = 1/(1+\exp(-x))$ pointwise sigmoid function



- $\mathbf{A}$: $V \times H$ matrix; $\mathbf{B}$: $H \times V$ matrix
- $H << V$
- Minimization problem:

$$\min_{\mathbf{A},\,\mathbf{B}} \frac{1}{T}\sum_{t=1}^{T} \ell(\mathbf{w}_t, f(\mathbf{B}\sigma(\mathbf{A}\mathbf{w}_{t-1})))$$

Neural *n*-gram model

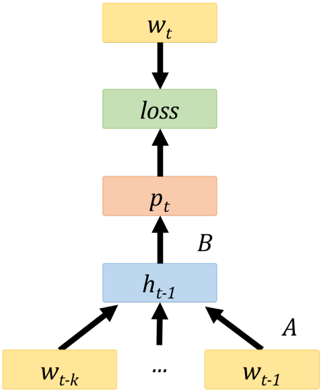
Generalization to any fixed *n*-gram:

- The input is the contactenation of previous words:

$$\mathbf{x}_t = [\mathbf{w}_{t-n+1}, \ldots, \mathbf{w}_{t-1}]$$

- **A**: $nV \times H$ matrix
- Minimization problem:

$$\min_{\mathbf{A},\ \mathbf{B}} \frac{1}{T} \sum_{t=1}^{T} \ell(w_t, f(\mathbf{B}\sigma(\mathbf{A}\mathbf{x}_t)))$$

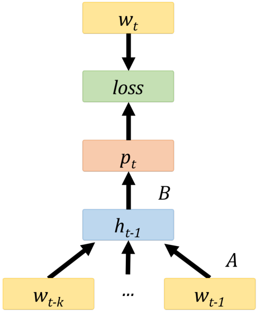


Recurrent Neural Network

- Recurrent network: Keep memory of past in the hidden variables

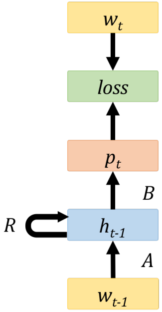
Feedforward

$$\mathbf{h}_{t-1} = \sigma\left(\mathbf{A}[\mathbf{w}_{t-k}, \ldots, \mathbf{w}_{t-1}]\right)$$
$$\mathbf{p}_t = f(\mathbf{B}\mathbf{h}_{t-1})$$

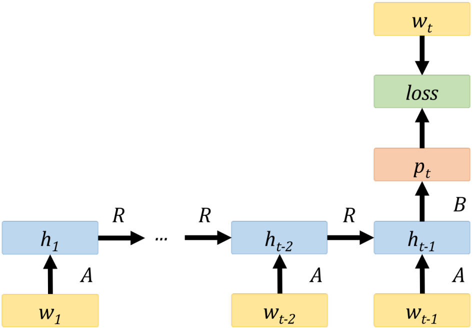


Recurrent Network

$$\mathbf{h}_{t-1} = \sigma\left(\mathbf{A}\mathbf{w}_{t-1} + \mathbf{R}\mathbf{h}_{t-2}\right)$$
$$\mathbf{p}_t = f(\mathbf{B}\mathbf{h}_{t-1})$$

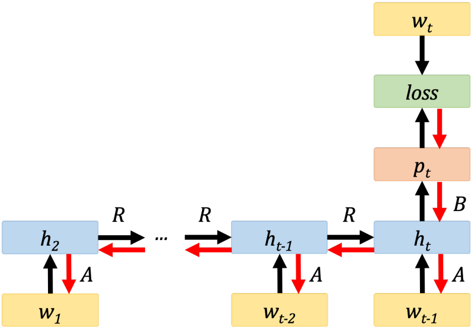


Recurrent Neural Network



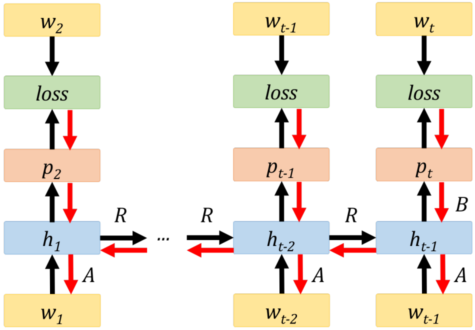
- Recurrent equation: $\mathbf{h}_t = \sigma\left(\mathbf{A}[\mathbf{h}_{t-1}, \mathbf{w}_t]\right)$
- Unfold over time: **very deep feedforward with weight sharing**
- Potentially capture long term dependencies

Recurrent Neural Network: training



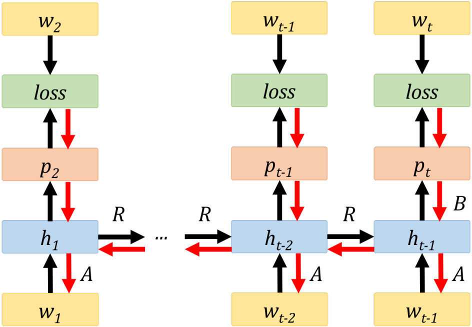
- **Backpropagation through time (BPTT)**: same as backpropagation through a very deepfeedforward network

Recurrent Neural Network: training



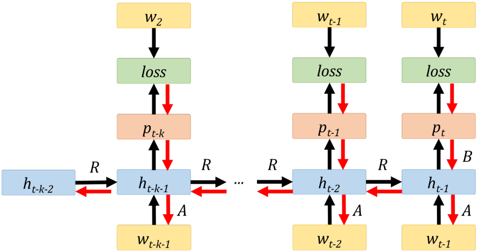
- **batch BPTT**: forward/backward for many words simultaneously

Recurrent Neural Network: training



- **Problem with BPTT**: Computing 1 gradient is $O(T)$. Too slow.

Recurrent Neural Network: training



- **Truncated BPTT**: Go back in time for k step: $O(k)$.

Transformer Networks

Motivation

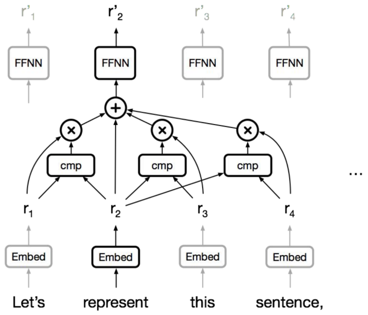
- In recurrent networks, we have

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, w_t).$$

- RNNs encode the whole history in single vector $\mathbf{h}_{t-1}$
- Instead, can we use all token representations to compute $\mathbf{h}_t$?
- Technical challenge:

  need to combine a variable number of representations!

Convolutional Neural Networks?



- Pros
  - easy to parallelize
  - exploits local context
- Cons
  - span of context increase linearly with number of layers
  - need to be very deep to have large context

---

from Vaswani and Huang:
`http://web.stanford.edu/class/cs224n/slides/`

Combining vectors with attention

- Solution: use the (self) attention mechanism
- Given a set of vectors $\mathbf{w}_1$, ..., $\mathbf{w}_T \in \mathbb{R}^d$ representing tokens

$$\mathbf{h}_t = \sum_{i=1}^{T} a_{it} \mathbf{V} \mathbf{w}_i$$

  where $\sum_{i=1}^{T} a_{it} = 1$.
- We could use $a_{it} = \frac{1}{T}$ and get a BoW

Combining vectors with attention

- Introducing matrix $\mathbf{W} \in \mathbb{R}^{d \times T}$ where columns correspond to $\mathbf{w}_i$,

$$\mathbf{h}_t = \mathbf{V}\mathbf{W}\mathbf{a}_t$$

- And finally

$$\mathbf{H} = \mathbf{V}\mathbf{W}\mathbf{A}$$

Combining vectors with attention

- How to compute the matrix $\mathbf{A}$?

$$\mathbf{A} = \text{softmax}(\mathbf{W}^\top \mathbf{K}^\top \mathbf{Q} \mathbf{W})$$

  where the softmax is applied column-wise.

- Why softmax? to get positive entries, and columns summing to 1.
- Why $\mathbf{W}^\top \mathbf{K}^\top \mathbf{Q} \mathbf{W}$? Bilinear form over the input

Combining vectors with attention

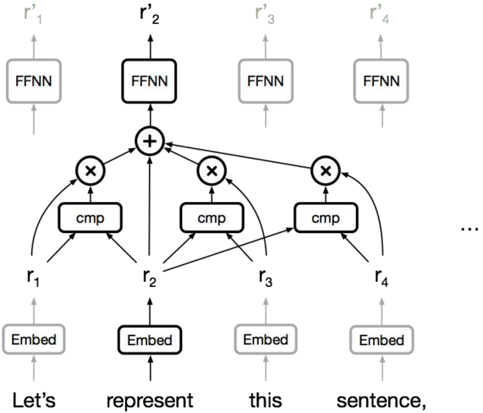
- Putting everything together:

$$\mathbf{H} = \mathbf{V}\mathbf{W}\text{softmax}(\mathbf{W}^\top \mathbf{K}^\top \mathbf{Q}\mathbf{W})$$

  where $\mathbf{H}, \mathbf{W} \in \mathbb{R}^{d \times T}$ and $\mathbf{V}, \mathbf{K}, \mathbf{Q} \in \mathbb{R}^{d \times d}$
- $\mathbf{V}, \mathbf{K}, \mathbf{Q}$ are parameters to be learned.
- This operation is called self-attention

- It can be generalized to multiple heads:
  - Split input vectors into n subvectors of dimension d/n,
  - Apply self attention (with different $\mathbf{V}, \mathbf{K}, \mathbf{Q}$) over these smaller vectors
  - Concatenate the results to get back d dimensional vectors

Combining vectors with attention

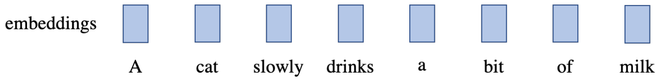


from Vaswani and Huang:
http://web.stanford.edu/class/cs224n/slides/

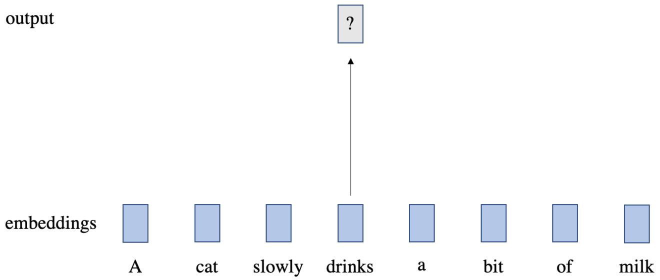
Combining vectors with attention

- Goal: use all the context to update a word
- Idea: look for the most important words in the context
- Solution: self-attention on the sequence of inputs

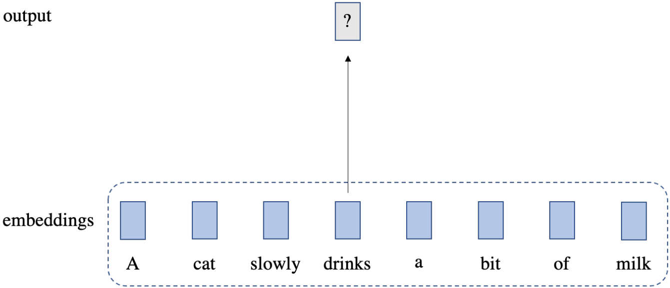
Combining vectors with attention



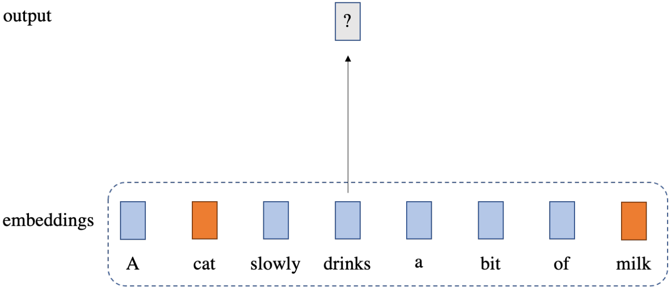
Combining vectors with attention



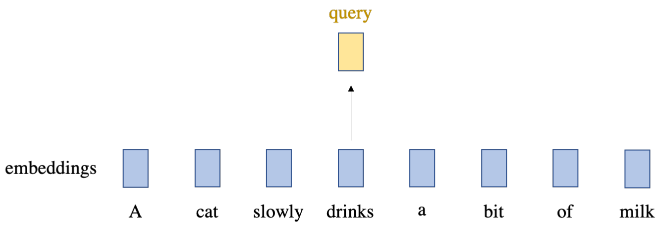
Combining vectors with attention



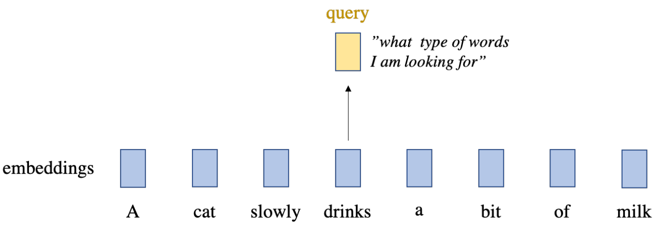
Combining vectors with attention



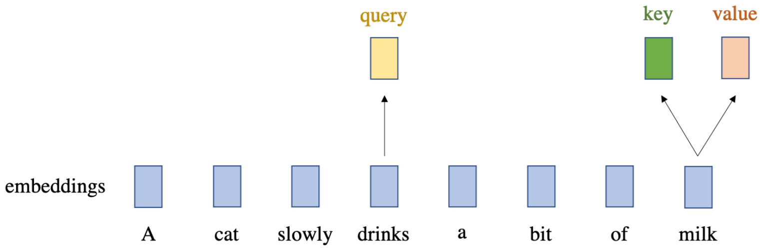
Combining vectors with attention



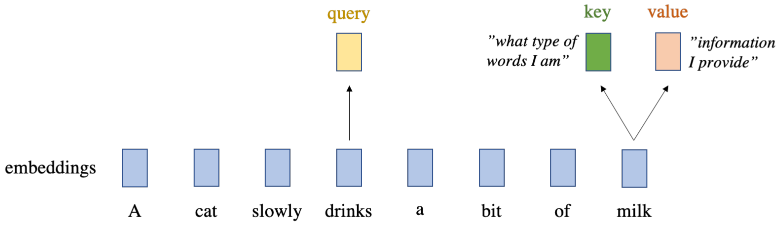
Combining vectors with attention



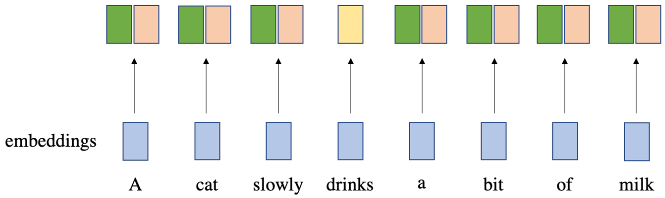
Combining vectors with attention



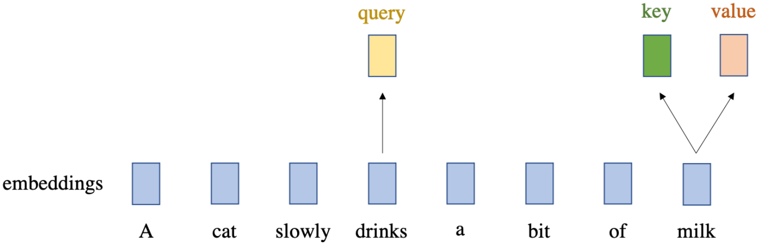
Combining vectors with attention



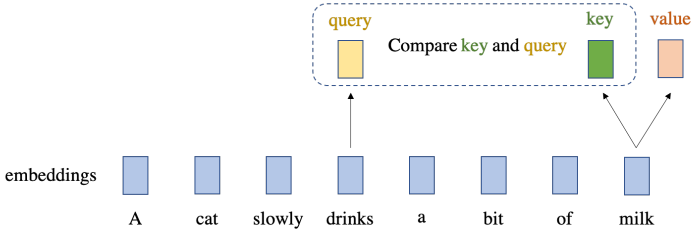
Combining vectors with attention



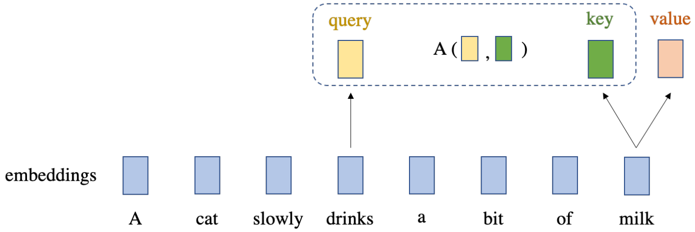
Combining vectors with attention



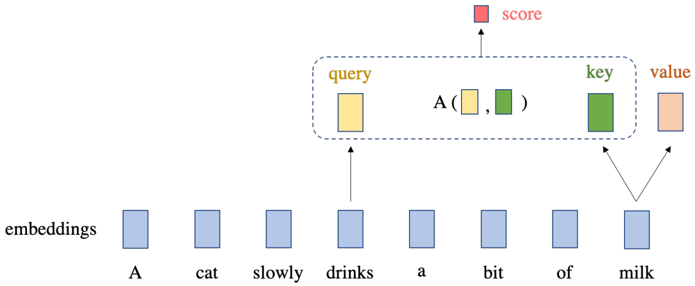
Combining vectors with attention



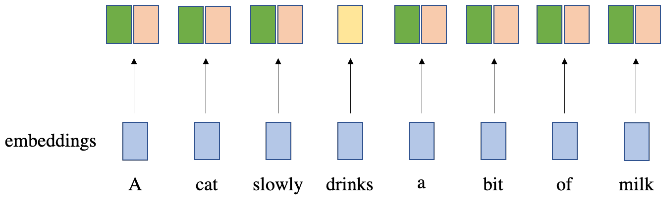
Combining vectors with attention



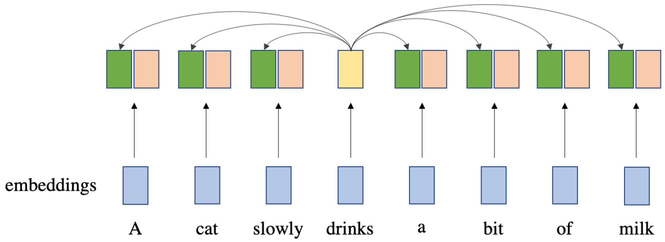
Combining vectors with attention



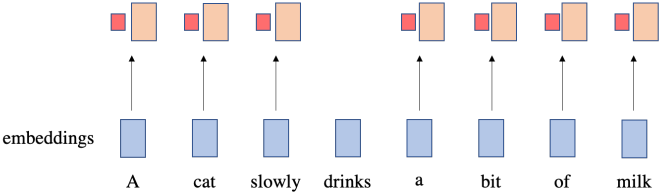
Combining vectors with attention



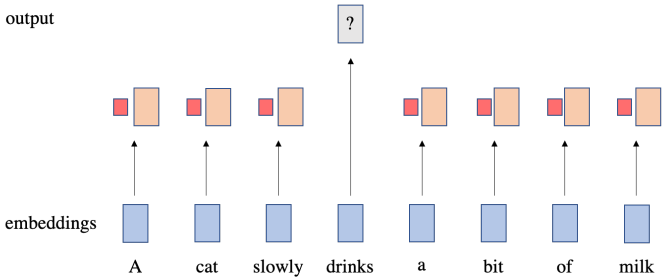
Combining vectors with attention



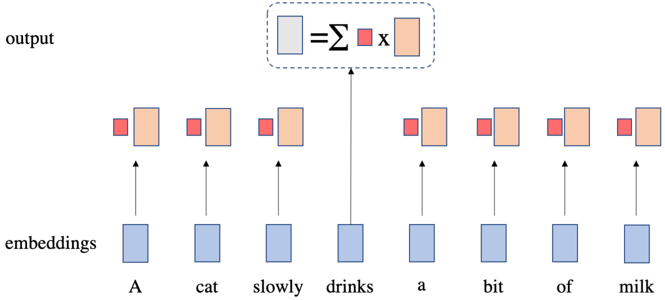
Combining vectors with attention



Combining vectors with attention



Combining vectors with attention



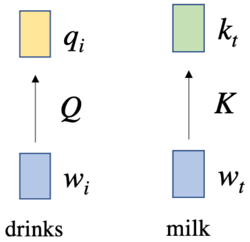
Combining vectors with attention

- "query vector" for word i ("drinks"):

$$\mathbf{q}_i = \mathbf{Q}\mathbf{w}_i$$

- "key vector" for word t ("milk"):

$$\mathbf{k}_t = \mathbf{K}\mathbf{w}_t$$



Combining vectors with attention

- "query vector" for word i ("drinks"):

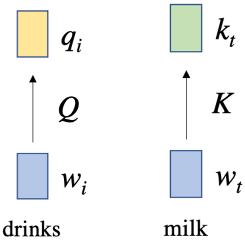
$$\mathbf{q}_i = \mathbf{Q}\mathbf{w}_i$$

- "key vector" for word t ("milk"):

$$\mathbf{k}_t = \mathbf{K}\mathbf{w}_t$$

- Their similarity score is then:

$$s_{it} = \mathbf{q}_i^\top \mathbf{k}_t$$



Combining vectors with attention

- "query vector" for word i ("drinks"):

$$\mathbf{q}_i = \mathbf{Q}\mathbf{w}_i$$

- "key vector" for word t ("milk"):

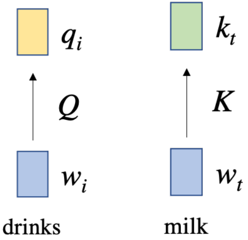
$$\mathbf{k}_t = \mathbf{K}\mathbf{w}_t$$

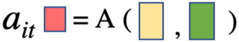
- Their similarity score is then:

$$s_{it} = \mathbf{q}_i^\top \mathbf{k}_t$$

- Normalize over sequence with softmax:

$$a_{it} = \frac{\exp(s_{it})}{\sum_k \exp(s_{ik})}$$

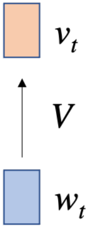




Combining vectors with attention

- "value vector" for word t ("milk"):

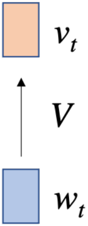
$$\mathbf{v}_t = \mathbf{V}\mathbf{w}_t$$



Combining vectors with attention

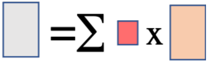
- "value vector" for word t ("milk"):

$$\mathbf{v}_t = \mathbf{V}\mathbf{w}_t$$



- Finally, compute output for "drinks":

$$\mathbf{y}_i = \sum_t a_{it}\mathbf{v}_t$$



Transformer network

Transformer block:

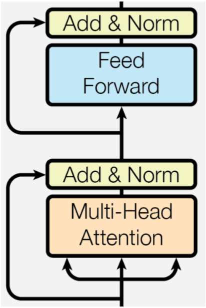
- Multi-head attention layer with skip connection and normalization
- Followed by feed forward with skip connection and normalization

Skip connection+normalization:

- Given a network block **nn** and input **x**
- The output **y** is computed as

$$\mathbf{y} = \mathbf{norm}(\mathbf{x} + \mathbf{nn}(\mathbf{x}))$$

where **norm** normalize the input



Vaswani et al. (2017)

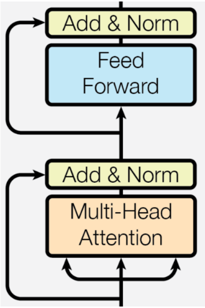
Transformer network

Feed forward block

- Two layer network, with ReLU activation

$$\mathbf{y} = \mathbf{W}_2 \texttt{ReLU}(\mathbf{W}_1 \mathbf{x})$$

- Usually, $\mathbf{W}_1 \in \mathbb{R}^{4d \times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{d \times 4d}$
- i.e. hidden layer of dimension $4d$.



Vaswani et al. (2017)

Position embeddings

- **Limitation:** self attention does not take position into account!
- Indeed, shuffling the input gives the same results

- **Solution:** add position encodings.
- Replace the matrix $\mathbf{W}$ by $\mathbf{W} + \mathbf{E}$, where $\mathbf{E} \in \mathbb{R}^{d \times T}$

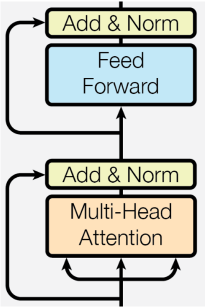
- $\mathbf{E}$ can be learned, or defined using sin and cos:

$$e_{2i,j} = \sin\left(\frac{j}{10000^{2i/d}}\right)$$
$$e_{2i+1,j} = \cos\left(\frac{j}{10000^{2i/d}}\right)$$

Transformer network: take away

Transformer network:

- Token embeddings + Position embeddings
- Then N transformer blocks (e.g. $N = 12$)
- Softmax classifier



Vaswani et al. (2017)

References I

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.