# Attention and Transformer

Natural Language Processing

Vladislav Lialin, Text Machine Lab

Feb 27, 2023

# Administrative

- **No classes next week, spring break**
- Weak extension to homework 4: you can add runs to wandb
- Next homework: implementing and training a transformer language model
  - Due after the spring break before the class, March 13
  - The homework is relatively hard, it will take at least full day to implement the architecture and it will take another day to train the models
  - Free Colab tier might be not enough for it. Your options:
    - Colab Pro ($10/mo) / Colab Pro+ ($50/mo)
    - Google Cloud Platform — $300 when you create an account
  - **There will be no extensions,** regular late submission policy applies

# Administrative

- **No classes next week, spring break**
- Next homework: implementing and training a transformer language model
  - Due after the spring break before the class, March 13
  - The homework is relatively hard, it will take at least full day to implement the architecture and it will take another day to train the models
  - Free Colab tier might be not enough for it. Your options:
    - Colab Pro ($10/mo) / Colab Pro+ ($50/mo)
    - Google Cloud Platform — $300 when you create an account
  - **There will be no extensions,** regular late sumission policy applies
- Sorry, we are going to have a midterm on March 27th

# What you should remember after this lecture

- Attention, KQV notation
- Language modeling
- Transformer Encoder

Quiz discussion

click

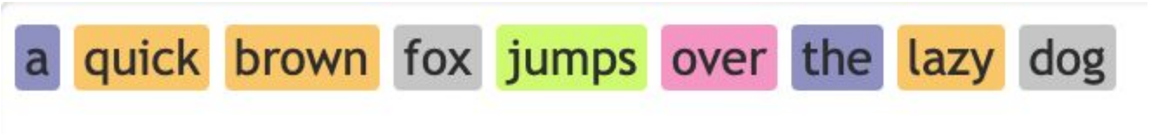
# Contextualized representations

# No bags of words anymore

- We want to learn complex features based on sequences of raw tokens
- Imagine a part-of-speech (PoS) tagging task

# No bags of words anymore

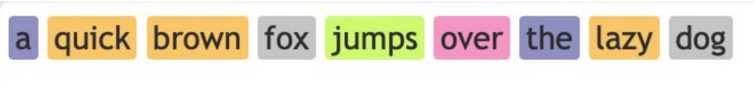
- We want to learn complex features based on sequences of raw tokens
- Imagine a part-of-speech (PoS) tagging task


# No bags of words anymore

- We want to learn complex features based on sequences of raw tokens
- Imagine a part-of-speech (PoS) tagging task
  - Different order of words — different expected output
  - But the bag of words for both of these sentences is the same

a quick brown fox jumps over the lazy dog




a dog jumps over the quick brown lazy fox

# Sequences of word vectors

# Sequences of word vectors

Matrix of shape
[seq_len, word_dim]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| -1 | 1 | -1 | 1 | -5 | 4 | -9 | -8 | 0 |
| 3 | 2 | 7 | 9 | 5 | 3 | 0 | 8 | 8 |
| 9 | 1 | 4 | 0 | 5 | 2 | 6 | 0 | 9 |
| -3 | -6 | -7 | -1 | -5 | -1 | -3 | -2 | -6 |
| 0 | 8 | 2 | 0 | 5 | 1 | 4 | 7 | 6 |
| 2 | 0 | 1 | 8 | 5 | 8 | 2 | 2 | 2 |
| 1 | 2 | 1 | 8 | 8 | 1 | 6 | 1 | 1 |

a quick brown fox jumps over the lazy dog

# What both word2vec and count vectors lack?

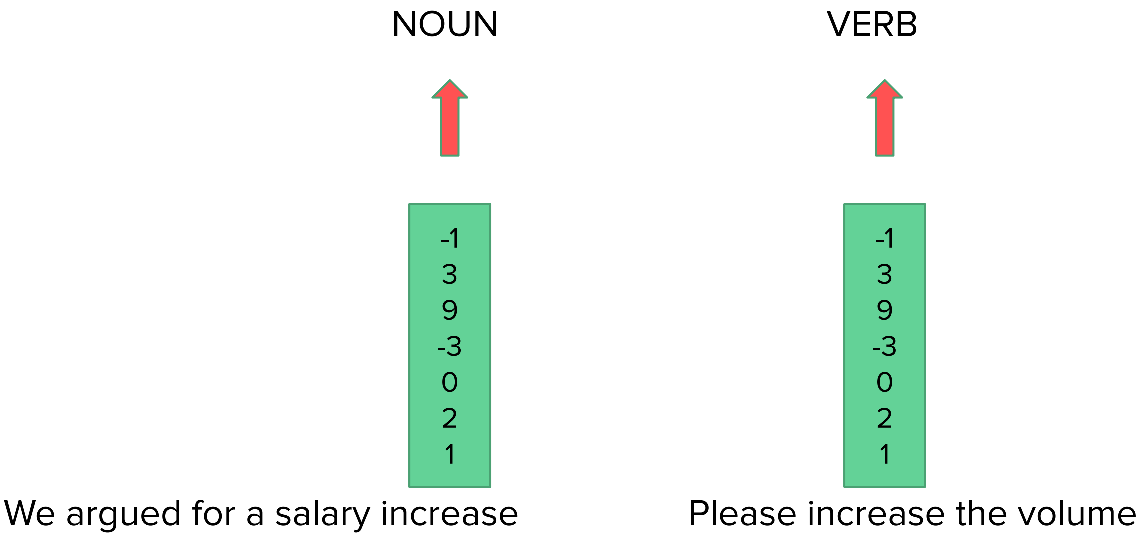
$$\begin{bmatrix} -1 \\ 3 \\ 9 \\ -3 \\ 0 \\ 2 \\ 1 \end{bmatrix}$$

We argued for a salary increase

$$\begin{bmatrix} -1 \\ 3 \\ 9 \\ -3 \\ 0 \\ 2 \\ 1 \end{bmatrix}$$

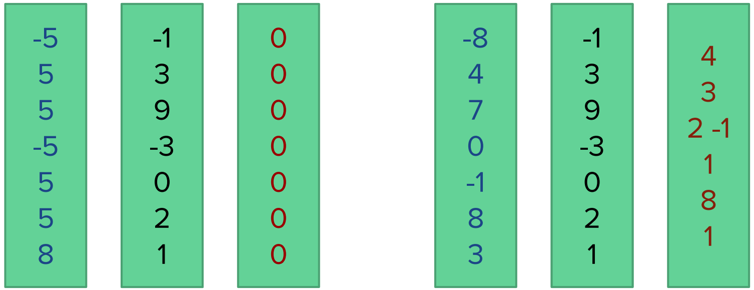
Please increase the volume

# What both word2ve and count vectors lack?



# Word-window context

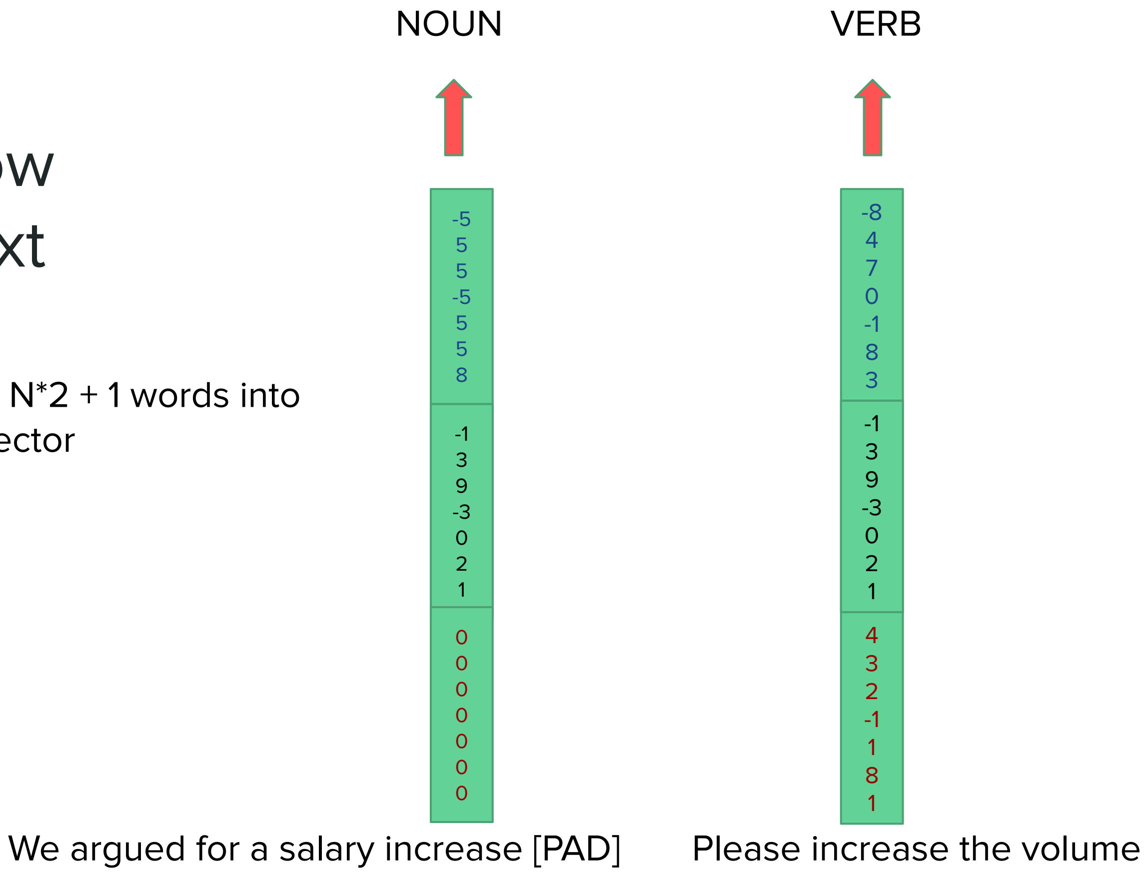
Use N words to the left and N words to the right




We argued for a salary increase [PAD]    Please increase the volume

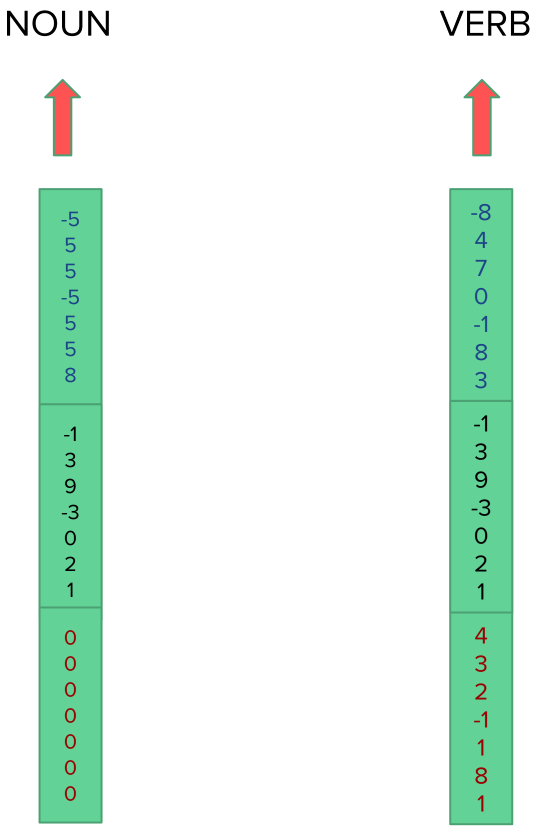
# Word window context

Concat all N*2 + 1 words into a single vector



# Word window context

Concat all N*2 + 1 words into a single vector




We argued for a salary increase [PAD]    Please increase the volume

padding the edges of texts
so that every word has N*2 neighbors

# Word-window context + linear model

$$\begin{bmatrix} -5 \\ 5 \\ 5 \\ -5 \\ 5 \\ 5 \\ 8 \\ \hline -1 \\ 3 \\ 9 \\ -3 \\ 0 \\ 2 \\ 1 \\ \hline 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} @ \, W$$

# Word-window context + linear model

$$\begin{bmatrix} -5 \\ 5 \\ 5 \\ -1 \\ 3 \\ 9 \\ 0 \\ 0 \\ 0 \end{bmatrix} @ W =$$

$$-5\,W_{0,0} + 5W_{1,0} + 5W_{2,0} - 1W_{3,0} + 3W_{4,0} + 9W_{5,0} + 0W_{6,0} + 0W_{7,0} + 0W_{8,0} = s_{\text{determiner}}$$

$$-5\,W_{0,1} + 5W_{1,1} + 5W_{2,1} - 1W_{3,1} + 3W_{4,1} + 9W_{5,1} + 0W_{6,1} + 0W_{7,1} + 0W_{8,1} = s_{\text{noun}}$$

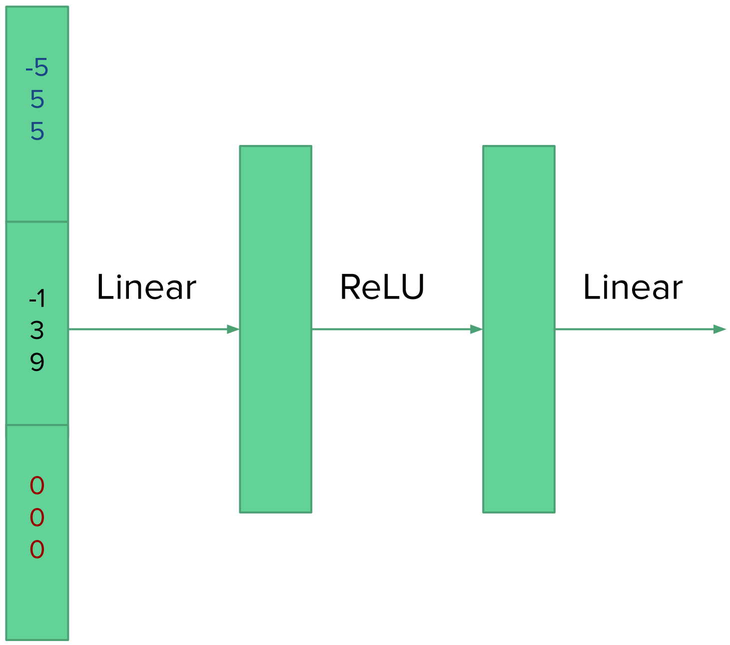
...

$$-5\,W_{0,k} + 5W_{1,k} + 5W_{2,k} - 1W_{3,k} + 3W_{4,k} + 9W_{5,k} + 0W_{6,k} + 0W_{7,k} + 0W_{8,k} = s_{\text{verb}}$$

Can we use a neural network instead of a linear model?

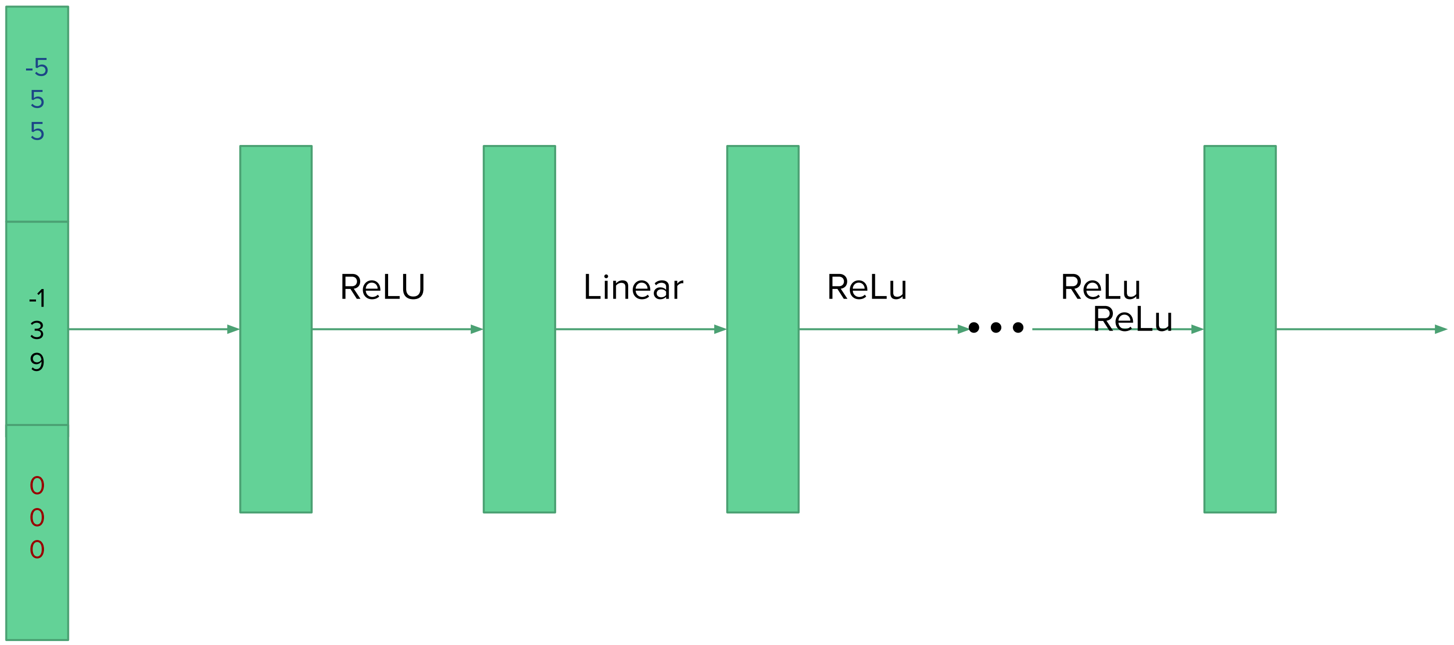
Yes, why not?

# Word-window context + FCN

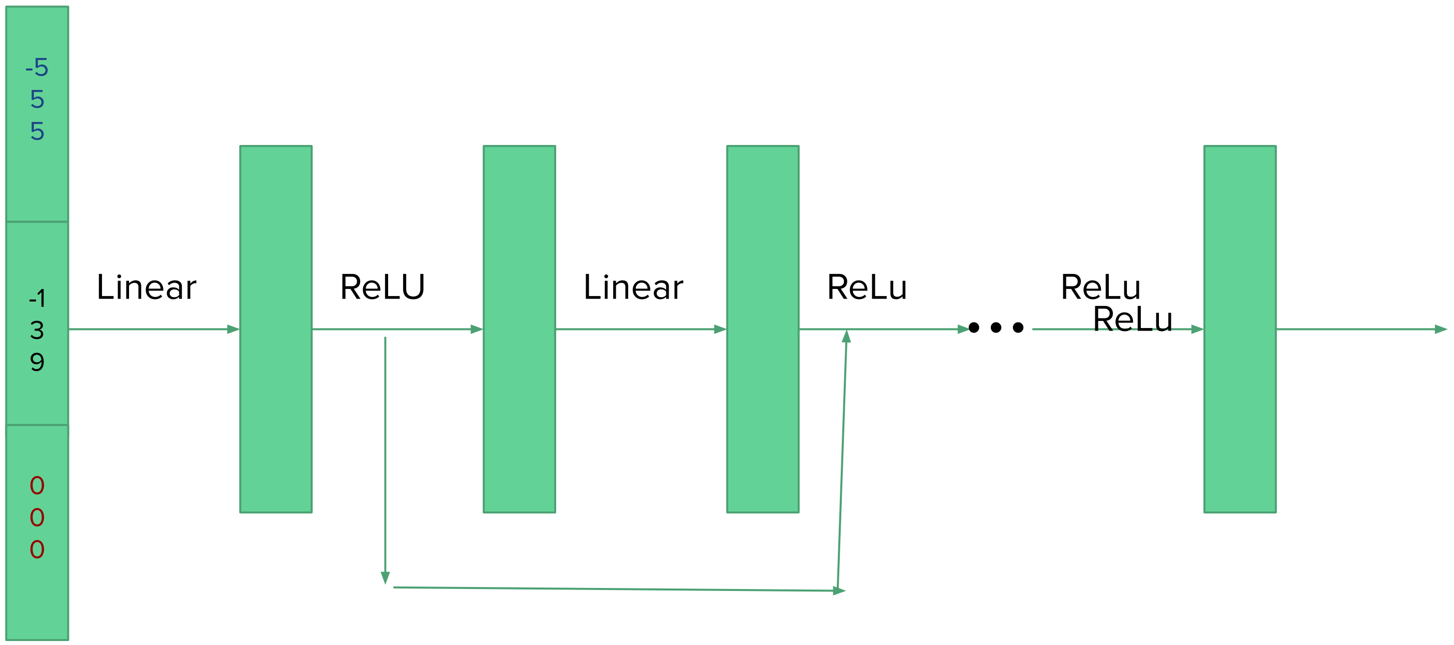


# Word-window context + FCN

# Word-window context + FCN



# Word-window context + FCN: residual connections



# What if we want a large N?

- Say N = 100
- Input size = (2N + 1) * word_dim = 201 * word_dim = input layer size = a lot
- Lots of padding
  - Even if text is of length 16, you still need to compute a vector of size 201 * word_dim
  - Computationally inefficient
- Parameters are per-position specific
  - Only a few long texts, the last ws are not updated a lot and undertrained
- Removing one word form the text completely changes it’s vector

# Learning relations

# What do we want?

- Text = matrix of **contextualized** embeddings
    - [seq_len, word_dim]
- Make our word representations **communicate**
    - Assume words “want” to communicate if they are related
    - How do we make our network able to **learn relations?**

# Learning relations

Assume we want to find all relations of the word “fox” to the rest of the words in a sentence.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| -1 | 1 | -1 | 1 | -5 | 4 | -9 | -8 | 0 |
| 3 | 2 | 7 | 9 | 5 | 3 | 0 | 8 | 8 |
| 9 | 1 | 4 | 0 | 5 | 2 | 6 | 0 | 9 |
| -3 | -6 | -7 | -1 | -5 | -1 | -3 | -2 | -6 |
| 0 | 8 | 2 | 0 | 5 | 1 | 4 | 7 | 6 |
| 2 | 0 | 1 | 8 | 5 | 8 | 2 | 2 | 2 |
| 1 | 2 | 1 | 8 | 8 | 1 | 6 | 1 | 1 |

a quick brown fox jumps over the lazy dog

# Learning relations: make a query

$$q_{fox} = \mathrm{Query}(w_{fox}) = \mathrm{FCN}(w_{fox})$$

# Learning relations: compute relation score

$$q_{fox} = \mathrm{Query}(w_{fox}) = \mathrm{FCN}_1(w_{fox})$$

$$\mathrm{relation}_a = FCN_2([q_{fox} : w_a])$$
$$\mathrm{relation}_{quick} = FCN_2([q_{fox} : w_{quick}])$$
$$\mathrm{relation}_{brown} = FCN_2([q_{fox} : w_{brown}])$$
$$\cdots$$

# Learning relations: compute relation score simpler

$$q_{fox} = \mathrm{Query}(w_{fox}) = \mathrm{FCN}_1(w_{fox})$$

$$\mathrm{relation}_a = q_{fox} w_a^T$$
$$\mathrm{relation}_{quick} = q_{fox} w_{quck}^T$$
$$\mathrm{relation}_{brown} = q_{fox} w_{brown}^T$$
$$\cdots$$

# Learning relations

$$\mathbf{x} = [w_a, w_{quick}, \ldots, w_{dog}]$$

$$\mathbf{q}_{fox} = \mathrm{FCN}_Q(\mathbf{w}_{fox})$$

$$\mathbf{s}_{fox} = \mathbf{q}_{fox} x^T$$

$$\mathbf{s}_{fox} = [\mathrm{relation}_a^{fox}, \mathrm{relation}_{quick}^{fox}, \mathrm{relation}_{brown}^{fox}, \mathrm{relation}_{fox}^{fox}, \ldots, \mathrm{relation}_{dog}^{fox}] \in \mathbb{R}^{\mathrm{seq\_len}}$$

# Learning relations: what to do with relation scores

$$\text{relation} = \text{argmax}_{seq\_len}(s)$$

$$\text{relation} = \text{softmax}_{seq\_len}(s) = \frac{e^{s}}{\sum_{j}^{seq\_len}(e^{s_j})}$$

# Learning relations: what to do with relation scores

$$\text{relation} = \text{argmax}_{seq\_len}(s)$$

$$\text{relation} = \text{softmax}_{seq\_len}(s) = \frac{e^{s}}{\sum_{j}^{seq\_len}(e^{s_j})}$$

# Learning relations: what to do with relation scores

$$\text{relation} = \text{argmax}_{seq\_len}(s) = [0,0,0,...,1,...,0]$$

$$\text{relation} = \text{softmax}_{seq\_len}(s) = \frac{e^s}{\sum_j^{seq\_len}(e^{s_j})} = [0.1,0.04,0.01,0.6,0.3,...,0.01]$$

# Learning relations: what to do with relation scores

$$\text{relation} = \text{argmax}_{seq\_len}(s) = [0,0,0,...,1,...,0]$$

$$\text{relation} = \text{softmax}_{seq\_len}(s) = \frac{e^s}{\sum_j^{seq\_len}(e^{s_j})} = [0.1,0.04,0.01,0.6,0.3,...,0.01]$$

# Learning relations: how to construct a contextualized vector

$$\mathbf{p} = \text{softmax}_{seq\_len}(\mathbf{s}) = \frac{e^{\mathbf{s}}}{\sum_j^{seq\_len}(e^{s_j})} = [0.05,0.36,0.40,0.06,0.13,...,0.01]$$

$$\mathbf{c}_{fox} = \mathbf{p} \cdot \mathbf{x}^T = 0.05w_a + 0.36\mathbf{w}_{quick} + 0.40\mathbf{w}_{brown} + 0.06\mathbf{w}_{fox} + \ldots + 0.01\mathbf{w}_{dog}$$

# Learning relations: how to construct a contextualized vector

Also called “attention probability”

$$\mathbf{p} = \text{softmax}_{seq\_len}(\mathbf{s}) = \frac{e^{\mathbf{s}}}{\sum_j^{seq\_len}(e^{s_j})} = [0.05, 0.36, 0.40, 0.06, 0.13, \ldots, 0.01]$$

$$\mathbf{c}_{fox} = \mathbf{p} \cdot \mathbf{x}^T = 0.05w_a + 0.36\mathbf{w}_{quick} + 0.40\mathbf{w}_{brown} + 0.06\mathbf{w}_{fox} + \ldots + 0.01\mathbf{w}_{dog}$$

# Learning relations

1. Transform the word “fox” into a **query**
2. Compute relations scores of “fox” to every word in the tet
   a. Dot-product between the query vector and the word-vectors
3. Apply softmax to the relation scores to get attention probabilities
4. Sum up word vectors in the text with the weights equal to attention probabilities

# Learning relations: our contextualizing function

$$\mathbf{x} = [w_a, w_{quick}, \ldots, w_{dog}]$$

$$\mathbf{q}_{fox} = \mathrm{FCN}_Q(\mathbf{w}_{fox})$$

$$\mathbf{s}_{fox} = \mathbf{q}_{fox} x^T$$

$$\mathbf{p} = \mathrm{softmax}_{seq\_len}(\mathbf{s}) = \frac{e^{\mathbf{s}}}{\sum_j^{seq\_len} (e^{s_j})}$$

$$\mathbf{c}_{fox} = \mathbf{p} \cdot \mathbf{x}^T$$

# Going more flexible

$$\mathbf{x} = [w_a, w_{quick}, \ldots, w_{dog}]$$

$$\mathbf{q}_{fox} = \mathrm{FCN}_Q(\mathbf{w}_{fox})$$

$$\boxed{\mathbf{V} = \mathrm{FCN}_V(\mathbf{x})}$$

$$\mathbf{s}_{fox} = \mathbf{q}_{fox} x^T$$

$$\mathbf{p} = \mathrm{softmax}_{seq\_len}(\mathbf{s}) = \frac{e^{\mathbf{s}}}{\sum_j^{seq\_len} (e^{s_j})}$$

$$\boxed{\mathbf{c}_{fox} = \mathbf{p} \cdot \mathbf{V}^T}$$

# Simplifying

$$\mathbf{X} = [w_a, w_{quick}, \ldots, w_{dog}]$$

$$\boxed{\begin{aligned} \mathbf{q}_{fox} &= \mathbf{w}_{fox} \cdot \mathrm{W}_Q \\ \mathbf{V} &= \mathbf{X} \cdot \mathrm{W}_V \end{aligned}}$$

$$\mathbf{s}_{fox} = \mathbf{q}_{fox} x^T$$

$$\mathbf{p} = \text{softmax}_{seq\_len}(\mathbf{s}) = \frac{e^{\mathbf{s}}}{\sum_j^{seq\_len} (e^{s_j})}$$

$$\mathbf{c}_{fox} = \mathbf{p} \cdot \mathbf{V}^T$$

# Final touch

$$\mathbf{X} = [w_a, w_{quick}, \ldots, w_{dog}]$$

$$\mathbf{q}_{fox} = \mathbf{w}_{fox} \cdot \mathrm{W}_Q$$

$$\mathbf{V} = \mathbf{X} \cdot \mathrm{W}_V$$

$$\boxed{\begin{aligned} \mathbf{K} &= \mathbf{X} \cdot \mathrm{W}_K \\ \mathbf{s}_{fox} &= \mathbf{q}_{fox} \cdot K^T \end{aligned}}$$

$$\mathbf{p} = \mathrm{softmax}_{seq\_len}(\mathbf{s}) = \frac{e^{\mathbf{s}}}{\sum_j^{seq\_len} (e^{s_j})}$$

$$\mathbf{c}_{fox} = \mathbf{p} \cdot \mathbf{V}^T$$

# Vectorize

$$\mathbf{X} = [w_a, w_{quick}, \ldots, w_{dog}]$$

$$\boxed{\mathbf{Q} = \mathbf{X} \cdot \mathrm{W}_Q}$$

$$\mathbf{V} = \mathbf{X} \cdot \mathrm{W}_V$$

$$\mathbf{K} = \mathbf{X} \cdot \mathrm{W}_K$$

$$\boxed{\begin{aligned} &\mathbf{S} = \mathbf{Q} \cdot K^T \\ &\mathbf{P} = \mathrm{softmax}_{seq\_len}(\mathbf{S}) = \frac{e^{\mathbf{S}}}{\sum_j^{seq\_len} (e^{S_j})} \\ &\mathbf{C} = \mathbf{P} \cdot \mathbf{V}^T \end{aligned}}$$

# Attention

| | |
|---|---|
| Input | $\mathbf{X} = [w_a, w_{quick}, \ldots, w_{dog}]$ |
| Query | $\mathbf{q}_{fox} = \mathbf{w}_{fox} \cdot \mathrm{W}_Q$ |
| Values | $\mathbf{V} = \mathbf{X} \cdot \mathrm{W}_V$ |
| Keys | $\mathbf{K} = \mathbf{X} \cdot \mathrm{W}_K$ |
| Scores | $\mathbf{s}_{fox} = \mathbf{q}_{fox} \cdot K^T$ |
| Attention probabilities | $\mathbf{p} = \mathrm{softmax}_{seq\_len}(\mathbf{s}) = \dfrac{e^{\mathbf{s}}}{\sum_j^{seq\_len}(e^{s_j})}$ |
| Output | $\mathbf{c}_{fox} = \mathbf{p} \cdot \mathbf{V}^T$ |

# Attention

| | |
|---|---|
| Input | $\mathbf{X} = [w_a, w_{quick}, \ldots, w_{dog}]$ |
| Query | $\mathbf{Q} = \mathbf{X} \cdot \mathrm{W}_Q$ |
| Values | $\mathbf{V} = \mathbf{X} \cdot \mathrm{W}_V$ |
| Keys | $\mathbf{K} = \mathbf{X} \cdot \mathrm{W}_K$ |
| Output | $\mathbf{C} = \mathrm{softmax}_{seq\_len}(\mathbf{Q} \cdot \mathbf{K}^{\mathbf{T}}) \cdot \mathbf{V}^T$ |

# Attention: visually

Input $\mathbf{X} = [w_a, w_{quick}, \ldots, w_{dog}]$

Query $\mathbf{Q} = \mathbf{X} \cdot \mathbf{W}_Q$

Values $\mathbf{V} = \mathbf{X} \cdot \mathbf{W}_V$

Keys $\mathbf{K} = \mathbf{X} \cdot \mathbf{W}_K$

Output $\mathbf{C} = \mathrm{softmax}_{seq\_len}(\mathbf{Q} \cdot \mathbf{K}^{\mathbf{T}}) \cdot \mathbf{V}^T$

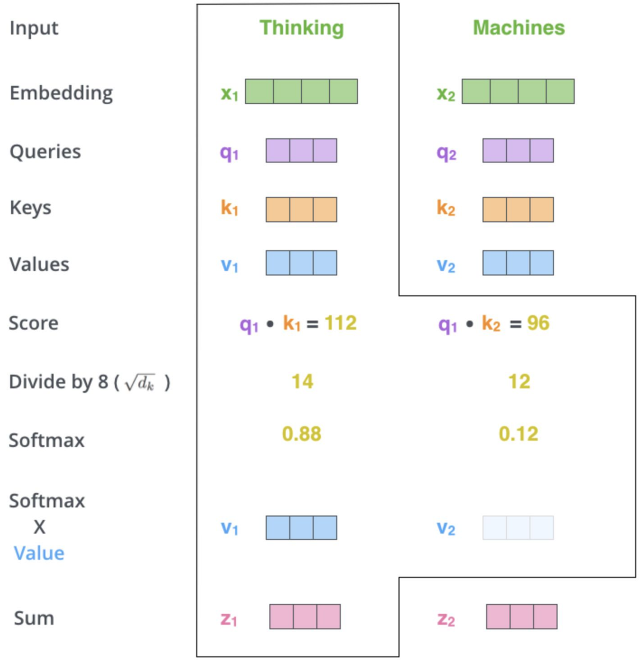
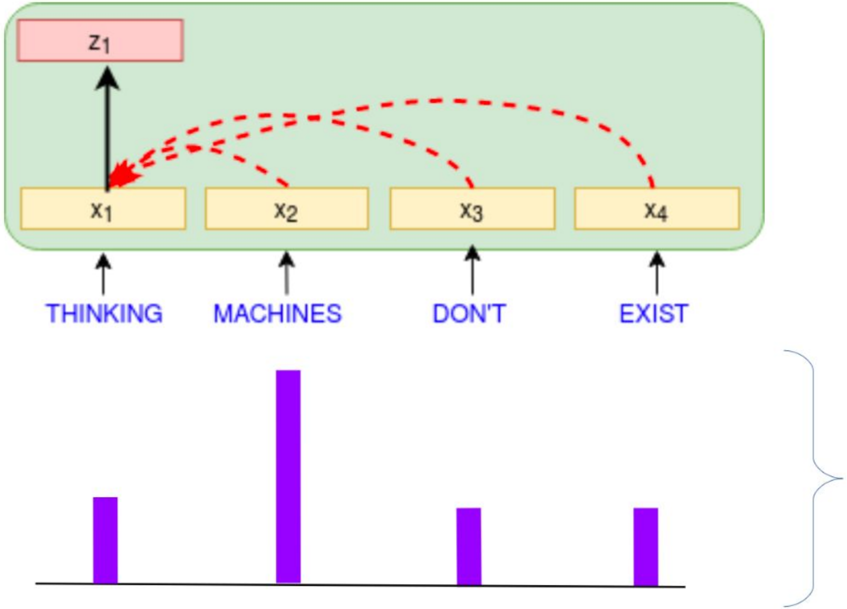



Image credit: Illustrated Transformer, Jay Allamar

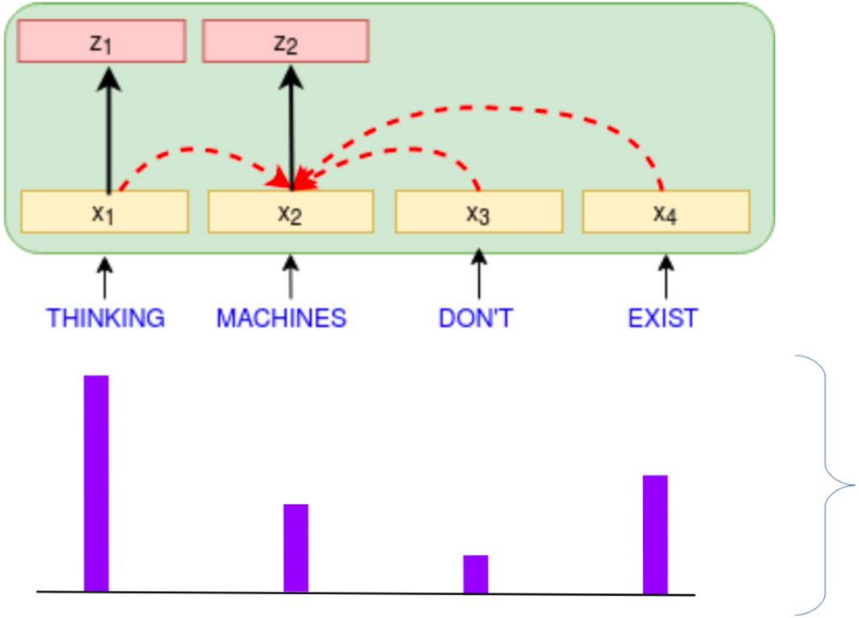
# Small visualization




- For each word, it computes a weighted combination of other words in the input passage.
- Self-attention layer recomputes the representation for every position simultaneously.

Distribution over context words

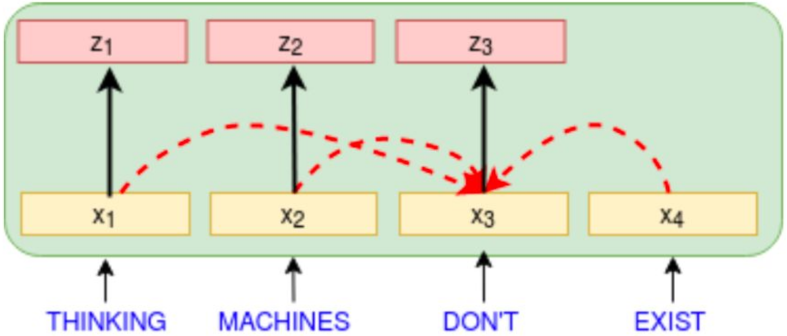
# Small visualization

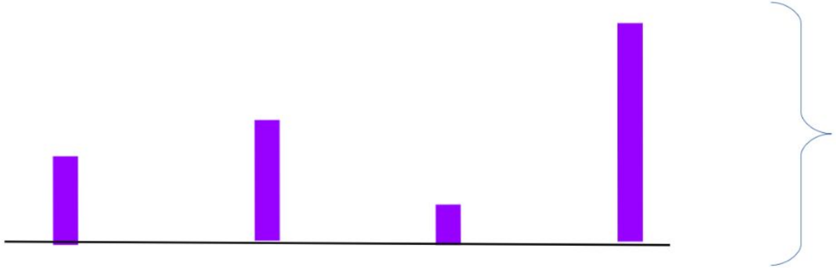



- For each word, it computes a weighted combination of other words in the input passage.
- Self-attention layer recomputes the representation for every position simultaneously.

Distribution over context words

# Small visualization

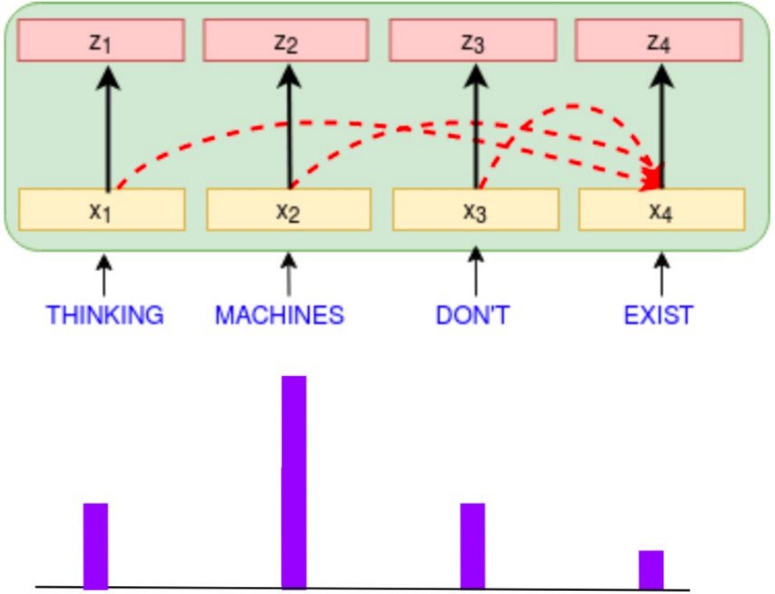


- For each word, it computes a weighted combination of other words in the input passage.
- Self-attention layer recomputes the representation for every position simultaneously.

Distribution over context words

# Small visualization




Distribution over input words

- For each word, it computes a weighted combination of other words in the input passage.
- Self-attention layer recomputes the representation for every position simultaneously.

# Benefits of attention

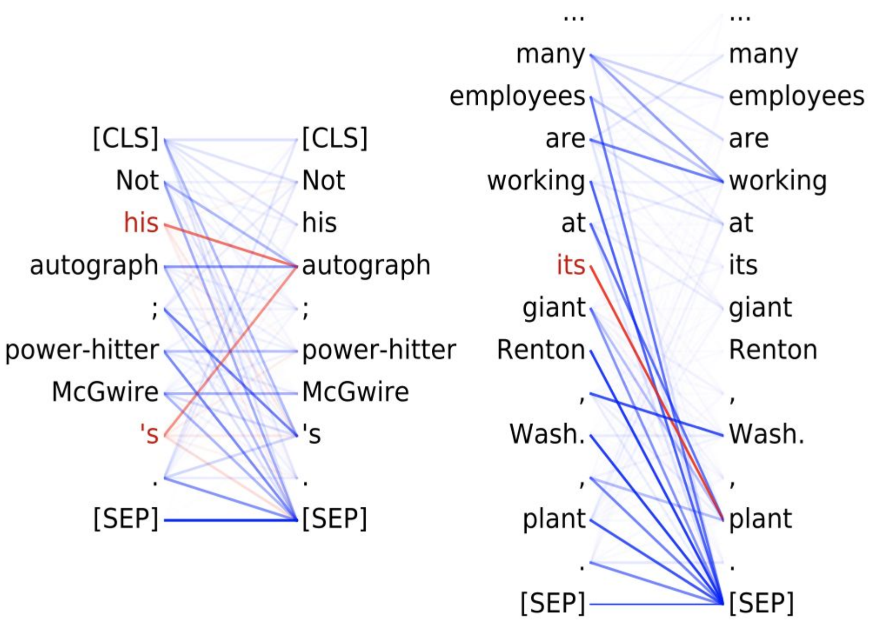
- You can processes input in parallel, much like convolutional network, but
- No need for a hierarchy of layers to process non-local dependencies – can "reach" any position in the input at any time (in every layer)

# Attention

- **Implicitly** learns the relations
  - You just use thi

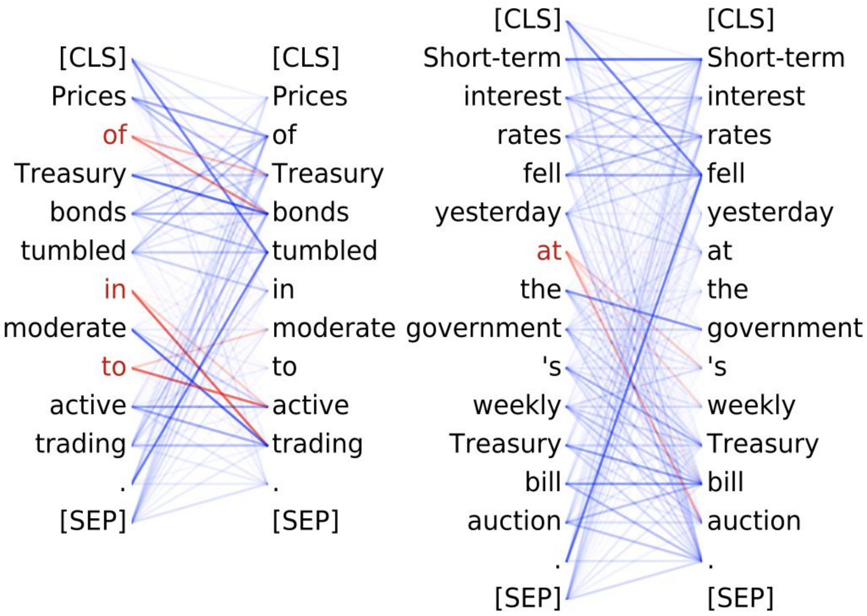
# Attention examples

- **Possessive pronouns** and apostrophes attend to the head of the corresponding NP
- 80.5% accuracy at the **poss** relation



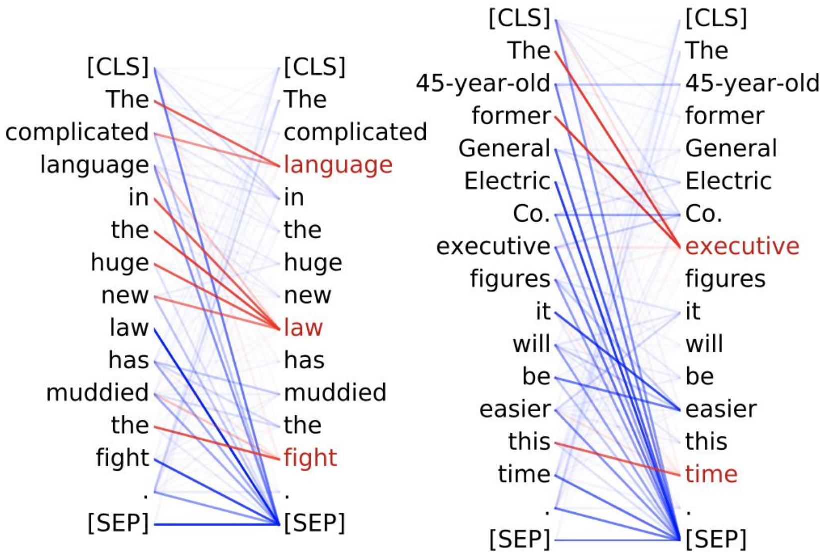
# Attention examples

- **Prepositions** attend to their objects
- 76.3% accuracy at the `pobj` relation



# Attention examples

- **Noun modifiers** (e.g., determiners) attend to their noun
- 94.3% accuracy at the `det` relation



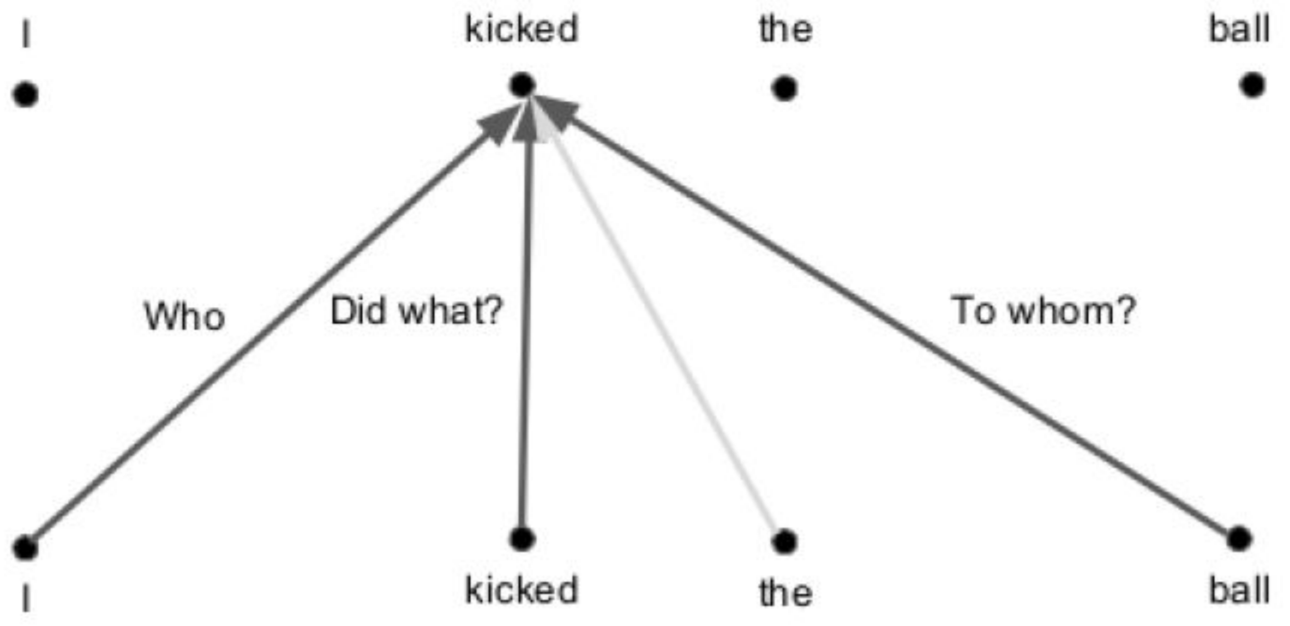
# Multiple heads are better than one

# Multi-head attention

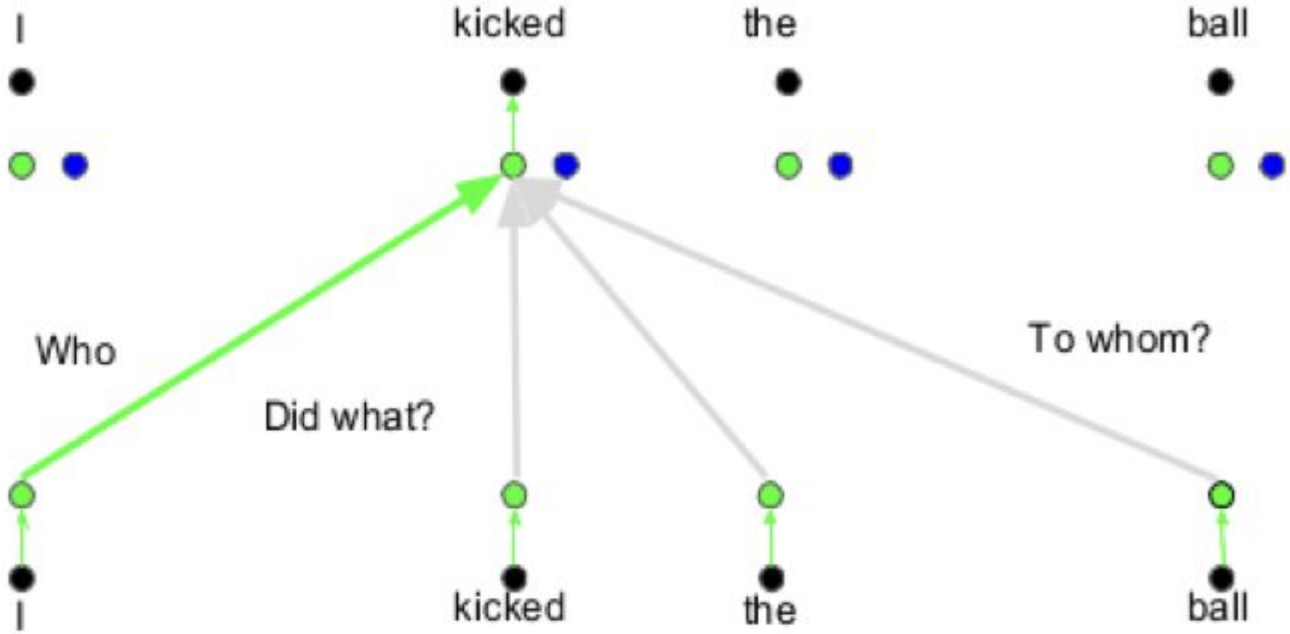
Multiple attentions running in parallel

- Catch multiple dependencies of the same word
- Catch more diverse dependencies

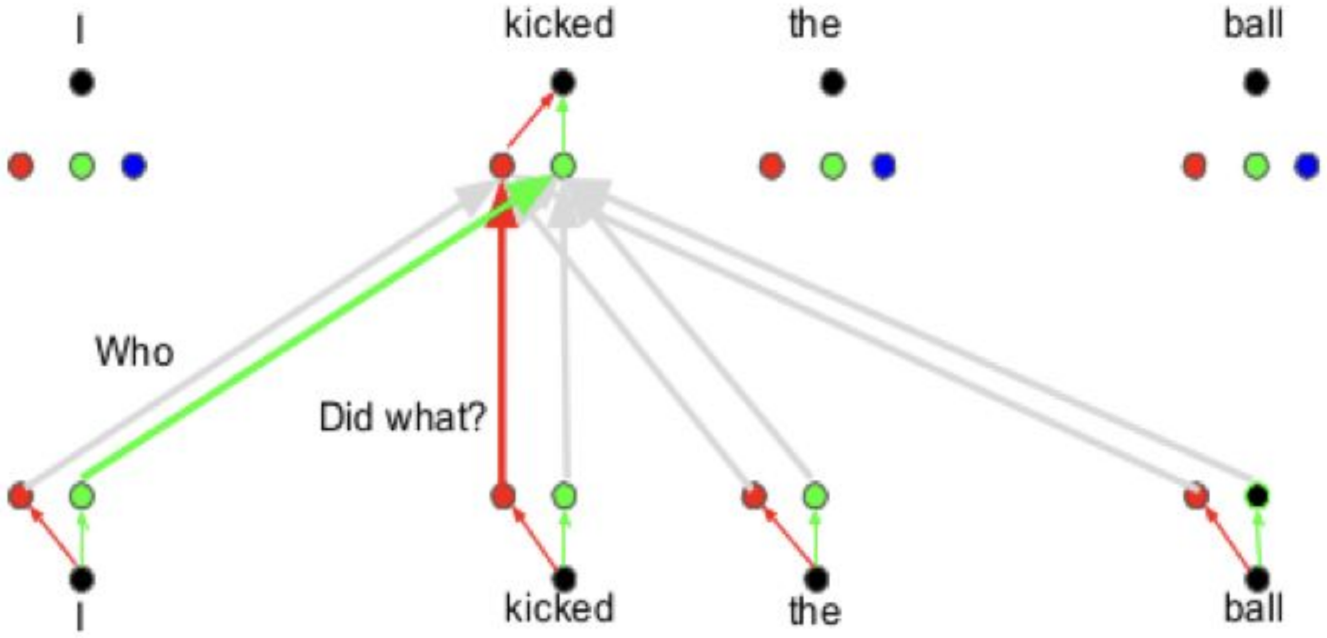
# Multi-head attention: why?



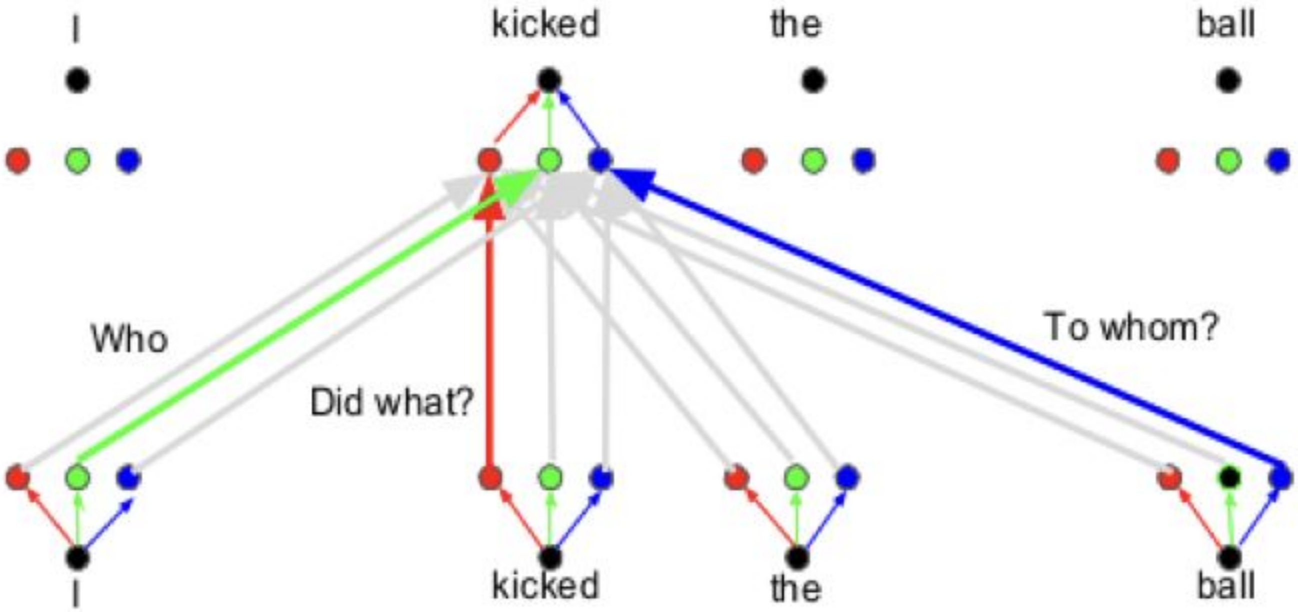
# Multi-head attention: why?



# Multi-head attention: why?



# Multi-head attention: why?




Different heads pick out different information, e.g. one head would pick out direct object of a predicate, the other subject, etc. – almost like feature extraction

# Multi-head attention

$$\mathbf{X} = [w_a, w_{quick}, \ldots, w_{dog}]$$

$$\mathbf{Q_{h1}} = \mathbf{X} \cdot \mathrm{W}_{Q_{h1}}$$

$$\mathbf{V_{h1}} = \mathbf{X} \cdot \mathrm{W}_{V_{h1}}$$

$$\mathbf{K_{h1}} = \mathbf{X} \cdot \mathrm{W}_{K_{h1}}$$

$$\mathbf{C_{h1}} = \mathrm{softmax}_{seq\_len}(\mathbf{Q_{h1}} \cdot \mathbf{K_{h1}^T}) \cdot \mathbf{V_{h1}}^T$$

# Multi-head attention

$$\mathbf{X} = [w_a, w_{quick}, \ldots, w_{dog}]$$

$$\mathbf{Q_{h1}} = \mathbf{X} \cdot \mathrm{W}_{Q_{h1}}$$

$$\mathbf{V_{h1}} = \mathbf{X} \cdot \mathrm{W}_{V_{h1}}$$

$$\mathbf{K_{h1}} = \mathbf{X} \cdot \mathrm{W}_{K_{h1}}$$

$$\mathbf{C_{h1}} = \mathrm{softmax}_{seq\_len}(\mathbf{Q_{h1}} \cdot \mathbf{K_{h1}^T}) \cdot \mathbf{V_{h1}}^T$$

$$\mathbf{Q_{h2}} = \mathbf{X} \cdot \mathrm{W}_{Q_{h2}}$$

$$\mathbf{V_{h2}} = \mathbf{X} \cdot \mathrm{W}_{V_{h2}}$$

$$\mathbf{K_{h2}} = \mathbf{X} \cdot \mathrm{W}_{K_{h2}}$$

$$\mathbf{C_{h2}} = \mathrm{softmax}_{seq\_len}(\mathbf{Q_{h2}} \cdot \mathbf{K_{h2}^T}) \cdot \mathbf{V_{h2}}^T$$

# Multi-head attention

$$\mathbf{X} = [w_a, w_{quick}, \ldots, w_{dog}]$$

$$\mathbf{Q_{h1}} = \mathbf{X} \cdot \mathrm{W}_{Q_{h1}}$$

$$\mathbf{V_{h1}} = \mathbf{X} \cdot \mathrm{W}_{V_{h1}}$$

$$\mathbf{K_{h1}} = \mathbf{X} \cdot \mathrm{W}_{K_{h1}}$$

$$\mathbf{C_{h1}} = \mathrm{softmax}_{seq\_len}(\mathbf{Q_{h1}} \cdot \mathbf{K_{h1}^T}) \cdot \mathbf{V_{h1}}^T$$

$$\mathbf{Q_{h2}} = \mathbf{X} \cdot \mathrm{W}_{Q_{h2}}$$

$$\mathbf{V_{h2}} = \mathbf{X} \cdot \mathrm{W}_{V_{h2}}$$

$$\mathbf{K_{h2}} = \mathbf{X} \cdot \mathrm{W}_{K_{h2}}$$

$$\mathbf{C_{h2}} = \mathrm{softmax}_{seq\_len}(\mathbf{Q_{h2}} \cdot \mathbf{K_{h2}^T}) \cdot \mathbf{V_{h2}}^T$$

$$\mathbf{C} = [\mathbf{C}_{h1} : \mathbf{C}_{h2}]$$

Concatenate the outputs

# Stack more layers

# Why a single multi-head attention is not enough

- Attention only extracts direct relations between a single word and a group of words
- No relations between groups of words
- No multi-hop relations

# Why a single multi-head attention is not enough

Motivating example:

Ban on nude dancing on Governor's desk

# Stacking attention layers

$$\begin{aligned}
\mathbf{X} &= [w_a, w_{quick}, \ldots, w_{dog}] \\
\mathbf{H}_1 &= \text{Attention}(\mathbf{X}) \\
\mathbf{H}_2 &= \text{Attention}(\mathbf{H_1}) \\
\mathbf{H}_3 &= \text{Attention}(\mathbf{H_2}) \\
&\cdots \\
\mathbf{H}_N &= \text{Attention}(\mathbf{H_{N-1}})
\end{aligned}$$

# Accounting for word position

# Attention is position-invariant

- Each word "can" has the same word embedding
  - Keys, queries, and values of each word can will be the same
  - $k = w_{can} @ W_K$, $q = w_{can} @ W_Q$, $v = w_{can} @ W_V$,
  - Scores will be the same
  - Output will be the same

Can a canner can a can?

# Adding positions to the vectors

- A new embedding matrix **P**
  - Not for words
  - For positions
- Add P to X
  - Add P0 to the first word in X
  - Add P1 to the second word in X
  - …

# Adding positions to the vectors

$$\hat{\mathbf{X}} = [\mathbf{w}_{can} + \mathbf{P}_0, \mathbf{w}_a + \mathbf{P}_1, \mathbf{w}_{canner} + \mathbf{P}_2, \mathbf{w}_{can} + \mathbf{P}_3, w_a + \mathbf{P}_4, \mathbf{w}_{can} + \mathbf{P}_5]$$

# Adding positions to the vectors

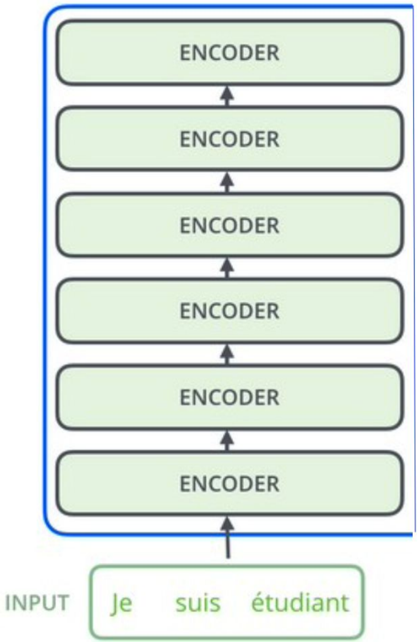
- A new embedding matrix **P**
  - Not for words
  - For positions
- Add P to X
  - Add P0 to the first word in X
  - Add P1 to the second word in X
  - ...
- Now different “can”s have different vectors and attention can distinguish between them

$$\hat{\mathbf{X}} = [\mathbf{w}_{can} + \mathbf{P}_0, \mathbf{w}_a + \mathbf{P}_1, \mathbf{w}_{canner} + \mathbf{P}_2, \mathbf{w}_{can} + \mathbf{P}_3, w_a + \mathbf{P}_4, \mathbf{w}_{can} + \mathbf{P}_5]$$

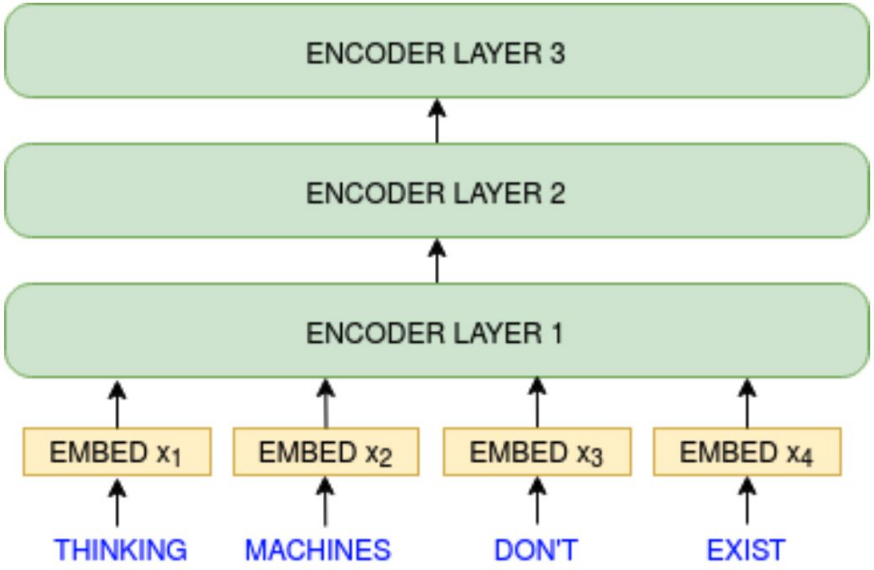
$$\hat{\mathbf{x}}_0 = \mathbf{w}_{can} + \mathbf{P}_0 \neq \hat{\mathbf{x}}_3 = \mathbf{w}_{can} + \mathbf{P}_3$$

# Transformer

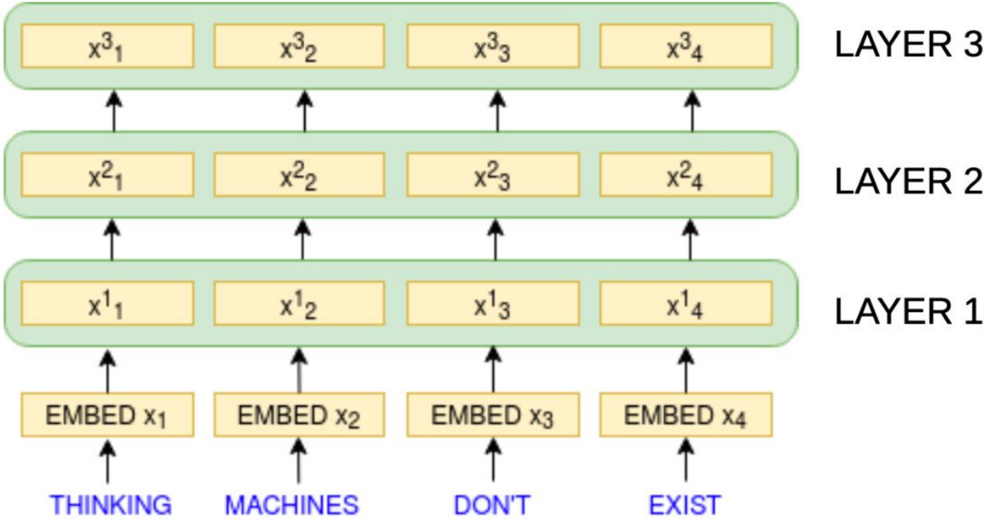
# Transformer



# Transformer



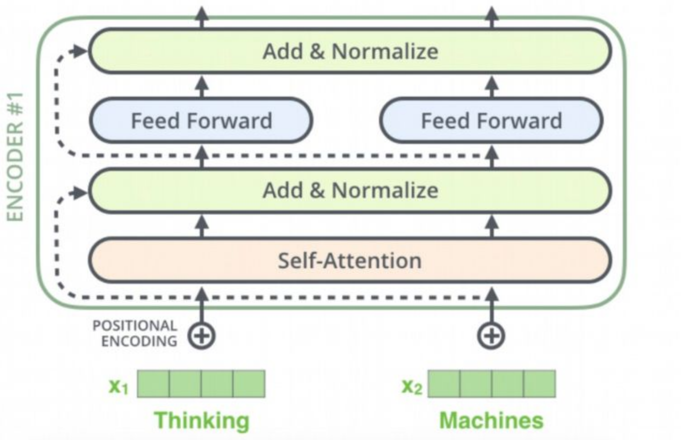
# Transformer




- A representation for each word is updated in each subsequent layer
- Each encoder layer computes a new representation for every position

# Transformer

- Main components (sublayers)
  - Position-wise feed-forward network
  - Multi-head self-attention
- Also used:
  - Skip connections around sublayers
  - Layer normalization



# Transformer

$$h = Attention(x)$$

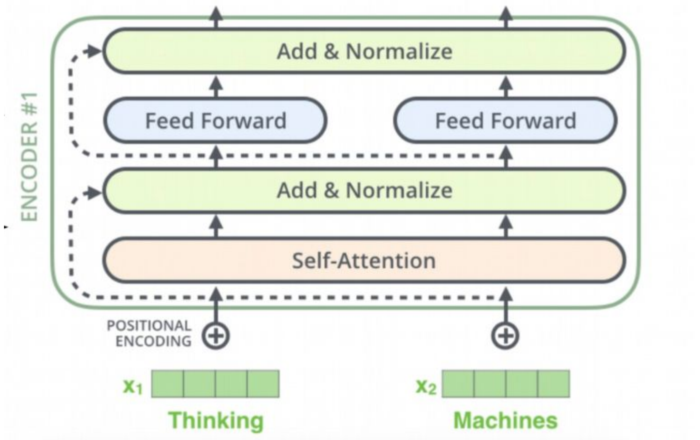
$$h = h + x$$ Skip connection

$$h = LayerNorm(h)$$

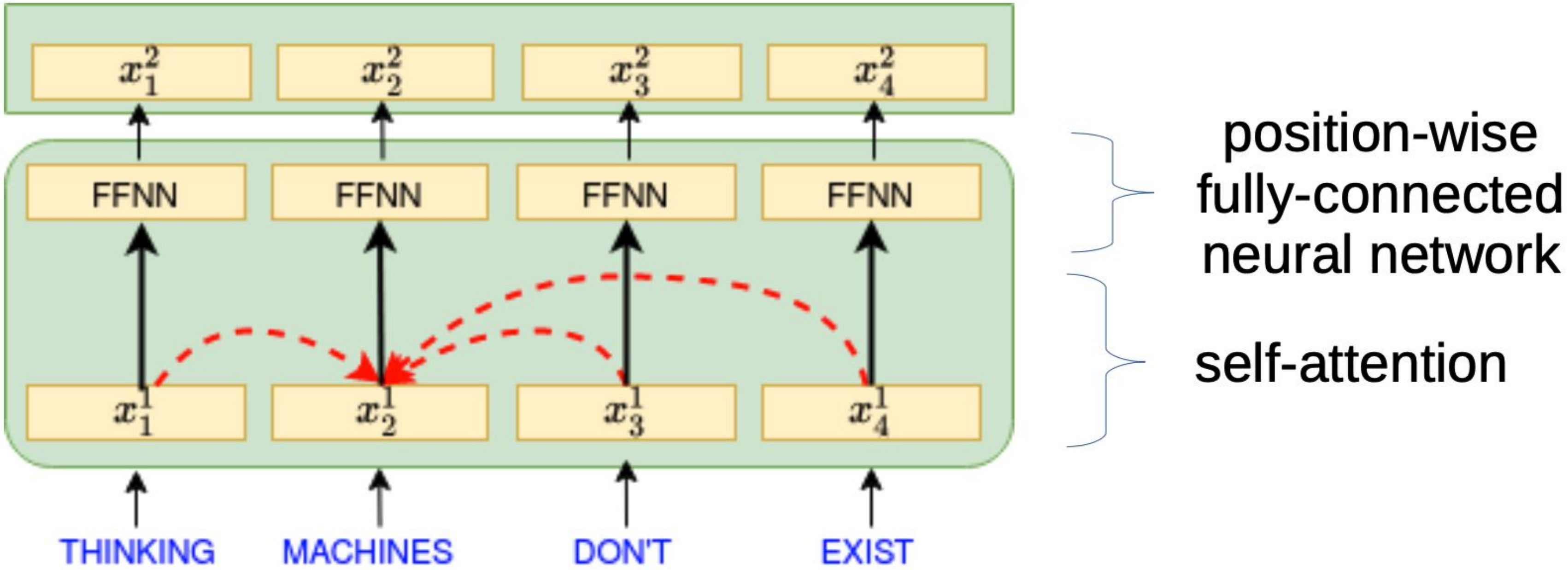
$$h2 = FCN(h)$$

$$h2 = h2 + h$$ Skip connection

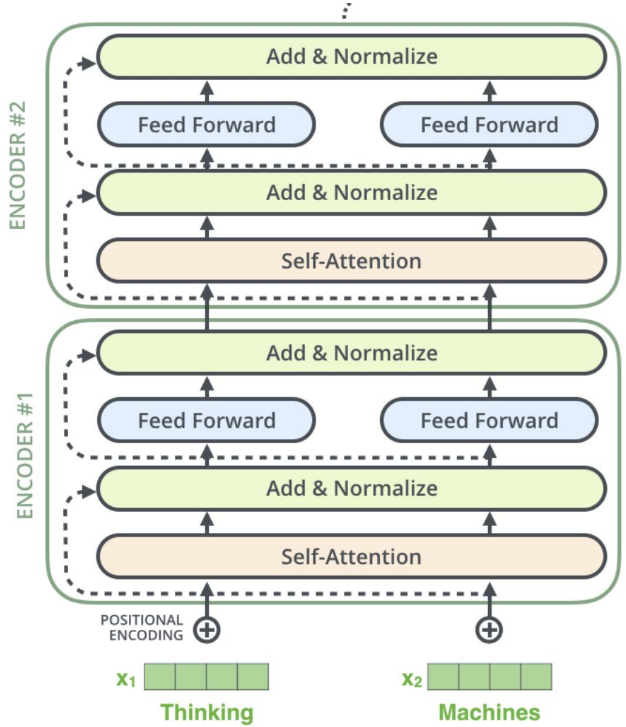
$$h2 = LayerNorm(h2)$$



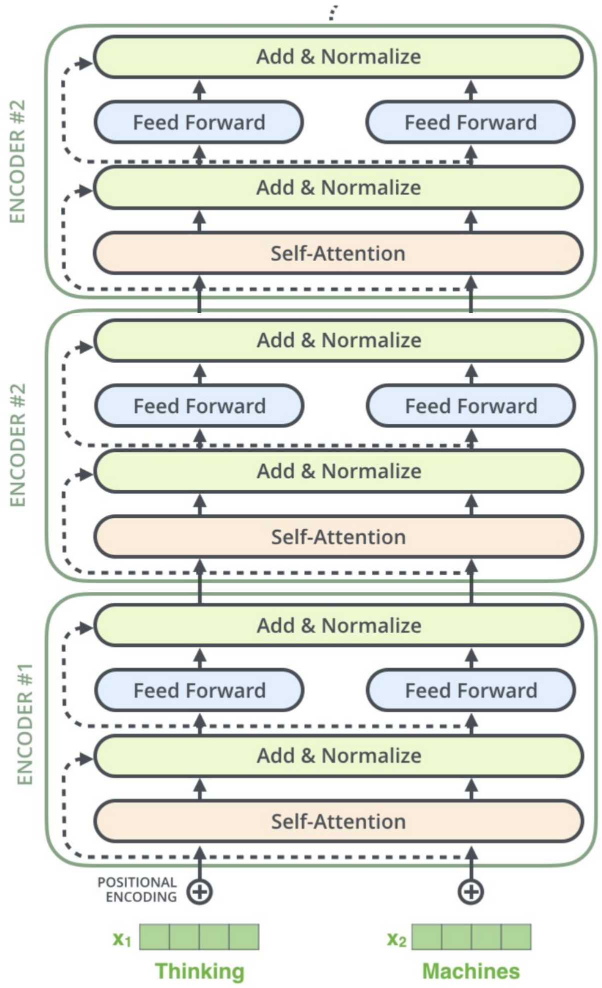
# Transformer



# Transformer



# Transformer



# Language modeling

# Language modeling task

- Given a prefix of n words, predict the next word

  A brown quick fox jumps over the lazy ...

# Language modeling task

- Given a prefix of n words, predict the next word

  A brown quick fox jumps over the lazy ...

- More formally: given a dataset D of sequences and a new sequence $x_1$, ..., $x_{n-1}$ compute the probability of the next element $x_n$ over all of the elements in your vocabulary

$$P(x_n \mid x_1, \ldots, x_{n-1})$$

# Language modeling task

$$P(x_{\textbf{a}} \mid \textbf{a quck brown fox})$$

$$P(x_{\textbf{the}} \mid \textbf{a quck brown fox})$$

$$P(x_{\textbf{she}} \mid \textbf{a quck brown fox})$$

$$\ldots$$

$$P(x_{\textbf{jumps}} \mid \textbf{a quck brown fox})$$

$$\ldots$$

$$P(x_{\textbf{zapateado}} \mid \textbf{a quck brown fox})$$

# Straightforward approach

$$P(x_{\textbf{jumps}} \mid \textbf{a quck brown fox}) = \frac{\text{Count}(\textbf{a quck brown fox jumps})}{\text{Count}(\textbf{a quck brown fox})}$$

# Neural approach

$$P(x_n \,|\, x_1, \ldots, x_{n-1}) = NN(x_1, \ldots, x_{n-1})$$

Given text prefix x1, ..., xn-1 classify this prefix over your vocabulary of words.

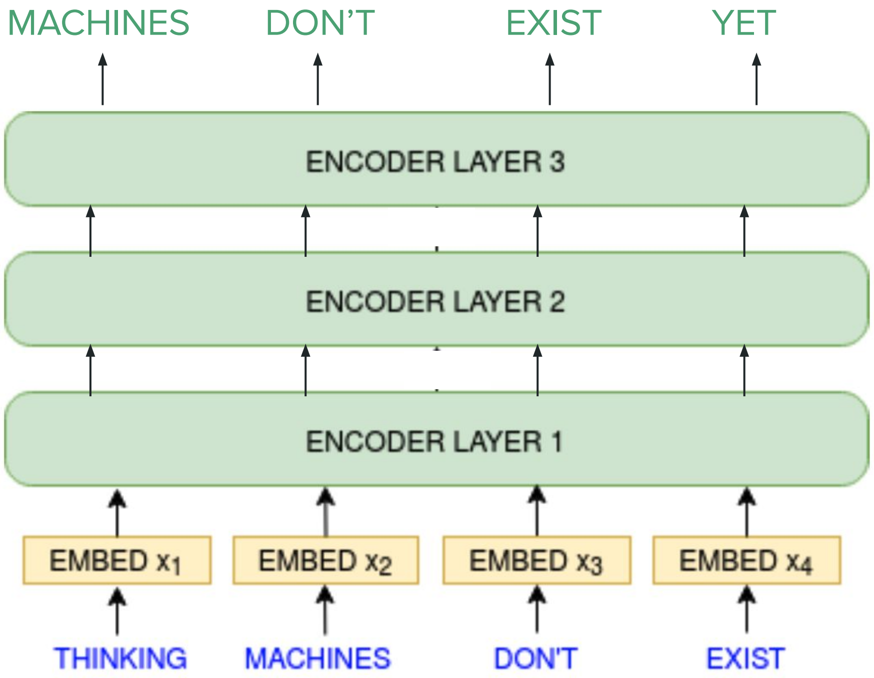
# Language modeling with Transformers

# Transformer approach

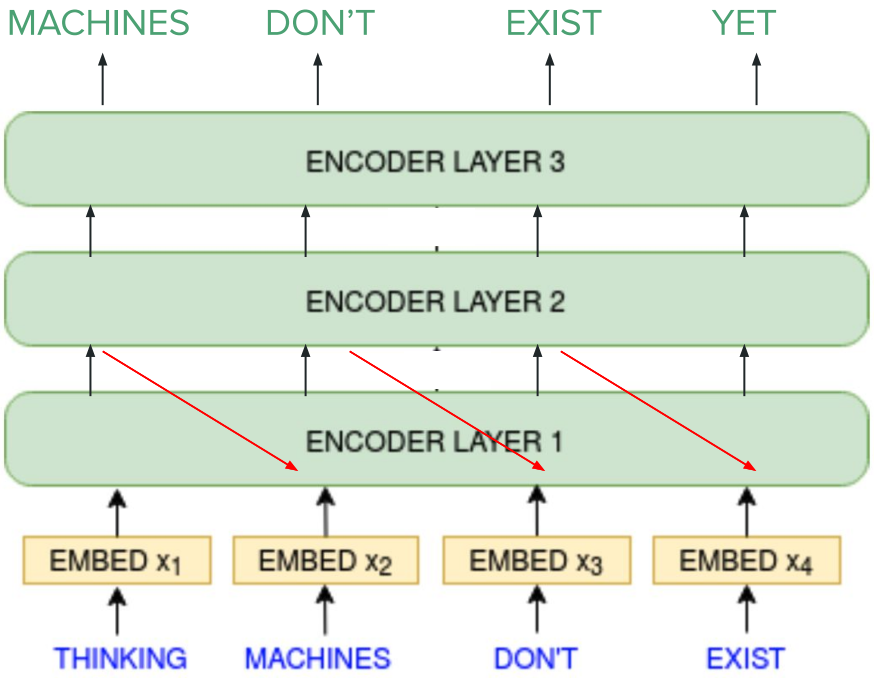
$$P(x_n \mid x_1, \ldots, x_{n-1}) = \text{Transformer}(x_1, \ldots, x_{n-1})$$

Given text prefix x1, ..., xn-1 classify this prefix over your vocabulary of words.

# Effective way of training



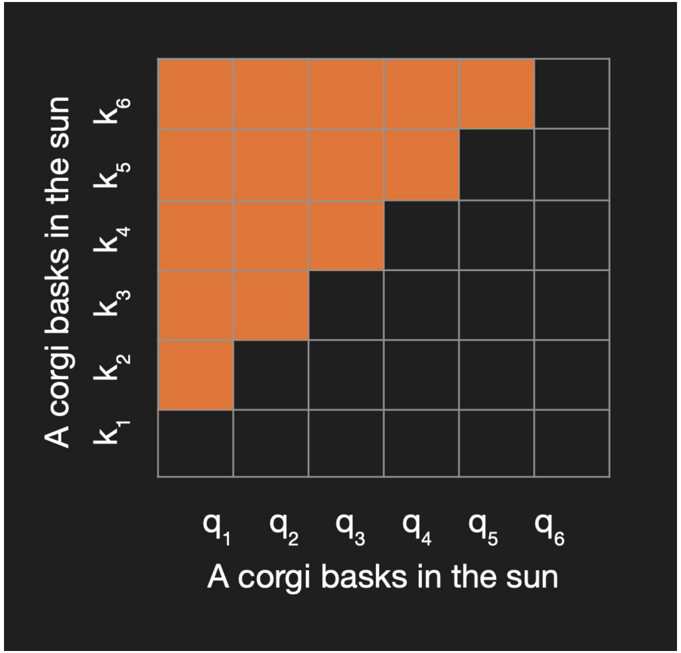
# Attention can cheat



# Limiting attention

$$S = QK^T$$

- Restrict attention from looking at the future tokens
- For token position i, attention probabilities of tokens with position j>i should be 0
- Technical solution:
  - Compute scores as usual
  - Add a masking matrix to the scores
  - 0 on the main diagonal and below it
  - -inf above the main diagonal



# Homework

# Homework

- Part 1: implementing multi-head attention in PyTorch
    - Due next Monday
- Part 2: implementing Transformer Encoder and training a language model
    - Due in two weeks
    - You need a GPU for this one.
      We will release a tutorial how to connect to DAN 417 machines via SSH.
    - Training language model will take **hours.**
      Your first run will fail because of a bug with 99% probability (this is always like this in practice)
      **You need to start early**
- **Will be published later this week**

# Homework. Extra materials

- Lecture notes (you will find them extremely useful)
- The Illustrated Transformer
  - Discusses transformer for seq2seq tasks, but you can read just the Encoder explanation