# Neural Networks for Natural Language Processing

**Prof. Sophie Fellenz**

Week 06 – Attention and Transformer based language models

# Agenda

- Motivation
- Example: RNN bottleneck problem and how to solve with Attention
- Attention and Transformers
- Decodings: Beam search
- Examples: BERT, GPT, PaLM

# Motivation

Do some text generation using GPT3 on:

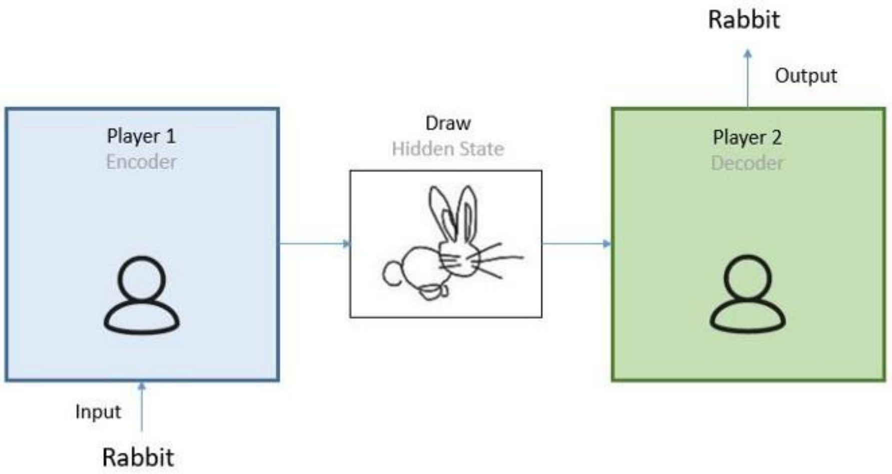
https://beta.openai.com/playground

Feel free to ask it to write anything you want! Let it do your homework! Ask it to write a short story about how an elephant broke into your room and stole your favourite watermelon!

# Setting

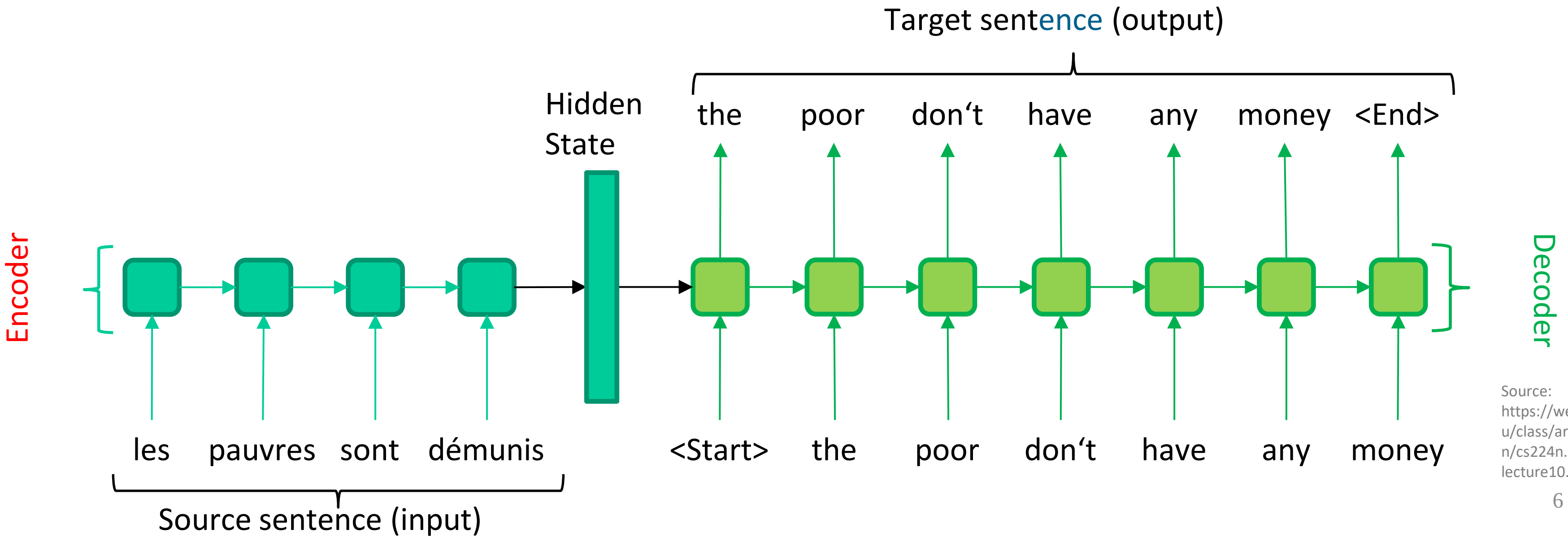
- Task: Translating text from one language to another (e.g. English to French)
- Sequence transduction:
  - Input: Sequence $(x_1, \ldots, x_n)$ (e.g. sequence of English words)
  - Output: Sequence $(z_1, \ldots, z_m)$ (e.g. sequence of French words)

# Encoder-Decoder Models



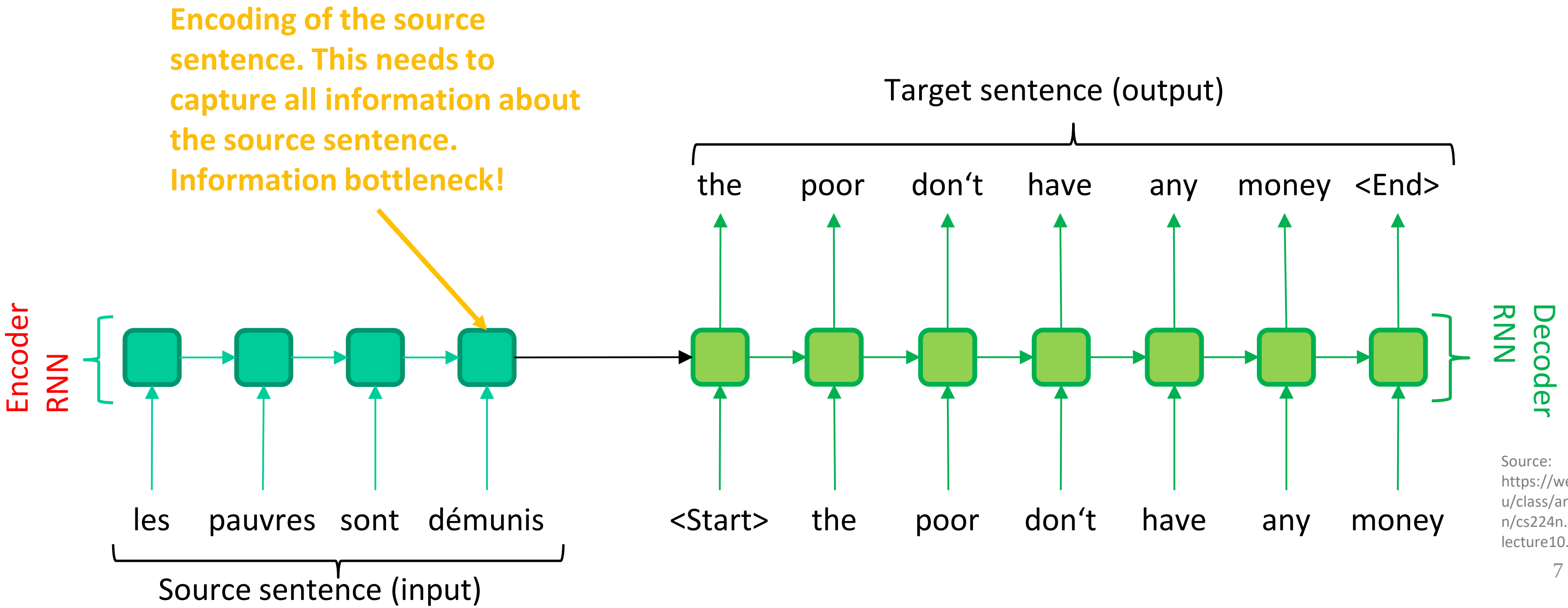

Source: https://towardsdatascience.com/what-is-an-encoder-decoder-model-86b3d57c5e1a

# Encoder-Decoder Models



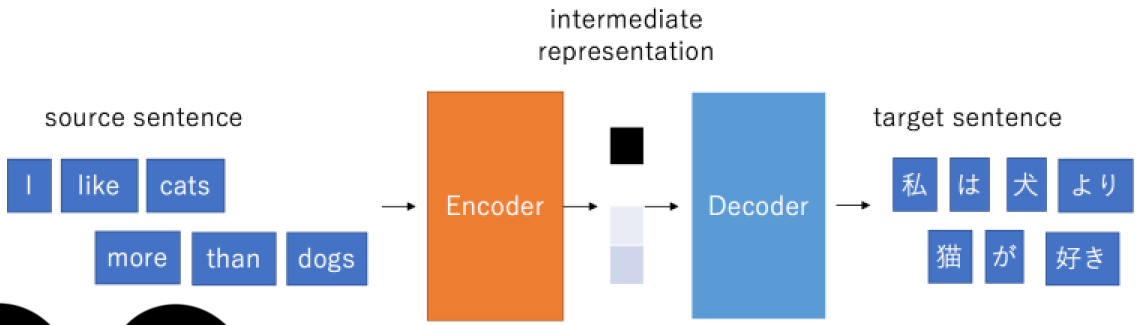

Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

# RNNs: The bottleneck problem




Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

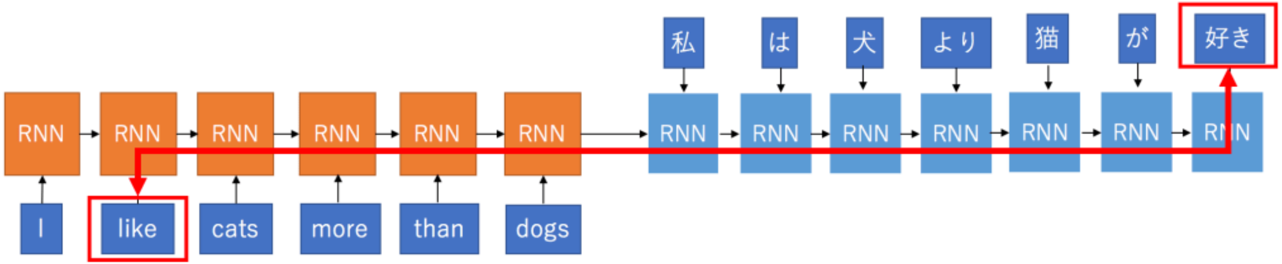
# Sentence Encoding




„You can‘t cram the meaning of a whole %&!$# sentence into a single $&!#* vector!“

Decoder needs different information at each time step.

# Sentence Encoding

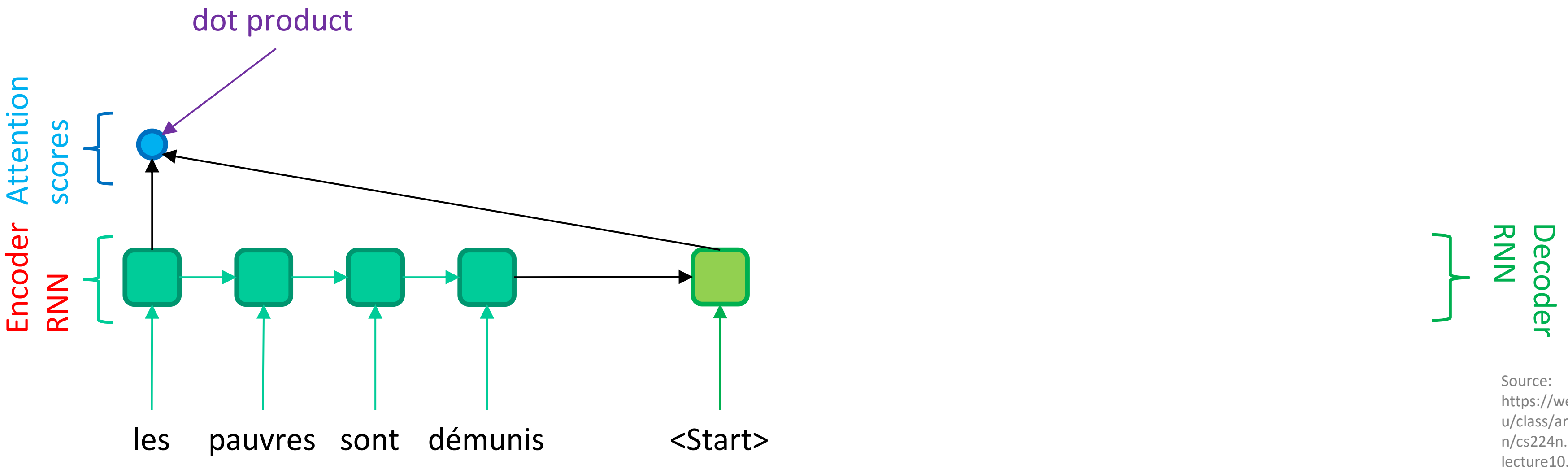



Source: mlexplained.com/2017/12/29/attention-is-all-you-need-explained

# Attention

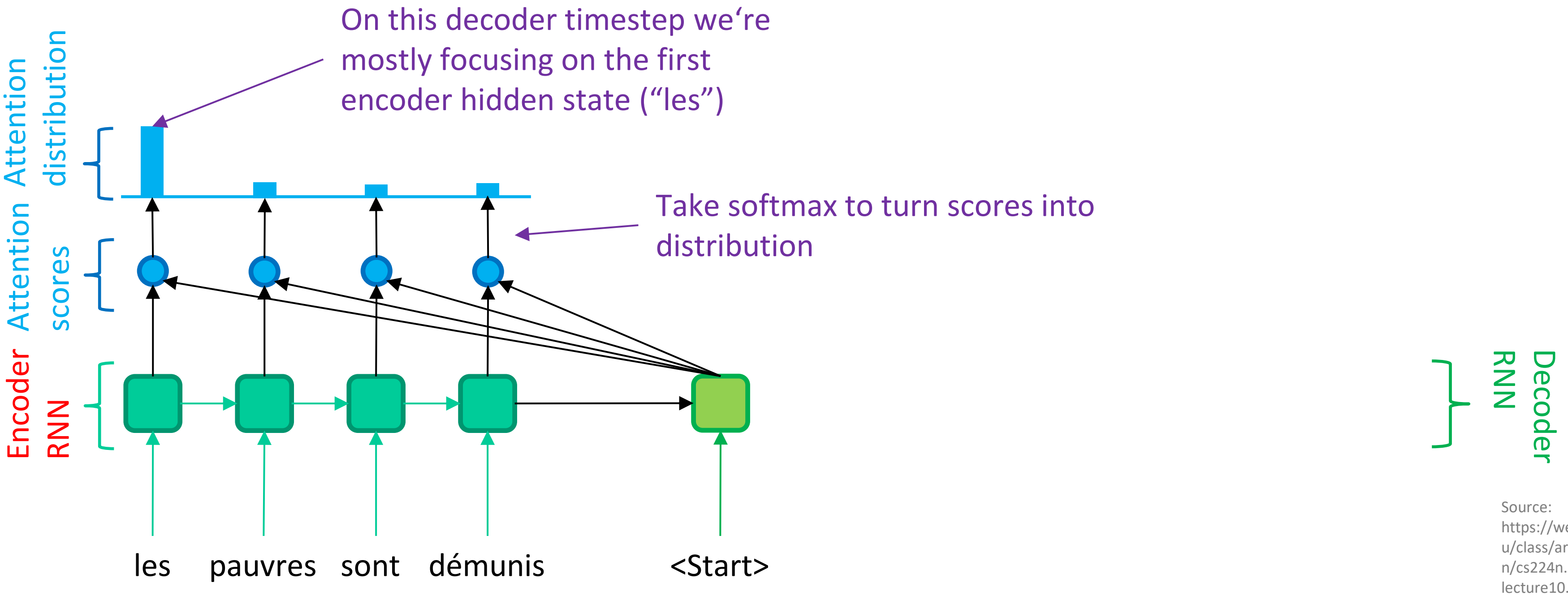
- Three kinds of dependencies are important:
  1. Between input and output tokens
  2. Between input tokens themselves (meanings become apparent in context: „she is taller than me“ vs. „I have no choice other than…“
  3. Between output tokens themselves („neither he nor I knew about deep learning“, „nor“ and „neither“ are dependent)
- **Solution**: Instead of going sequentially through the text with RNNs, why not show all of the input at once and model the dependencies directly!

# Example: Attention Model



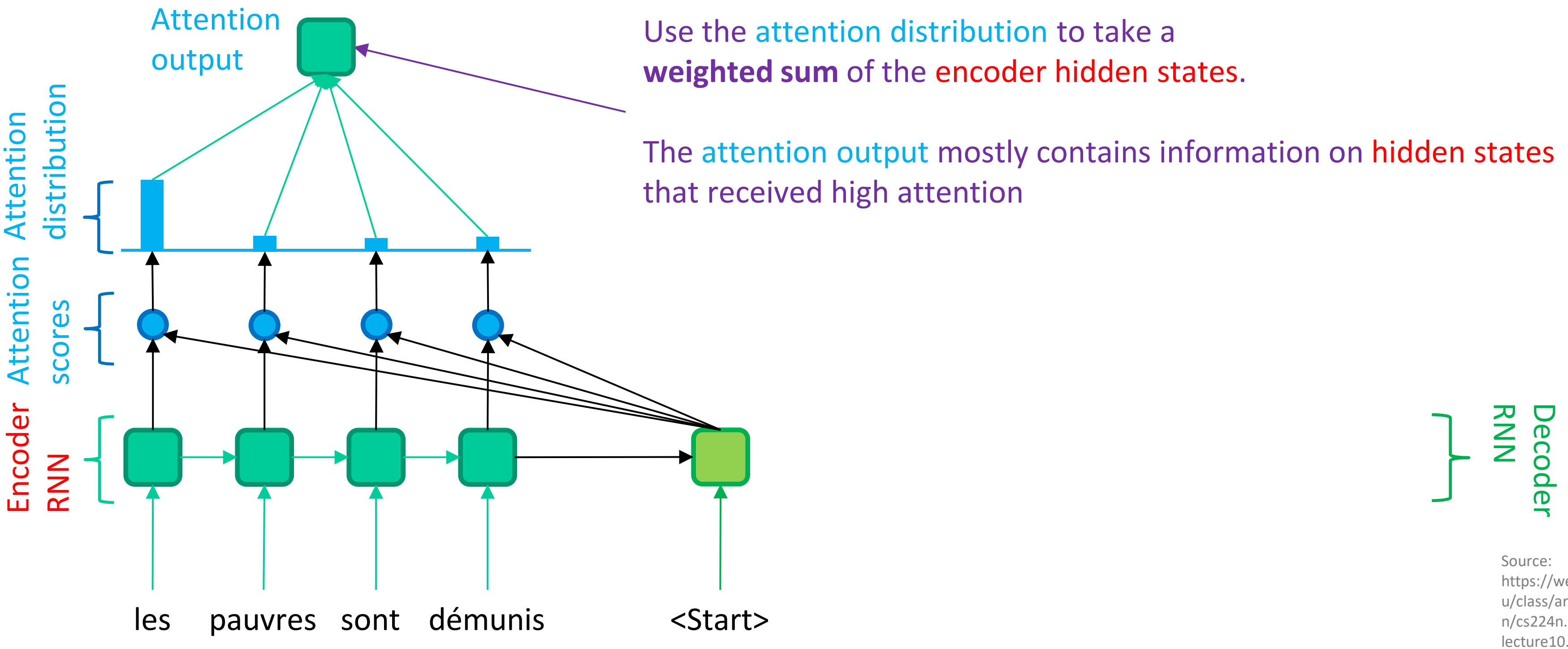

Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

# Example: Attention Model




Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

# Example: Attention Model

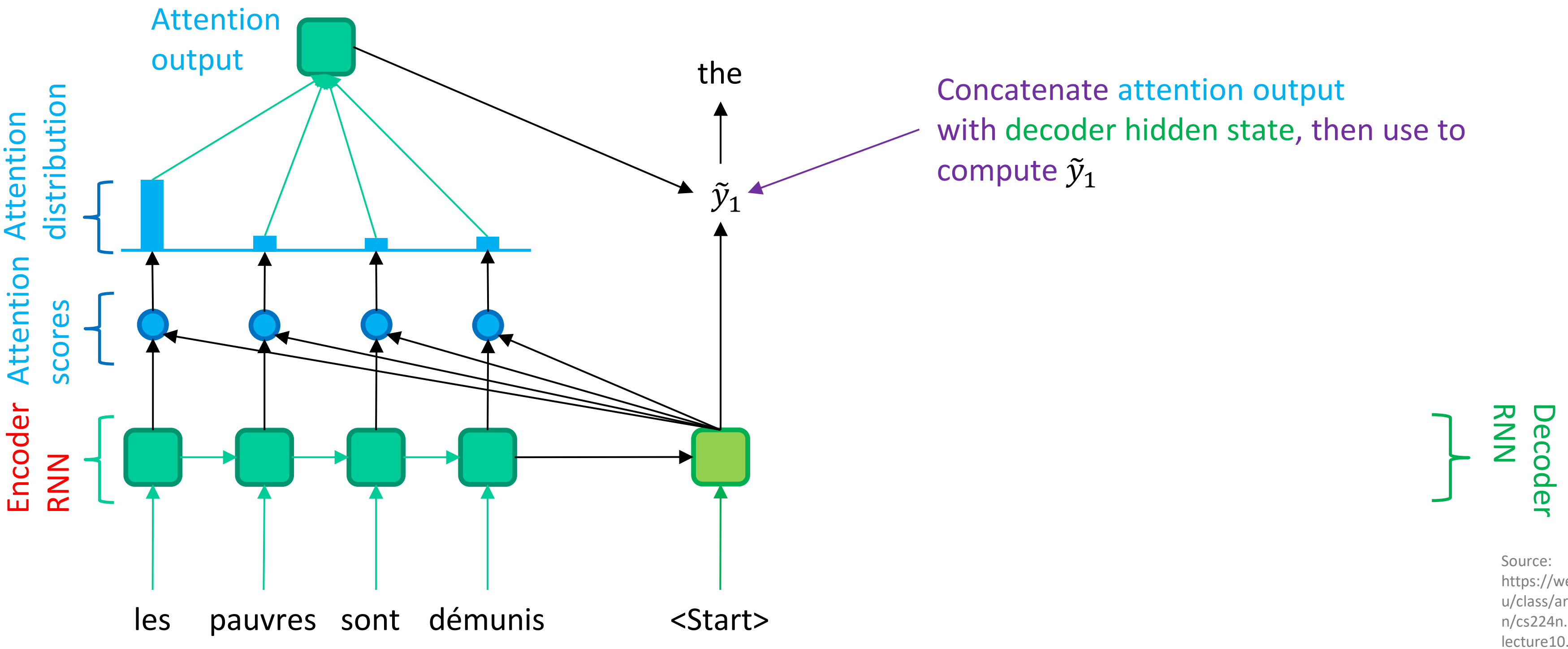



Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information on hidden states that received high attention

Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

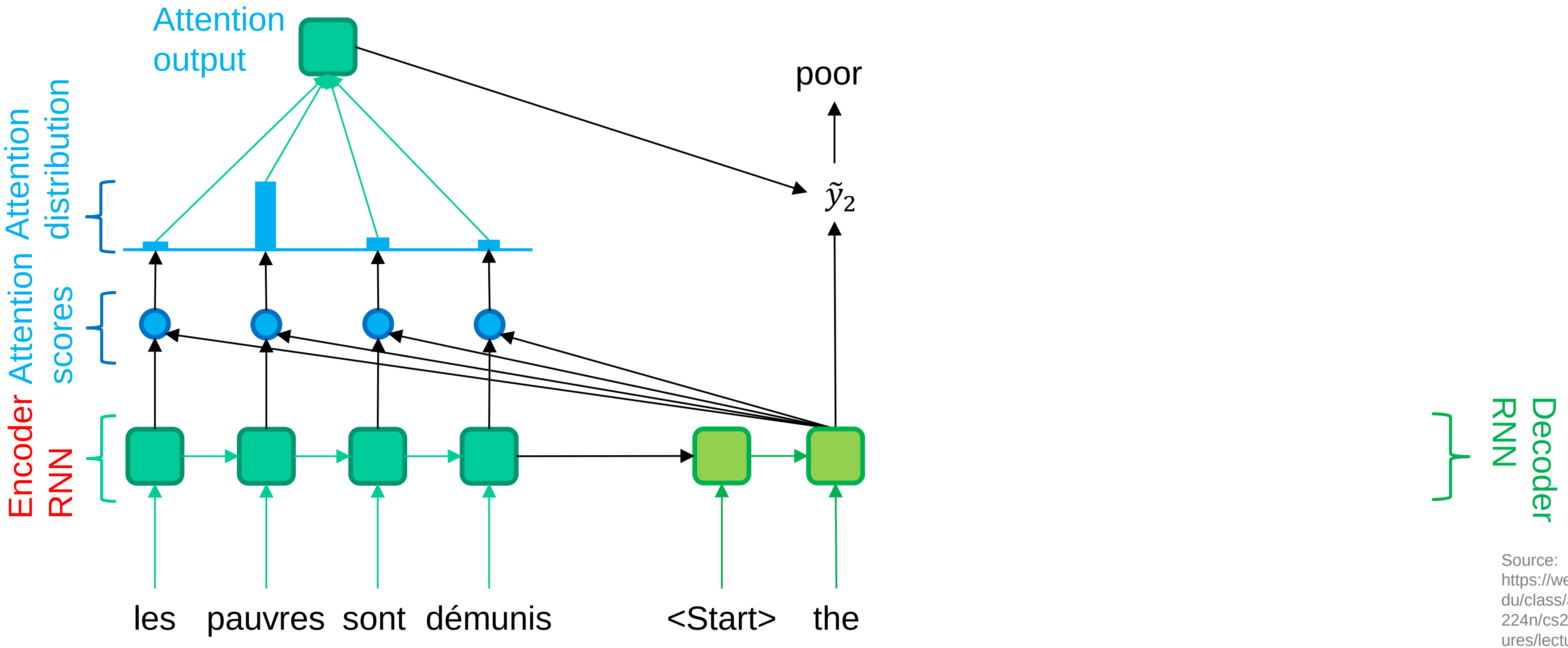
# Example: Attention Model




Concatenate attention output with decoder hidden state, then use to compute $\tilde{y}_1$

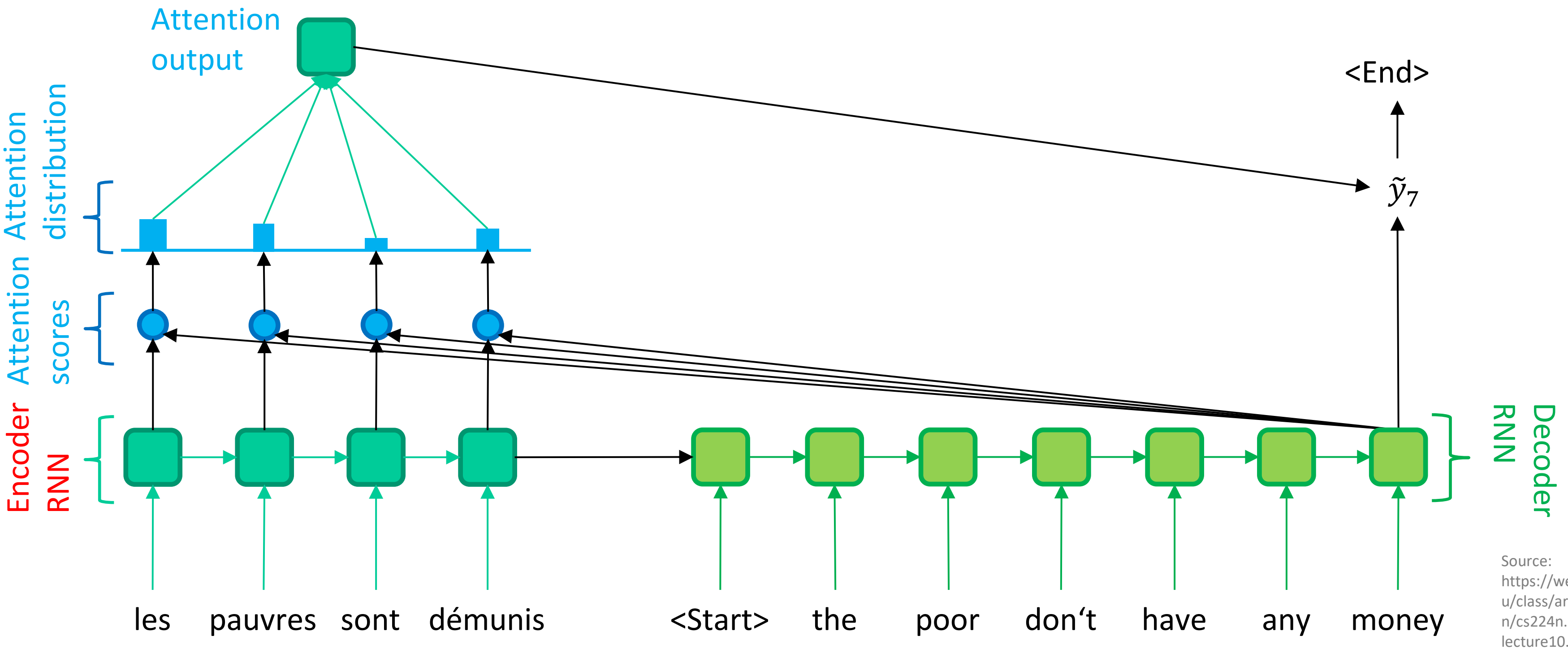
Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

# Example: Attention Model




Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

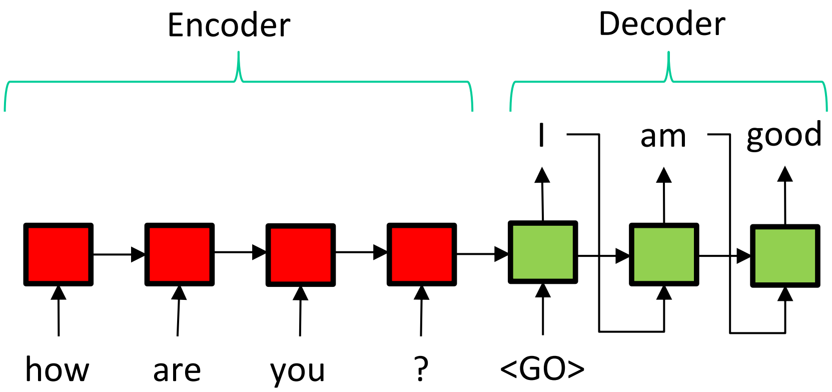
# Example: Attention Model




Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

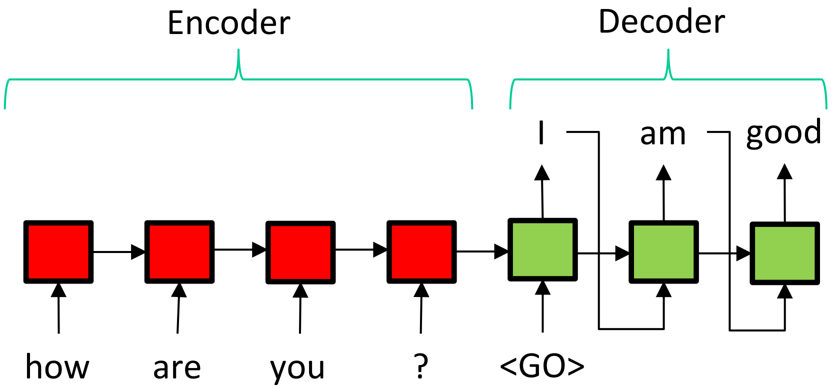
# Why attention?

RPTU




- Before attention we used RNNs (Recurrent Neural Networks)
- The input words are fed sequentially into the model
- Longer sentences are difficult:
  - Inputs get diluted
  - Encoding time increases linearly with the length of the sentence
  - Sequential Computation ⇒ Cannot be sped up

# Why attention?




- Attention can analyze words across a sentence no matter the length
- Calculations with attention can largely be done in parallel
- RNNs only read the input in some ordering (e.g. left to right) whereas attention can connect all words

# Is attention all you need?

https://www.isattentionallyouneed.com/

## Is Attention All You Need?

### Current Status: Yes

Time Remaining: 1141d 0h 59m 13s

### Proposition:

*On January 1, 2027, a Transformer-like model will continue to hold the state-of-the-art position in most benchmarked tasks in natural language processing.*

### For the Motion

Jonathan Frankle

### Against the Motion

Sasha Rush

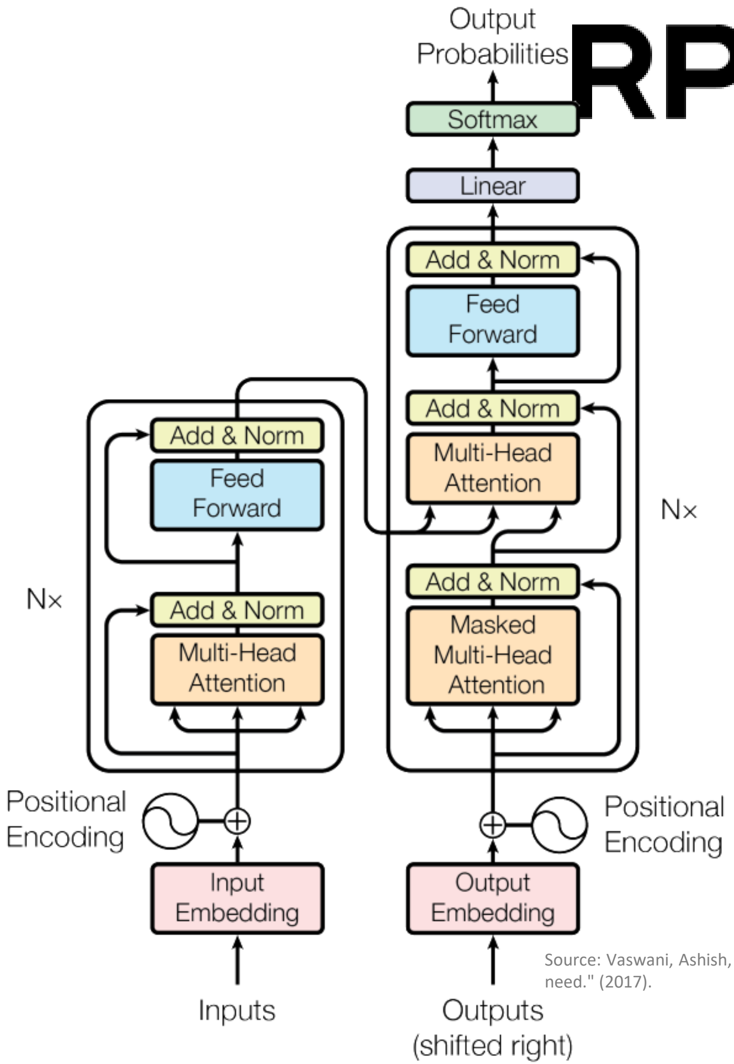
# Transformer

Encoder (left):

- Input: Sequence of words (e.g. English sentence)
- Output: Vector for each word

Decoder (right):

- Input: Previous outputs (e.g. already translated words)
- Output: Probabilities over all possible outputs (e.g. French words)




Source: Vaswani, Ashish, et al. "Attention is all you need." (2017).

# Attention

Problem: We want to extract relations between words

Example: “The girl took the ball and threw it away.” Here “it” refers to the ball.

Solution: Calculate how related any two words are within a sentence

# Scaled Dot-Product Attention

RPTU

- Relation between two words $q, k \in \mathbb{R}^{\mathrm{d_k}}$ (query, key) is given by
- $Attention(q, k, v) = \frac{q^T k}{\sqrt{d_k}} v$
- $q^T k$ large if aligned in same direction $\Rightarrow$ similar words have high attention
- Scale by $1/\sqrt{d_k}$ so gradient doesn't explode during training
- Attention weight is multiplied by some value vector $v$

# Scaled Dot-Product Attention

- For multiple queries $q_{1,\dots,n}$, keys $k_{1,\dots,m}$ and values $v_{1,\dots,m}$ concatenate to matrices $Q \in \mathbb{R}^{\mathrm{n} \times d_k}, K \in \mathbb{R}^{m \times d_k}, V \in \mathbb{R}^{m \times d_v}$

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \in \mathbb{R}^{\mathrm{n} \times d_v}$$

- Here $softmax$ is taken row-wise for each query.

# Scaled Dot-Product Attention

Example:

- Consider input sentence $x_1, \ldots, x_n \in \mathbb{R}^{d_k}$ which we use as queries, keys and values
- The Attention weight calculates how related a query word is to any of the keys

| | | My | dog | loves | good | belly | rubs |
|---|---|---|---|---|---|---|---|
| | | $k_1$ | $k_2$ | $k_3$ | $k_4$ | $k_5$ | $k_6$ |
| My | $q_1$ | $q_1^T k_1$ | … | … | … | … | $q_1^T k_6$ |
| dog | $q_2$ | … | … | … | … | … | … |
| loves | $q_3$ | … | … | … | … | … | … |
| good | $q_4$ | … | … | … | … | … | … |
| belly | $q_5$ | … | … | … | … | … | … |
| rubs | $q_6$ | $q_6^T k_1$ | … | … | … | … | $q_6^T k_6$ |

$QK^T$

# Scaled Dot-Product Attention

$\frac{1}{\sqrt{d_k}}$ and $softmax$

| | | | | | |
|---|---|---|---|---|---|
| $q_1^T k_1$ | … | … | … | … | $q_1^T k_6$ |
| … | … | … | … | … | … |
| … | … | … | … | … | … |
| … | … | … | … | … | … |
| … | … | … | … | … | … |
| $q_6^T k_1$ | … | … | … | … | $q_6^T k_6$ |


$$softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

# Scaled Dot-Product Attention

- For each $q_i$ we output an Attention weighted average of $x_1, \dots, x_n$

| | $k_1$ | $k_2$ | $k_3$ | $k_4$ | $k_5$ | $k_6$ | **Output** |
|---|---|---|---|---|---|---|---|
| $q_1$ | | | | | | | $w_{1,1}v_1 + \cdots + w_{1,6}v_6$ |
| $q_2$ | | | | | | | ... |
| $q_3$ | | | | | | | ... |
| $q_4$ | | | | | | | ... |
| $q_5$ | | | | | | | ... |
| $q_6$ | | | | | | | $w_{6,1}v_1 + \cdots + w_{6,6}v_6$ |

Attention Weights $W = (w_{i,j})$

# Multi-Head Attention

- Problem: Attention for the same word is highest since then $q = k$ which isn‘t useful.
- Solution: First use linear transformation on $Q, K$ and $V$. Calculate attention for multiple transformations and take weighted average.

# Multi-Head Attention

- Linear transformation on $Q, K, V$:

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

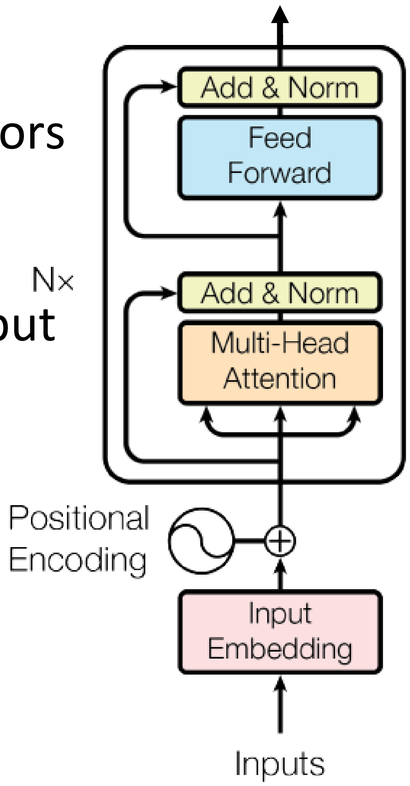
- Take weighted average:

$$Multihead(Q, K, V) = Concat(head_1, \dots, head_h)W^O \in \mathbb{R}^{n \times d_k},$$

- Matrices $W_i^Q, W_i^K, W_i^V$ and $W^O$ are trainable linear transformations.
- Outputs a vector for each query

# Encoder

- Input: Sequence of words
- Embed and add positional information to obtain vectors
- Use vectors as queries, keys and values for Multi-Headed Attention
- Multi-Headed Attention outputs a vector for each input word
- Add more complexity with Feed Forward Layers
- Output vectors of the Multi-Head Attention are fed through the Feed Forward Layer independently
  $\Rightarrow$ Output vector for each word
- We stack multiple of these Encoder blocks (e.g. $N$=8)




Source: Vaswani, Ashish, et al. "Attention is all you need." (2017).

# Positional Encoding

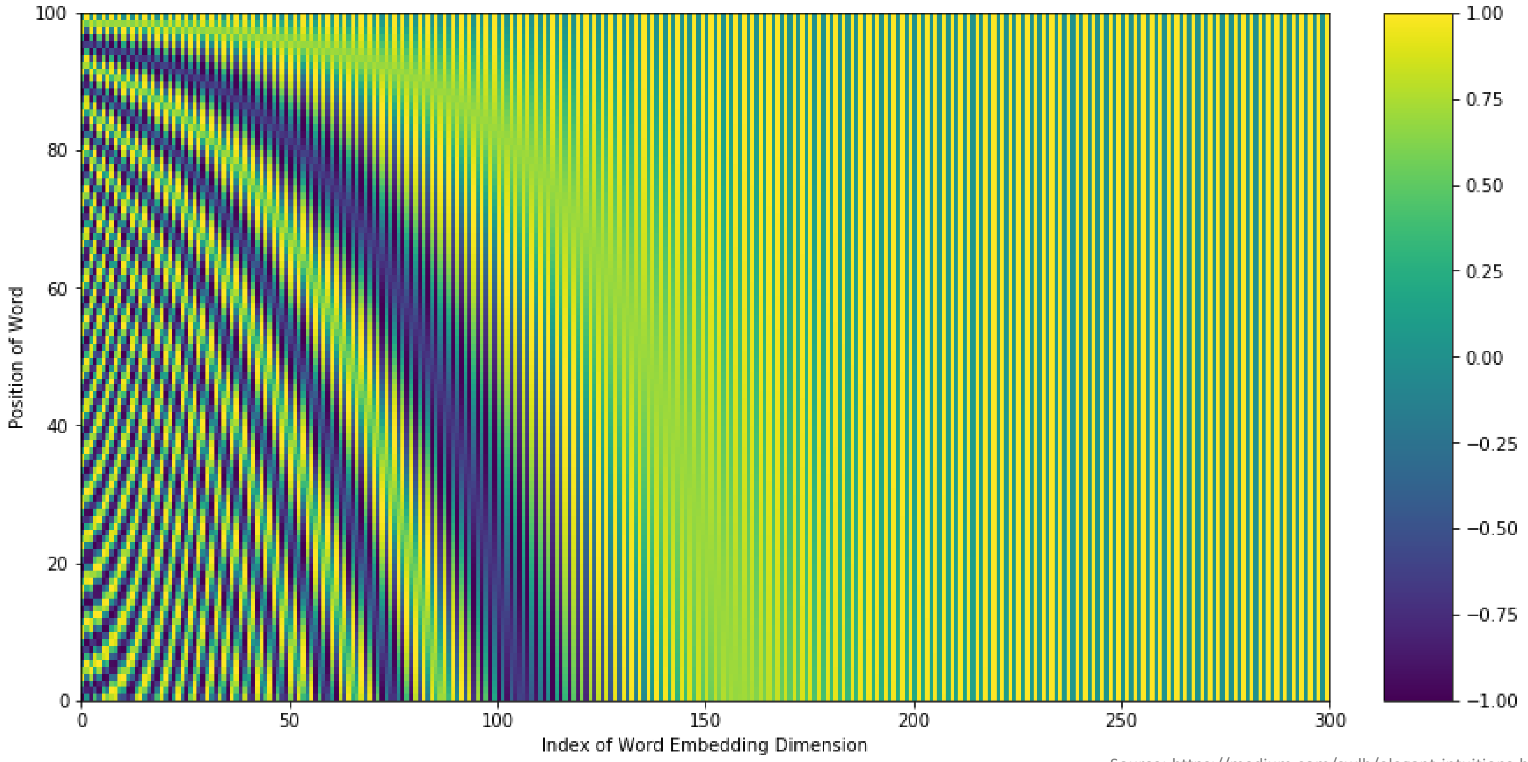
- Problem: Transformers do not take position into account
- Solution: Add positional encoding to each embedded input vector

# Positional Encoding

- Example:
  - Input embedding: $(x_1, \dots, x_n)$
  - Embedding for one word: $x_k = (x_{k,1}, \dots, x_{k,d}) \in \mathbb{R}^d$
  - Add positional encoding $PE(k,i)$ to $x_{k,i}$

$$PE(k,i) = \sin\left(k/10000^{\frac{2i}{d}}\right) \text{ if } i \text{ is even}$$
$$PE(k,i) = \cos\left(k/10000^{\frac{2i}{d}}\right) \text{ else.}$$

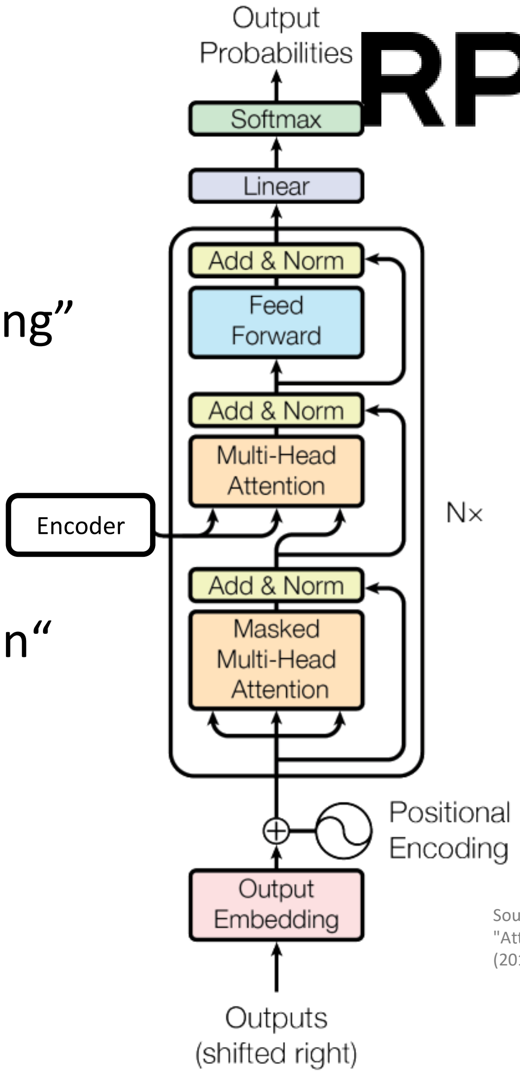
# Positional Encoding




Source: https://medium.com/swlh/elegant-intuitions-behind-positional-encodings-dc48b4a4a5d1

# Decoder

- Assume we are translating “The dog is running” into French (“Le chien court vite”)
- Generate translation word by word in the decoder
- Assume we have already generated „Le chien“
- Input: Already generated words (Le chien)
- Embed words and add positional encoding




Source: Vaswani, Ashish, et al. "Attention is all you need." (2017).

# Masked Multi-Head Attention

- Aim: Learn to generate next word for translation given original sentence and all of the previously generated words
- Consequence: Attention weight for key words after the query word are set to 0
- Example: $Attention(Chien, Le, v) \in \mathbb{R}^{d_v}$

$$Attention(Le, Chien, v) = 0$$

# Scaled Dot-Product Attention + Mask

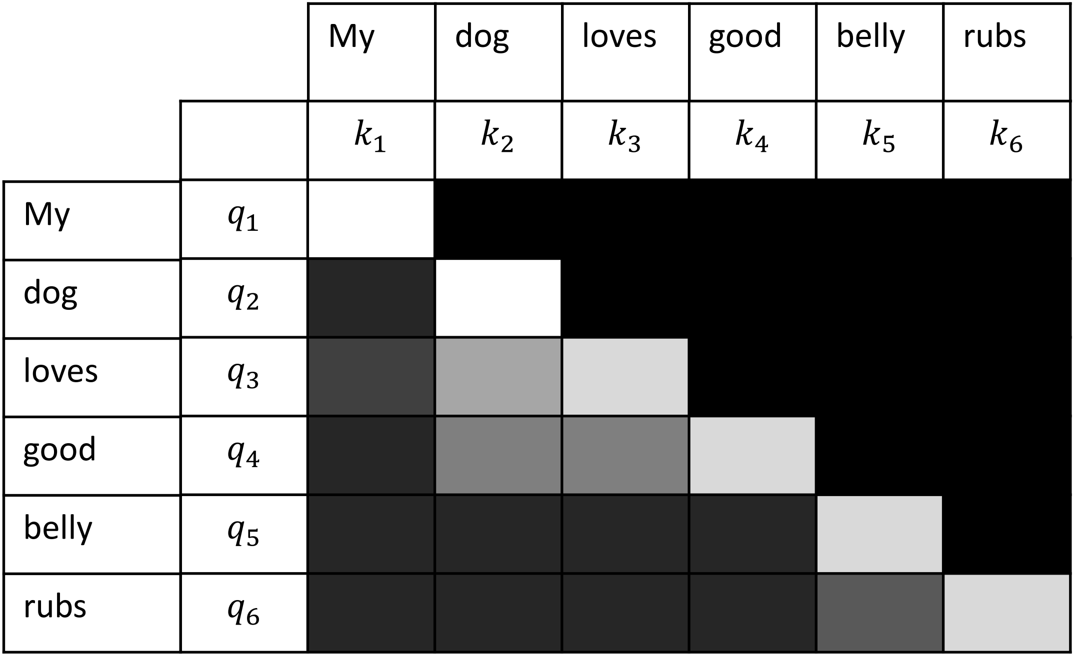
- For multiple queries $q_{1,\ldots,n}$, keys $k_{1,\ldots,m}$ and values $v_{1,\ldots,m}$ concatenate to matrices $Q \in \mathbb{R}^{n \times d_k}, K \in \mathbb{R}^{m \times d_k}, V \in \mathbb{R}^{m \times d_v}$

$$Attention(Q, K, V) = softmax\left(\frac{QK^T + M}{\sqrt{d_k}}\right)V \in \mathbb{R}^{n \times d_v}$$

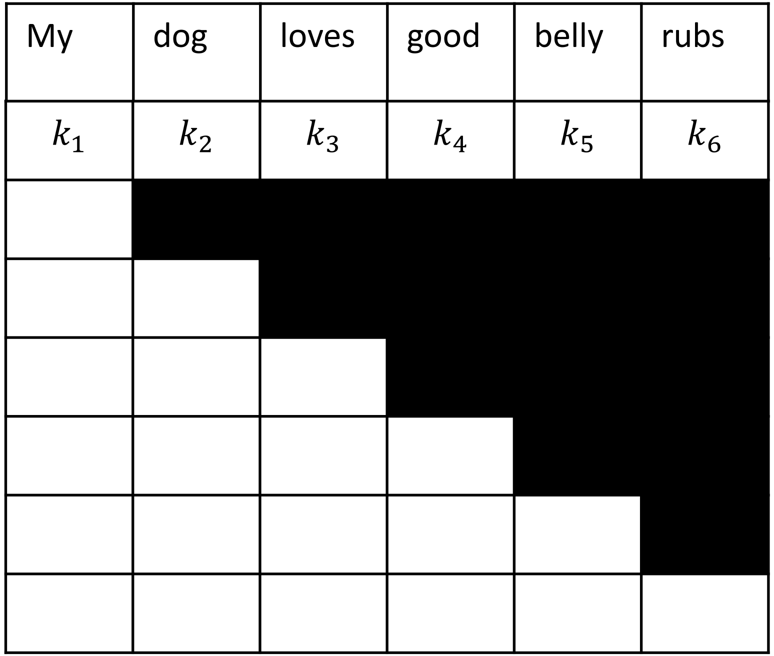
- Here $softmax$ is taken row-wise for each query.
- Mask $M \in \mathbb{R}^{n \times m}$
- Set those values we want to mask to –inf to obtain a 0 after the softmax

# Masked Multi-Head Attention

**Attention matrix**: black is zero attention

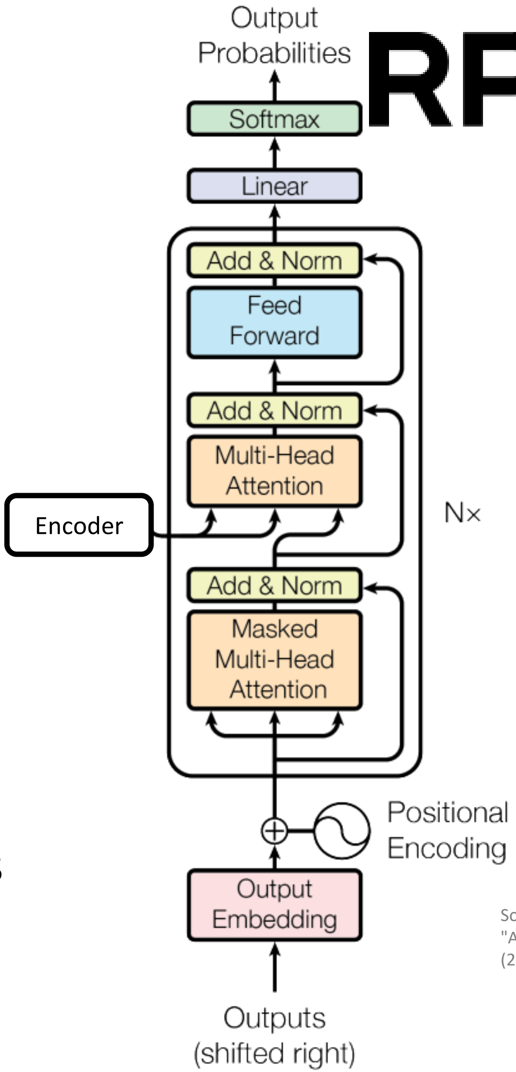



**Mask**: black is –inf, white is 0



# Decoder

- Next Multi-Head Attention:
  $K, V$ from the Encoder Output
  $Q$ from the Masked Multi-Head Attention
- Use Feed Forward layers as in the Encoder
- Stack multiple Decoder blocks (e.g. $N = 8$)
- Use Classification Layer over French vocabulary:
  - Linear Layer with output size as large as number of French words
  - Softmax ⇒ Distribution over French words
- Sample word with highest probability (e.g. “court”) to generate next word




Source: Vaswani, Ashish, et al. "Attention is all you need." (2017).

# Attention versions

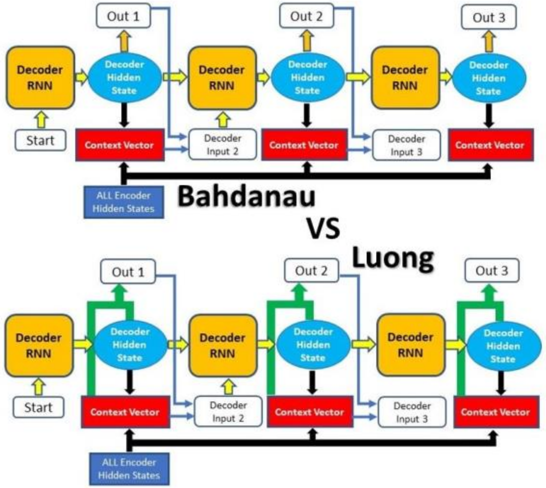
- Reminder: $Attention(q,k,v) = \frac{q^T k}{\sqrt{d_k}} v$
- Basic dot-product attention: $q^T k$
- Multiplicative attention: $q^T W k$

  Here $W$ is a weight matrix

  We know this from Multi-Head-Attention
- Additive attention: $W_3^T \tanh(W_1 k + W_2 q)$

  $W_1, W_2$ are weight matrices, $W_3$ is weight vector

# Attention




Query: decoder state $s_t$
Key: all encoder states $h_i$
Value: all encoder states $h_i$

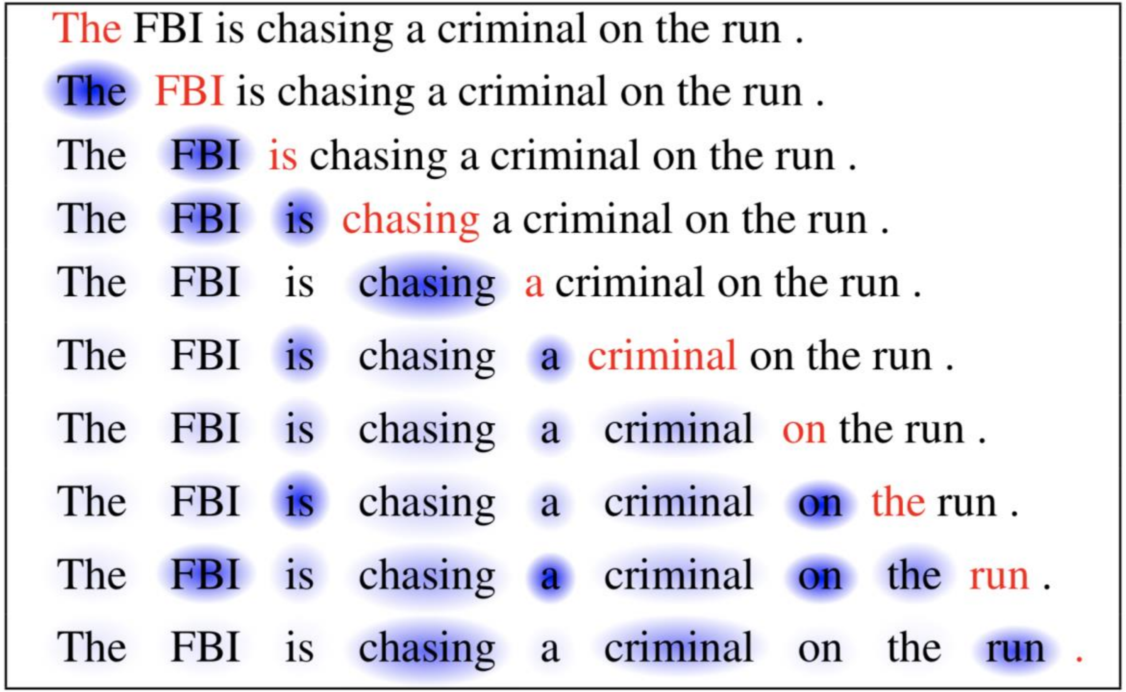
| **Name** | **Alignment Score Function** | **Citation** |
|---|---|---|
| Content-based attention | $score(s_t, h_i) = cosine[s_t, h_i]$ | Graves 2014 |
| Additive/Concat | $score(s_t, h_i) = v_a^T \tanh(W_a[s_t; h_i])$ | Bahdanau 2015 |
| Location-Based | $\alpha_{t,i} = softmax(W_a s_t)$ | Luong 2015 |
| General | $score(s_t, h_i) = s_t^T W_a h_i$ | Luong 2015 |
| Dot-Product | $score(s_t, h_i) = s_t^T h_i$ | Luong 2015 |
| Scaled Dot-Product | $score(s_t, h_i) = \frac{s_t^T h_i}{\sqrt{n}}$ | Vaswani 2017 |

Source: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

# Attention versions

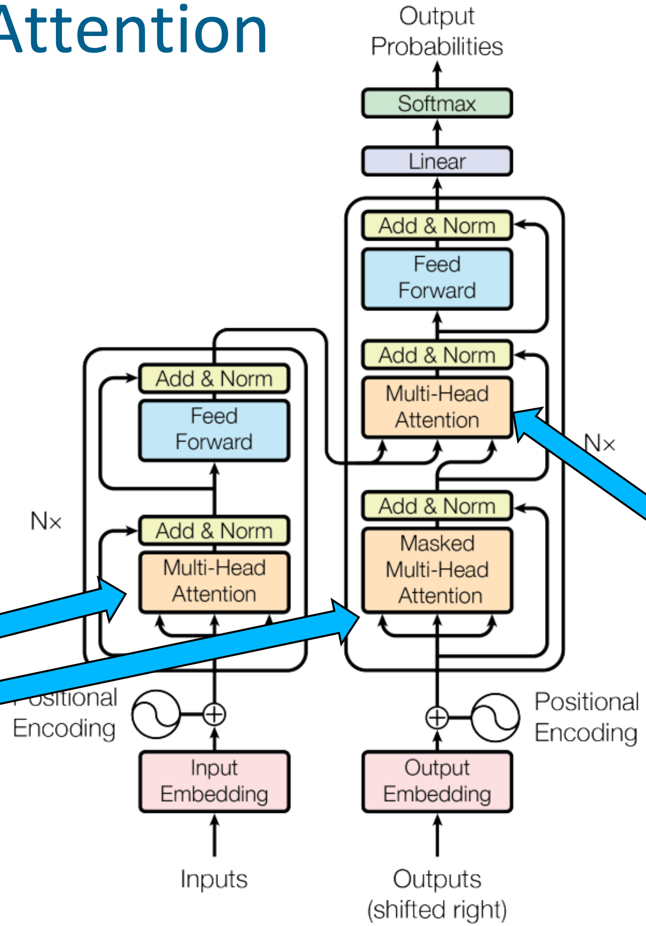
- Self-Attention
  - Relating different positions of the same input sequence. Theoretically the self-attention can adopt any score function, but just replace the target sequence with the same input sequence (as in decoder-only language models)
- Cross Attention
  - Relating a vector to positions from a different input sequence (as in a translation task)
- Global/Soft
  - Attending to the entire input state space
- Local/Hard
  - Attending to a part of input state space; i.e. a patch of the input image

# Self Attention




Source: Cheng et al. 2016

# Self Attention vs Cross Attention




**Self Attention**
key=query=value

**Cross Attention**
Key and value from encoder, Query from decoder

Source: Cheng et al. 2016

# Attention Computation Summary

- Encode each token in the input sentence into vectors
- When decoding, perform a linear combination of these vectors, weighted by „attention weights“
$$a = softmax(alignment\ scores)$$
- Query: decoder state
  Key: all encoder states
  Value: all encoder states

# Attention Computation Summary

- Comes from retrieval systems: when typing a query to search for a video, e.g. on Youtube, the search engine maps the query against a set of keys (video title, descriptions etc.) associated with candidate videos in the database, then present you the best matched videos (values)

# Attention: Image Caption Generation

*Figure 2.* Attention over time. As the model generates each word, its attention changes to reflect the relevant parts of the image. “soft” (top row) vs “hard” (bottom row) attention. (Note that both models generated the same captions in this example.)

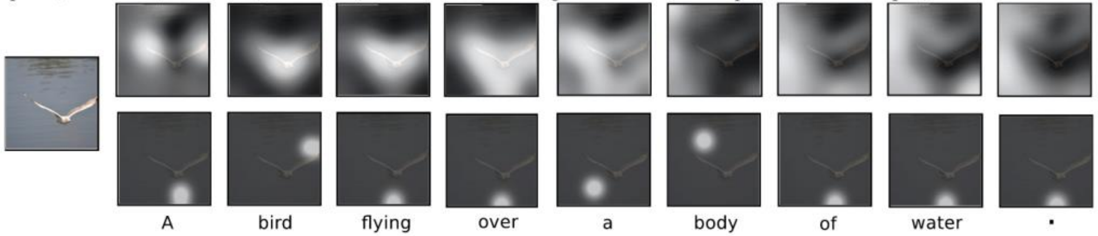
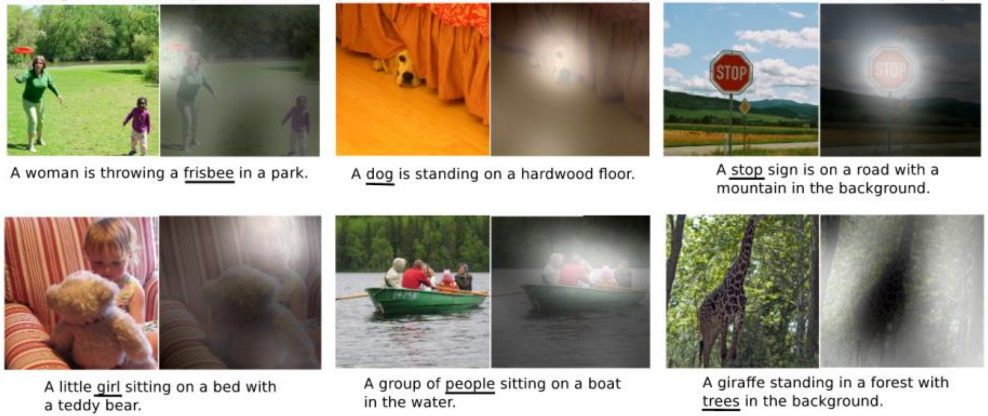


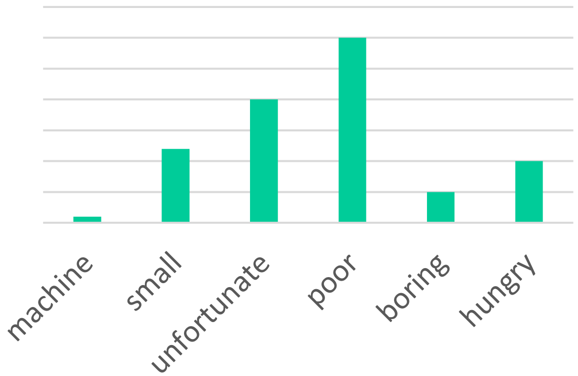

*Figure 3.* Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)




Source: Xu et al. (2016) Show, Attend and Tell: Neural Image Caption Generation with Visual Attention

# Greedy decoding

Inputs




argmax

poor

# Greedy decoding

Greedy decoding has no way to undo decisions

les pauvres sont démunis (the poor don’t have any money)

-> the _

-> the poor _

-> the poor are _

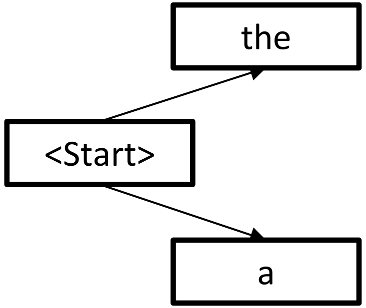
# Beam search

Beam search: Keep track of the $k$ most likely partial translations

$k$ is the beam size

# Beam search

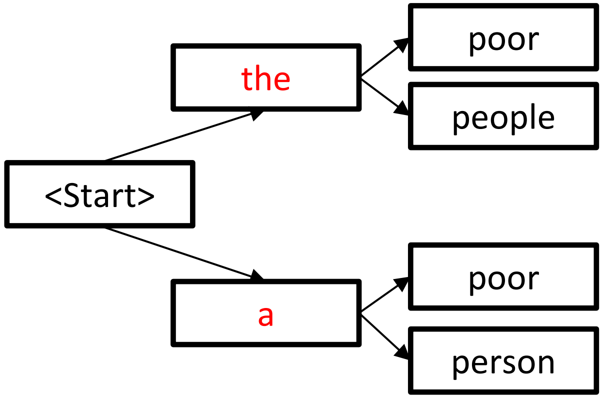
Beam size: 2




Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

# Beam search

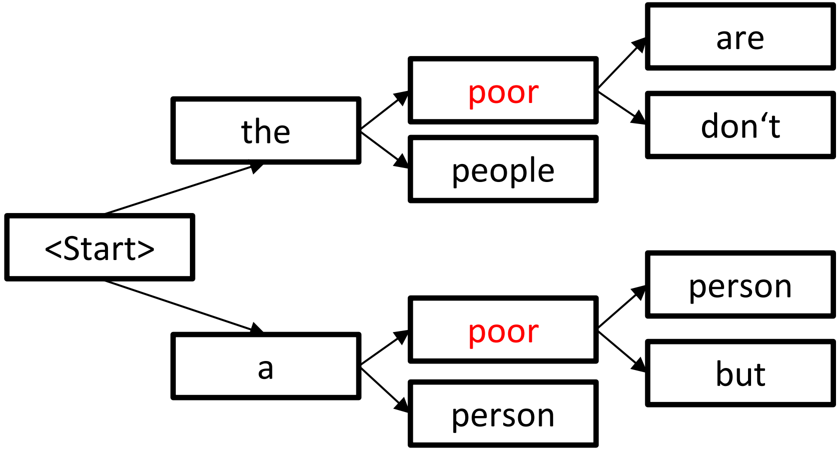
Beam size: 2




Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

# Beam search

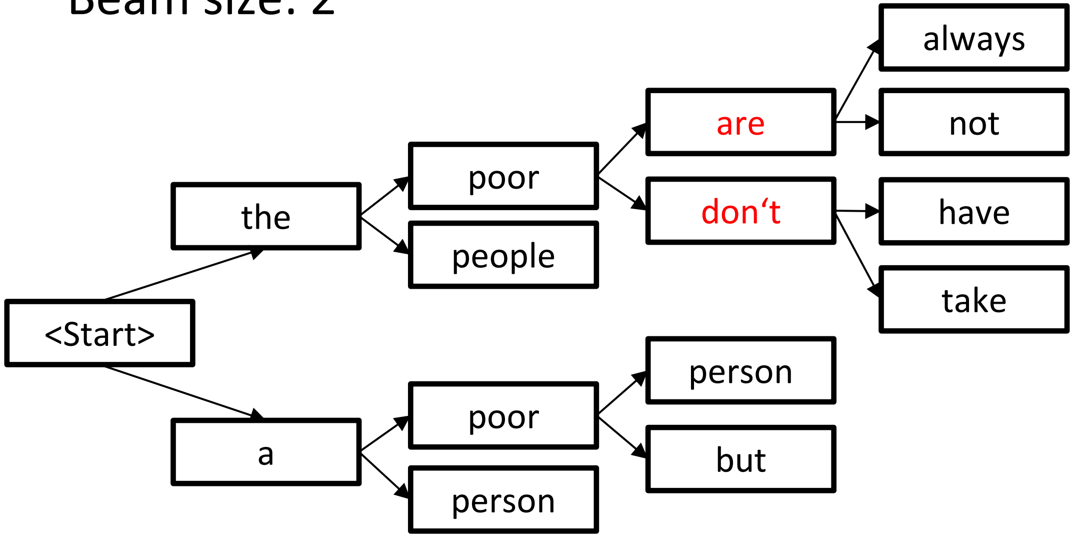
Beam size: 2




Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

# Beam search

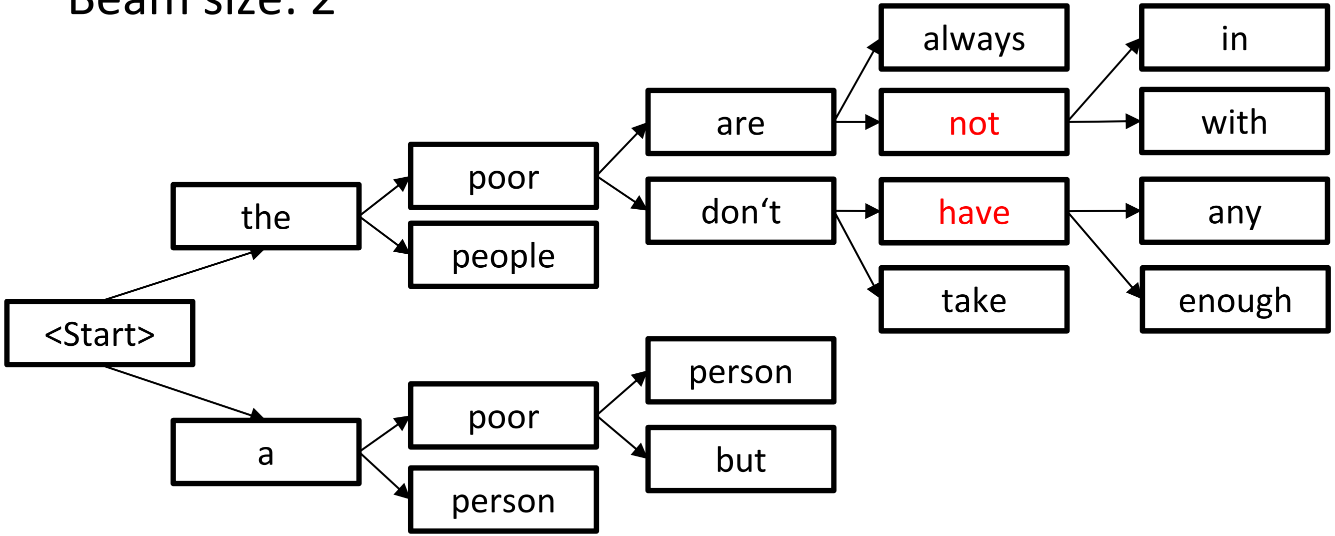
Beam size: 2




Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

# Beam search

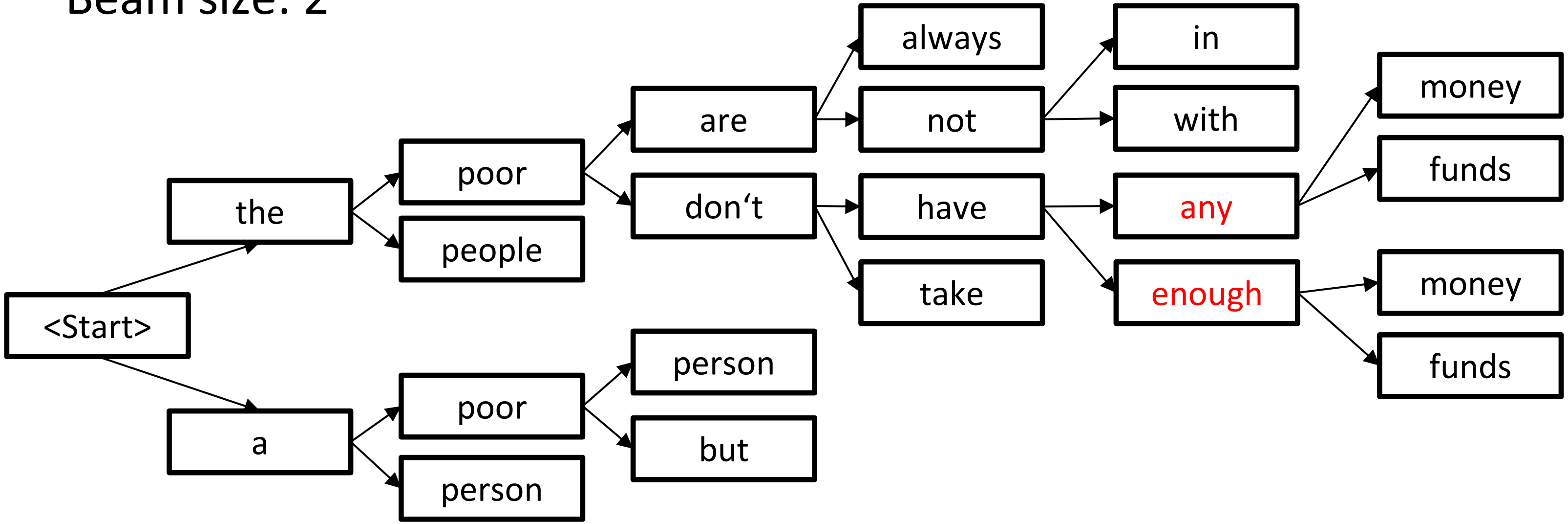
Beam size: 2




Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

# Beam search

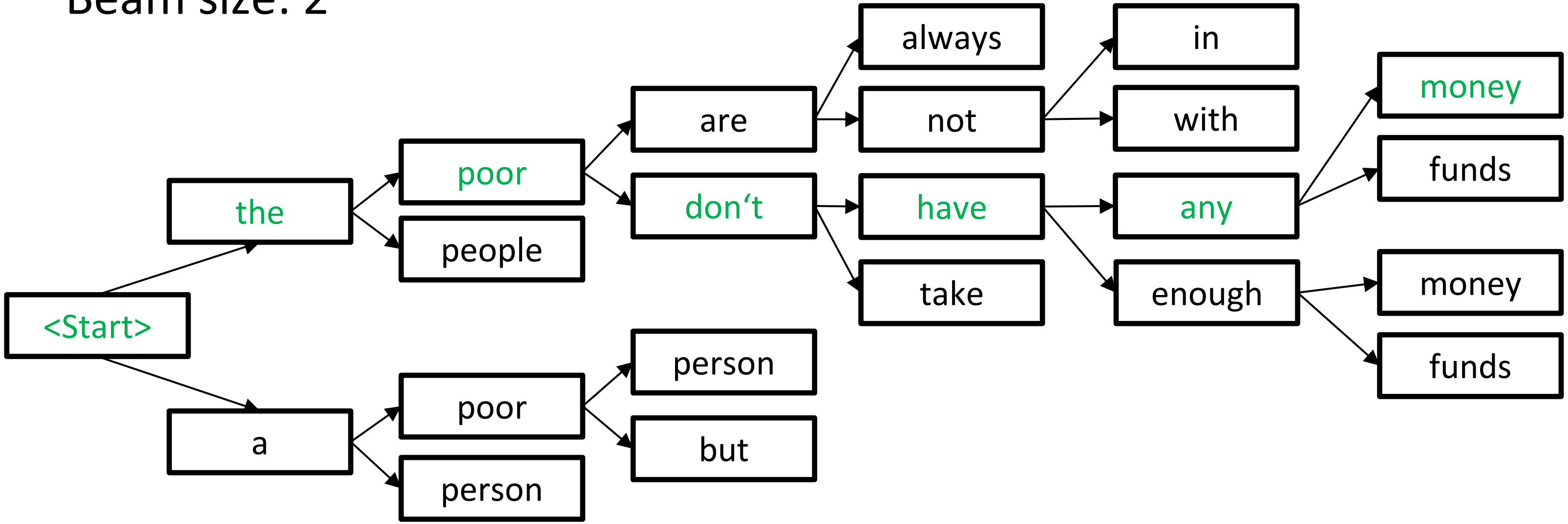
Beam size: 2




Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

# Beam search

Beam size: 2




Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

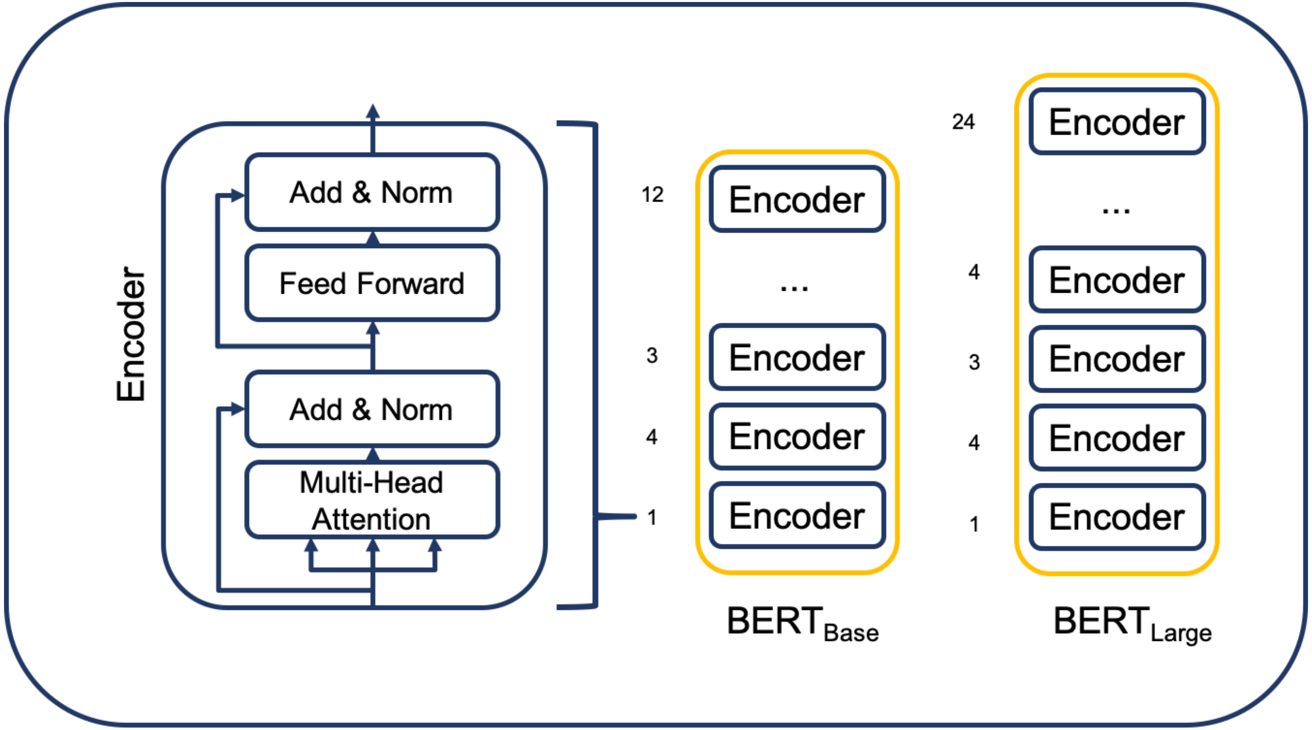
# Word embeddings with Context

- Problem: Word embeddings vectorize words independently of context
- Encoder outputs vector for each word after attending to whole sentence
- Idea: Stack Encoders to get word representation with context

# BERT

- BERT (Bidirectional Encoder Representations from Transformers) is a language representation model (Devlin et al. 2019)
- Transforms words into vectors while retaining context
- Model consists of stacked Transformer Encoder blocks

# BERT

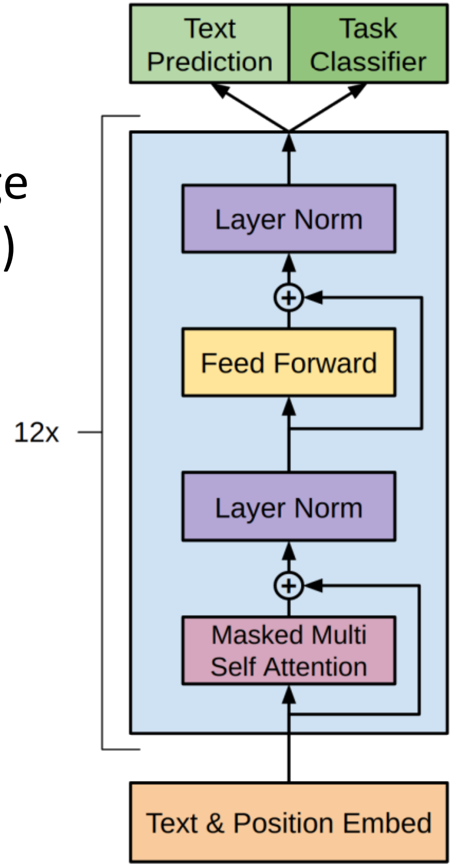



Bidirectional: Attention takes in account previous and following words

Source: https://humboldt-wi.github.io/blog/research/information_systems_1920/bert_blog_post/

# GPT

- GPT (Generative Pre-Trained Transformer) is a language model to produce human-like text (Radford et al. 2018)
- Model consists of stacked Transformer Decoder blocks
- Similarly to BERT we pre-train and then fine-tune




Source: Radford, Alec, et al. "Improving language understanding by generative pre-training.„ (2018).

# PaLM

- PaLM (Pathways Language Model) is a language model and consists of stacked Transformer Decoder blocks like GPT (Chowdhery et al. 2022)
- Modifications like a different activation function or reduction of dimensionality in Key and Value vectors were done to train more efficiently

# PaLM

- The optimizations allowed to train a very large model of 540B parameters with a huge dataset (780 billion tokens)

| Data source | Proportion of data |
|---|---|
| Social media conversations (multilingual) | 50% |
| Filtered webpages (multilingual) | 27% |
| Books (English) | 13% |
| GitHub (code) | 5% |
| Wikipedia (multilingual) | 4% |
| News (English) | 1% |

# PaLM

**Input:** I was supposed to start writing the paper at 5:00 PM. But then I started playing with this cool new language model for 10 minutes. 10 minutes later, it's suddenly 9:30 PM!

**Model Output:** This joke is about how time flies when you're having fun. The person was supposed to start writing a paper at 5:00 PM, but they started playing with a language model instead. They thought they were only playing with it for 10 minutes, but it was actually 4.5 hours.

Source: Chowdhery, Aakanksha, et al. "Palm: Scaling language modeling with pathways." (2022).

# PaLM

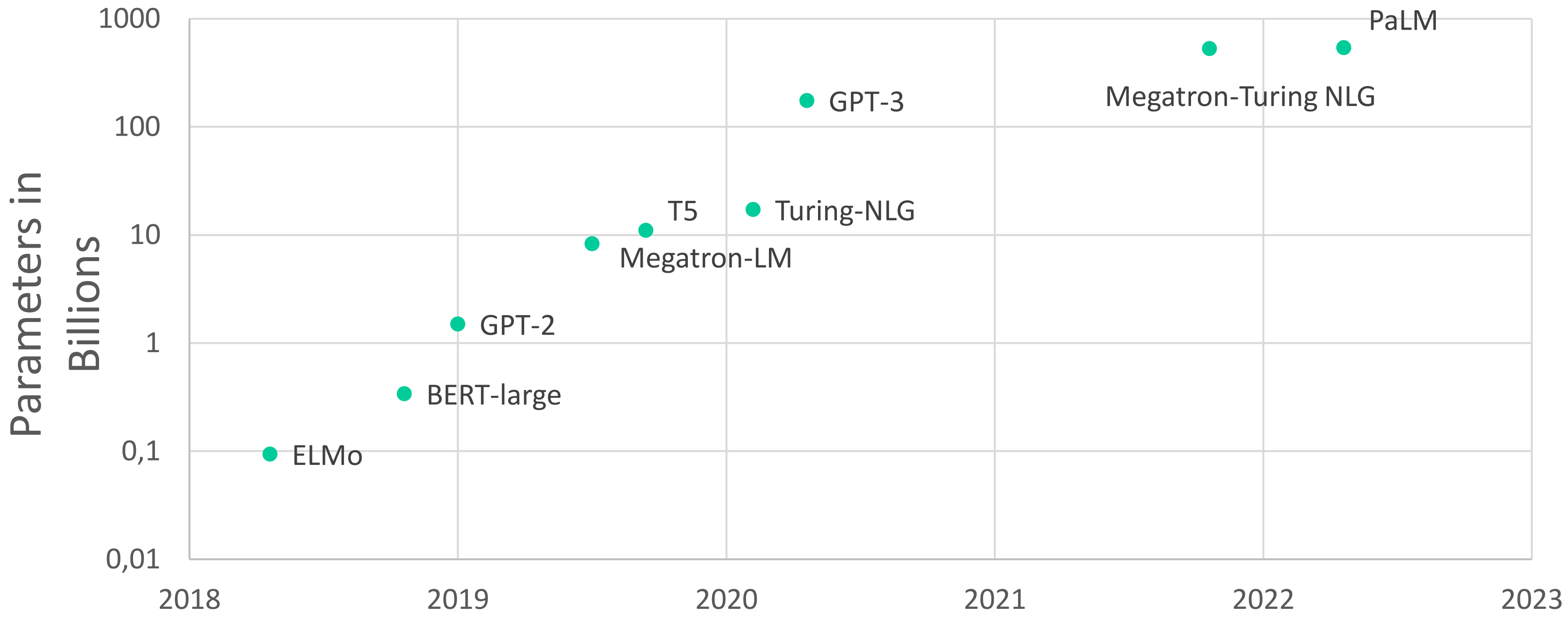
RPTU

**Input:** I was going to fly to visit my family on April 6th. My mom said "Oh great, your step dad's poetry reading is that night!" So now I'm flying in on April 7th.

**Model Output:** The joke is that the speaker's mother is trying to get them to go to their step dad's poetry reading, but the speaker doesn't want to go, so they are changing their flight to the day after the poetry reading.

Source: Chowdhery, Aakanksha, et al. "Palm: Scaling language modeling with pathways." (2022).

# Language Models



# Summary

- Encoder-decoder models
- Attention can model relations between all words
- Transformers are building blocks of larger language models
- BERT: Bidirectional/Contextual embeddings
- GPT: Unidirectional embeddings ideal for generation

# References


- Vaswani, Ashish, et al. "Attention is all you need." (2017).
- Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." (2018).
- Radford, Alec, et al. "Improving language understanding by generative pre-training." (2018).
- Chowdhery, Aakanksha, et al. "Palm: Scaling language modeling with pathways." (2022).

# Acknowledgements

- Feibai Huang