Lecture 14. Self-attention Mechanism and Transformers

Bao Wang
Department of Mathematics
Scientific Computing and Imaging Institute
University of Utah

Math 5750/6880, Fall 2023

Embeddings and Language Models

Converting words to vectors

- In natural language processing (NLP), our inputs are sequences of words, but deep learning needs vectors.

- How to convert words to vectors?

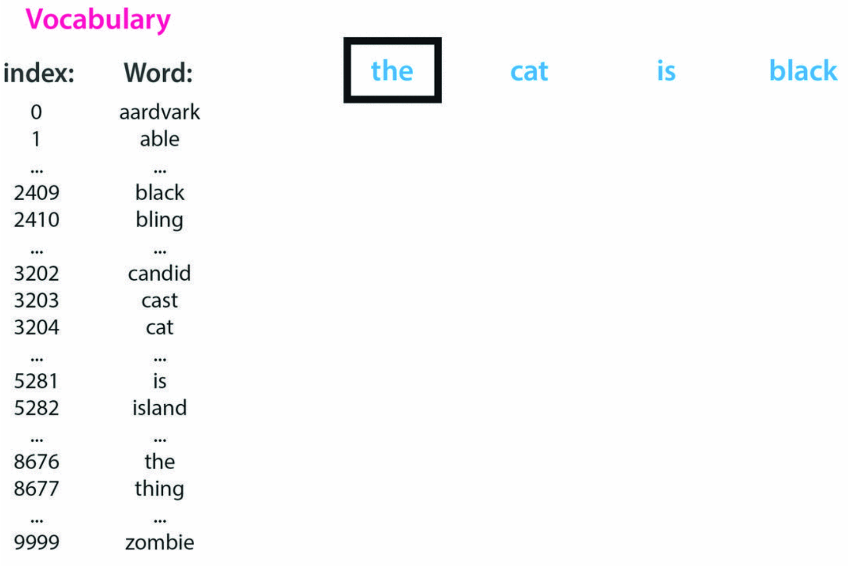
Converting words to vectors

- In natural language processing (NLP), our inputs are sequences of words, but deep learning needs vectors.

- How to convert words to vectors?

- Simplest idea: one-hot encoding.

One-hot encoding



Vocabulary

index:	Word:
0	aardvark
1	able
...	...
2409	black
2410	bling
...	...
3202	candid
3203	cast
3204	cat
...	...
5281	is
5282	island
...	...
8676	the
8677	thing
...	...
9999	zombie

$\boxed{\text{the}}$ cat is black

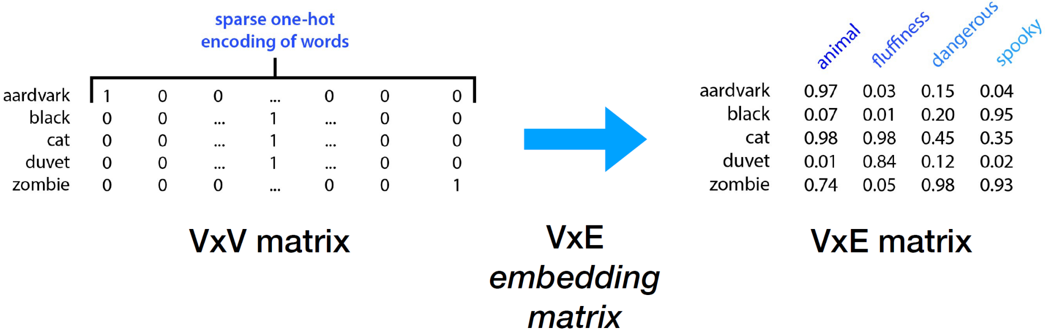
Problems with one-hot encoding

- Scales poorly with vocabulary size.

- Very high-dimensional sparse vectors: neural network operations work poorly.

- Violates what we know about word similarity (e.g. “run” is as far away from “running” as from “poetry”).

Map one-hot to dense vectors



sparse one-hot encoding of words

aardvark	1	0	0	...	0	0	0
black	0	0	...	1	...	0	0
cat	0	0	...	1	...	0	0
duvet	0	0	...	1	...	0	0
zombie	0	0	0	...	0	0	1

VxV matrix

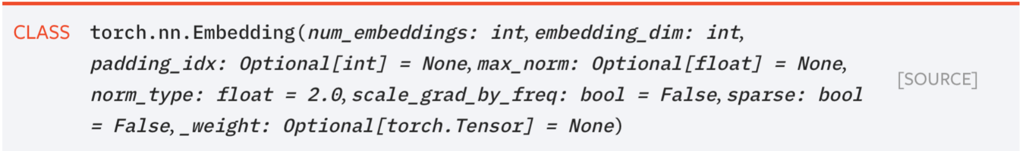
VxE
embedding matrix

	animal	fluffiness	dangerous	spooky
aardvark	0.97	0.03	0.15	0.04
black	0.07	0.01	0.20	0.95
cat	0.98	0.98	0.45	0.35
duvet	0.01	0.84	0.12	0.02
zombie	0.74	0.05	0.98	0.93

VxE matrix

Problem: how do we find the values of the embedding matrix?

Solution: Learn as part of the task

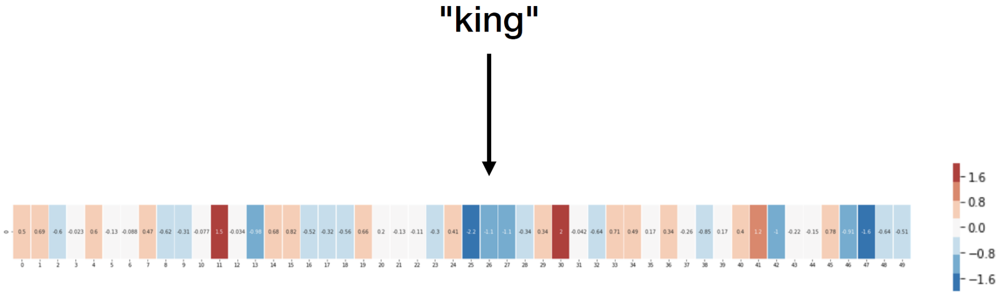


```
CLASS torch.nn.Embedding(num_embeddings: int, embedding_dim: int,
      padding_idx: Optional[int] = None, max_norm: Optional[float] = None,
      norm_type: float = 2.0, scale_grad_by_freq: bool = False, sparse: bool
      = False, _weight: Optional[torch.Tensor] = None)                      [SOURCE]
```

- A simple lookup table that stores embeddings of a fixed dictionary and size.

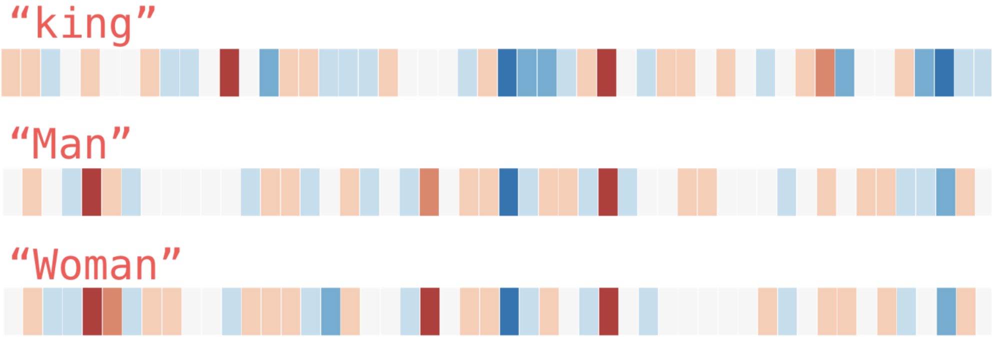
- This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.

Word2Vec – 1



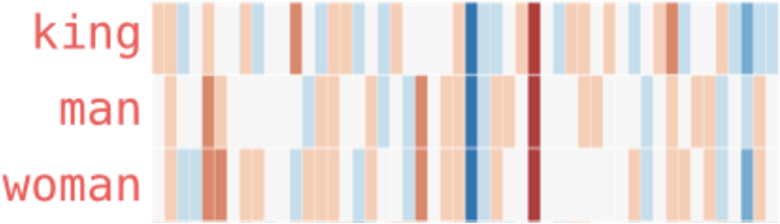
"king"

Word2Vec – 2



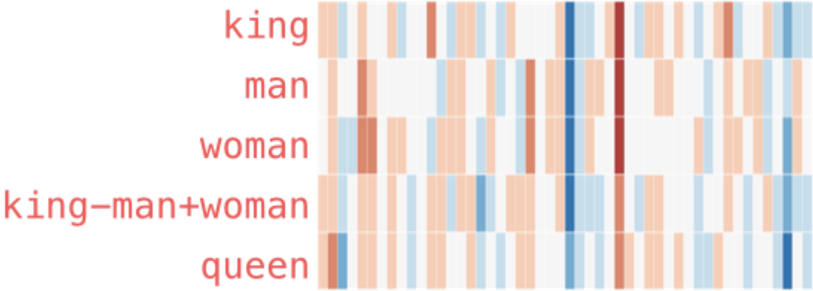
Word2Vec – 3

king − man + woman



Word2Vec – 4

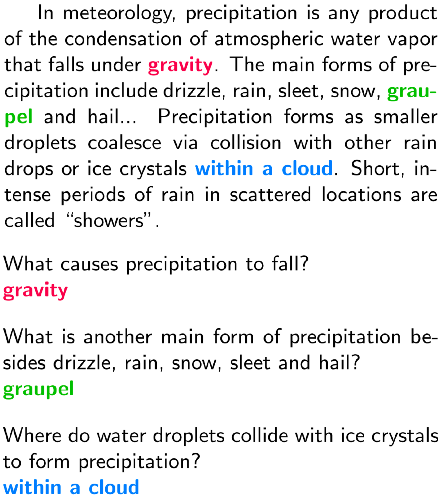
king – man + woman ~= queen



Some NLP Applications

Q&A: SQuAD

- 100K question-answer pairs
- Answers are always spans in the question



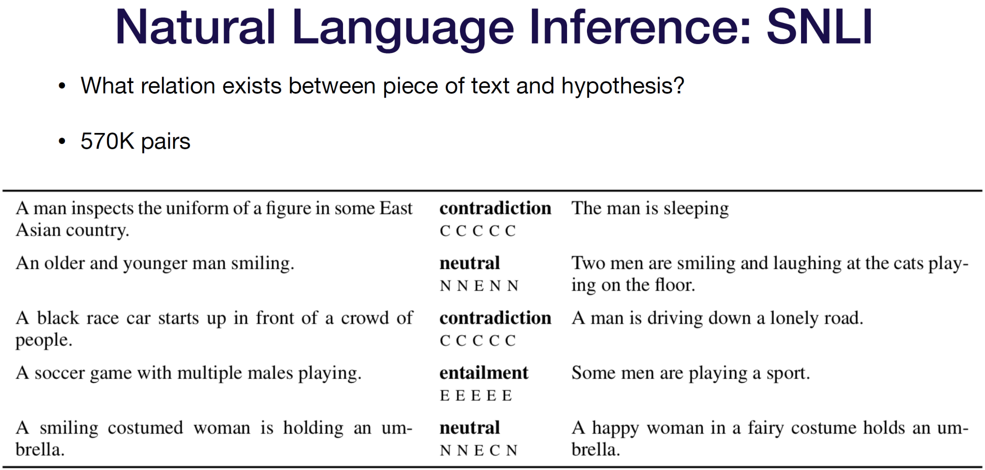
In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called "showers".

What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

Natural Language Inference: SNLI

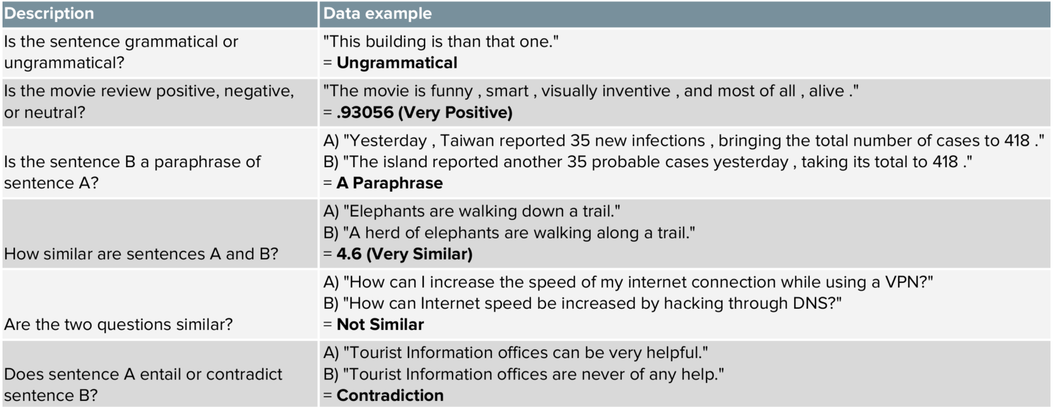


Natural Language Inference: SNLI

- What relation exists between piece of text and hypothesis?
- 570K pairs

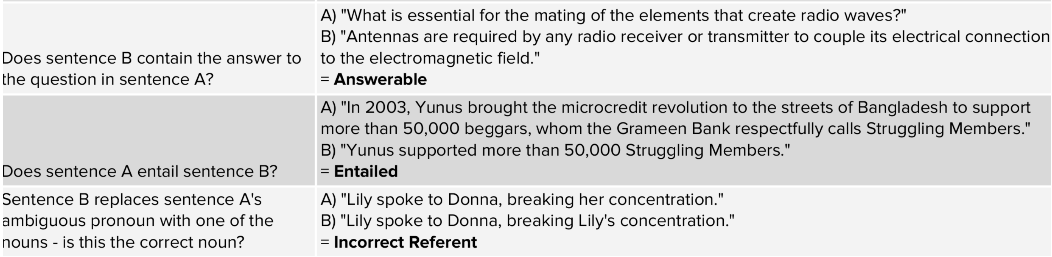
A man inspects the uniform of a figure in some East Asian country.	**contradiction** C C C C C	The man is sleeping
An older and younger man smiling.	**neutral** N N E N N	Two men are smiling and laughing at the cats playing on the floor.
A black race car starts up in front of a crowd of people.	**contradiction** C C C C C	A man is driving down a lonely road.
A soccer game with multiple males playing.	**entailment** E E E E E	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	**neutral** N N E C N	A happy woman in a fairy costume holds an umbrella.

GLUE – 1



Description	Data example
Is the sentence grammatical or ungrammatical?	"This building is than that one." = **Ungrammatical**
Is the movie review positive, negative, or neutral?	"The movie is funny , smart , visually inventive , and most of all , alive ." = **.93056 (Very Positive)**
Is the sentence B a paraphrase of sentence A?	A) "Yesterday , Taiwan reported 35 new infections , bringing the total number of cases to 418 ." B) "The island reported another 35 probable cases yesterday , taking its total to 418 ." = **A Paraphrase**
How similar are sentences A and B?	A) "Elephants are walking down a trail." B) "A herd of elephants are walking along a trail." = **4.6 (Very Similar)**
Are the two questions similar?	A) "How can I increase the speed of my internet connection while using a VPN?" B) "How can Internet speed be increased by hacking through DNS?" = **Not Similar**
Does sentence A entail or contradict sentence B?	A) "Tourist Information offices can be very helpful." B) "Tourist Information offices are never of any help." = **Contradiction**

GLUE – 2



Does sentence B contain the answer to the question in sentence A?	A) "What is essential for the mating of the elements that create radio waves?" B) "Antennas are required by any radio receiver or transmitter to couple its electrical connection to the electromagnetic field." = **Answerable**
Does sentence A entail sentence B?	A) "In 2003, Yunus brought the microcredit revolution to the streets of Bangladesh to support more than 50,000 beggars, whom the Grameen Bank respectfully calls Struggling Members." B) "Yunus supported more than 50,000 Struggling Members." = **Entailed**
Sentence B replaces sentence A's ambiguous pronoun with one of the nouns - is this the correct noun?	A) "Lily spoke to Donna, breaking her concentration." B) "Lily spoke to Donna, breaking Lily's concentration." = **Incorrect Referent**

- 9 tasks, model score is averaged across them.

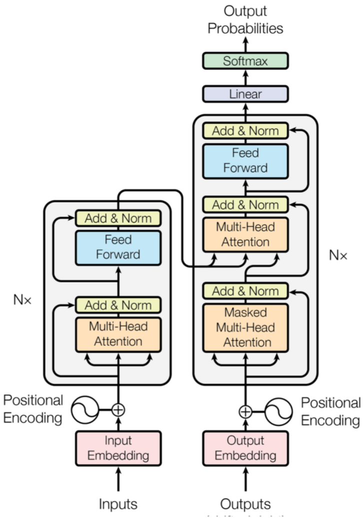
Large Language Models

- ChatGPT
- Bard

. . .

Transformers

Attention is all you need

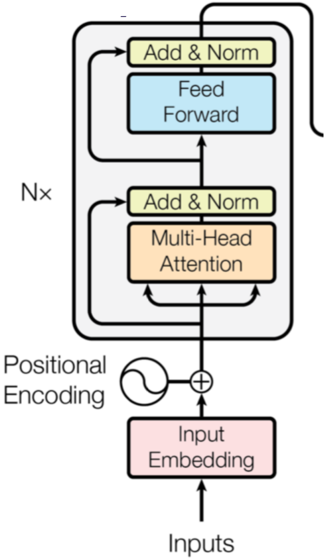


- Encoder-decoder with only attention and fully connected layers (no recurrent or convolutions).

- Set new SOTA on translation datasets and many more.

Vaswani et al., Attention is all you need, NeurIPS 2017.

Attention is all you need



- For simplicity, we can focus just on the encoder. For instance, BERT is just the encoder.
- The components:
  - (Masked) Self-attention
  - Positional encoding
  - Layer normalization

Basic self-attention

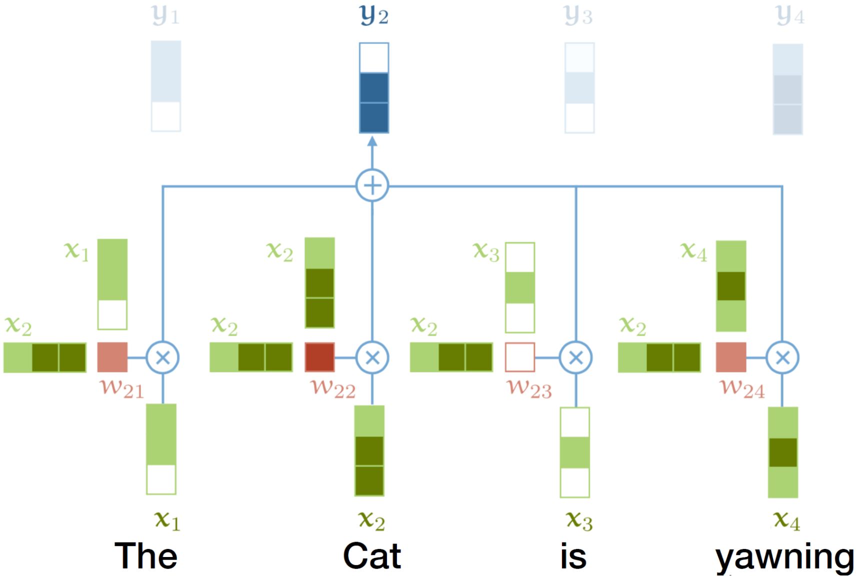
- **Input:** sequence of tensors $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_t$.
- **Output:** sequence of tensors, each one a weighted sum of the input sequence:

$$\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_t, \quad \text{where} \quad \boldsymbol{y}_i = \sum_j w_{ij} \boldsymbol{x}_j.$$

  - w_{ij} is a function of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$

$$w_{ij} = \frac{\exp(w'_{ij})}{\sum_j \exp(w'_{ij})}, \quad \text{where} \quad w'_{ij} = \boldsymbol{x}_i^\top \boldsymbol{x}_j.$$

Basic self-attention

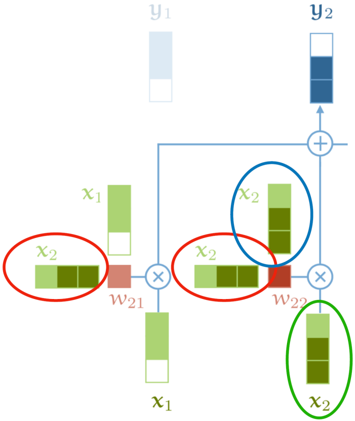


Basic self-attention – Problems

- **No learned weights** → Let us learn some weights!

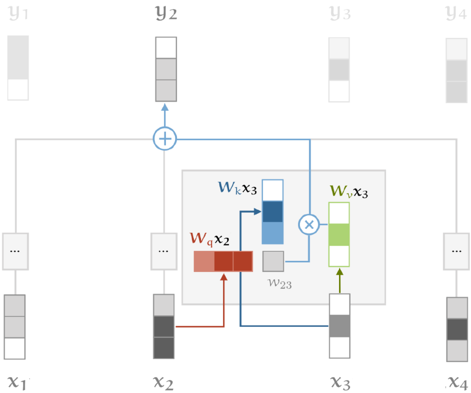
- Order of the sequence does not affect the result of computations. (Permutation invariance)

Query, key, value



- Every input vector $\boldsymbol{x}_i$ is used in 3 ways:
  - Compared to every other vector to compute attention weights for its own output y_i (query).
  - Compared to every other vector to compute attention weight w_{ij} for output y_j (key).
  - Summed with other vectors to form the result of the attention-weighted sum (value).

Query, key, value



- We can process each input vector to fulfill the three roles with matrix multiplication.

- Learning the matrices, i.e., learning attention Each vector is mapped to three vectors: query, key, and value.

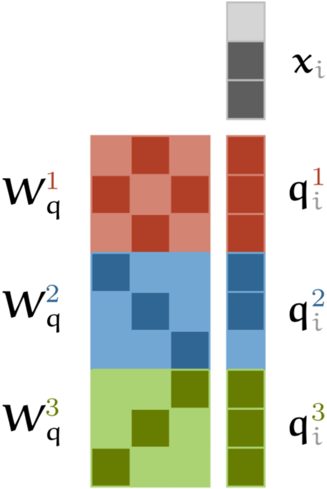
$$\boldsymbol{q}_i = \boldsymbol{W}_q \boldsymbol{x}_i, \quad \boldsymbol{k}_i = \boldsymbol{W}_k \boldsymbol{x}_i, \quad \boldsymbol{v}_i = \boldsymbol{W}_v \boldsymbol{x}_i$$

$$w'_{ij} = \boldsymbol{q}_i^\top \boldsymbol{k}_j,$$

$$w_{ij} = \mathrm{softmax}(w'_{ij})$$

$$\boldsymbol{y}_i = \sum_j w_{ij} \boldsymbol{v}_j.$$

Multi-head attention

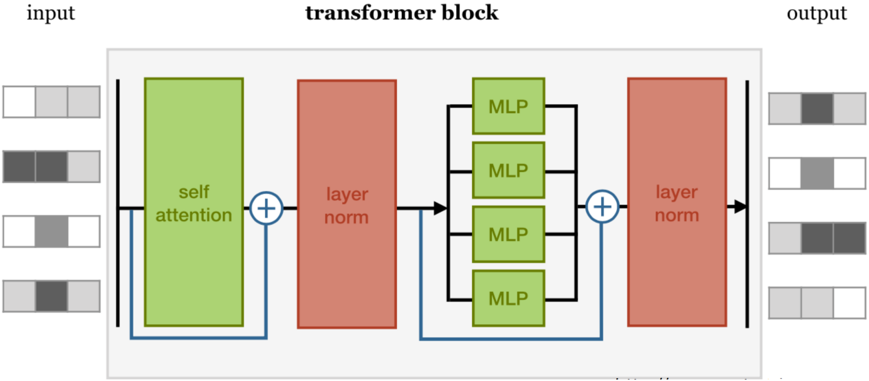


- Multiple "heads" of attention just means learning different sets of $\boldsymbol{W}_q$, $\boldsymbol{W}_k$, and $\boldsymbol{W}_v$ matrices simultaneously.

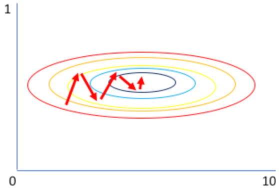
- Implemented as just a single matrix anyway ...

Transformer

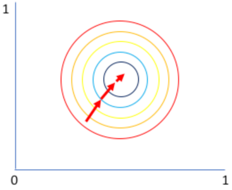
- Self-attention layer → Layer normalization → Dense layer



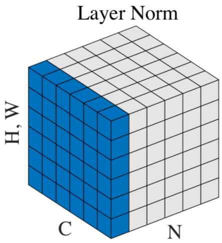
Layer normalization



Gradient of larger parameter dominates the update



Both parameters can be updated in equal proportions



- Neural net layers work best when input vectors have uniform mean and std in each dimension.

- As inputs flow through the network, means and std's get blown out.

- Layer normalization is a hack to reset things to when we want them in between layers.

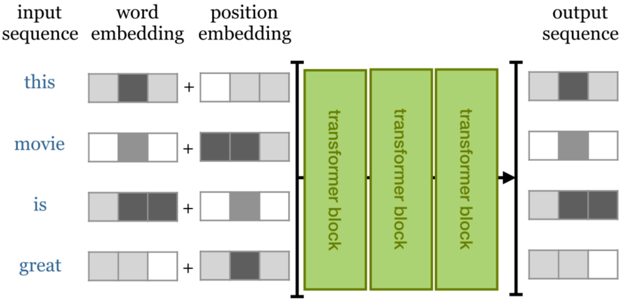
Transformer

So far:

- Learned query, key, value weights.
- Multiple heads.
- **Order of the sequence does not affect result of computations:** Let us encode each vector with position.

Transformer

- Position embedding: just what it sounds!

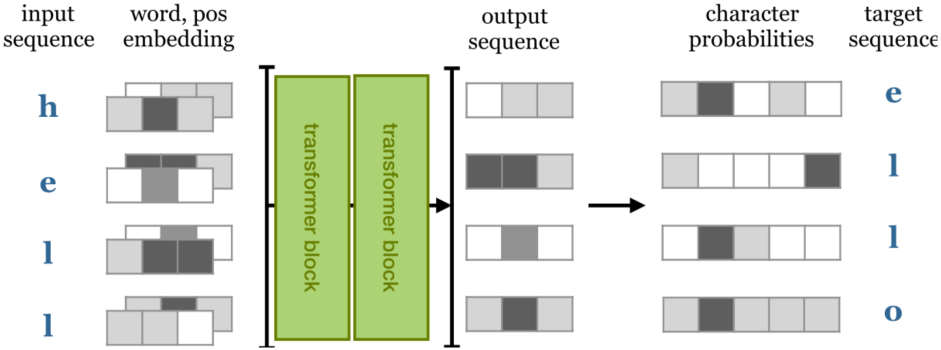


For instance, let d be the dimension of the word embedding, then one particular position embedding scheme is

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right); \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

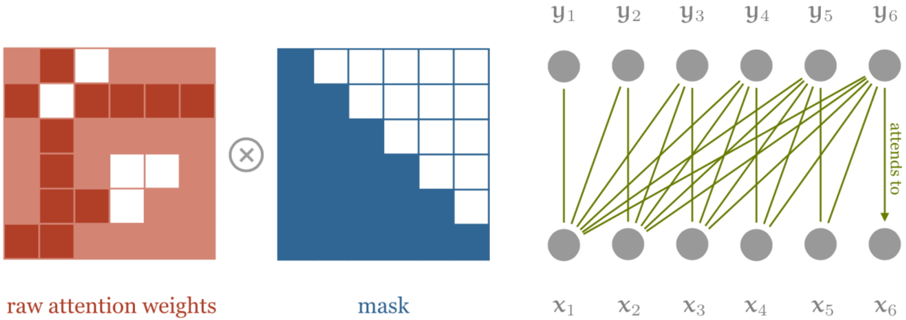
Transformer: last trick

- Since the Transformer sees all inputs at once, to predict next vector in sequence (e.g. generate text), we need to mask the future.



Transformer: last trick

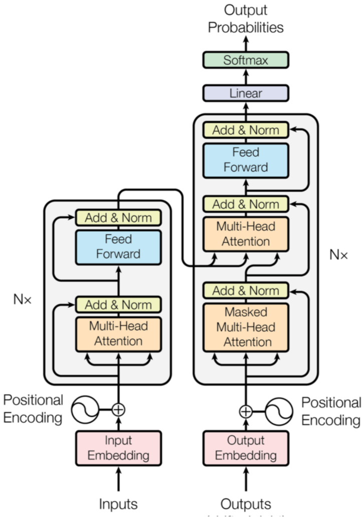
- Since the Transformer sees all inputs at once, to predict next vector in sequence (e.g. generate text), we need to mask the future.



$\boldsymbol{y}_i$ cannot see $\boldsymbol{x}_j$ for $j > i$.

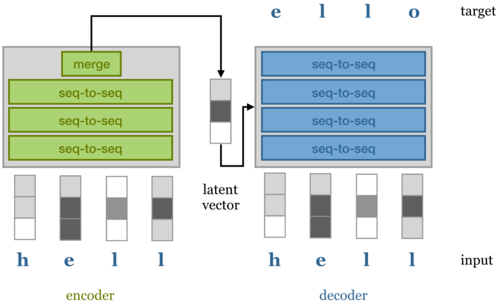
Transformers-based AI Models

Transformers: Recap



- Encoder-decoder for translation.

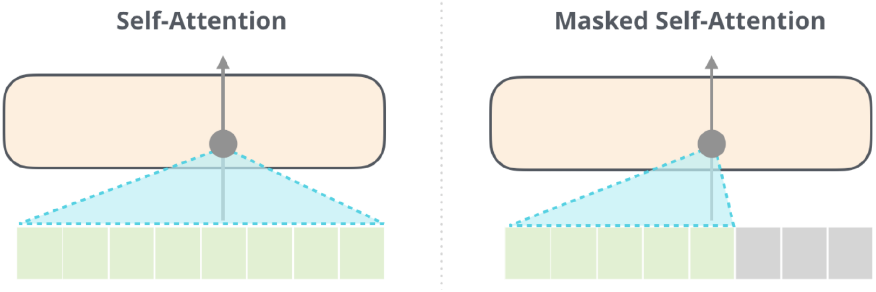
Transformers: Recap



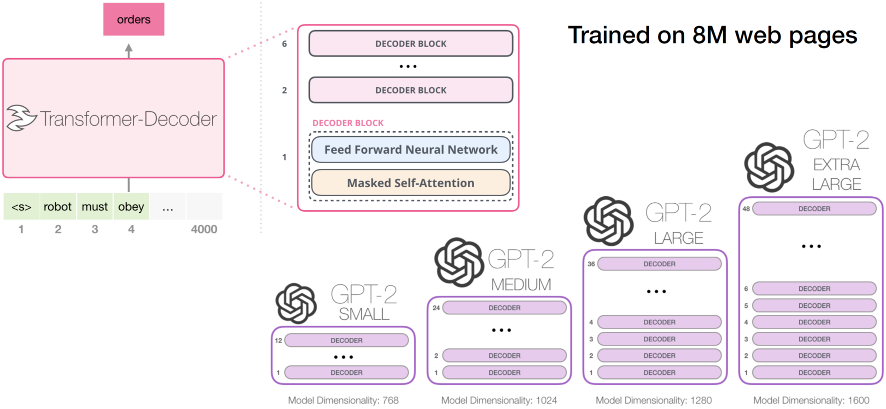
- Encoder-decoder for translation.
- Later models made it mostly just the encoder or just the decoder.
- ... but then the latest models are back to encoder-decoder.

GPT/GPT-2

- Generative Pre-trained Transformer
- GPT learns to predict the next word in the sequence
- Since it conditions only on preceding words, it uses masked self-attention



GPT/GPT-2



Trained on 8M web pages

117M params

345M params

762M params

1.5B params

BERT

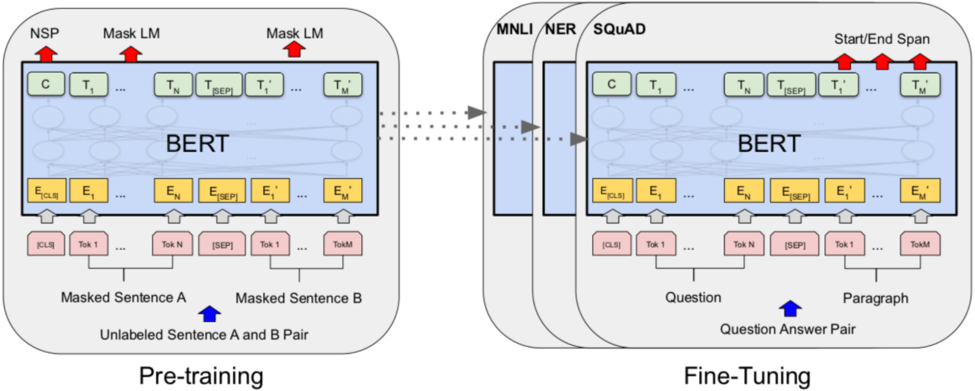
- Bidirectional Encoder Representations from Transformers

- Encoder blocks only (no masking)

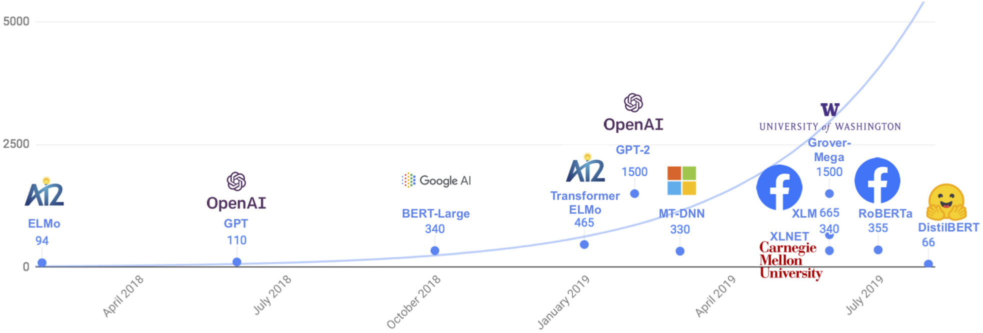
- BERT involves pre-training on a lot of text with 15% of all words masked out. Also, sometimes predicting whether one sentence follows another.

- 340M parameters: 24 transformer blocks, embedding dim of 1024, 16 attention heads.

BERT


NSP
Mask LM
Mask LM
C
T1
...
TN
T[SEP]
T1'
...
TM'
BERT
E[CLS]
E1
...
EN
E[SEP]
E1'
...
EM'
[CLS]
Tok 1
...
Tok N
[SEP]
Tok 1
...
TokM
Masked Sentence A
Masked Sentence B
Unlabeled Sentence A and B Pair
Pre-training
MNLI
NER
SQuAD
Start/End Span
C
T1
...
TN
T[SEP]
T1'
...
TM'
BERT
E[CLS]
E1
...
EN
E[SEP]
E1'
...
EM'
[CLS]
Tok 1
...
Tok N
[SEP]
Tok 1
...
TokM
Question
Paragraph
Question Answer Pair
Fine-Tuning

More transformer-based AI models



Linearized Self-Attention

Katharopoulos et al. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention, ICML 2020.

Self-attention mechanism: Recap

Self-attention mechanism: Transforms sequences $\boldsymbol{X} := [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N \times D_x}$ as follows:

Step 1. Project $\boldsymbol{X}$ into $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q^\top;\ \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K^\top;\ \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V^\top,$$

where $\boldsymbol{W}_Q, \boldsymbol{W}_K \in \mathbb{R}^{D \times D_x}$, and $\boldsymbol{W}_V \in \mathbb{R}^{D_v \times D_x}$ are learnable. $\boldsymbol{Q} := [\boldsymbol{q}_1, \cdots, \boldsymbol{q}_N]^\top$, similar for $\boldsymbol{K}$ and $\boldsymbol{V}$.

Step 2. Output sequence $\hat{\boldsymbol{V}} := [\hat{\boldsymbol{v}}_1, \cdots, \hat{\boldsymbol{v}}_N]$, where

$$\hat{\boldsymbol{v}}_i = \sum_{j=1}^{N} \mathrm{softmax}\Big(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{D}}\Big)\boldsymbol{v}_j.$$

Self-attention mechanism: Bottlenecks

$$\hat{\boldsymbol{v}}_i = \sum_{j=1}^{N} \mathrm{softmax}\left(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{D}}\right)\boldsymbol{v}_j, \iff \hat{\boldsymbol{V}} = \mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{D}}\right)\boldsymbol{V} := \boldsymbol{A}\boldsymbol{V}.$$

Bottleneck: Store $\boldsymbol{A}$ takes $\mathcal{O}(N^2)$ memory and compute $\boldsymbol{A}\boldsymbol{V}$ costs $\mathcal{O}(N^2)$ time.

Relaxation

Note that

$$\hat{\boldsymbol{v}}_i = \sum_{j=1}^{N} \mathrm{softmax}\left(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{D}}\right) \boldsymbol{v}_j,$$

can be written as

$$\hat{\boldsymbol{v}}_i = \frac{\sum_{j=1}^{N} \mathrm{sim}(\boldsymbol{q}_i, \boldsymbol{k}_j) \boldsymbol{v}_j}{\sum_{j=1}^{N} \mathrm{sim}(\boldsymbol{q}_i, \boldsymbol{k}_j)},$$

where $\mathrm{sim}(\boldsymbol{q}_i, \boldsymbol{k}_j) = \exp\left(\frac{\boldsymbol{q}^\top \boldsymbol{k}}{\sqrt{D}}\right)$.

Relaxation

Note that

$$\hat{\boldsymbol{v}}_i = \sum_{j=1}^{N} \mathrm{softmax}\left(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{D}}\right) \boldsymbol{v}_j,$$

can be written as

$$\hat{\boldsymbol{v}}_i = \frac{\sum_{j=1}^{N} \mathrm{sim}(\boldsymbol{q}_i, \boldsymbol{k}_j) \boldsymbol{v}_j}{\sum_{j=1}^{N} \mathrm{sim}(\boldsymbol{q}_i, \boldsymbol{k}_j)},$$

where $\mathrm{sim}(\boldsymbol{q}_i, \boldsymbol{k}_j) = \exp\left(\frac{\boldsymbol{q}^\top \boldsymbol{k}}{\sqrt{D}}\right)$.

Key idea: Replace $\mathrm{sim}(\boldsymbol{q}_i, \boldsymbol{k}_j)$ with a kernel $k(\boldsymbol{q}_i, \boldsymbol{k}_j)$ that can be represented as the inner product of a feature map on $\boldsymbol{q}_i$ and $\boldsymbol{k}_j$, i.e., $k(\boldsymbol{q}_i, \boldsymbol{k}_j) = \phi(\boldsymbol{q}_i)^\top \phi(\boldsymbol{k}_j)$.

Linearized self-attention

$$\hat{\boldsymbol{v}}_i = \frac{\sum_{j=1}^{N} \mathrm{sim}(\boldsymbol{q}_i, \boldsymbol{k}_j)\boldsymbol{v}_j}{\sum_{j=1}^{N} \mathrm{sim}(\boldsymbol{q}_i, \boldsymbol{k}_j)} \approx \frac{\sum_{j=1}^{N} k(\boldsymbol{q}_i, \boldsymbol{k}_j)\boldsymbol{v}_j}{\sum_{j=1}^{N} k(\boldsymbol{q}_i, \boldsymbol{k}_j)} = \frac{\phi(\boldsymbol{q}_i)^\top \sum_{j=1}^{N} \phi(\boldsymbol{k}_j)\boldsymbol{v}_j^\top}{\phi(\boldsymbol{q}_i)^\top \sum_{j=1}^{N} \phi(\boldsymbol{k}_j)}.$$

Advantages of linearized self-attention

$$\hat{\boldsymbol{V}} = \boldsymbol{A}\boldsymbol{V} \approx (\boldsymbol{a}\boldsymbol{b}^\top)\boldsymbol{V} = \boldsymbol{a}(\boldsymbol{b}^\top\boldsymbol{V}),$$

– $\boldsymbol{a}\boldsymbol{b}^\top$ is the rank-one approximation of the matrix $\boldsymbol{A}$.

Remark. Computing $\boldsymbol{A}\boldsymbol{V}$ requires $\mathcal{O}(N^2)$ complexity in computational time and memory footage. However, compute $\boldsymbol{a}(\boldsymbol{b}^\top\boldsymbol{V})$ only requires $\mathcal{O}(N)$ complexity in computational time and memory footage.

Remark. Rank-one approximation of $\boldsymbol{A}$ is a quite poor approximation.

FMMformer: Efficient and Flexible Transformers with Decomposed Near-field and Far-field Attention

T. Nguyen, V. Suliafu, S. Osher, L. Chen, and B. Wang, FMMformer: Efficient and Flexible Transformers with Decomposed Near-field and Far-field Attention, NeurIPS 2021.

Gravitational force calculation

Physics analogue

$$\hat{\boldsymbol{v}}_i = \sum_{j=1}^{N} \mathrm{softmax}\left(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{D}}\right)\boldsymbol{v}_j, \iff \hat{\boldsymbol{v}}_i = \underbrace{\sum_{j=1}^{N} k(\boldsymbol{q}_i, \boldsymbol{k}_j)\boldsymbol{v}_j.}_{\text{Force calculation?}}$$

Simplest idea: cutoff ⇒ sparse (local) attention. Problematic if $k(\boldsymbol{q}_i, \boldsymbol{k}_j) \gtrsim \frac{1}{\|\boldsymbol{q}_i - \boldsymbol{k}_j\|}$.

Physics analogue: gravitational and electrostatics force calculation. Long-range force where the potential decays at the rate $1/\|\boldsymbol{q}_i - \boldsymbol{k}_j\|$.

Computational math toolbox: particle mesh Ewald (PME) (Ewald, Ann. Phys. 1921.); fast multipole method (FMM) (Greengard and Rokhlin, JCP, 1987.)

PME: calculate near-field interaction in real-space and calculate far-field interaction in the k-space.
FMM: direct calculation of near-field interaction and coarse-grain far-field interaction.

Fast multipole method & its algebraic interpretation

Key idea of FMM: *far-field interaction can be well-approximated by separable low-rank matrices* while the near-field interaction can be calculated directly.

Let $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ encodes interaction among all particles, where $\boldsymbol{A}(i,j) = g(\|\boldsymbol{q}_i - \boldsymbol{k}_j\|)$ with g be smooth except at 0 and $g(st) = g(s)g(t)$. E.g., $g(\|\boldsymbol{q}_i - \boldsymbol{k}_j\|) = 1/\|\boldsymbol{q}_i - \boldsymbol{k}_j\|$.

Def. [Well-separation] Let us partition $\{1, \cdots, N\}$ into two groups $\{T_1, T_2\}$, then T_1 is called well-separated from T_2 if $\exists \boldsymbol{k}^*$ and a number δ s.t.

$$\|\boldsymbol{k}_j - \boldsymbol{k}^*\| \leq \delta \|\boldsymbol{q}_i - \boldsymbol{k}^*\| \quad \forall i \in T_1, j \in T_2,$$

e.g., $\boldsymbol{k}^*$ is the center of vectors in T_2.

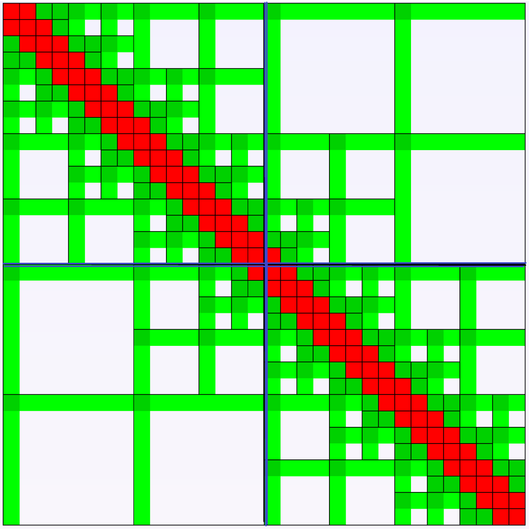
In this case, $|\boldsymbol{q}_i - \boldsymbol{k}_j| \approx |\boldsymbol{q}_i - \boldsymbol{k}^*|$ for any $j \in T_2$. The i^{th} row of $\boldsymbol{A}$ will be a constant after excluding a banded matrix $\boldsymbol{D}$ from $\boldsymbol{A}$, i.e., $\boldsymbol{A} - \boldsymbol{D}$ is low-rank.

Fast multipole method & $\mathcal{H}$-matrix

Proposition. [Informal] Let $\{T_1, T_2\}$ be two well-separated index sets. If g satisfies certain conditions, for any $\varepsilon > 0$, the sub-matrix $A(T_1, T_2)$ can be approximated by a rank p matrix to a relative tolerance $\varepsilon > 0$ in the sense that: there exists rank p matrices $\boldsymbol{U} \in \mathbb{R}^{|T_1| \times p}, \boldsymbol{V} \in \mathbb{R}^{|T_2| \times p}$, with $p \geq C|\log \epsilon|$ for some constant C, such that

$$|\boldsymbol{A}(i,j) - (\boldsymbol{U}\boldsymbol{V}^\top)(i,j)| \leq \epsilon, \quad \forall i \in T_1, j \in T_2.$$

Taking above well-separated set partitioning recursively, we get the following $\mathcal{H}$-matrix.



Low-rank approximation via kernel tricks

Benefits of low-rank approximation: $\boldsymbol{LV}$ requires $\mathcal{O}(N^2)$ computational time and memory costs if $\boldsymbol{L} \in \mathbb{R}^{N\times N}$. However, if $\boldsymbol{L}$ is rank r with $r \ll N$, then

$$\boldsymbol{LV} = \underbrace{(\boldsymbol{a}_1\boldsymbol{b}_1^\top + \boldsymbol{a}_2\boldsymbol{b}_2^\top + \cdots + \boldsymbol{a}_r\boldsymbol{b}_r^\top)\boldsymbol{V}}_{\mathcal{O}(N^2)} = \underbrace{\boldsymbol{a}_1(\boldsymbol{b}_1^\top\boldsymbol{V}) + \boldsymbol{a}_2(\boldsymbol{b}_2^\top\boldsymbol{V}) + \cdots + \boldsymbol{a}_r(\boldsymbol{b}_r^\top\boldsymbol{V})}_{\mathcal{O}(N)},$$

Practical low-rank attention:

$$\hat{\boldsymbol{v}}_i = \frac{\sum_{j=1}^N k(\boldsymbol{q}_i, \boldsymbol{k}_j)\boldsymbol{v}_j}{\sum_{j=1}^N k(\boldsymbol{q}_i, \boldsymbol{k}_j)} = \frac{\sum_{j=1}^N \phi(\boldsymbol{q}_i)^\top\phi(\boldsymbol{k}_j)\boldsymbol{v}_j}{\sum_{j=1}^N \phi(\boldsymbol{q}_i)^\top\phi(\boldsymbol{k}_j)} = \frac{\phi(\boldsymbol{q}_i)^\top\sum_{j=1}^N \phi(\boldsymbol{k}_j)\boldsymbol{v}_j^\top}{\phi(\boldsymbol{q}_i)^\top\sum_{j=1}^N \phi(\boldsymbol{k}_j)},$$

i.e.,

$$\hat{\boldsymbol{V}} = \frac{\phi(\boldsymbol{Q})(\phi(\boldsymbol{K})^\top\boldsymbol{V})}{\phi(\boldsymbol{Q})\phi(\boldsymbol{K})^\top}.$$

Select a set of linearly independent feature maps $\{\phi_l(\cdot)\}_{l=1}^r \Rightarrow$ **rank-r attention.**

Banded matrix modeling of near-field attention

We model the near-field attention with the following banded matrix

$$\boldsymbol{D} = \mathrm{softmax}\left(\mathrm{band}_k\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{D}}\right)\right), \tag{1}$$

with $k \ll N$.

FMMformer

Original self-attention mechanism:

$$\hat{\boldsymbol{v}}_i = \sum_{j=1}^{N} \mathrm{softmax}\Big(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{D}}\Big)\boldsymbol{v}_j, \iff \hat{\boldsymbol{V}} = \mathrm{softmax}\Big(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{D}}\Big)\boldsymbol{V} := \boldsymbol{A}\boldsymbol{V}.$$

FMMformer:

$$\hat{\boldsymbol{V}} = \underbrace{w_1(\boldsymbol{D}\boldsymbol{V})}_{\text{banded: near-field attention}} + \underbrace{w_2\Bigg(\sum_{l=1}^{r}\frac{\phi_l(\boldsymbol{Q})(\phi_l(\boldsymbol{K})^\top\boldsymbol{V})}{\phi_l(\boldsymbol{Q})\phi_l(\boldsymbol{K})^\top}\Bigg)}_{\text{rank r: far-field attention}},$$

where w_1 and w_2 are two learnable weights with positivity constraints.

Random Fourier Features

Rahimi and Recht, Random Features for Large-Scale Kernel Machines, NeurIPS, 2007.

Question

Given a kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x})^\top \phi(\boldsymbol{y})$, where $\phi(\cdot)$ is a feature map which can be infinite-dimensional. How to construct a finite-dimensional explicit feature map $\boldsymbol{z}(\cdot)$ such that $k(\boldsymbol{x}, \boldsymbol{y}) \approx \boldsymbol{z}(\boldsymbol{x})^\top \boldsymbol{z}(\boldsymbol{y})$. In particular, consider the following kernels:

$$\exp(\boldsymbol{x}^\top \boldsymbol{y}),$$

and

$$\exp(\|\boldsymbol{x} - \boldsymbol{y}\|_2^2).$$

Random features

$$k(\boldsymbol{x}, \boldsymbol{y}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle_{\mathcal{V}} \approx \boldsymbol{z}(\boldsymbol{x})^\top \boldsymbol{z}(\boldsymbol{y}).$$

• The idea was inspired by the following observation. Let $\boldsymbol{w} \in \mathbb{R}^D$ be a random vector such that

$$\boldsymbol{w} \sim \mathcal{N}_D(0, \boldsymbol{I}). \tag{2}$$

Now define h as

$$h : \boldsymbol{x} \to \exp(i \boldsymbol{w}^\top \boldsymbol{x}). \tag{3}$$

Above, i is the imaginary unit.

- Importantly, recall that the complex conjugate of e^{ix} is e^{-ix}. Then note

$$\begin{aligned}\mathbb{E}_{\boldsymbol{w}}[h(\boldsymbol{x})h(\boldsymbol{y})^*] = \mathbb{E}_{\boldsymbol{w}}[\exp(i\boldsymbol{w}^\top(\boldsymbol{x}-\boldsymbol{y}))] &= \int_{\mathbb{R}^D} p(\boldsymbol{w})\exp(i\boldsymbol{w}^\top(\boldsymbol{x}-\boldsymbol{y}))d\boldsymbol{w} \\ &= \exp\Big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{y})^\top(\boldsymbol{x}-\boldsymbol{y})\Big),\end{aligned} \tag{4}$$

where the superscript $*$ denote the complex conjugate. In other words, the expected value of $h(\boldsymbol{x})h(\boldsymbol{y}^*)$ is the Gaussian kernel.

Gaussian kernel derivation, i.e., Equation (4)

Let $\boldsymbol{\delta} = \boldsymbol{x} - \boldsymbol{y}$:

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{w}}[h(\boldsymbol{x})h(\boldsymbol{y})^*] &= \mathbb{E}_{\boldsymbol{w}}[\exp(i\boldsymbol{w}^\top \boldsymbol{x})\exp(-i\boldsymbol{w}^\top \boldsymbol{y})] \\
&= \mathbb{E}_{\boldsymbol{w}}[\exp(i\boldsymbol{w}^\top \boldsymbol{\delta})] = \int_{\mathbb{R}^D} p(\boldsymbol{w})\exp(i\boldsymbol{w}^\top \boldsymbol{\delta})d\boldsymbol{w} \\
&= (2\pi)^{-D/2}\int_{\mathbb{R}^D} \exp\Big(-\frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w}\Big)\exp(i\boldsymbol{w}^\top \boldsymbol{\delta})d\boldsymbol{w} \\
&= (2\pi)^{-D/2}\int_{\mathbb{R}^D} \exp\Big(-(\frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} - i\boldsymbol{w}^\top \boldsymbol{\delta})\Big)d\boldsymbol{w} \\
&= (2\pi)^{-D/2}\int_{\mathbb{R}^D} \exp\Big(-\frac{1}{2}(\boldsymbol{w}^\top \boldsymbol{w} - 2i\boldsymbol{w}^\top \boldsymbol{\delta} - \boldsymbol{\delta}^\top \boldsymbol{\delta}) - \frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{\delta}\Big)d\boldsymbol{w} \\
&= (2\pi)^{-D/2}\exp\Big(-\frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{\delta}\Big)\underbrace{\int_{\mathbb{R}^D} \exp\Big(-\frac{1}{2}(\boldsymbol{w} - i\delta)^\top(\boldsymbol{w} - i\boldsymbol{\delta})\Big)d\boldsymbol{w}}_{(2\pi)^{D/2}} \\
&= \exp(-\frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{\delta}) = k(\delta).
\end{aligned} \tag{5}$$

In other words, $k(\cdot)$ is the Gaussian kernel with $p(\boldsymbol{w})$ be a spherical Gaussian.

Bochner's theorem

The above result is a specific instance of Bochner's theorem.

Bochner's theorem. A continuous kernel $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y})$ on $\mathbb{R}^D$ is positive definite if and only if $k(\Delta)$ is the Fourier transform of a non-negative measure.

The Fourier transform of a non-negative measure, call it $p(\boldsymbol{w})$, is

$$k(\Delta) = \int p(\boldsymbol{w}) \exp(i \boldsymbol{w} \Delta) d\boldsymbol{w}. \tag{6}$$

Remark. Bochner's theorem gives us a general framework to approximate any shift invariant kernel (Gaussian, Laplace, and Cauchy kernels) by re-defining $h(\cdot)$ in (3) to depend on $\boldsymbol{w}$ from any non-negative measure $p(\boldsymbol{w})$, not just the spherical Gaussian in (2). Furthermore, is we sample R i.i.d. realizations $\{\boldsymbol{w}_r\}_{r=1}^{R}$, we can lower the variance of this approximation:

$$
\begin{aligned}
k(\boldsymbol{x},\boldsymbol{y}) = k(\boldsymbol{x}-\boldsymbol{y}) &= \int p(\boldsymbol{w})\exp(i\boldsymbol{w}^\top(\boldsymbol{x}-\boldsymbol{y}))d\boldsymbol{w} \\
&= \mathbb{E}_{\boldsymbol{w}}\big[\exp(i\boldsymbol{w}^\top(\boldsymbol{x}-\boldsymbol{y}))\big] \\
&\underbrace{\approx}_{1} \frac{1}{R}\sum_{r=1}^{R}\exp(i\boldsymbol{w}_r^\top(\boldsymbol{x}-\boldsymbol{y})) \\
&= \begin{pmatrix} \frac{1}{\sqrt{R}}\exp(i\boldsymbol{w}_1^\top\boldsymbol{x}) \\ \frac{1}{\sqrt{R}}\exp(i\boldsymbol{w}_2^\top\boldsymbol{x}) \\ \vdots \\ \frac{1}{\sqrt{R}}\exp(i\boldsymbol{w}_R^\top\top\boldsymbol{x}) \end{pmatrix}^\top \begin{pmatrix} \frac{1}{\sqrt{R}}\exp(-i\boldsymbol{w}_1^\top\boldsymbol{x}) \\ \frac{1}{\sqrt{R}}\exp(-i\boldsymbol{w}_2^\top\boldsymbol{x}) \\ \vdots \\ \frac{1}{\sqrt{R}}\exp(-i\boldsymbol{w}_R^\top\top\boldsymbol{x}) \end{pmatrix} \\
&\underbrace{=}_{2} \boldsymbol{h}(\boldsymbol{x})\boldsymbol{h}(\boldsymbol{y})^*.
\end{aligned}
\tag{7}
$$

- Step 1 is a Monte Carlo approximation of the expectation ($\boldsymbol{w}_r$s are sampled from the distribution $p(\boldsymbol{w})$).

- Step 2 is the definition of a random map $\boldsymbol{h}:\mathbb{R}^D \to \mathbb{R}^R$, so an R-vector of normalized $h(\cdot)$ transformations.

$$
\begin{aligned}
k(\boldsymbol{x}, \boldsymbol{y}) &= k(\boldsymbol{x} - \boldsymbol{y}) \\
&= \begin{pmatrix} \frac{1}{\sqrt{R}} \exp(i \boldsymbol{w}_1^\top \boldsymbol{x}) \\ \frac{1}{\sqrt{R}} \exp(i \boldsymbol{w}_2^\top \boldsymbol{x}) \\ \vdots \\ \frac{1}{\sqrt{R}} \exp(i \boldsymbol{w}_R^\top \top \boldsymbol{x}) \end{pmatrix}^\top \begin{pmatrix} \frac{1}{\sqrt{R}} \exp(-i \boldsymbol{w}_1^\top \boldsymbol{x}) \\ \frac{1}{\sqrt{R}} \exp(-i \boldsymbol{w}_2^\top \boldsymbol{x}) \\ \vdots \\ \frac{1}{\sqrt{R}} \exp(-i \boldsymbol{w}_R^\top \top \boldsymbol{x}) \end{pmatrix} \\
&\underbrace{=}_{2} \boldsymbol{h}(\boldsymbol{x}) \boldsymbol{h}(\boldsymbol{y})^*.
\end{aligned}
$$

Remark. Note that we've talked about the dot product $\boldsymbol{z}(\boldsymbol{x})^\top \boldsymbol{z}(\boldsymbol{y})$, but above we have $\boldsymbol{h}(\boldsymbol{x})\boldsymbol{h}(\boldsymbol{y})^*$. As we will see next, the imaginary part of our random map will disappear, and the new transform is what we used in machine learning.

Fine tuning

We have discussed the big idea of a low-dimensional, randomized map and why it might work, let us get into the weeds.

- First, note that since both our distribution $\mathcal{N}_D(0, \boldsymbol{I})$ and the kernel $k(\Delta)$ are real-valued, we can write

$$\exp(i\boldsymbol{w}^\top(\boldsymbol{x}-\boldsymbol{y})) \underbrace{=}_{\text{Euler's formula}} \cos(\boldsymbol{w}^\top(\boldsymbol{x}-\boldsymbol{y})) - \cancel{i\sin(\boldsymbol{w}^\top(\boldsymbol{x}-\boldsymbol{y}))} = \cos(\boldsymbol{w}^\top(\boldsymbol{x}-\boldsymbol{y})) \quad (8)$$

- We can then define $z_{\boldsymbol{w}}(\boldsymbol{x})$–note that this is still not yet the bolded $\boldsymbol{z}$–without the imaginary unit as

$$\begin{aligned} \boldsymbol{w} &\sim p(\boldsymbol{w}) \\ b &\sim \mathrm{Uniform}(0, 2\pi) \\ z_{\boldsymbol{w}}(\boldsymbol{x}) &= \sqrt{2}\cos(\boldsymbol{w}^\top\boldsymbol{x} + b). \end{aligned} \quad (9)$$

This works because

$$\begin{aligned}\mathbb{E}_{\boldsymbol{w}}[z_{\boldsymbol{w}}(\boldsymbol{x})z_{\boldsymbol{w}}(\boldsymbol{y})] &= \mathbb{E}_{\boldsymbol{w}}[\sqrt{2}\cos(\boldsymbol{w}^\top \boldsymbol{x} + b)\sqrt{2}\cos(\boldsymbol{w}^\top \boldsymbol{y} + b)] \\ &\underbrace{=}_{(1)} \mathbb{E}_{\boldsymbol{w}}[\cos(\boldsymbol{w}^\top(\boldsymbol{x}+\boldsymbol{y}) + 2b)] + \mathbb{E}_{\boldsymbol{w}}[\cos(\boldsymbol{w}^\top(\boldsymbol{x}-\boldsymbol{y}))] \\ &\underbrace{=}_{(2)} \mathbb{E}_{\boldsymbol{w}}[\cos(\boldsymbol{w}^\top(\boldsymbol{x}-\boldsymbol{y}))]\end{aligned} \tag{10}$$

- (1) is because of the following trigonometry identity

$$\cos(x+y) = \cos(x)\cos(y) - \sin(x)\sin(y).$$

- (2) uses the fact that since $b \sim \mathrm{Uniform}(0, 2\pi)$, the expectation w.r.t. b is zero:

Lemma.

$$\mathbb{E}_{\boldsymbol{w}}[\cos(\boldsymbol{w}^\top(\boldsymbol{x}+\boldsymbol{y}) + 2b)] = \mathbb{E}_{\boldsymbol{w}}[\mathbb{E}_b[\cos(\boldsymbol{w}^\top(\boldsymbol{x}+\boldsymbol{y}) + 2b) | \boldsymbol{w}]] = 0. \tag{11}$$

Proof of the Lemma. Note that

$$\mathbb{E}_{\boldsymbol{w}}[\cos(\boldsymbol{w}^\top(\boldsymbol{x}+\boldsymbol{y})+2b)] = \mathbb{E}_{\boldsymbol{w}}[\mathbb{E}_b[\cos(\boldsymbol{w}^\top(\boldsymbol{x}+\boldsymbol{y})+2b)|\boldsymbol{w}]]$$

holds by the law of total expectation. We claim the inner conditional expectation is zero. To ease notation, let $\boldsymbol{t} = \boldsymbol{w}^\top(\boldsymbol{x}-\boldsymbol{y})$. Then

$$\begin{aligned}
\mathbb{E}_b[\cos(\boldsymbol{t}+2b)|\boldsymbol{w}] &= \int_0^{2\pi} \frac{\cos(\boldsymbol{t}+2b)}{2\pi} db \\
&= \frac{1}{2\pi}\int_0^{2\pi} \cos(\boldsymbol{t}+2b) db \\
&= \frac{1}{2\pi}\left[\sin(\boldsymbol{t}+2b)|_0^{2\pi}\right] \\
&= \frac{1}{2\pi}\left[\sin(\boldsymbol{t}) - \sin(\boldsymbol{t}+4\pi)\right] \\
&= 0
\end{aligned}$$

The last step holds because $\sin(\boldsymbol{t}) = \sin(\boldsymbol{t} \pm 2\pi k)$ for any integer k.

Fine tuning

We are now ready to define the random map $\boldsymbol{z} : \mathbb{R}^D \to \mathbb{R}^R$ such that (??) holds. Let

$$\boldsymbol{z}(\boldsymbol{x}) = \begin{pmatrix} \frac{1}{\sqrt{R}} z_{\boldsymbol{w}_1}(\boldsymbol{x}) \\ \frac{1}{\sqrt{R}} z_{\boldsymbol{w}_2}(\boldsymbol{x}) \\ \vdots \\ \frac{1}{\sqrt{R}} z_{\boldsymbol{w}_R}(\boldsymbol{x}) \end{pmatrix} \tag{12}$$

and therefore

$$\begin{aligned} \boldsymbol{z}(\boldsymbol{x})^\top \boldsymbol{z}(\boldsymbol{y}) &= \frac{1}{R} \sum_{r=1}^{R} z_{\boldsymbol{w}_r}(\boldsymbol{x}) z_{\boldsymbol{w}_r}(\boldsymbol{y}) = \frac{1}{R} \sum_{r=1}^{R} 2\cos(\boldsymbol{w}_r^\top \boldsymbol{x} + b_r)\cos(\boldsymbol{w}_r^\top \boldsymbol{y} + b_r) \\ &= \frac{1}{R} \sum_{r=1}^{R} \cos(\boldsymbol{w}_r^\top (\boldsymbol{x} - \boldsymbol{y})) \approx \mathbb{E}_{\boldsymbol{w}}[\cos(\boldsymbol{w}^\top (\boldsymbol{x} - \boldsymbol{y}))] = k(\boldsymbol{x}, \boldsymbol{y}). \end{aligned} \tag{13}$$

We now have a simple algorithm to estimate a shift invariant, positive definite kernel. Draw R samples of $\boldsymbol{w} \sim p(\boldsymbol{w})$ and $b \sim \mathrm{Uniform}(0, 2\pi)$ and then compute $\boldsymbol{z}(\boldsymbol{x})^\top \boldsymbol{z}(\boldsymbol{x})$.

Alternative random Fourier features

An alternative version of random Fourier feature is

$$z_{\boldsymbol{w}_r}(\boldsymbol{x}) = \begin{pmatrix} \cos(\boldsymbol{w}_r^\top \boldsymbol{x}) \\ \sin(\boldsymbol{w}_r^\top \boldsymbol{x}) \end{pmatrix} \tag{14}$$

.

Draw $R' = R/2$ samples

$$\boldsymbol{w}_r \sim p(\boldsymbol{w}). \tag{15}$$

Then

$$\begin{aligned} \frac{1}{R'} \sum_{r=1}^{R'} \sum_{r=1}^{R'} z_{\boldsymbol{w}_r}(\boldsymbol{x})^\top z_{\boldsymbol{w}_r}(\boldsymbol{y}) &\equiv \frac{2}{R} \sum_{r=1}^{R/2} \left(\begin{pmatrix} \cos(\boldsymbol{w}_r^\top \boldsymbol{x}) \\ \sin(\boldsymbol{w}_r^\top \boldsymbol{x}) \end{pmatrix}^\top \begin{pmatrix} \cos(\boldsymbol{w}_r^\top \boldsymbol{y}) \\ \sin(\boldsymbol{w}_r^\top \boldsymbol{y}) \end{pmatrix} \right) \\ &= \frac{2}{R} \sum_{r=1}^{R/2} \cos(\boldsymbol{w}_r^\top \boldsymbol{x}) \cos(\boldsymbol{w}_r^\top \boldsymbol{y}) + \sin(\boldsymbol{w}_r^\top \boldsymbol{x}) \sin(\boldsymbol{w}_r^\top \boldsymbol{y}) \qquad (16) \\ &\underbrace{=}_{*} \frac{2}{R} \sum_{r=1}^{R/2} \cos(\boldsymbol{w}_r^\top \boldsymbol{x} - \boldsymbol{w}_r^\top \boldsymbol{y}) \approx \mathbb{E}_{\boldsymbol{w}}[\cos(\boldsymbol{w}^\top(\boldsymbol{x} - \boldsymbol{y}))] = k(\boldsymbol{x}, \boldsymbol{y}). \end{aligned}$$

$*$ in the last equation dues to the product identities from trigonometry:

$$2\sin(x)\sin(y) = \cos(x-y) - \cancel{\cos(x+y)}; \quad 2\cos(x)\cos(y) = \cos(x-y) + \cancel{\cos(x+y)}. \tag{17}$$

The right-most terms above cancel in (16), and we get $2\cos(x-y)$.

Example: Gaussian kernel approximation

Let us first approximate a Gaussian kernel using random Fourier features. Sample R i.i.d. $\boldsymbol{w}$ variables from a spherical Gaussian and then compute

$$\boldsymbol{z}(\boldsymbol{x})^\top \boldsymbol{z}(\boldsymbol{y}) = \frac{1}{R}\sum_{r=1}^{R} z_{\boldsymbol{w}_r}(\boldsymbol{x})^\top z_{\boldsymbol{w}_r}(\boldsymbol{y}) = \frac{1}{R}\sum_{r=1}^{R} \cos(\boldsymbol{w}_r^\top(\boldsymbol{x}-\boldsymbol{y})). \tag{18}$$

for each $(\boldsymbol{x}, \boldsymbol{y})$ pair in the data. The result $N \times N$ matrix is the approximate covariance matrix induced by the Gaussian kernel function. Concretely, let $\boldsymbol{Z}_X$ denote $\boldsymbol{z}(\cdot)$ applied to all N samples $\boldsymbol{x}_n$. Thus, $\boldsymbol{Z}_X$ is $N \times R$ and therefore

$$\boldsymbol{K}_X \approx \begin{bmatrix} \boldsymbol{z}(\boldsymbol{x}_1) \\ \vdots \\ \boldsymbol{z}(\boldsymbol{x}_N) \end{bmatrix} \begin{bmatrix} \boldsymbol{z}(\boldsymbol{x}_1) & \cdots & \boldsymbol{z}(\boldsymbol{x}_N) \end{bmatrix} = \boldsymbol{Z}_X \boldsymbol{Z}_X^\top, \tag{19}$$

because

$$\begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix} \approx \begin{pmatrix} \mathbf{z}(\mathbf{x}_1)^\top \mathbf{z}(\mathbf{x}_1) & \cdots & \mathbf{z}(\mathbf{x}_1)^\top \mathbf{z}(\mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \mathbf{z}(\mathbf{x}_N)^\top \mathbf{z}(\mathbf{x}_1) & \cdots & \mathbf{z}(\mathbf{x}_N)^\top \mathbf{z}(\mathbf{x}_N) \end{pmatrix} \tag{20}$$

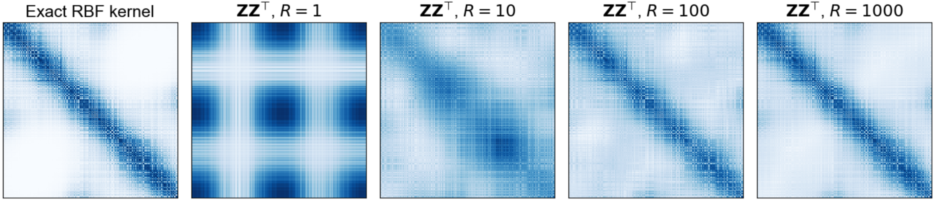


Figure: As R increases, the covariance matrix approximation improves because each cell value uses more Monte Carlo samples to estimate the basis function $\phi(\cdot)$ associated with $k(\cdot, \cdot)$ for the pair of samples associated with that cell.