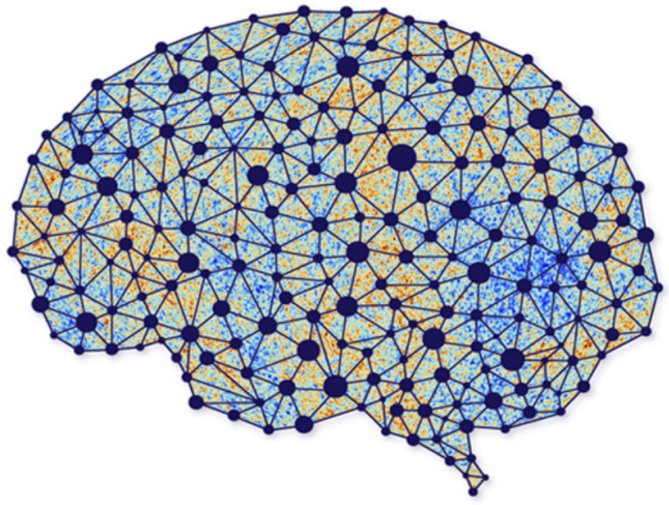
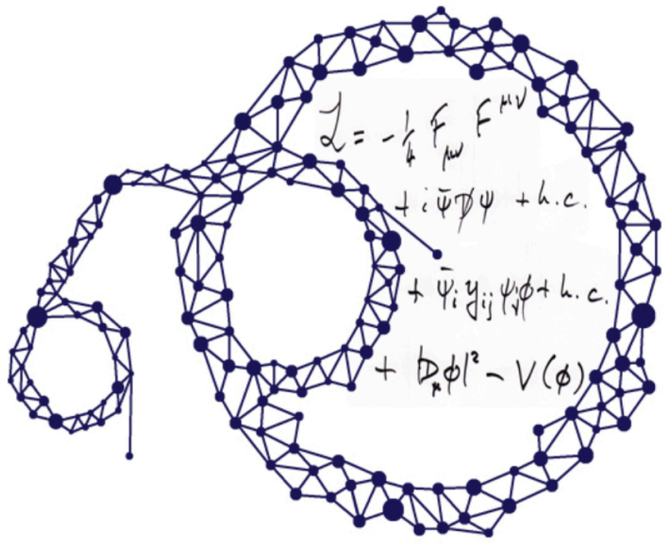
Physics 361 - Machine Learning in Physics

Lecture 15 – Transformers

March 11th 2025



AI
∩
Universe



Moritz Münchmeyer (with slides from Gary Shiu)

Advertisement 1

Events at Physics

<< Fall 2024 Spring 2025 Summer 2025 >>

Subscribe your calendar or receive email announcements of events

<< March 2025 >>

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
>>							1
>>	2	3	4	5	6	7	8
>>	9	10	11	12	13	14	15
>>	16	17	18	19	20	21	22
>>	23	24	25	26	27	28	29
>>	30	31					

Physics ∩ ML Seminar
Thinking Fast & Slow with AI

Add an Event Edit This Event

Date: **Wednesday, March 12th**
Time: 10:00 am - 11:00 am
Place: Chamberlin 5280
Speaker: Sash Sarangi, EMAlpha
Abstract: We will discuss various contexts where the AI/ML model must "think" fast and slow (a la Danny Kahneman) to solve problems effectively. After defining the contexts, we will discuss a couple of approaches to getting the models to think in this way.
Host: Gary Shiu

Add this event to your calendar

Advertisement 2

- The X-ray astronomy group in the physics department is building a sounding-rocket instrument to measure the spectrum of million-degree interstellar gas with an unprecedented resolution of 1 eV over the 70-1000 eV range. We are in immediate **need of a student** looking for research experience to assist with the design of the superconducting and normal magnetic shielding of these new detectors, which operate at a temperature of 0.05 K and are very sensitive to magnetic fields.

- The student would be building models using the COMSOL multi-physics package in close collaboration with graduate student Sophia Nowak. Smaller models can be run on our computer, more complex ones and large amounts of optimization would be run at the Center for High Throughput computing.

- Starts: immediately

- time commitment: 10 hrs/week (6 hrs/week minimum) - schedule is somewhat flexible and will allow for classes, exams, etc, but need to average these hrs/week.

- requires:
  - Good familiarity with computers and an operating system. Linux preferred, but windows OK.
  - Good familiarity with Matplotlib, Gnuplot, or some similarly flexible plotting program.
  - Solid Works or Pro-E experience a plus.
  - Availability extending into summer and/or next fall a plus (by this point you should be working more independently).

- Pay: Start $15.00/hour, depending on experience.

- Conctact: Dan McCammon mccammon@physics.wisc.edu <mailto:mccammon@physics.wisc.edu> or Sophia Nowak srnowak2@wisc.edu

Final project

- You will write **a paper on an application of machine learning to physics** of your choice. Your paper needs to contain a computational analysis, which generally will mean applying a machine learning method to some data set.
- You can **work alone or in groups of up to four people. For larger groups we will expect a little bit more total effort.**
- The paper should be **5 to 10 pages** and contain the following:
    - A short review of at least one research paper related to your topic. This is to encourage you to learn how to browse the literature.
    - A description of the data set you will be working with and its properties.
    - A brief description of the machine learning method you will use. Don’t re-explain basics such as how CNNs work, rather describe the detailed properties of your approach.
    - Train the model and put the results in your paper. Explore some variations such as different hyper parameters.
    - Describe successes and problems in your analysis.
- If you are already doing research in physics or a related field, you can write the paper on this topic if you wish.

Final project

- You can **use machine learning methods either from the lecture or ones that we have not covered**. Major topics which we have not yet covered but will be covering in the next weeks are Generative models (GANs, Diffusion, Normalizing Flows), Simulation Based Inferences, and Transformers and LLMs.

- The project should take you a few days of work, spread over the rest of the semester.

- We will have an **intermediate check-in. Format TBA.**

- Your **paper will be due on Sunday May 4th at midnight**.

- We want to **know your topic by March 11th**. You can discuss your topic ideas with Yurii or with me, after the lecture, or during office hours.
  - **Please send an Email to myself and Yurii with your proposed topic and team members.**

- **We will have a brief ~1 slide presentation of your results in the lecture on April 29th.**

Transformers

Introduction

Introduction

- Transformer is one of the most talked about ML architecture (e.g. ChatGPT).
- Initially targeted at natural language processing (NLP) problems, transformers are now being used quite generally on unstructured data representations (texts, images, audio, video, and their combo).
- These ML models are known as transformers because they transform a set of vectors in some representation space into a corresponding set of vectors, having the same dimensionality, in some new space.
- The new space has a richer internal representation that is better suited to solving downstream tasks.
- Reference: “Deep learning: Foundations and Concepts” by Chris Bishop with Hugh Bishop, Chapter 12: https://www.bishopbook.com/

Why should you care?

- Math and Physics problems are language problems, expressed in terms of formulae. Your tasks are to translate questions to answers.

- Numerous applications of transformers in math and theoretical physics. Applications of ML are not limited to experimental areas.

- Some success in solving college level physics and math problems (see talks by Guy Gur-Ari and Francois Charton at http://www.physicsmeetsml.org/)

- AI Does Math as Well as Math Olympians: https://www.scientificamerican.com/article/ai-matches-the-abilities-of-the-best-math-olympians/

- Examples of research level problems:

  - https://deepmind.google/discover/blog/funsearch-making-new-discoveries-in-mathematical-sciences-using-large-language-models/

  - https://nips.cc/virtual/2023/76132

Foundational Model

- A large-scale model that can be adapted to solve multiple different tasks is known as a foundation model, e.g., https://polymathic-ai.org/

- Transformers can be trained in a self-supervised way using unlabeled data, which is especially effective with language models since there are vast quantities of text available from the internet.

- The scaling hypothesis asserts that simply by increasing the number of learnable parameters and training on a commensurately large data set, significant improvements in performance can be achieved.

- Transformers are quite suited for massively parallel processing hardware, e.g., GPU. Models with 10^{12} parameters can be trained in reasonable time.

- The pre-trained models can then be fine-tuned for specific tasks.

Natural Language Processing

- Language datasets share some similarities with image data:
  - The number of input variables can be very large.
  - The statistics are similar at every position; not sensible to re-learn the meaning of dog at every possible position in a body of text.
- These are the reasons for introducing CNN: instead of fully connected NN, a CNN employs parameter sharing.
- However, language datasets have varying lengths in text sequences. There is no easy way to resize them.

An Illustrative Example

- Consider the following restaurant review

> The restaurant refused to serve me a ham sandwich because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambiance was just as good as the food and service.

- How to process texts like this into a representation suitable for downstream tasks (positive/negative review? is steak served?)

- Three problems to overcome:
  - Inputs are large: 37 words represented by an embedding vector of length 1024 has a 37x1024 = 37888 dimensional input.
  - Inputs have different lengths: not obvious how to apply fully connected NNs; how to share parameters across words at different positions?
  - Language is ambiguous: it refers to the restaurant and not to ham sandwich. A successful ML model should pay *attention* to the word restaurant. There are connections between words and the strength of these connections depends on the words themselves. The word their also refers to the restaurant.

Transformers

Attention

Attention is all you need

https://arxiv.org/abs/1706.03762

- Originally developed as an enhancement to RNNs for machine translation: https://arxiv.org/abs/1409.0473
- https://arxiv.org/abs/1706.03762) showed that the RNN structure can be eliminated; instead focus exclusively on the attention mechanism.
- Consider the following two sentences:

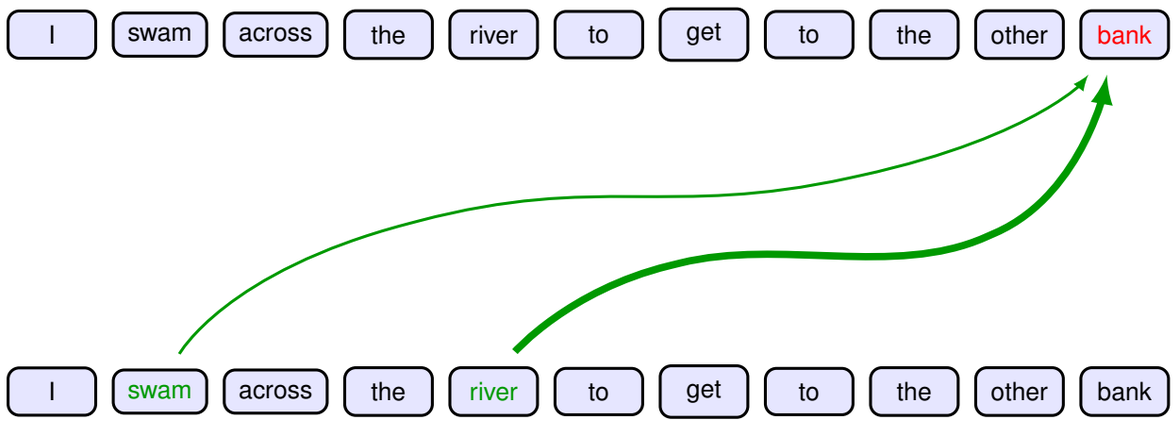
I swam across the river to get to the other bank.

I walked across the road to get cash from the bank.

- The word “bank” has different meanings which can be detected by looking at other words in the sentence.
- In the first sentence, the words “swam” and “river” most strongly indicate that “bank” refers to the side of a river, while in the second sentence, the word “cash” is a strong indicator that “bank” refers to a financial institution.

Attention is all you need

- A NN processing a sentence should attend to specific words from the rest of the sequence:



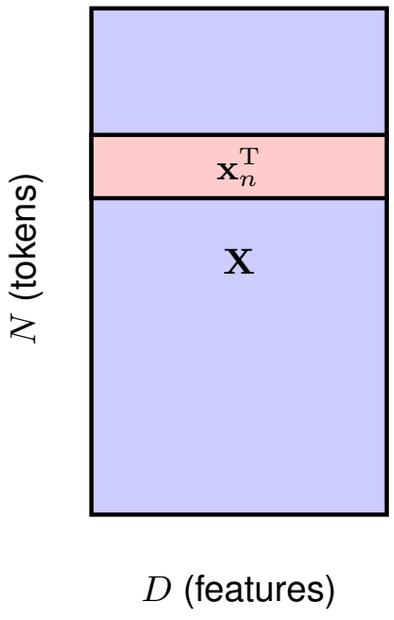
- The specific locations that should receive more attention depends on the input sequence itself.
- In a standard NN, once a network is trained, the weights are independent on the input data.
- By contrast, attention uses weighting factors whose values depend on the specific input data.

Word Embedding

- Words are mapped into vectors in an embedding space.
- Words with similar meanings are mapped to nearby locations in the embedding space.
- A transformer is a richer form of embedding in which a given vector is mapped to a location that depends on other vectors in the sequence.
- The vector representing “bank” is mapped to a location close to “water” in the embedding space in the first sentence, and close to “money” in the second sentence.
- Not only for words: a protein is a 1d sequence of amino acids (22 possibilities). A protein can comprise hundreds or thousands of such amino acids. Amino acids that are widely separated in the 1d sequence can be physically close in 3d space if the proton folds. A transformer model allows distant amino acids to attend to each other for modeling 3d structure.
- For similar reasons, transformers have been used for modeling molecular dynamics.

Transformer Processing

- Input data is a set of vectors $\{\mathbf{x}_n\}$ of dimensionality D, $n = 1, \ldots, N$.
- These data vectors are known as **tokens** (e.g., a word within a sentence, a patch within an image, or an amino acid within a protein).
- The elements x_{ni} of the tokens are called **features**.
- Transformers can handle a mix of different data types by combining the data variables into a joint set of tokens.
- Combining the data vectors into a matrix X of dimensions $N \times D$.



$$\widetilde{\mathbf{X}} = \mathrm{TransformerLayer}\,[\mathbf{X}]$$

same dimensionality as X

Apply multiple transformer layer
to learn rich internal representations.

Attention Coefficients

- A set of input tokens $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ is mapped to a set of output tokens $\{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$.
- With attention, this dependence should be stronger for those inputs $\mathbf{x}_m$ that are particularly important for determining $\mathbf{y}_n$.
- Consider the map:

$$\mathbf{y}_n = \sum_{m=1}^{N} a_{nm}\mathbf{x}_m$$

where a_{nm} are called attention weights. $a_{nm} \approx 0$ for input tokens $\mathbf{x}_m$ that have little influence on the output $\mathbf{y}_n$ and large otherwise.

- The attention weights satisfy two constraints:

$$a_{nm} \geqslant 0$$

avoid cancellation from large coefficients of opposite signs.

$$\sum_{m=1}^{N} a_{nm} = 1.$$

normalize the total attention.

Self-attention

- Consider the problem of choosing which movie to watch on Netflix.
- Associate each movie with a list of attributes: genre, names of leading actors, length of movie, etc.
- Search though a catalogue to find a movie that matches preferences.
- Encode the attributes of each movie in a vector called the **key**.
- The corresponding movie file is called a **value**.
- The user's personal vector of attributes is called the **query**.
- Netflix compares the query vector with all the key vectors to find the best match, and send the user the corresponding movie (value) file.
- Hard attention: a single value vector is returned.

Dot-Product Self-attention

- For transformer, we generalize this info retrieval to **soft attention**.
- Use continuous variables to measure the degree of match between queries and keys, then use these variables to weight the influence.
- Transformer function is differentiable, trainable by gradient descent.
- To satisfy the two constraints on the attention weights, we define:

$$a_{nm} = \frac{\exp(\mathbf{x}_n^{\mathrm{T}}\mathbf{x}_m)}{\sum_{m'=1}^{N}\exp(\mathbf{x}_n^{\mathrm{T}}\mathbf{x}_{m'})}$$

- In matrix notation:

$$\mathbf{Y} = \mathrm{Softmax}\left[\mathbf{X}\mathbf{X}^{\mathrm{T}}\right]\mathbf{X}$$

where $\mathrm{Softmax}[\mathbf{L}]$ is an operator that takes the exponential of every element of a matrix $\mathbf{L}$ then normalizes each row independently to sum to 1.

- **Dot-product self-attention** (using the same sequence to determine the queries, keys, and values; measure of similarity is given by dot product).

Network Parameters

- Transformation from $\{\mathbf{x}_n\}$ to $\{\mathbf{y}_n\}$ is fixed, with no capacity to learn from data because it has no adjustable parameters.
- Each feature within a token vector $\{\mathbf{x}_n\}$ plays an equal role in determining a_{nm}. Want flexibility to focus on some features vs others.
- We can address both issues if we define modified feature vectors:

$$\widetilde{\mathbf{X}} = \mathbf{X}\mathbf{U}$$

- $\mathbf{U}$ is a $D \times D$ matrix of learnable weight parameters, analogous to a layer in a standard NN. This gives a modified transformation:

$$\mathbf{Y} = \mathrm{Softmax}\left[\mathbf{X}\mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\right]\mathbf{X}\mathbf{U}$$

- This has more flexibility, but still the matrix $\mathbf{X}\mathbf{U}\mathbf{U}^{\mathbf{T}}\mathbf{X}^{\mathbf{T}}$ is symmetric.

Network Parameters

- The attention mechanism should support significant asymmetry, e.g., “chisel” is strongly associated with “tool”, but not the other way round.
- Although the softmax function means the attention weights matrix is not symmetric (NB normalization), we can create more flexibility by allowing queries & keys to have independent parameters.
- Define query, key, & value matrices each w/ different transformations:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{(\mathrm{q})}$$
$$\mathbf{K} = \mathbf{X}\mathbf{W}^{(\mathrm{k})}$$
$$\mathbf{V} = \mathbf{X}\mathbf{W}^{(\mathrm{v})}$$

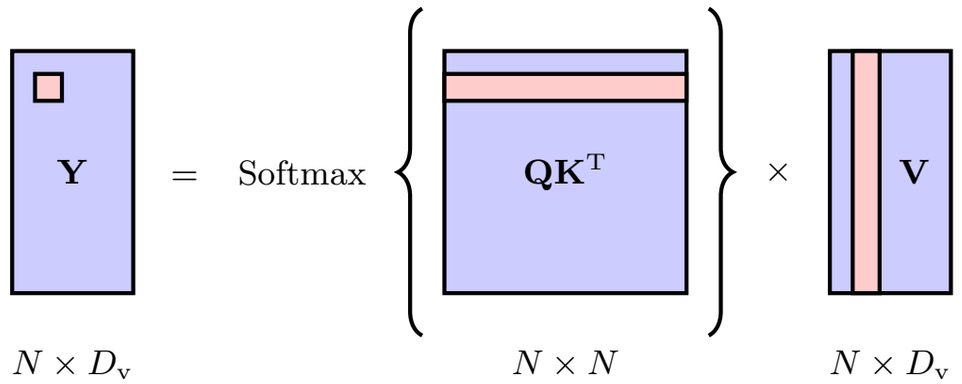
the weight matrices $\mathbf{W}^{(q)}, \mathbf{W}^{(k)}, \mathbf{W}^{(v)}$ represent parameters that will be **learned** during the training of the transformer architecture.

- $\mathbf{W}^{(q)}, \mathbf{W}^{(k)}, \mathbf{W}^{(v)}$ are matrices of dim. $D \times D_k, D \times D_q, D \times D_v$. Setting $D_k = D_q$ allows for dot-products between query and key while $D_v = D$ allows multiple transformer layers to be stacked. We set $D_k = D_q = D_v = D$.

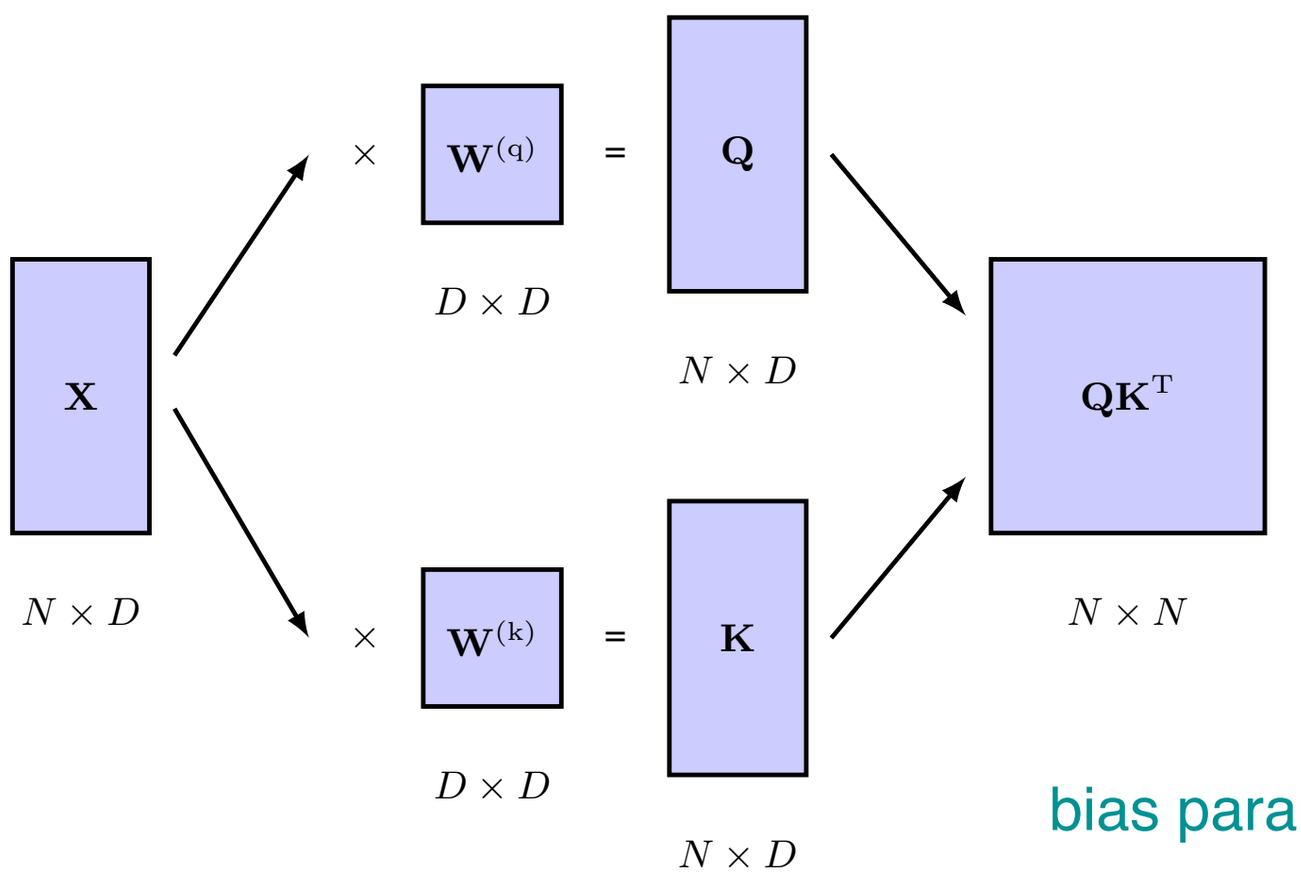
Network Parameters

- The transformation is now generalized to:

$$\mathbf{Y} = \text{Softmax}\left[\mathbf{Q}\mathbf{K}^{\text{T}}\right]\mathbf{V}$$



whereas the dot-product can be computed by:



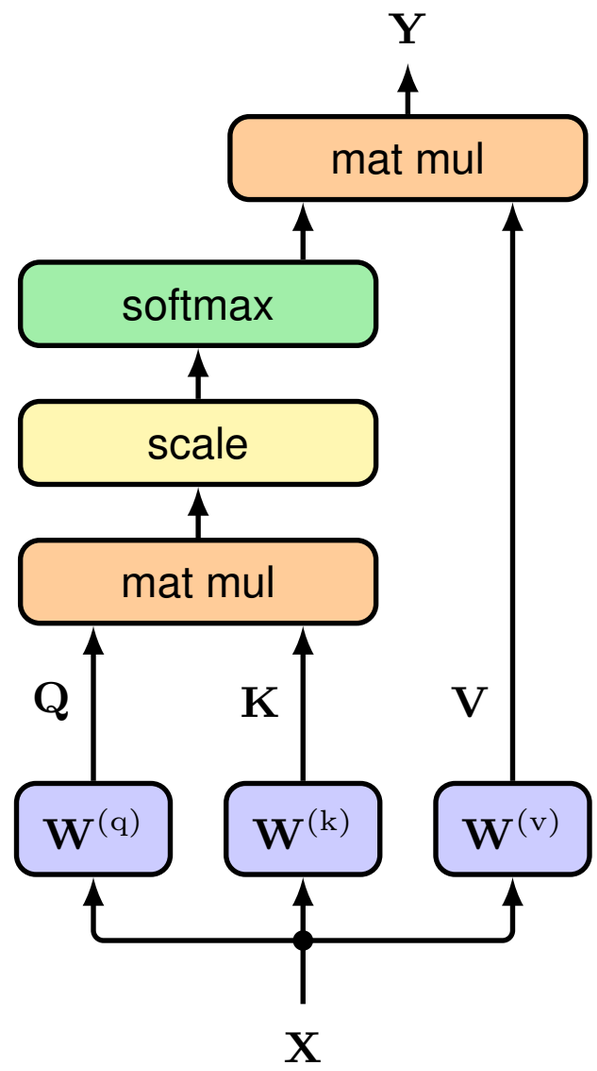
bias parameters are implicit

Scaled self-attention

- Gradient of Softmax becomes exponentially small for inputs of high magnitude, c.f. $\tanh$ or sigmoid activation; trouble with grad descent.
- Rescale the product of the query and key vectors before Softmax.
- If the elements of the query and key vectors were all independent random numbers with zero mean and unit variance, then the variance of the dot product would be D_k.
- Normalizing the argument to the softmax using the standard deviation given by $\sqrt{D_k}$:

$$\mathbf{Y} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \equiv \text{Softmax}\left[\frac{\mathbf{Q}\mathbf{K}^{\text{T}}}{\sqrt{D_{\text{k}}}}\right]\mathbf{V}.$$

- This is the **scaled dot-product self-attention**.



Algorithm 12.1: Scaled dot-product self-attention

Input: Set of tokens $\mathbf{X} \in \mathbb{R}^{N \times D} : \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$

Weight matrices $\{\mathbf{W}^{(\mathrm{q})}, \mathbf{W}^{(\mathrm{k})}\} \in \mathbb{R}^{D \times D_\mathrm{k}}$ **and** $\mathbf{W}^{(\mathrm{v})} \in \mathbb{R}^{D \times D_\mathrm{v}}$

Output: $\mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \in \mathbb{R}^{N \times D_\mathrm{v}} : \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$

$\mathbf{Q} = \mathbf{X}\mathbf{W}^{(\mathrm{q})}$ `// compute queries` $\mathbf{Q} \in \mathbb{R}^{N \times D_\mathrm{k}}$

$\mathbf{K} = \mathbf{X}\mathbf{W}^{(\mathrm{k})}$ `// compute keys` $\mathbf{K} \in \mathbb{R}^{N \times D_\mathrm{k}}$

$\mathbf{V} = \mathbf{X}\mathbf{W}^{(\mathrm{v})}$ `// compute values` $\mathbf{V} \in \mathbb{R}^{N \times D}$

return $\mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{Softmax}\left[\dfrac{\mathbf{Q}\mathbf{K}^\mathrm{T}}{\sqrt{D_\mathrm{k}}}\right]\mathbf{V}$

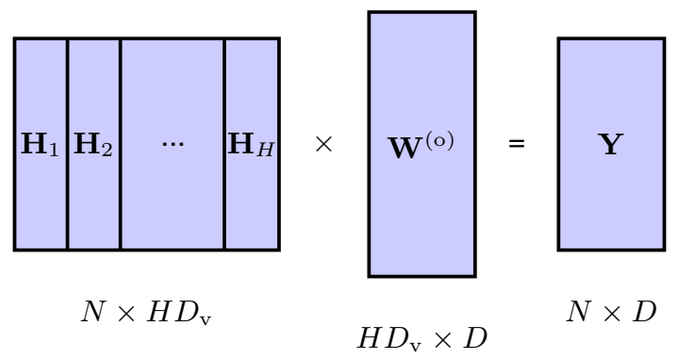
Multi-head attention

- There might be multiple patterns of attention relevant at the same time, e.g., some associated with tenses, some with vocabulary.
- Single “attention head” averages out these effects. Instead use multiple attention heads in parallel; analogous to channels in CNN.
- Suppose we have H heads indexed by $h = 1, \ldots, H$:

$$\mathbf{H}_h = \text{Attention}(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h)$$

- The heads are concatenated into a single matrix, and the result is then linearly transformed to give a combined output:

$$\mathbf{Y}(\mathbf{X}) = \text{Concat}\left[\mathbf{H}_1, \ldots, \mathbf{H}_H\right] \mathbf{W}^{(\text{o})}$$



- The matrix $\mathbf{W}^{(\text{o})}$ is learned along with the weight matrices $\mathbf{W}^{(q)}$, $\mathbf{W}^{(k)}$, $\mathbf{W}^{(v)}$.

Algorithm 12.2: Multi-head attention

Input: Set of tokens $\mathbf{X} \in \mathbb{R}^{N \times D} : \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$
Query weight matrices $\{\mathbf{W}_1^{(\mathrm{q})}, \ldots, \mathbf{W}_H^{(\mathrm{q})}\} \in \mathbb{R}^{D \times D}$
Key weight matrices $\{\mathbf{W}_1^{(\mathrm{k})}, \ldots, \mathbf{W}_H^{(\mathrm{k})}\} \in \mathbb{R}^{D \times D}$
Value weight matrices $\{\mathbf{W}_1^{(\mathrm{v})}, \ldots, \mathbf{W}_H^{(\mathrm{v})}\} \in \mathbb{R}^{D \times D_{\mathrm{v}}}$
Output weight matrix $\mathbf{W}^{(\mathrm{o})} \in \mathbb{R}^{HD_{\mathrm{v}} \times D}$

Output: $\mathbf{Y} \in \mathbb{R}^{N \times D} : \{\mathbf{y}_1, \ldots, \mathbf{x}_N\}$

// compute self-attention for each head (Algorithm 12.1)

for $h = 1, \ldots, H$ **do**

$\mathbf{Q}_h = \mathbf{X}\mathbf{W}_h^{(\mathrm{q})}, \quad \mathbf{K}_h = \mathbf{X}\mathbf{W}_h^{(\mathrm{k})}, \quad \mathbf{V}_h = \mathbf{X}\mathbf{W}_h^{(\mathrm{v})}$

$\mathbf{H}_h = \mathrm{Attention}\left(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h\right)$ // $\mathbf{H}_h \in \mathbb{R}^{N \times D_{\mathrm{v}}}$

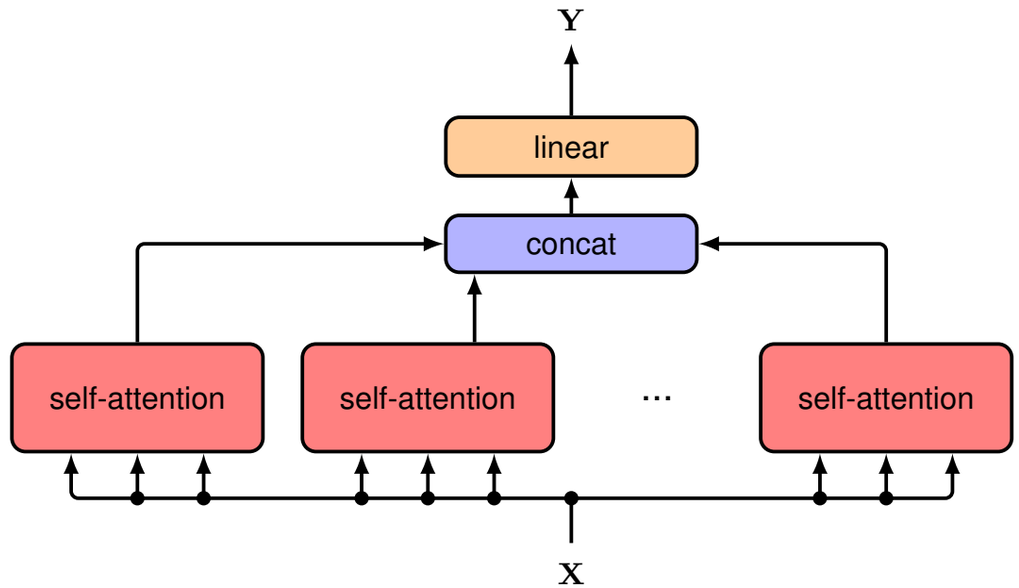
end for

$\mathbf{H} = \mathrm{Concat}\left[\mathbf{H_1}, \ldots, \mathbf{H_N}\right]$ // concatenate heads

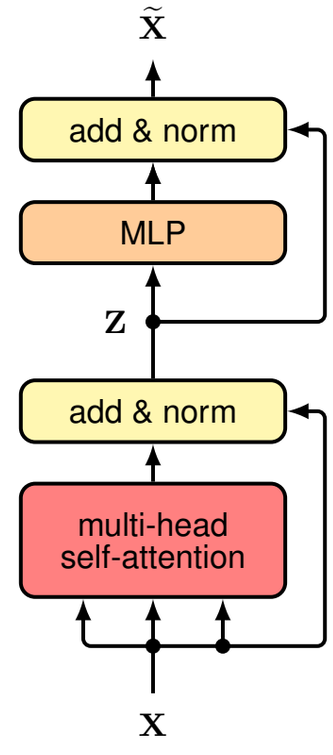
return $\mathbf{Y}(\mathbf{X}) = \mathbf{H}\mathbf{W}^{(\mathrm{o})}$

Transformer Layers

- NNs benefit greatly from depth, so we can stack self-attention layers (like the right) on top of each other.
- To improve efficiency, transformer layers are followed by **layer normalization**: https://arxiv.org/abs/1607.06450



- Output of an attention layer are constrained to be linear combinations of the inputs, though non-linearities enter through the attention weights.
- Enhance flexibility by post-processing the output of each layer using non-linear network denoted by MLP (e.g., fully connected NN with ReLu activation.



Algorithm 12.3: Transformer layer

Input: Set of tokens $\mathbf{X} \in \mathbb{R}^{N \times D} : \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$

Multi-head self-attention layer parameters

Feed-forward network parameters

Output: $\widetilde{\mathbf{X}} \in \mathbb{R}^{N \times D} : \{\widetilde{\mathbf{x}}_1, \ldots, \widetilde{\mathbf{x}}_N\}$

$\mathbf{Z} = \text{LayerNorm}\left[\mathbf{Y}(\mathbf{X}) + \mathbf{X}\right]$ // $\mathbf{Y}(\mathbf{X})$ from Algorithm 12.2

$\widetilde{\mathbf{X}} = \text{LayerNorm}\left[\text{MLP}\left[\mathbf{Z}\right] + \mathbf{Z}\right]$ // shared neural network

return $\widetilde{\mathbf{X}}$

Position Encoding

- The weight matrices $\mathbf{W}^{(q)}, \mathbf{W}^{(k)}, \mathbf{W}^{(v)}$ are shared among the input tokens, so transformer is equivariant w.r.t. input permutations.
- Token ordering is important in sequential processing: *"The professor failed the students"* is different from *"The students failed the professor"*.
- Construct a position encoding vector $\mathbf{r}_n$ and combine with the input token embedding $\mathbf{x}_n$. Concatenation would increase the dim of input space and significantly increase computational cost. Instead:

$$\widetilde{\mathbf{x}}_n = \mathbf{x}_n + \mathbf{r}_n.$$

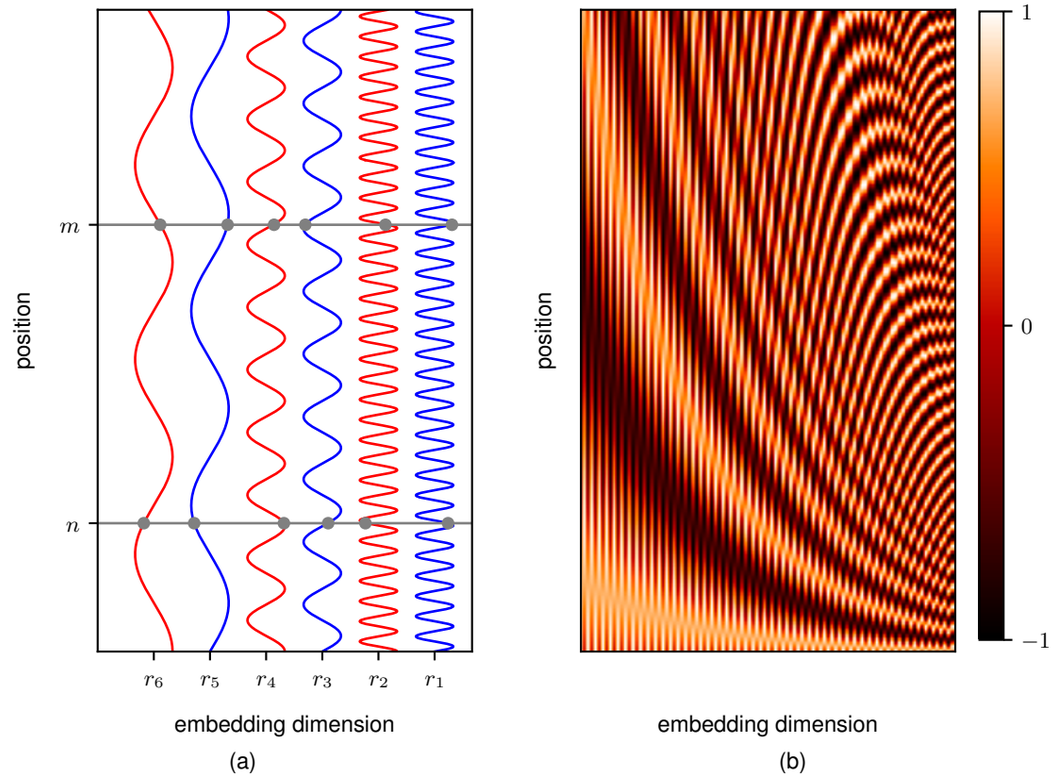
- The position & input vectors have the same dim. Two randomly chosen uncorrelated vectors tend to be nearly orthogonal in high dim.
- Associate an integer $1,2,3,\ldots$ to each position has the problem of corrupting the input vector because the length is unbounded, and vary among training sets. May not recognize new longer input sequence.

Position Encoding

- Assigning a # between $(0,1)$ to each token in the sequence does not work as the rep. is not unique for a given position (depends on sequence length).
- Is there an encoding that provides a unique rep. for each position, is bounded, generalizable to longer sequences, & capture relative positions?
- Use sinusoidal functions (Vaswani et al):

$$r_{ni} = \begin{cases} \sin\left(\frac{n}{L^{i/D}}\right), & \text{if } i \text{ is even,} \\ \cos\left(\frac{n}{L^{(i-1)/D}}\right), & \text{if } i \text{ is odd.} \end{cases}$$

- Because of the properties of sine and cosine, the encoding allows the network to attend to relative positions.



Similar to binary reps of integers, except that r_{ni} is continuous:

1 :	0	0	0	1
2 :	0	0	1	0
3 :	0	0	1	1
4 :	0	1	0	0
5 :	0	1	0	1
6 :	0	1	1	0
7 :	0	1	1	1
8 :	1	0	0	0
9 :	1	0	0	1

Transformers

Natural Language

Transformer for NLP

- A typical NLP pipeline starts with a **tokenizer** that splits the text into words or word fragments. Using words as tokens may not be ideal:
  - Some words (e.g. names, technical terms) aren't in the vocabulary.
  - How about punctuation? A question mark contains info to encode.
  - The vocabulary would need different tokens for different versions of the same word with different suffices (e.g., walk, walks, walked, walking), and there is no way to clarify these variations are related.
- Then each of the tokens is mapped to a learned **embedding**.
  - The whole vocabulary is stored in a matrix $\Omega_e \in \mathbb{R}^{D \times |\mathscr{V}|}$ where $|\mathscr{V}|$ is the vocabulary size; this vocabulary matrix is learned.
- These embeddings are passed thru a series of **transformer layers**.

Tokenization

- One approach is to use letters and punctuations as the vocabulary. But this requires the subsequent network to re-learn the relations between the very small pieces.
- A compromise is **sub-word tokenizer** such as **byte pair encoding** that greedily merges sub-strings based on their frequencies.
- Consider the following nursery rhyme:

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

- The tokens are initially just the characters & whitespace (represented by an underscore), and their frequencies given in the table.

Byte pair encoding

- At each iteration, the sub-word tokenizer looks for the most commonly occurring adjacent pair of tokens and merges them. This creates a new token & decreases the counts for the original tokens.

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

- At the second iteration, the algorithm merges e and the whitespace character_. The last character of the first token to be merged cannot be whitespace, which prevents merging across words.

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	se	a	e_	t	o	h	l	u	b	d	e	w	c	s	f	i	m	n	p	r
21	13	12	12	11	8	6	6	4	3	3	3	3	2	2	1	1	1	1	1	1

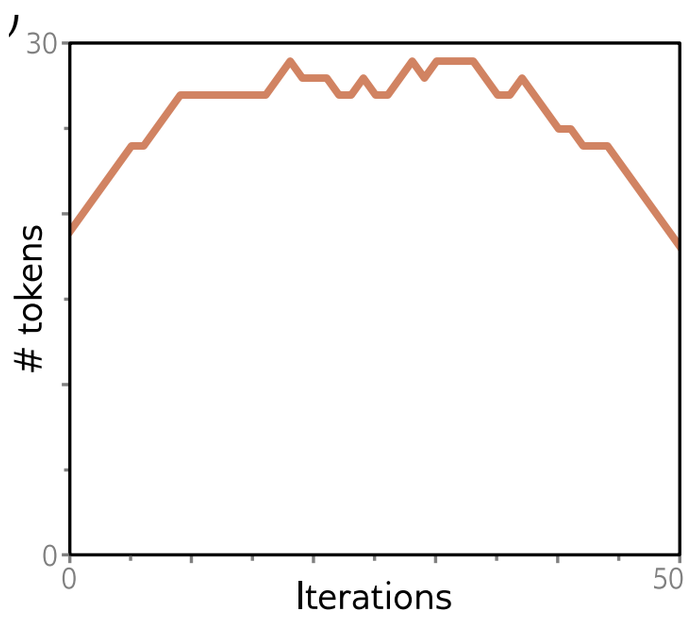
Byte pair encoding (continued)

- After 22 iterations, the tokens consist of a mix of letters, word fragments, and commonly occurring words:

see_	sea_	e	b	l	w	a	could_	hat_	he_	o	t	t_	the_	to_	u	a_	d	f	m	n	p	s	sailor_	to
7	6	4	3	3	3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1

- If we continue this process indefinitely, the tokens eventually represent the full words:

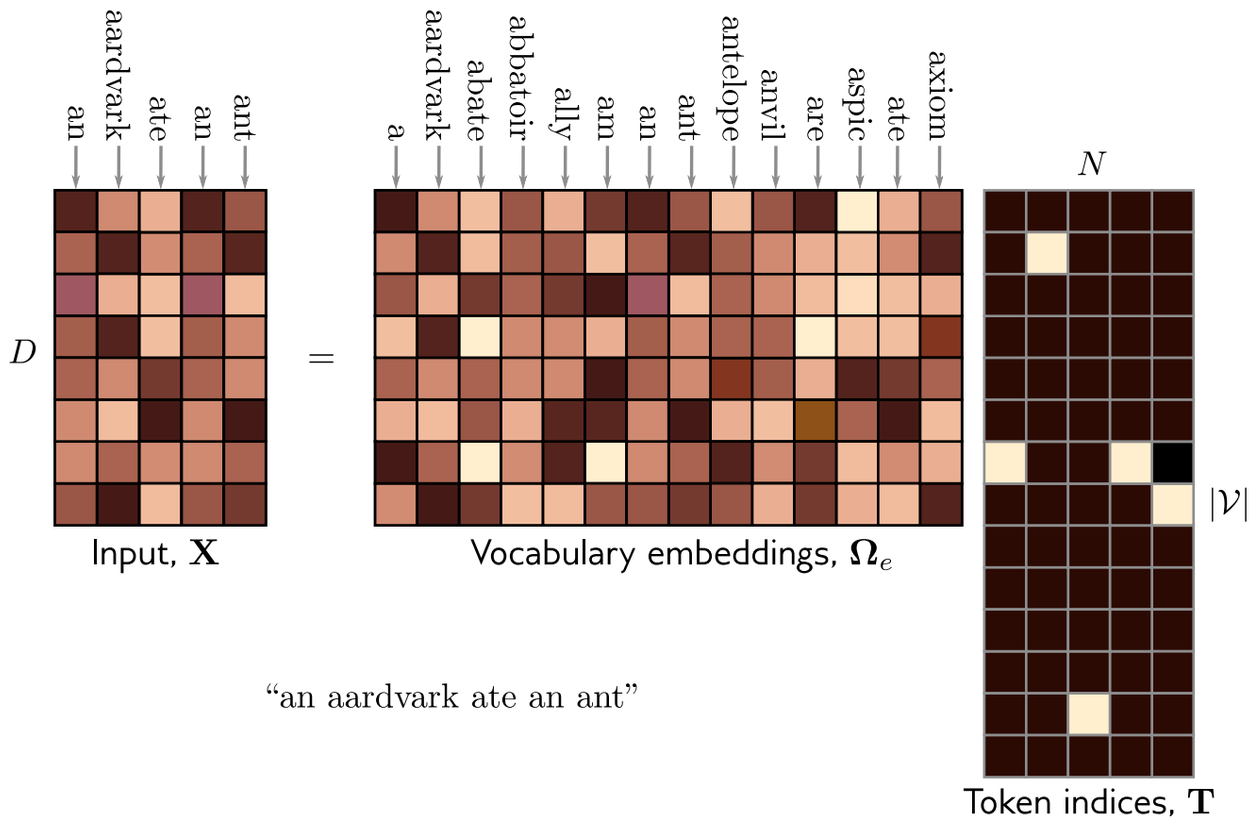
see_	sea_	could_	he_	the_	a_	all_	blue_	bottom_	but_	deep_	of_	sailor_	that_	to_	was_	went_	what_
7	6	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1



The number of tokens increases as we add word fragments to the letters and then decreases again as we merge these fragments.

Learned Embeddings

- Each token is mapped to a unique word embedding; the embeddings for the whole vocabulary are storied in a matrix $\Omega_e \in \mathbb{R}^{D \times |\mathscr{V}|}$



- The matrix Ω_e is learned like any other network parameter.

- A typical embedding size D is 1024 and a typical total vocabulary size $|\mathscr{V}|$ is $30{,}000$. Many parameters in Ω_e to learn.

Course logistics

- **Reading for this lecture:**
  - This lecture was based in part on the book by Bishop, linked on the website.