Large Language Models

Search Engines

Master Degree in Computer Engineering
Master Degree in Data Science
Academic Year 2023/2024

Nicola Ferro

Intelligent Interactive Information Access (IIIA) Hub
Department of Information Engineering
University of Padua

Outline

- Static Word Embeddings
- Feedforward Neural Networks
  - FF as Language Models
- Recurrent Neural Networks
  - RNN as Language Models
  - Encoder-decoder Models
  - Attention
- Transformers
  - Self-attention
- Decoder-only Models
  - Large Language Models (ChatGPT)
- Encoder-only Models
  - Large Language Models (BERT)

Static Word Embeddings

BoW: Sparse Representation

Coronavirus Early Symptoms

Covid Early Symptoms

	d_1	d_2	d_3	…	d_{47137}
animal					
coronavirus	**1**				
covid		**1**			
early					1
hypertension					
quarantine					
reopening					
symptoms			1		

q_1
1
1
1

q_2
1
1
1

BoW: Sparse Representation

Coronavirus Early Symptoms

Covid Early Symptoms

	d_1	d_2	d_3	…	d_{47137}
animal					
coronavirus	**1**				
covid		**1**			
early					1
hypertension					
quarantine					
reopening					
symptoms			1		

	q_1
	1
	1
	1

	q_2
	1
	1
	1

- **Representations** are **sparse** since most of their components are zero and do not appear in the query/document
  - **One-hot encoding** in Natural Language Processing is even sparser

BoW: Sparse Representation

Coronavirus Early Symptoms ≠ Covid Early Symptoms

	d_1	d_2	d_3	…	d_{47137}
animal					
coronavirus	1				
covid		1			
early					1
hypertension					
quarantine					
reopening					
symptoms			1		

	q_1
animal	
coronavirus	1
covid	
early	1
hypertension	
quarantine	
reopening	
symptoms	1

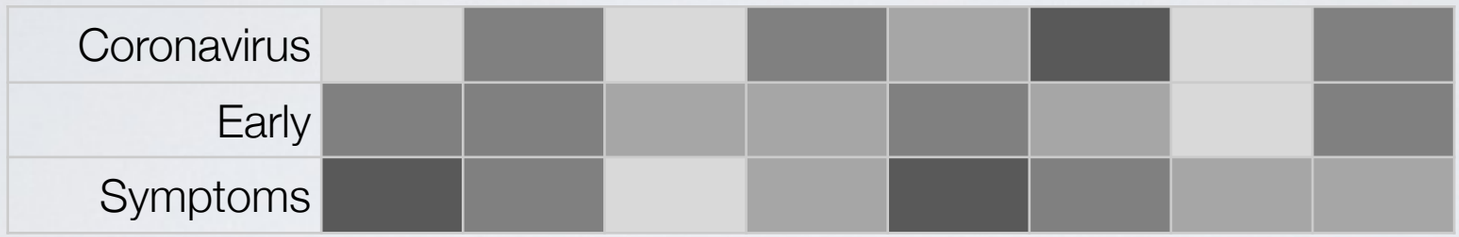
≠

	q_2
animal	
coronavirus	
covid	1
early	1
hypertension	
quarantine	
reopening	
symptoms	1

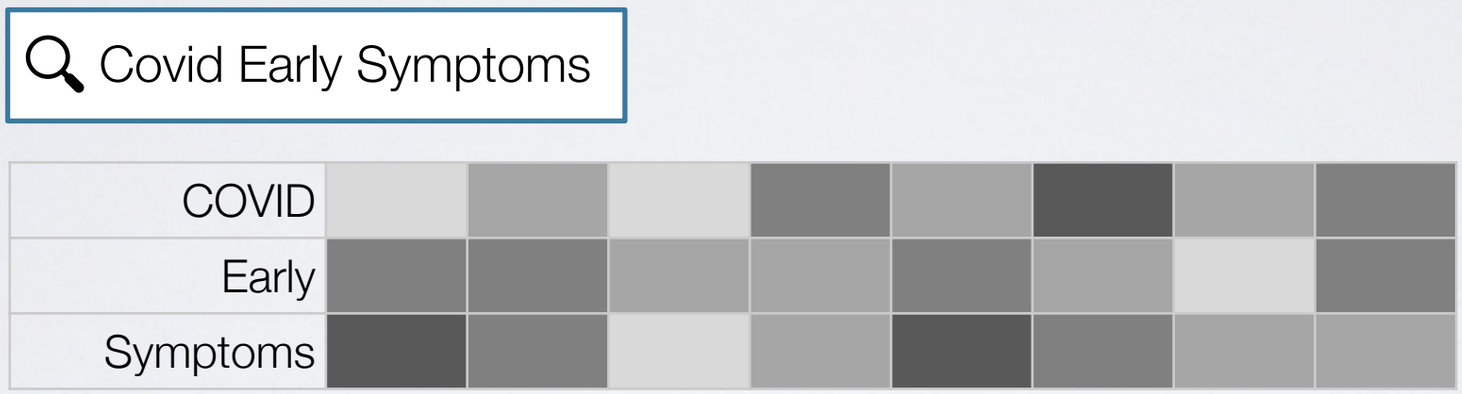
- **Representations** are **sparse** since most of their components are zero and do not appear in the query/document
  - **One-hot encoding** in Natural Language Processing is even sparser
- **Semantically close** element may end up in completely **different vector representations**

Word Embeddings: Dense Representation

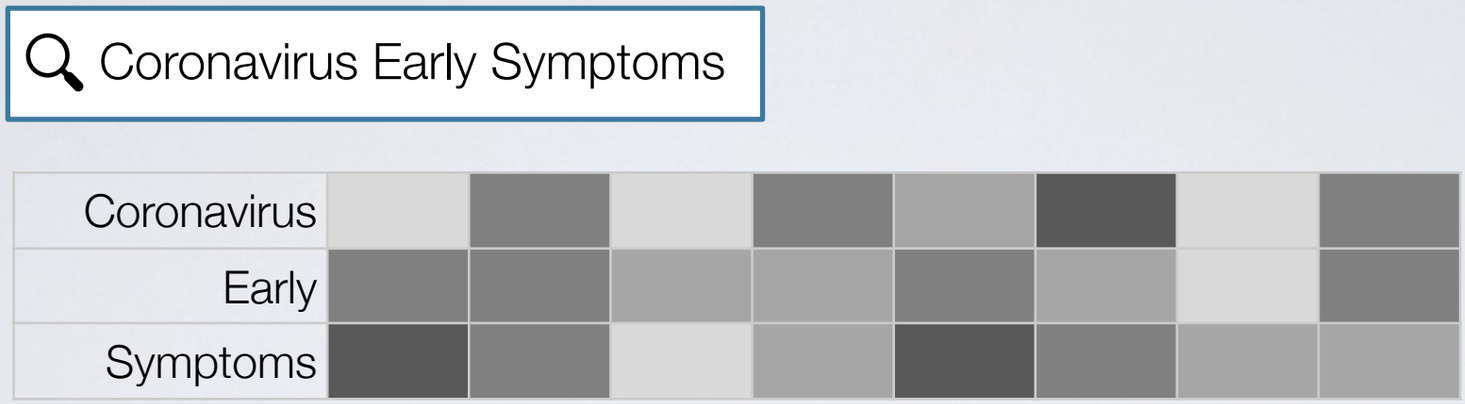
Coronavirus Early Symptoms



Covid Early Symptoms



Word Embeddings: Dense Representation



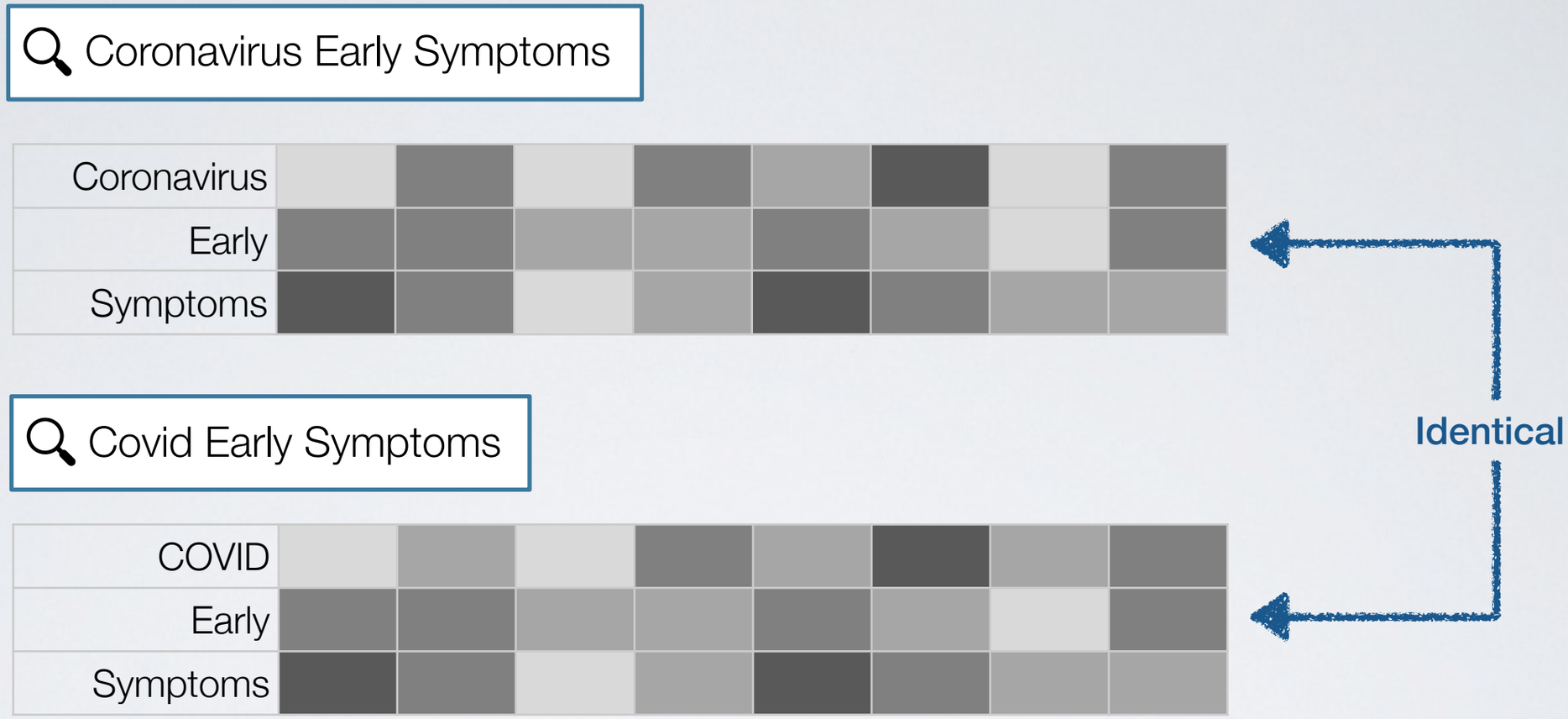
- **Representations** are **dense** since all their components contain a real value (rarely zero)
  - The dimension of the vector is fixed ahead

⇒ in sparse, dipende da # documenti in collezione

Word Embeddings: Dense Representation

- **Representations** are **dense** since all their components contain a real value (rarely zero)
  - The dimension of the vector is fixed ahead

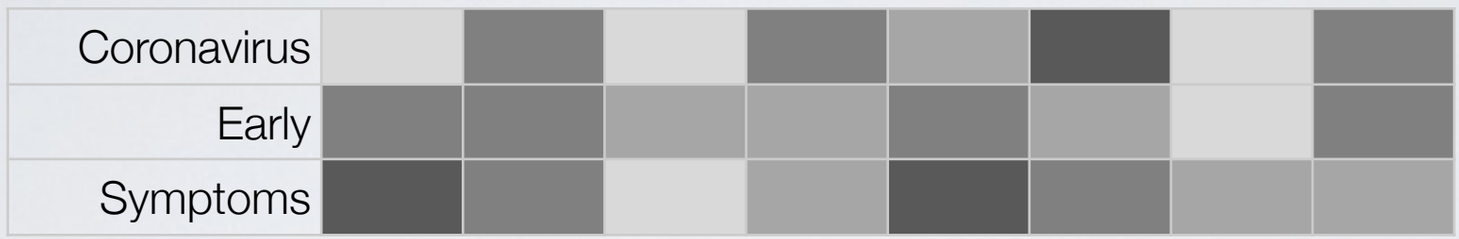
Word Embeddings: Dense Representation



- **Representations** are **dense** since all their components contain a real value (rarely zero)
  - The dimension of the vector is fixed ahead

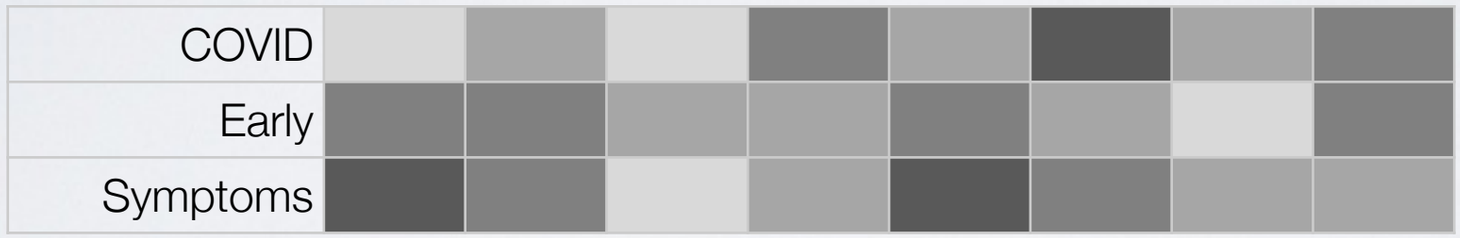
Word Embeddings: Dense Representation

Coronavirus Early Symptoms



Not identical but close

Covid Early Symptoms



- **Representations** are **dense** since all their components contain a real value (rarely zero)
  - The dimension of the vector is fixed ahead
- **Semantically close** element end up in **close vector representations**

Word Embeddings and Distributional Semantics

- A **word** is represented as a real-valued **vector**, called **embedding**
  - Word embeddings depend on notion of **semantic similarity**
- When a word appears in a text, its **context** is the set of words that appear nearby (within a **fixed-size window**)
  - Based on the Distributional Hypothesis
- Use many contexts of the same word to build up its representation
  - **Banks** are also of interest in navigation, where the term can refer either to a barrier island or a submerged plateau, such as an ocean **bank**
  - Europe needs unified **banking** regulation to replace the hodgepodge...
  - The plane **banks** as if to return to the airport

Joos, M. (1950). Description of Language Design. *The Journal of the Acoustical Society of America (JASA)*, 22(6):701–707.
Harris, Z. S. (1954). Distributional Structure. *Word*, 10(2–3):146–162.
Firth, J. R. (1957). *Studies in linguistic analysis. Special Volume of Philological Society*. Blackwell, Oxford, UK.

Word Embeddings and Distributional Semantics

Distributional Hypothesis

"The linguist's 'meaning' of a morpheme. . . is by definition the set of conditional probabilities of its occurrence in context with all other morphemes" [Joos, 1950]
"Words that occur in the same contexts tend to have similar meanings" [Harris, 1954]
"A word is characterized by the company it keeps" [Firth, 1957]

- A **word** is represented as a real-valued **vector**, called **embedding**
  - Word embeddings depend on notion of **semantic similarity**
- When a word appears in a text, its **context** is the set of words that appear nearby (within a **fixed-size window**)
  - Based on the Distributional Hypothesis
- Use many contexts of the same word to build up its representation
  - **Banks** are also of interest in navigation, where the term can refer either to a barrier island or a submerged plateau, such as an ocean **bank**
  - Europe needs unified **banking** regulation to replace the hodgepodge...
  - The plane **banks** as if to return to the airport

Joos, M. (1950). Description of Language Design. *The Journal of the Acoustical Society of America (JASA)*, 22(6):701–707.
Harris, Z. S. (1954). Distributional Structure. *Word*, 10(2–3):146–162.
Firth, J. R. (1957). *Studies in linguistic analysis. Special Volume of Philological Society*. Blackwell, Oxford, UK.

Word Embeddings: Example

- Consider encountering a new word:
  balåstraykan
  - [C1] A bottle of **balåstraykan** is on the table
  - [C2] Everybody likes **balåstraykan**
  - [C3] Don't have **balåstraykan** before you drive
  - [C4] We make **balåstraykan** out of corn
- What is a balåstraykan?

$$\text{balåstraykan} = \begin{bmatrix} 0.21179 \\ 0.78311 \\ -0.17114 \\ 0.56921 \\ -0.81557 \\ \dots \end{bmatrix}$$

The representation is called **latent** because the dimensions of the embedding do not (necessarily) match the human meanings/ senses/contexts.

Word Embeddings: Example

- Consider encountering a new word:
  balåstraykan
  - [C1] A bottle of **balåstraykan** is on the table
  - [C2] Everybody likes **balåstraykan**
  - [C3] Don't have **balåstraykan** before you drive
  - [C4] We make **balåstraykan** out of corn
- What is a balåstraykan?

$$\text{balåstraykan} = \begin{bmatrix} 0.21179 \\ 0.78311 \\ -0.17114 \\ 0.56921 \\ -0.81557 \\ \dots \end{bmatrix}$$

The representation is called **latent** because the dimensions of the embedding do not (necessarily) match the human meanings/ senses/contexts.

terms	context				similarity
	C1	C2	C3	C4	
balåstraykan	1	1	1	1	
loud	0	0	0	0	**0**
motor oil	1	0	0	1	**2**
tortillas	0	1	0	1	**2**
choices	0	1	0	0	**1**
wine	1	1	1	0	**3**

Word Embeddings

- A word embedding is a (latent) **vector space** in which we can do math
  - the dimension of the space is much smaller than the vocabulary size $\mathbb{R}^n, n << |\mathcal{V}|$, typically in the range of 50-1,000 elements
- A common operation is to measure the **distance** between two points
  - In a word embedding, measuring the **distance** is like measuring **similarity of words**
  - Standard measure is the **cosine similarity**
- Two main types of embeddings *da sparse a dense representations*
  - **Static word embeddings**: they compute **global representations** of the terms, i.e. a **single** fixed embedding (**vector**) **for each term** in the vocabulary and map terms with multiple senses into an **average** or most common **sense representation**. Most common are `word2vec`, `fasttext`, and `GloVe`
  - **Contextualised word embeddings**: compute **local representations** of the terms, which depend on the other tokens used together with a given term, i.e. its **context**; **each term** in the vocabulary is associated with a **different vector** every time it appears in a document, **depending on the surrounding tokens**. Most common are `BERT` (Bidirectional Encoder Representations from Transformers), `RoBERTa` (Robustly Optimized BERT Approach), and `GPT` (Generative Pre-Training)

Word2vec: Intuition

c w

c w

- We do not need explicit labels to learn but we can adopt a **self-supervised approach** [Bengio et al., 2003], leveraging a **large corpus of real text**
- Every **word** in a fixed vocabulary is represented by a **vector** $\mathbf{w}$
- Go through each position in the text, which has a **center word** w and **context words** c
- Use the **similarity** $\mathbf{c} \cdot \mathbf{w}$ of the word vectors to calculate the **probability of** w **given** c (continuous bag-of-word) or viceversa **probability of** c **given** w (skip-gram)
- Download at: `https://code.google.com/archive/p/word2vec/`

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K., editors, *Proc. 26th Annual Conference on Neural Information Processing Systems (NeurIPS 2013)*, pages 3111–3119. `https://proceedings.neurips.cc/paper_files/paper/2013`.
Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv.org, Computation and Language (cs.CL)*, arXiv:1301.3781.
Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research (JMLR)*, 3:1137–1155.

Word2vec: Intuition

ci servono molti esempi

The intuition of `word2vec` is that instead of counting how often each word c occurs near to a target word w, we instead train a classifier on a binary prediction task: "Is word c likely to show up near w?" We do not actually care about this prediction task; instead we take the learned classifier weights as the word embeddings

- We do not need explicit labels to learn but we can adopt a **self-supervised approach** [Bengio et al., 2003], leveraging a **large corpus of real text**
- Every **word** in a fixed vocabulary is represented by a **vector** $\mathbf{w}$
- Go through each position in the text, which has a **center word** w and **context words** c
- Use the **similarity** $\mathbf{c} \cdot \mathbf{w}$ of the word vectors to calculate the **probability of** w **given** c (continuous bag-of-word) or viceversa **probability of** c **given** w (skip-gram)
- Download at: `https://code.google.com/archive/p/word2vec/`

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K., editors, *Proc. 26th Annual Conference on Neural Information Processing Systems (NeurIPS 2013)*, pages 3111–3119. `https://proceedings.neurips.cc/paper_files/paper/2013`.
Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv.org, Computation and Language (cs.CL)*, arXiv:1301.3781.
Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research (JMLR)*, 3:1137–1155.

Skip-Gram with Negative Sampling Algorithm

... lemon, a [tablespoon of apricot jam, a] pinch ...
c1 c2 w c3 c4


(apricot, jam) $\mathbb{P}(+|w,c)$

(apricot, aardvark) $\mathbb{P}(-|w,c) = 1 - \mathbb{P}(+|w,c)$


- Treat the target word w and a neighboring context word c as **positive examples**
- Randomly sample other words c in the lexicon to get **negative samples**
- Use **logistic regression** to train a classifier to distinguish those two cases
- Use the **learned weights** as the **embeddings**

Jurafsky, D. and Martin, J. H. (2024). *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. https://web.stanford.edu/~jurafsky/slp3/, 3rd, draft edition.

How to Estimate Probabilities?

- Two vectors are **similar** if they have a high **dot product**
  - Cosine is just a normalized dot product

$$\text{similarity}(\mathbf{w}, \mathbf{c}) \sim \mathbf{c} \cdot \mathbf{w}$$

- We need to normalize to get a probability by using the **sigmoid function** from logistic regression

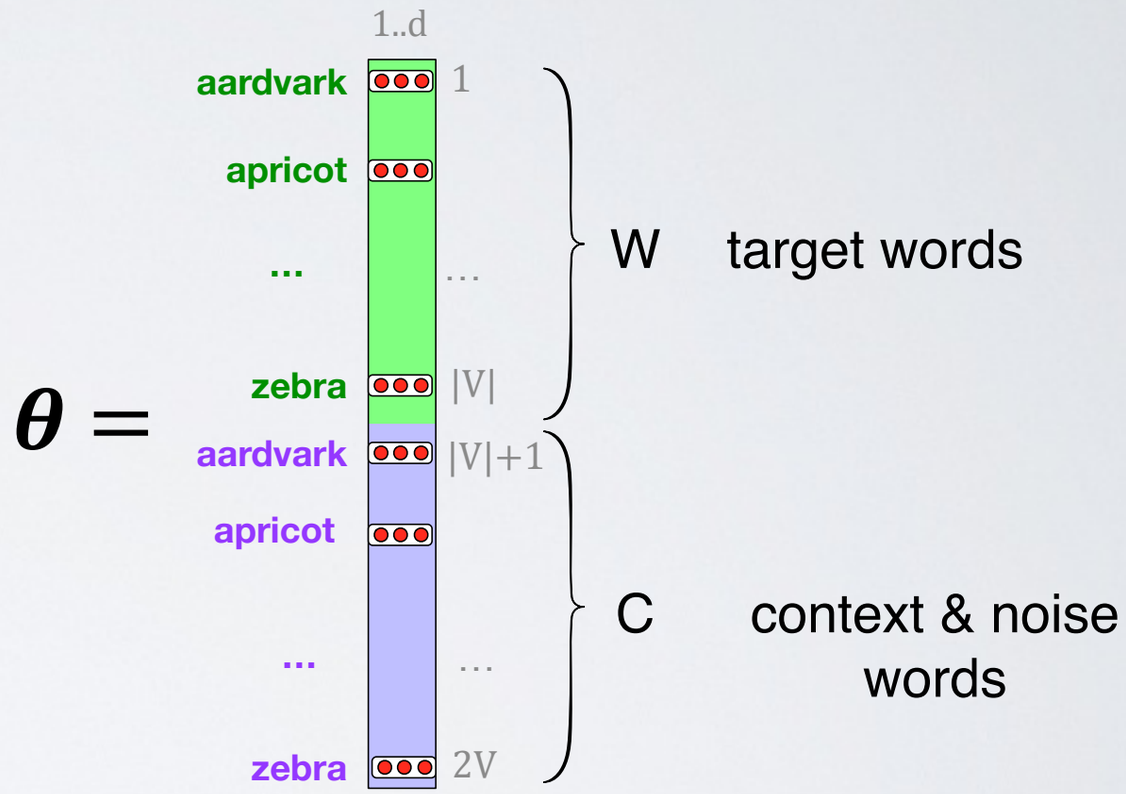
$$\mathbb{P}(+|w, c) = \sigma(\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{c} \cdot \mathbf{w})}$$

$$\mathbb{P}(-|w, c) = 1 - \mathbb{P}(+|w, c) = \sigma(-\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1 + \exp(\mathbf{c} \cdot \mathbf{w})}$$

How to Estimate Probabilities for the Whole Context Window?

$$\mathbb{P}(+|w, c_{1:L}) = \prod_{i=1}^{L} \sigma(-\mathbf{c_i} \cdot \mathbf{w})$$

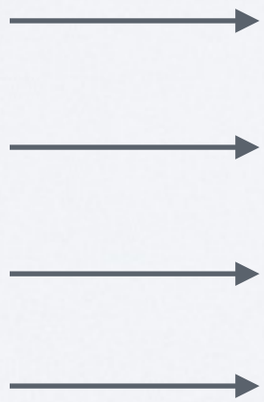
- Assume independence of the words
  - Skip-gram trains a probabilistic classifier that, given a test target word w and its context window of L words $c_{1:L}$, assigns a probability based on **how similar this context window is to the target word**
  - To compute this probability, we need **embeddings** for each target word and context word in the vocabulary V
  - Skip-gram stores two embeddings of size d for each word, one for the **word as a target**, and one for the **word** considered **as context**. Thus the parameters we need to learn are two matrices $\mathbf{W}$ and $\mathbf{C}$, each containing an embedding for every one of the $|V|$ words



Skip-Gram Setup

```
... lemon, a [tablespoon of apricot jam, a] pinch ...
              c1      c2     w     c3   c4
```

- The **embeddings** of the $|V|$ words are **randomly initialized**
- For each positive example, we decide how many negative examples k to take
  - the noise words are chosen according to their **weighted unigram frequency**, attributing more probability to rare noise words



Skip-Gram Setup

```
... lemon, a [tablespoon of apricot jam, a] pinch ...
              c1      c2     w     c3   c4
```

- The **embeddings** of the $|V|$ words are **randomly initialized**
- For each positive example, we decide how many negative examples k to take
  - the noise words are chosen according to their **weighted unigram frequency**, attributing more probability to rare noise words

Positive Examples +	
w	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Skip-Gram Setup

```
... lemon, a [tablespoon of apricot jam, a] pinch ...
              c1      c2    w     c3   c4
```

- The **embeddings** of the $|V|$ words are **randomly initialized**
- For each positive example, we decide how many negative examples k to take
  - the noise words are chosen according to their **weighted unigram frequency**, attributing more probability to rare noise words

Positive Examples +	
w	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Negative Examples -			
w	c	w	c
apricot	aardvark	apricot	seven
apricot	where	apricot	dear
apricot	Tolstoy	apricot	matrix
apricot	coaxial	apricot	if

Skip-Gram: The Loss Function

$$L_{CE} = -\log\left[\mathbb{P}(+|w, c_{pos})\prod_{i=1}^{k}\mathbb{P}(-|w, c_{neg_i})\right] = -\left[\log\left(\sigma(\mathbf{c}_{pos}\cdot\mathbf{w})\right) + \sum_{i=1}^{k}\log\left(\sigma(-\mathbf{c}_{neg_i}\cdot\mathbf{w})\right)\right]$$

- The goal of the learning algorithm is to adjust the embeddings to
  - **Maximize** the similarity of the target word, context word pairs (w, c_{pos}) drawn from the **positive examples**
  - **Minimize** the similarity of the (w, c_{neg}) pairs from the **negative examples**
- Therefore, for one word/context pair, we aim at minimizing the above **cross-entropy loss function** (hence the negative sign)
  - We use **stochastic gradient descent**

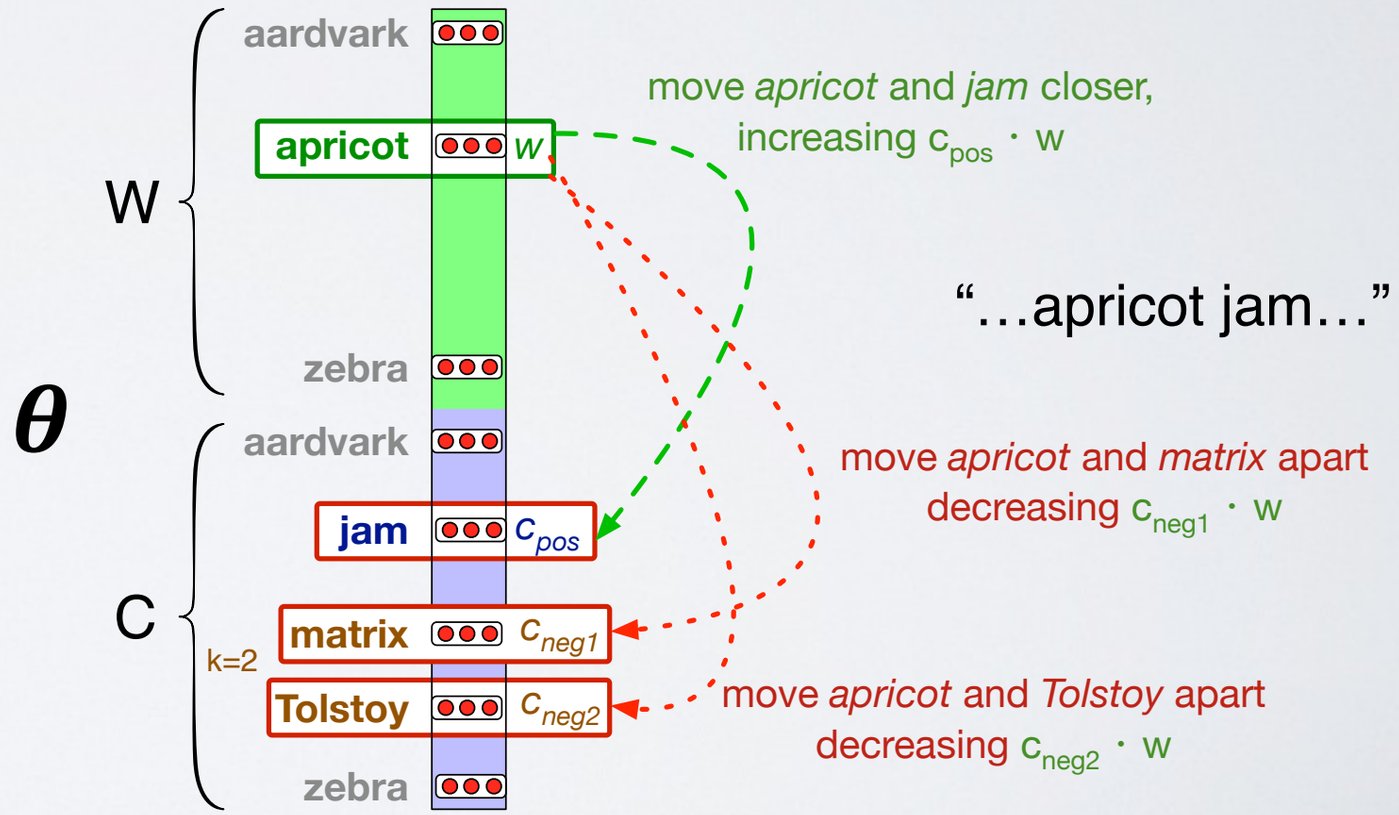
Skip-Gram: The Gradient Descent

$$\mathbf{c}_{pos}^{t+1} = \mathbf{c}_{pos}^{t} - \eta \left[\sigma\left(\mathbf{c}_{pos}^{t} \cdot \mathbf{w}^{t}\right) - 1\right] \mathbf{w}^{t}$$

$$\mathbf{c}_{neg}^{t+1} = \mathbf{c}_{neg}^{t} - \eta \left[\sigma\left(\mathbf{c}_{neg}^{t} \cdot \mathbf{w}^{t}\right)\right] \mathbf{w}^{t}$$

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta \left[\left[\sigma\left(\mathbf{c}_{pos}^{t} \cdot \mathbf{w}^{t}\right) - 1\right] \mathbf{c}_{pos}^{t} + \sum_{i=1}^{k} \left[\sigma\left(\mathbf{c}_{neg_i}^{t} \cdot \mathbf{w}^{t}\right)\right] \mathbf{c}_{neg_i}^{t} \right]$$

- The skip-gram model tries to shift embeddings so the target embeddings (here for `apricot`) are closer to (have a higher dot product with) context embeddings for nearby words (here `jam`) and further from (lower dot product with) context embeddings for noise words that don't occur nearby (here `Tolstoy` and `matrix`)
- η is the learning rate

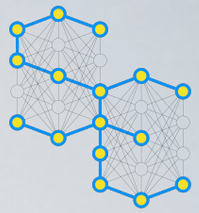


Skip-Gram: Two Sets of Embeddings

- Skip-Gram with Negative Sampling learns two sets of embeddings for each word
  - The target embedding matrix $\mathbf{W}$
  - The context embedding matrix $\mathbf{C}$

- What to pick as **final embedding of a word**?
  - We just keep the embedding of the target matrix $\mathbf{w}_i$
  - Or, we add them $\mathbf{w}_i + \mathbf{c}_i$

Other Static Word Embeddings: GloVe and fastText



- **GloVe (Global Vectors)** [Pennington et al., 2014] is based on ratios of probabilities from the word-word co-occurrence matrix.
  - It is trained only on the nonzero elements in a word-word co-occurrence matrix, rather than on the entire sparse matrix or on individual context windows in a large corpus
  - Download at: `http://nlp.stanford.edu/projects/glove/`
- **fastText** [Bojanowski et al., 2017] addresses some limitations of `word2vec` which cannot deal with **out-of-vocabulary words** and suffers from **word sparsity**, especially in morphologically-rich languages
  - `fastText` uses **sub-word models**, representing each **word as itself** plus a bag of constituent **n-grams**, with special boundary symbols < and > added to each word.
    - For example, with n = 3 the word `where` would be represented by the sequence `<where>` plus the character n-grams: `<wh`, `whe`, `her`, `ere`, `re>`
  - Then a skip-gram embedding is learned for each constituent n-gram, and a **word** is represented by the **sum of all of the embeddings** of its constituent n-grams
  - Out-of-vocabulary words are then represented by the sum of the constituent n-grams
  - Download at: `http://www.fasttext.cc/` (pre-trained for 157 languages)

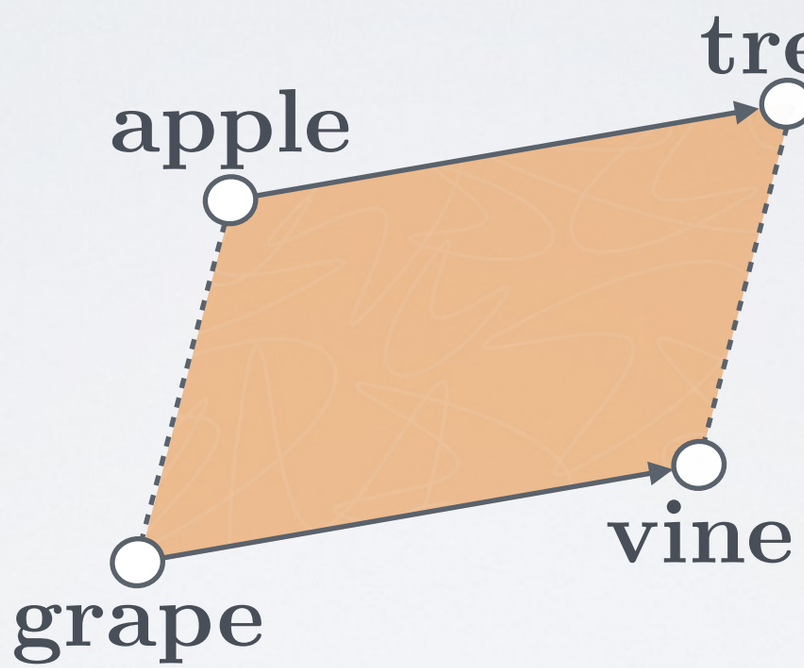
Pennington, J., Socher, R., and Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1532–1543. Association for Computational Linguistics, USA.
Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics (TACL)*, 5:135–146.

Properties of Embeddings: Window Size

- **Small windows** ($L = \pm 2$): nearest words are **syntactically similar words** in same taxonomy
  - *Hogwarts* nearest neighbors are other fictional schools *Sunnydale*, *Evernight*, *Blandings*
- **Large windows** ($L = \pm 5$): nearest words are related words in **same semantic field**
  - *Hogwarts* nearest neighbors are Harry Potter world: *Dumbledore*, *half-blood*, *Malfoy*

Properties of Embeddings: Analogical Relations

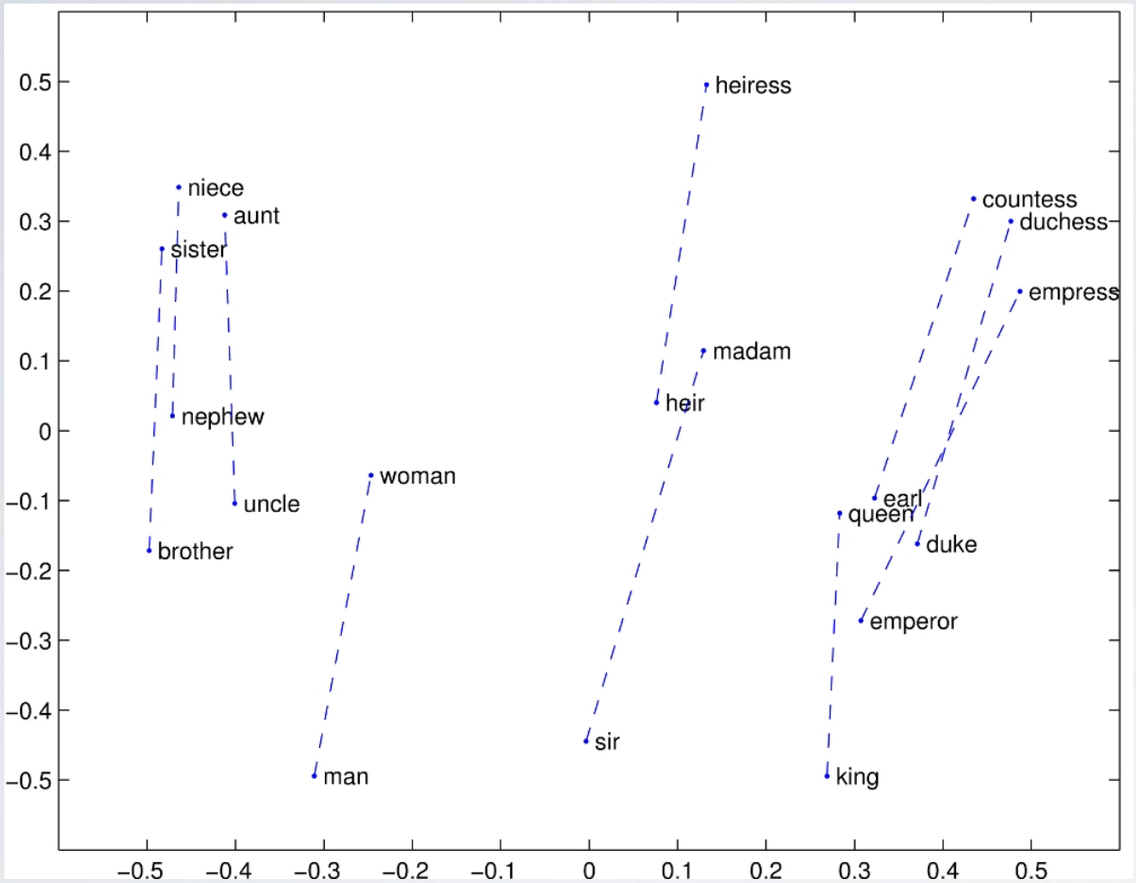
- The classic **parallelogram model of analogical reasoning**
  - To solve: "`apple` is to `tree` as `grape` is to `_____`"
  - Add $(\mathbf{tree} - \mathbf{apple})$ to $\mathbf{grape}$ to get $\mathbf{vine}$



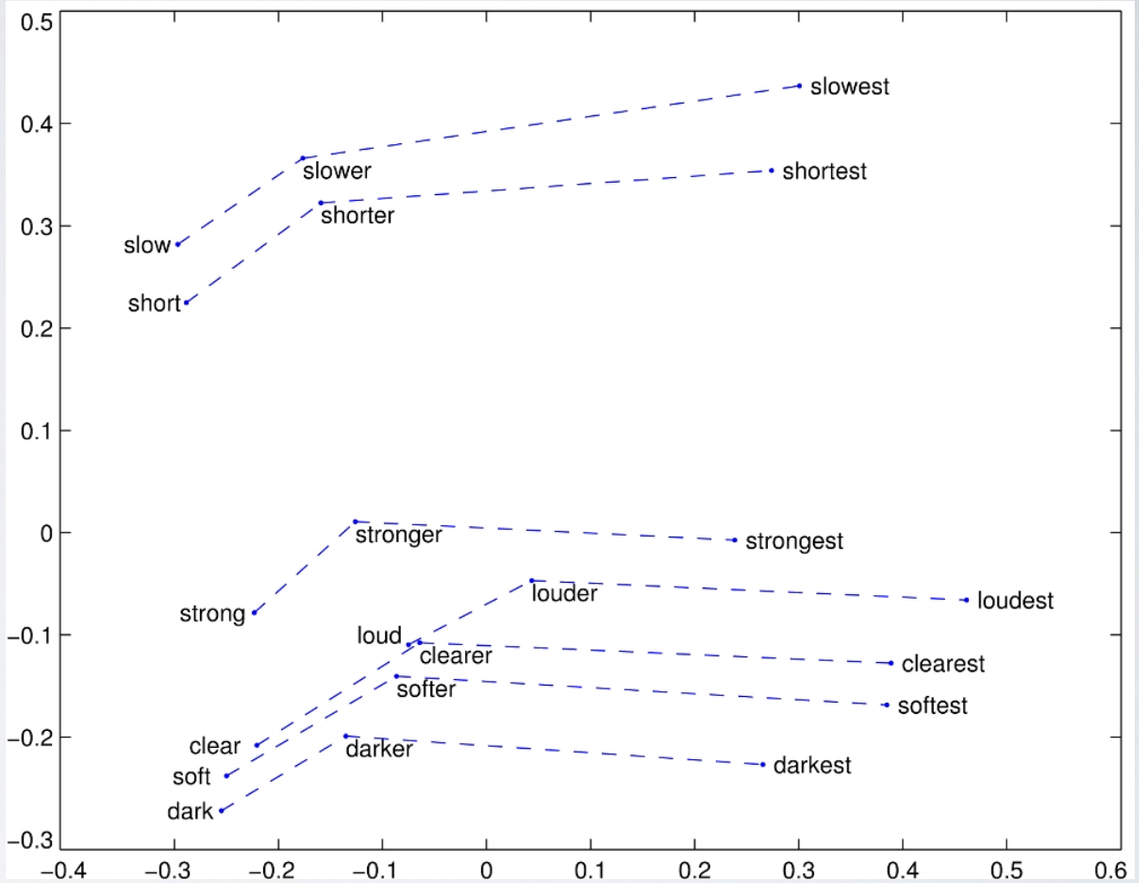
$$a : a* \;::\; b : b* \;\rightarrow\; \hat{\mathbf{b}}* = \underset{\mathbf{x}}{\operatorname{argmin}}\ \operatorname{distance}\big(\mathbf{x}, \mathbf{b} + (\mathbf{a}^* - \mathbf{a})\big)$$

Rumelhart, D. E. and Abrahamson, A. A. (1973). A Model for Analogical Reasoning. *Cognitive Psychology*, 5(1):1–28.

Properties of Embeddings: Analogical Relations



(a)

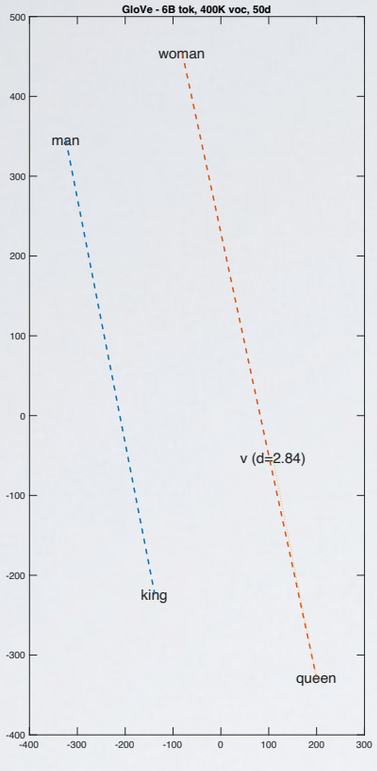


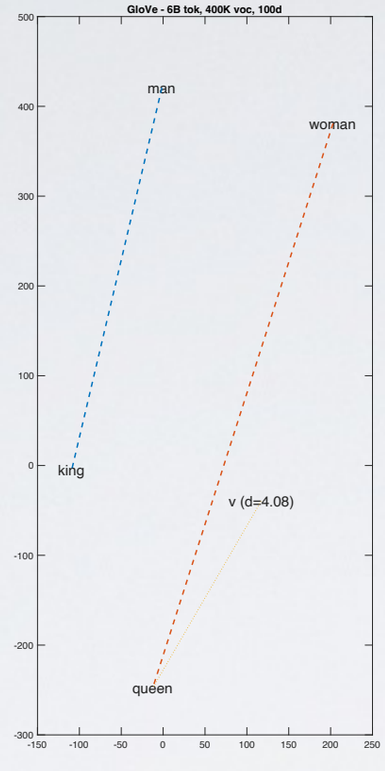
(b)

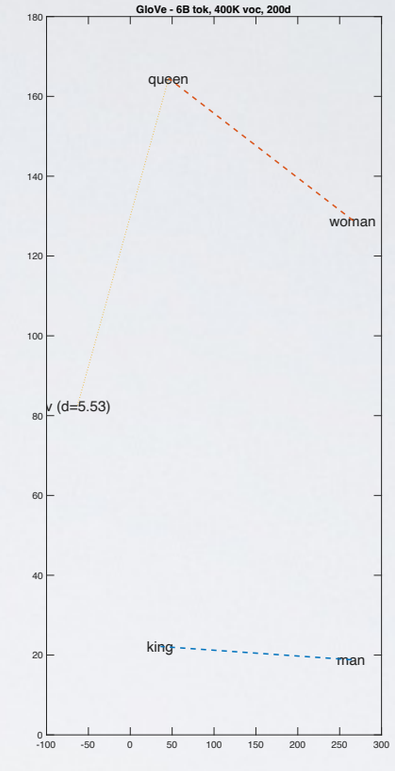
- Relational properties of the GloVe vector space, shown by projecting vectors onto two dimensions
  - (a) **king** – **man** + **woman** is close to **queen**
  - (b) offsets seem to capture comparative and superlative morphology
- It only seems to work for frequent words, small distances and certain relations (relating countries to capitals, or parts of speech), but not others

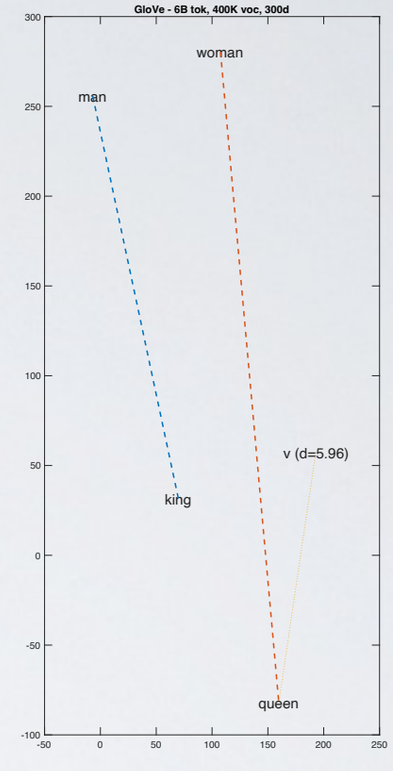
Jurafsky, D. and Martin, J. H. (2024). *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. https://web.stanford.edu/~jurafsky/slp3/, 3rd, draft edition.

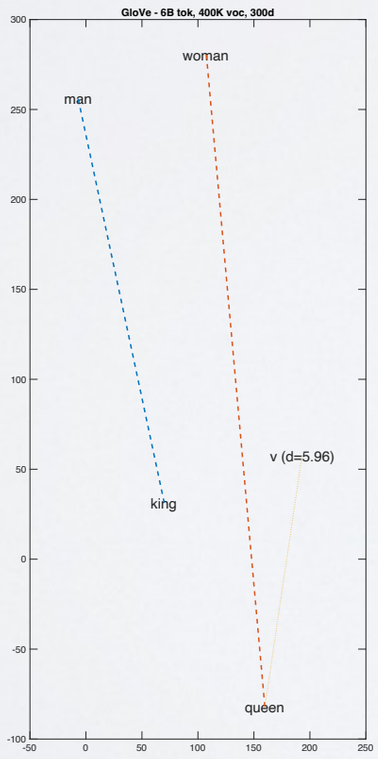
Glove Hands-on

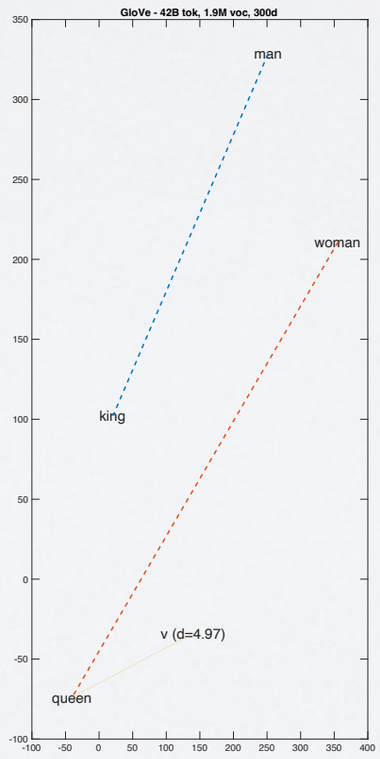


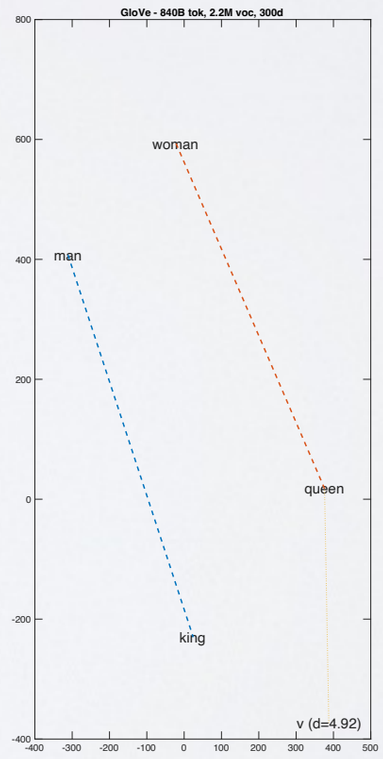




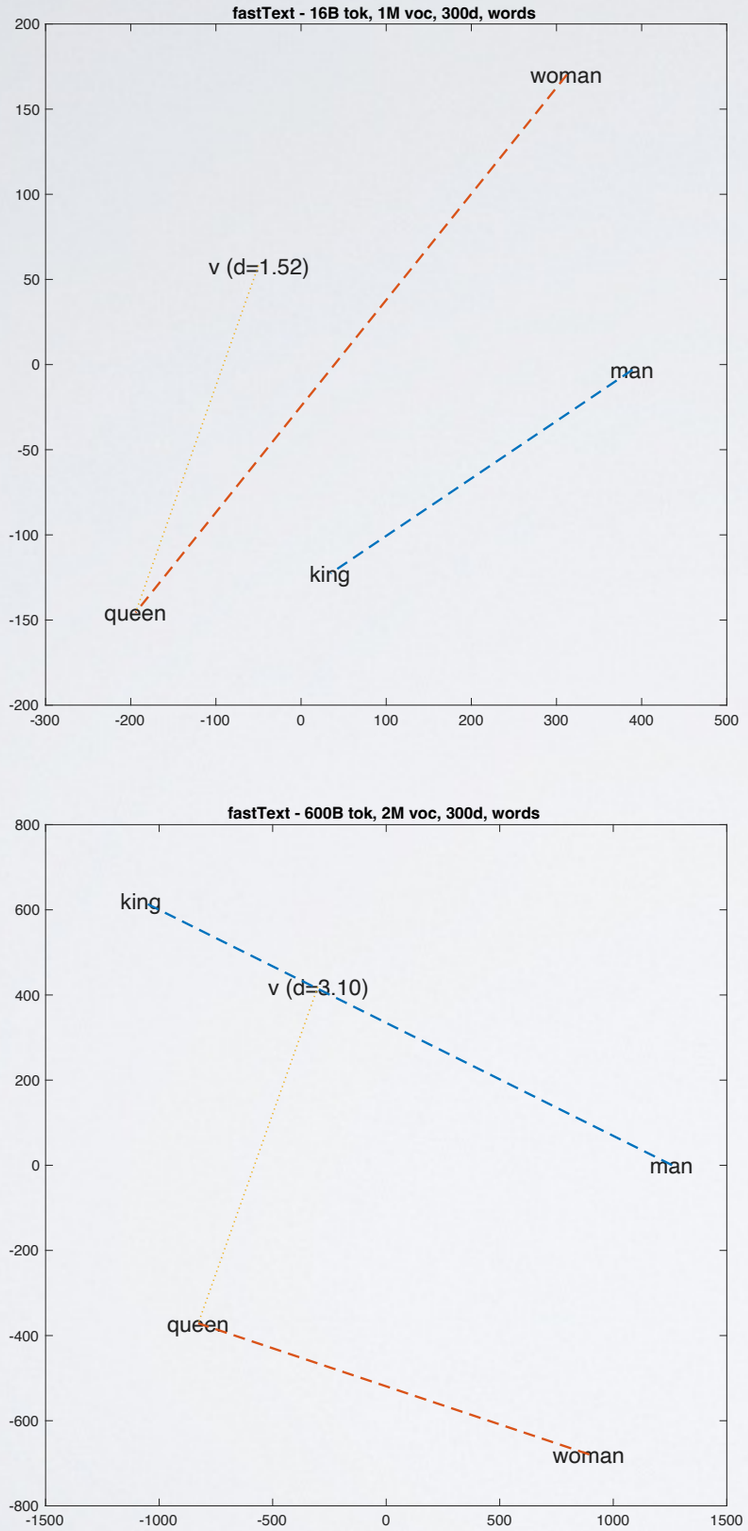


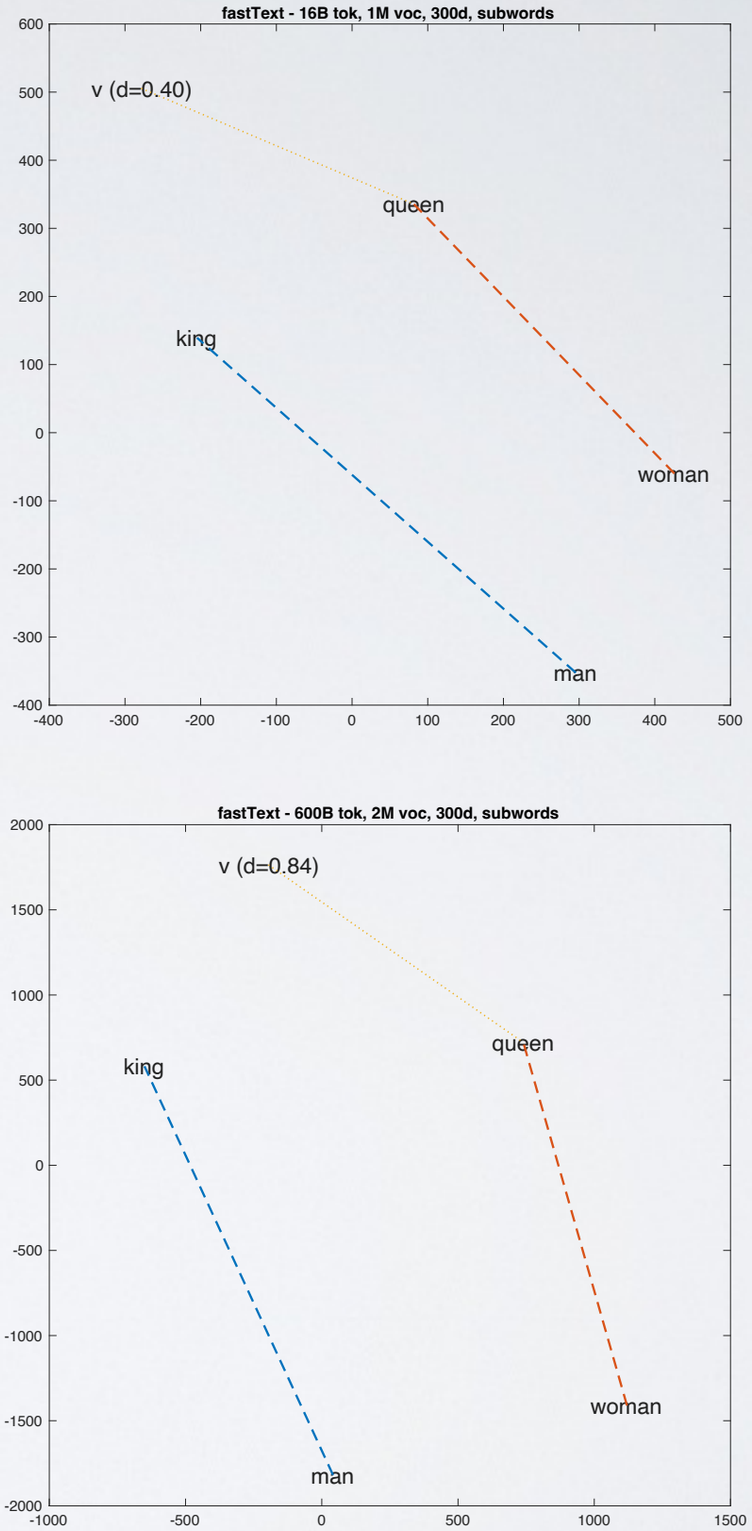




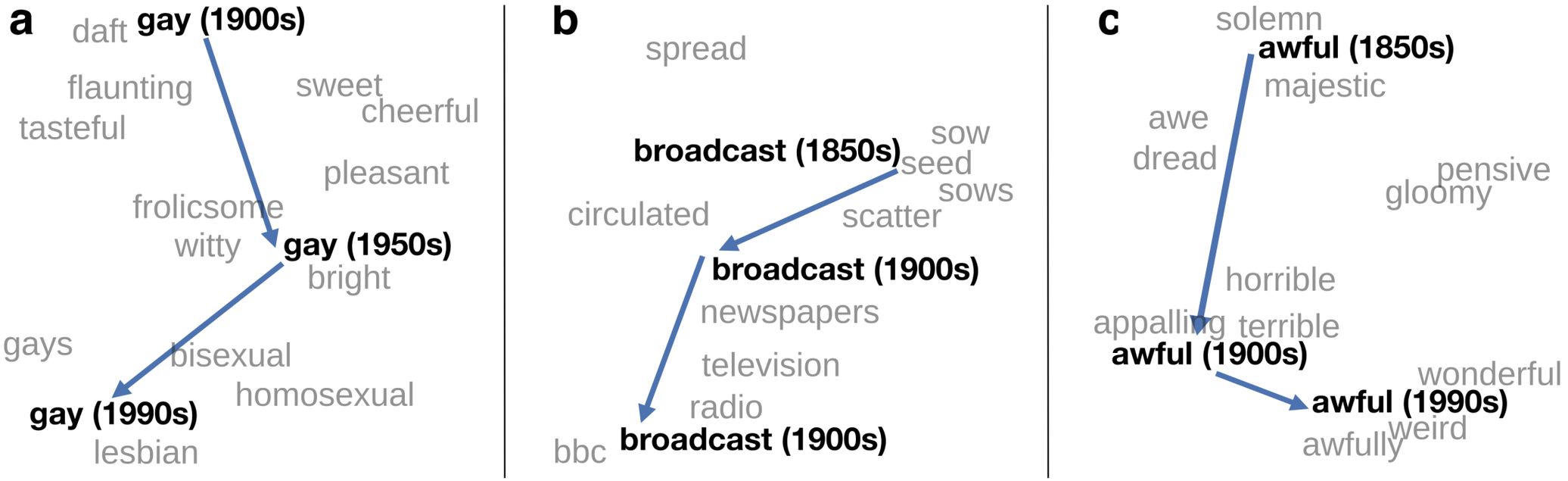


FastText Hands-on





Static Embeddings and Historical Semantics



- Visualization of changes in meaning in English words over the last two centuries, computed by building separate embedding spaces for each decade from historical corpora like Google N-grams and the Corpus of Historical American English
  - the changes in the word *gay* from meanings related to “cheerful” or “frolicsome” to referring to homosexuality
  - The development of the modern “transmission” sense of *broadcast* from its original sense of sowing seeds
  - The pejoration of the word *awful* as it shifted from meaning “full of awe” to meaning “terrible or appalling”

Hamilton, W. L., Leskovec, J., and Jurafsky, D. (2016). Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. In Erk, K. and Smith, N. A., editors, *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*, pages 1489–1501. Association for Computational Linguistics, USA.

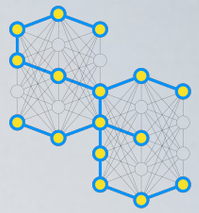
Embeddings Reflect Cultural Bias

- Embeddings also reproduce the **implicit biases** and **stereotypes** that were latent in the text
  - They do not just reflect the statistics of their input, but also bias **amplify bias**; gendered terms become more gendered in embedding space than they amplification were in the input text statistics

- Ask “Paris : France :: Tokyo : x”
  - x = Japan
- Ask “father : doctor :: mother : x”
  - x = nurse
- Ask “man : computer programmer :: woman : x”
  - x = homemaker

Bolukbasi, T., Chang, K.-W., Zou, J. Y., Saligrama, V., and Kalai, A. T. (2016). Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Proc. 29th Annual Conference on Neural Information Processing Systems (NeurIPS 2016)*. https://proceedings.neurips.cc/paper_files/paper/2016.

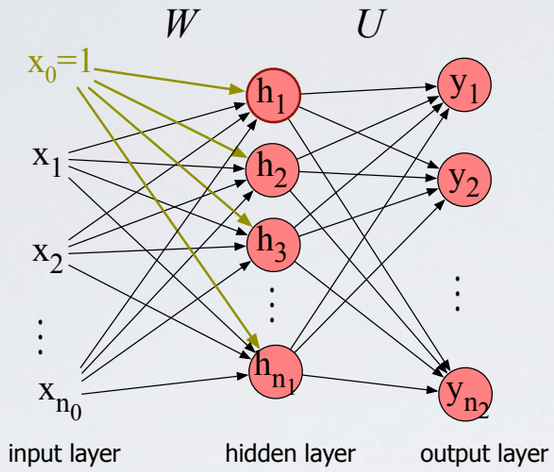
Limitations of Static Word Embeddings



- The **ordering** & l**ocal context** is important: “not good” vs. “not bad”
- Creating bi-gram or tri-gram embeddings is **not feasible**
  - Sparsity problem
  - Not enough training data: no connection between “quite good” and “very good”
- Word2vec and other approaches always **map one word to one vector**
  - This is good for analysis purposes and constrained resource environments
- There is **no contextualization** in the vector after training
  - The vector of a word **does not change** based on other words in the sentence around it
  - Words with multiple senses depending on context are squashed together in an **average or most common sense** in the training data
- But many words do have **many senses** based on the context
  - E.g., bank, jaguar, glass, leg, make, …

Feedforward Neural Networks

Feedforward Neural Networks



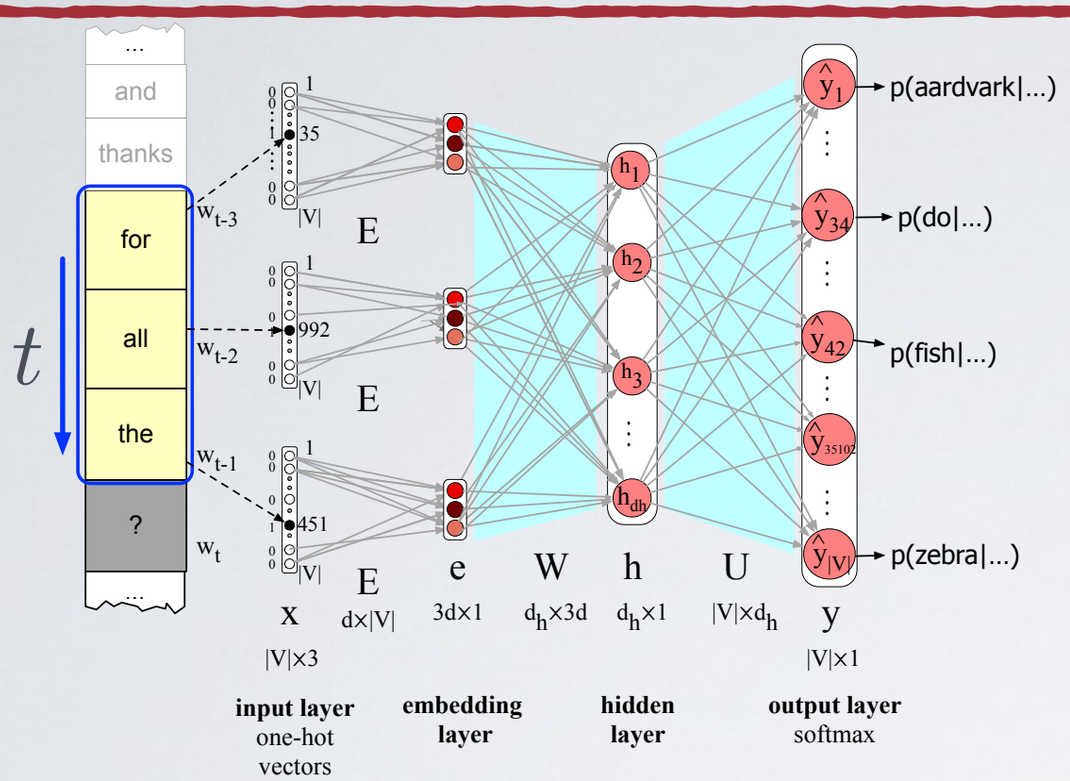
$$\mathbf{h} = \mathrm{g}(\mathbf{W}\mathbf{x})$$

$$\mathbf{z} = \mathbf{U}\mathbf{h}$$

$$\mathbf{y} = \mathrm{softmax}(\mathbf{z})$$

- A **feedforward network** is a multilayer network in which the units are connected with **no cycles**
  - The core of the neural network is the **hidden layer** $\mathbf{h}$ which forms a **representation** of the input
  - The role of the **output layer** is to take this new representation $\mathbf{h}$ and compute a final output; in case of a classification decision we need to turn it into a probability distribution
- g is the **activation function**, which introduces the **non-linearity** in the neural network
  - Sigmoid $\sigma(a) = \dfrac{1}{1+\exp(-a)} \in [0,1]$
  - ReLU (Rectified Linear Unit) $\mathrm{ReLU}(a) = \max(0, a)$
  - Tanh $\tanh(a) = \dfrac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} \in [-1,1]$
- The `softmax` function normalize a vector of real values into a **probability distribution** $\mathrm{softmax}(\mathbf{z}_i) = \dfrac{\exp(\mathbf{z}_i)}{\sum_{j=1}^{d}\exp(\mathbf{z}_j)}$
- A **neural network classifier** with one hidden layer builds a vector $\mathbf{h}$ which is a **representation of the input** and then runs standard **multinomial logistic regression** on the features that the network develops in $\mathbf{h}$
  - Rather than forming the features by feature templates, the layers of **the network induce the feature representations** themselves

Feedforward Neural Language Models



$$\mathbf{e} = [\mathbf{E}\mathbf{x}_{t-3}; \mathbf{E}\mathbf{x}_{t-2}; \mathbf{E}\mathbf{x}_{t-1}]$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{e})$$

$$\mathbf{z} = \mathbf{U}\mathbf{h}$$

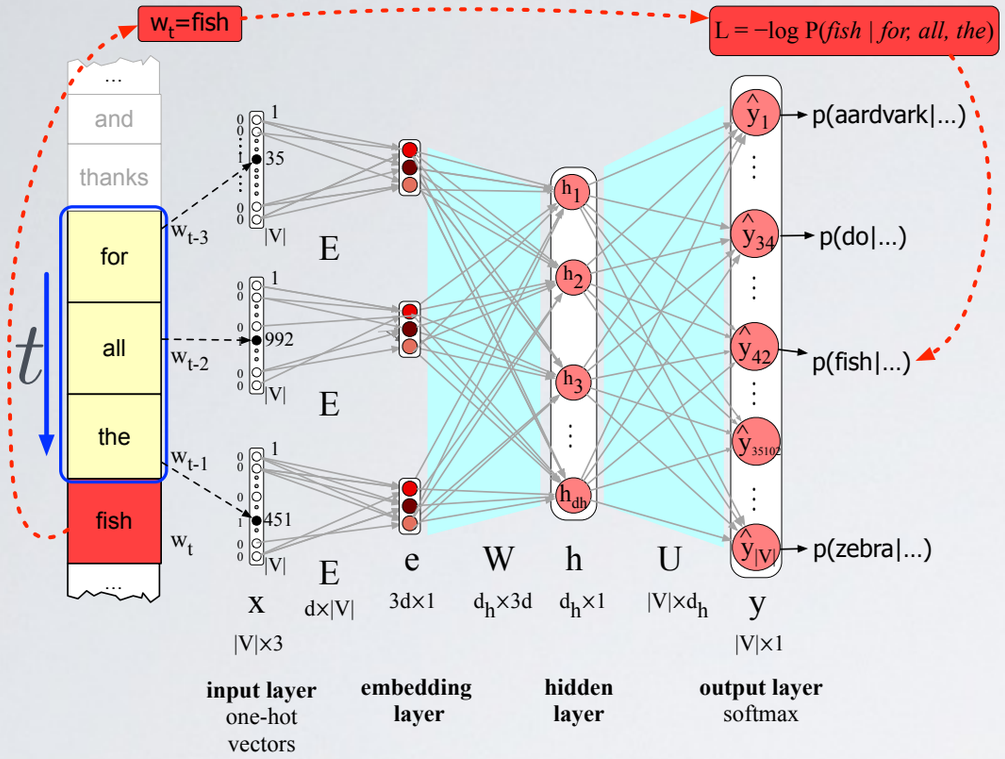
$$\mathbf{y} = \text{softmax}(\mathbf{z})$$

$$\mathbb{P}(w_t = i | w_{t-3}, w_{t-2}, w_{t-1})$$

- Assume a **fixed context window**, e.g. 3 preceding words
- At each timestep t the network computes a d-dimensional **embedding** (`word2vec`, `GloVe`, random,…)-initialized for each context word, by multiplying a **one-hot vector** by the embedding matrix $\mathbf{E}$ and concatenates the 3 resulting embeddings to get the **embedding input layer** $\mathbf{e}$
- The embedding vector $\mathbf{e}$ is multiplied by a weight matrix $\mathbf{W}$ and then an **activation function** is applied element-wise to produce the hidden layer $\mathbf{h}$, which is then multiplied by another weight matrix $\mathbf{U}$
- Finally, a `softmax` output layer predicts at each node i the probability that the next word w_t will be vocabulary word V_i

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research (JMLR)*, 3:1137–1155.

Training Feedforward Neural Language Models



$$L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i=1}^{|V|} \mathbf{y}_i \log \hat{\mathbf{y}}_i =$$

$$= -\sum_{i=1}^{|V|} \mathbb{1}\{\mathbf{y}_i = 1\} \log \hat{\mathbf{y}}_i =$$

$$= -\log \hat{\mathbf{y}}_c \quad (\text{where } c \text{ is the correct class}) =$$

$$= -\log \frac{\exp(\mathbf{z}_c)}{\sum_{i=1}^{|V|} \exp(\mathbf{z}_i)} =$$

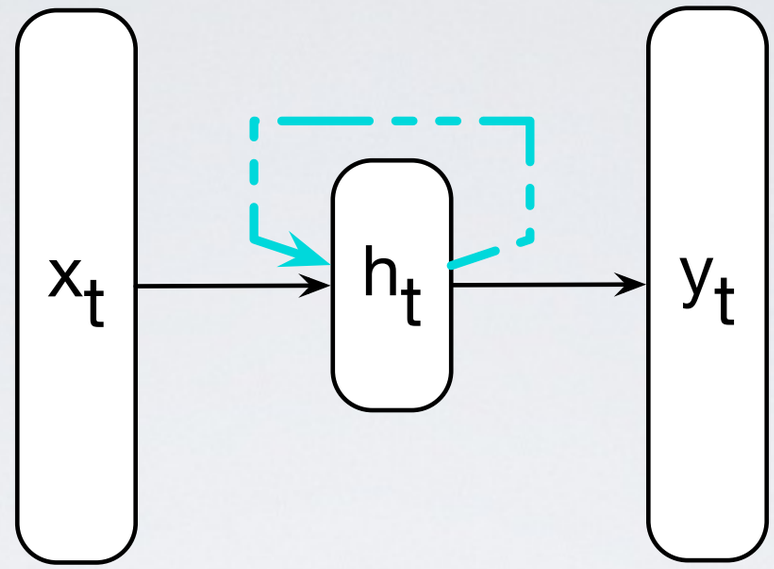
$$= -\log \mathbb{P}(w_t | w_{t-1}, \ldots, w_{t-n+1})$$

- We use the **self-supervision approach**, where training proceeds by taking as input a very long text, concatenating all the sentences, starting with random weights, and then iteratively moving through the text predicting each word
- At each word w_t, we use the **cross-entropy** (negative log likelihood) **loss**
- We use **stochastic gradient descent** to update parameters from step (**epoch**) s to $s + 1$ with **learning rate** η
  - We use **backpropagation** to update the weights of the network
  - Training the parameters to minimize loss will result both in an algorithm for language modeling (a word predictor) but also a new set of embeddings that can be used as word representations for other tasks (optional)

$$\theta^{s+1} = \theta^s - \eta \frac{\partial L_{CE}}{\partial \theta}, \quad \theta = \mathbf{E}, \mathbf{W}, \mathbf{U}$$

Recurrent Neural Networks and Attention

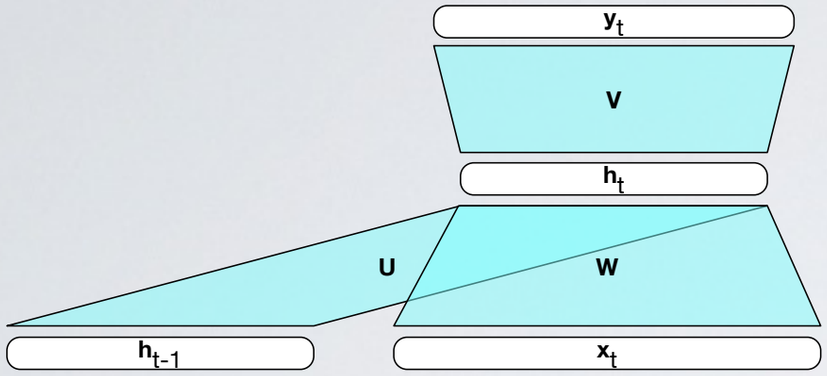
Recurrent Neural Networks (RNN)



- **Sequences** are processed by presenting one item $\mathbf{x}_t$ at a time t to the network
- As with ordinary feedforward networks, an input vector representing the current input $\mathbf{x}_t$ is multiplied by a weight matrix and then passed through a non-linear activation function to compute the values for a layer of hidden units $\mathbf{h}_t$. This hidden layer is then used to calculate a corresponding output $\mathbf{y}_t$
- Differently from a feedforward network, the **recurrent link** augments the input to the computation at the hidden layer $\mathbf{h}_t$ with the value of the hidden layer from the preceding point in time $\mathbf{h}_{t-1}$
- The hidden layer from the previous time step $\mathbf{h}_{t-1}$ provides a form of **memory**, or **context**, that encodes earlier processing and informs the decisions to be made at later points in time
  - This approach does **not** impose a **fixed-length** limit on this prior **context**; the context embodied in the previous hidden layer can include information extending back to the beginning of the sequence

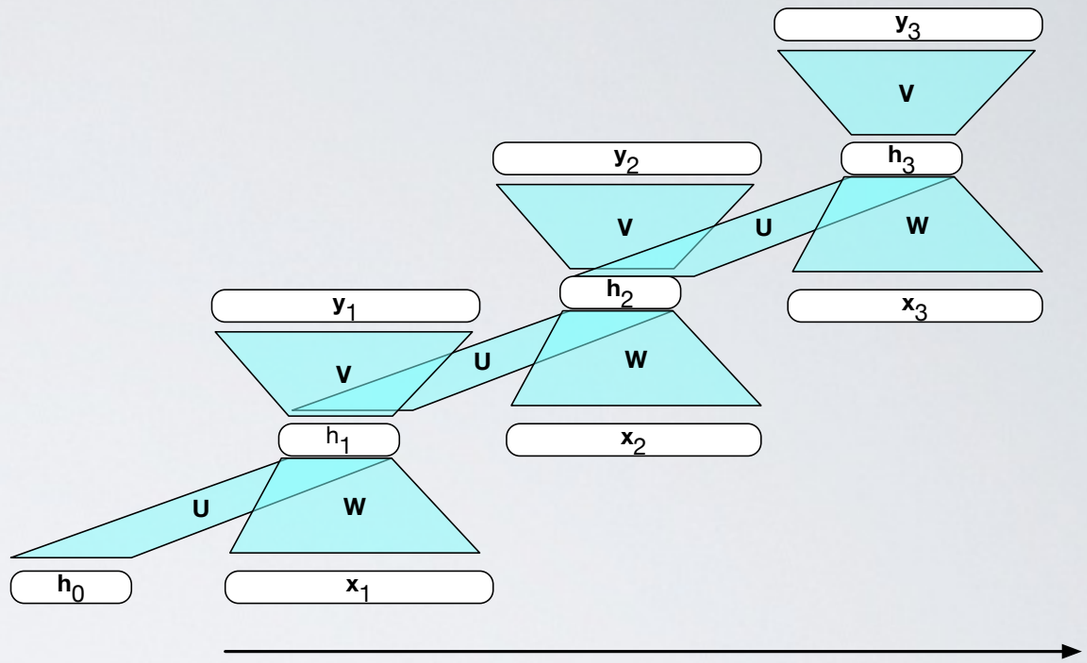
Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science*, 14(2):179–211.

RNN Computation and Unrolling



$$\mathbf{h}_t = \mathrm{g}(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$$

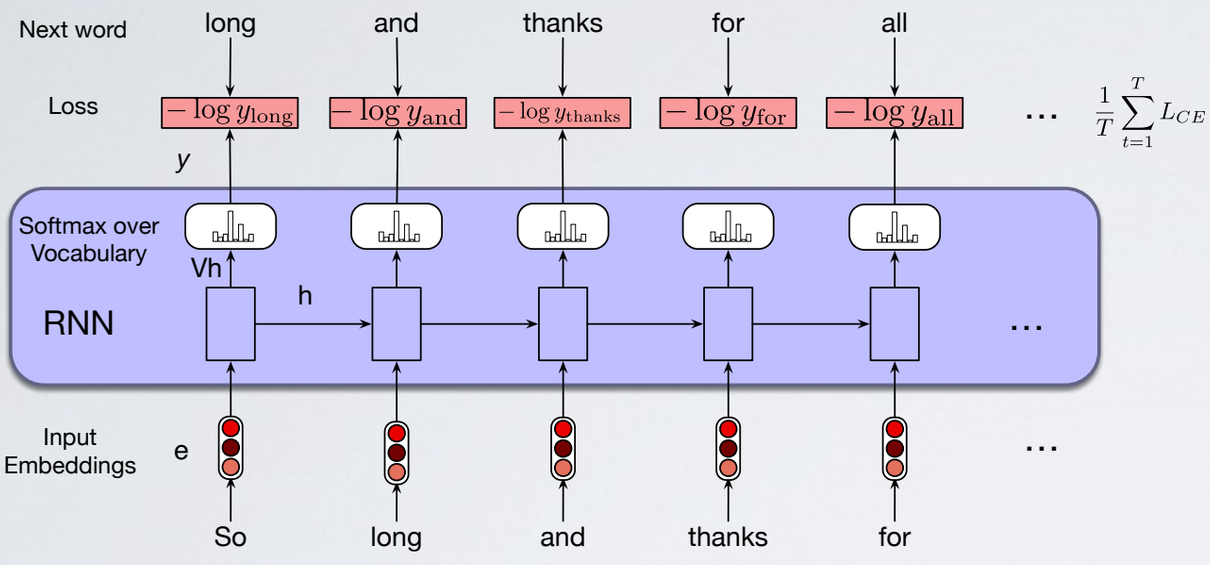
$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{V}\mathbf{h}_t)$$



- g is a suitable **activation function**, e.g. `ReLU`, `sigmoid`, or `tanh`
- In the case of classification, `softmax` provides a probability distribution over the possible output classes
- The weight matrices $\mathbf{U}$, $\mathbf{W}$, and $\mathbf{V}$ are shared across time
- The **sequential nature** of RNN can be seen by **unrolling** the network in time
  - Unrolling allows for training via **backpropagation**
  - For long sequences, unrolling an entire input sequence may not be feasible. In these cases, we can unroll the input into manageable fixed-length segments and treat each segment as a distinct training item

RNN as Language Model

$$\mathbb{P}(w_{t+1} = k | w_1, \ldots, w_t) = \mathbf{y}_t[k]$$

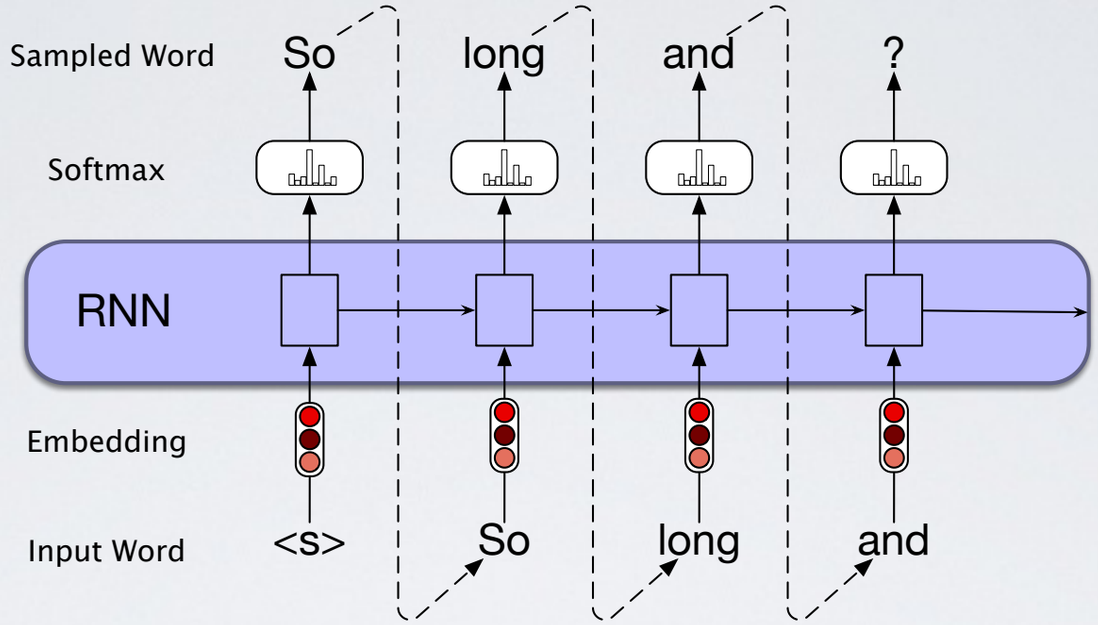


$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t [w_{t+1}]$$

- The probability that a particular word in the vocabulary is the next word is represented by $\mathbf{y}_t[k]$, the k-th component of $\mathbf{y}_t$
- To **train an RNN as a language model**, we use **self-supervision**: we take a corpus of text as training material and at each time step t ask the model to predict the next word
  - The **cross-entropy loss** measures the difference between a predicted probability distribution and the correct distribution
  - At each word position t of the input, the model takes as input the correct sequence of tokens $w_{1:t}$ (**teacher forcing**), and uses them to compute a probability distribution over possible next words so as to compute the model's loss for the next token w_{t+1}
  - The weights in the network are adjusted to **minimize the average CE loss** over the training sequence via **gradient descent**

Mikolov, T., Karafiát, M., Burget, L., Černocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In Kobayashi, T., Hirose, K., and Nakamura, S., editors, *Proc. 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010)*, pages 1045–1048. International Speech Communication Association (ISCA).

Generation with RNN-based Language Model



dopo training, ora si genera

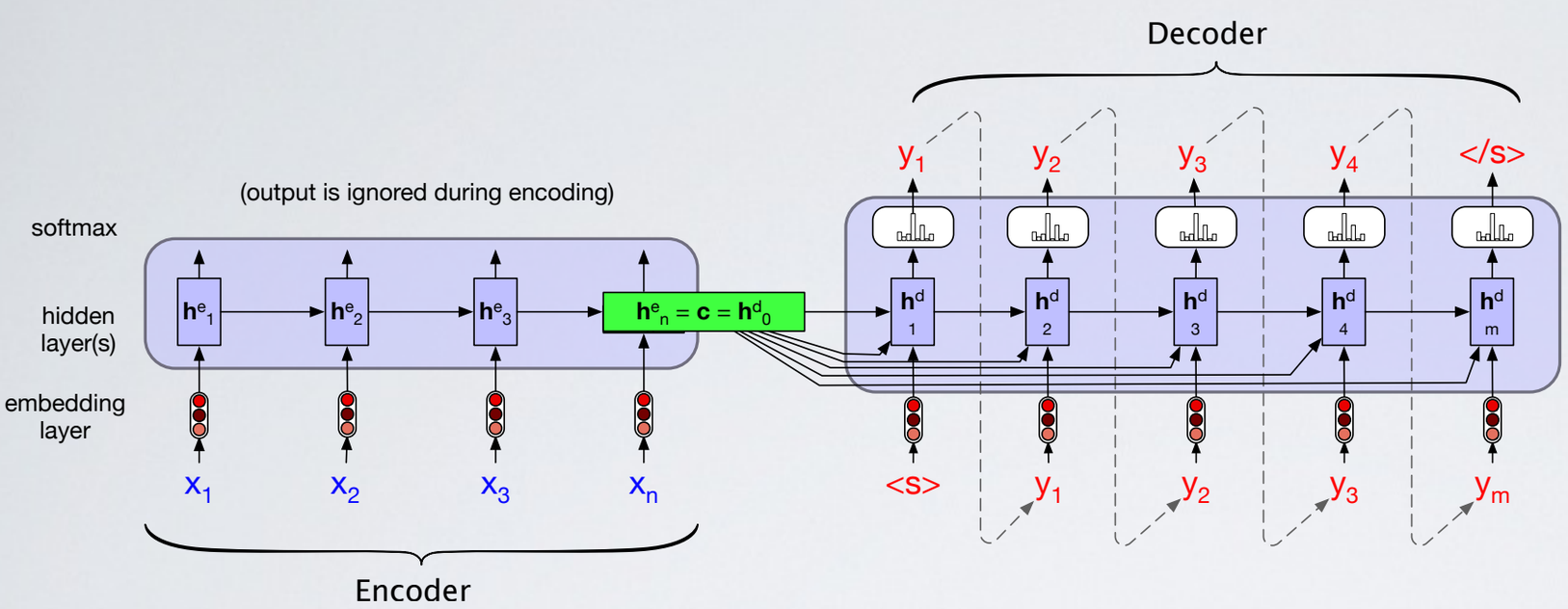
- **Autoregressive generation** (**causal LM generation**): we use a language model to incrementally generate words by repeatedly sampling the next word conditioned on our previous choices
  - Sample a word in the output from the `softmax` distribution that results from using the **beginning of sentence marker**, `<s>`, as the first input
  - Use the word embedding for that first word as the input to the network at the next time step, and then sample the next word in the same fashion
  - Continue generating until the end of sentence marker, `</s>`, is sampled or a fixed length limit is reached
- Autoregressive generation is applied to machine translation, summarization, and question answering. The key to these approaches is to **prime the generation component with an appropriate context**. That is, instead of simply using `<s>` to get things started we can provide a richer task-appropriate context:
  - for **translation** the context is the sentence in the source language
  - for **summarization** the context is the long text we want to summarize
  - for **question answering** the context is the question we need to answer

RNN Limitations

`The flights the airline was canceling were full.`

- Consider the above sentence
  - Assigning a high probability to `was` following `airline` is straightforward since `airline` provides a strong local context for the singular agreement
  - Assigning an appropriate probability to `were` is quite difficult, not only because the plural `flights` is quite distant, but also because the singular noun `airline` is closer in the intervening context
  - A network should be able to retain the **distant information** about plural `flights` until it is needed, while still processing the **intermediate parts** of the sequence correctly
- RNN suffer from **inability to carry forward critical information**
  - The hidden layer performs two tasks simultaneously: provide information useful for the **current decision**, and updating and carrying forward information required for **future decisions**
  - During the backward pass of training, the hidden layers are subject to repeated multiplications, as determined by the length of the sequence, causing the vanishing gradients problem
- RNN are intrinsically **sequential** and their computation **cannot** be **parallelized**
- **Long short-term memory (LSTM)** networks divide the context management problem into two subproblems: removing information no longer needed from the context, and adding information likely to be needed for later decision making
  - Add an explicit **context layer** to the architecture, in addition to the usual recurrent hidden layer
  - Use specialized neural units, called, **gates** to control the flow of information into and out of the units that comprise the network layers

Encoder-Decoder Networks



$$\mathbf{c} = \mathbf{h}^e_n$$

$$\mathbf{h}^d_0 = \mathbf{c}$$

$$\mathbf{h}^d_t = \mathrm{g}(\hat{y}_{t-1}, \mathbf{h}^d_{t-1}, \mathbf{c})$$

$$\mathbf{z}_t = \mathrm{f}(\mathbf{h}^d_t)$$

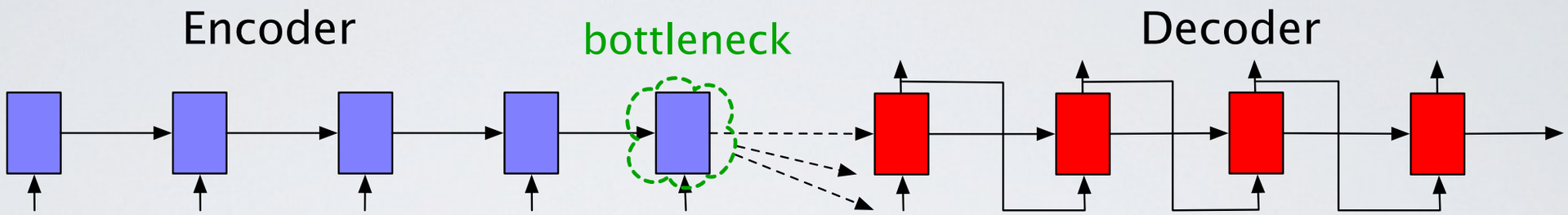
$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{z}_t)$$

encoded: nuovo input

- **Encoder-decoder networks**, sometimes called **sequence-to-sequence networks**, consist of three conceptual components:
  - An **encoder** that accepts an input sequence and generates a corresponding sequence of **contextualized representations**
    - RNN, LSTM, convolutional networks, and transformers can all be employed as encoders
  - A **context vector** $\mathbf{c}$ which conveys the essence of the input to the decoder.
  - A **decoder**, which accepts $\mathbf{c}$ as input and generates an arbitrary length sequence of hidden states, from which a corresponding sequence of output states can be obtained
    - Just as with encoders, decoders can be realized by any kind of sequence architecture
- Encoder-decoder architectures are trained end-to-end: each training example is a tuple of paired strings, a source and a target, concatenated with a separator <s>

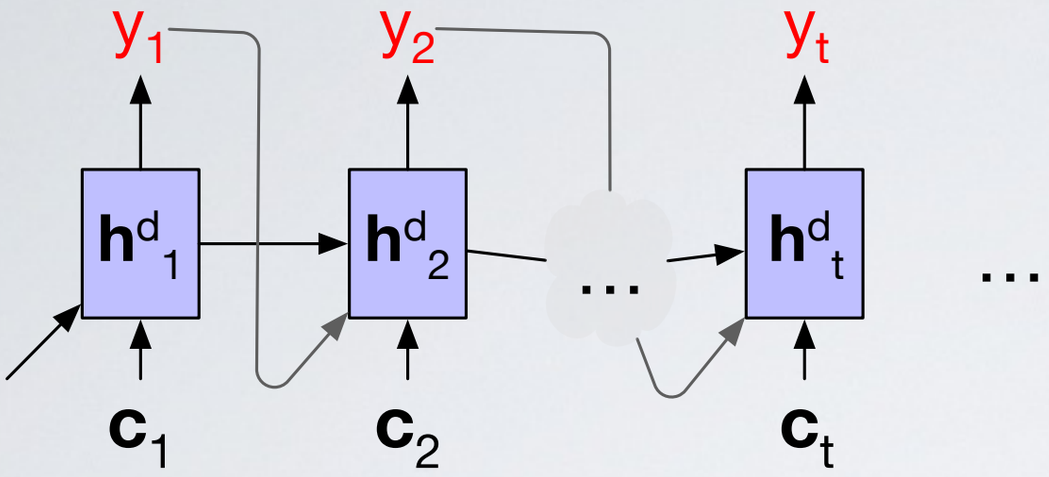
Cho, K., van Merrïenboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1724–1734. Association for Computational Linguistics, USA.
Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Proc. 27th Annual Conference on Neural Information Processing Systems (NeurIPS 2014)*. https://proceedings.neurips.cc/paper_files/paper/2014.

The Encoder-Decoder Bottleneck



- The final hidden state is the context vector and acts as a **bottleneck**: it must represent absolutely everything about the meaning of the source text, since the only thing the decoder knows about the source text is what is in this context vector
  - Information at the beginning of the sentence, especially for long sentences, may not be equally well represented in the context vector
- The **attention mechanism** is a solution to the bottleneck problem, a way of allowing the decoder to **get information from all the hidden states** of the encoder, not just the last hidden state

Attention



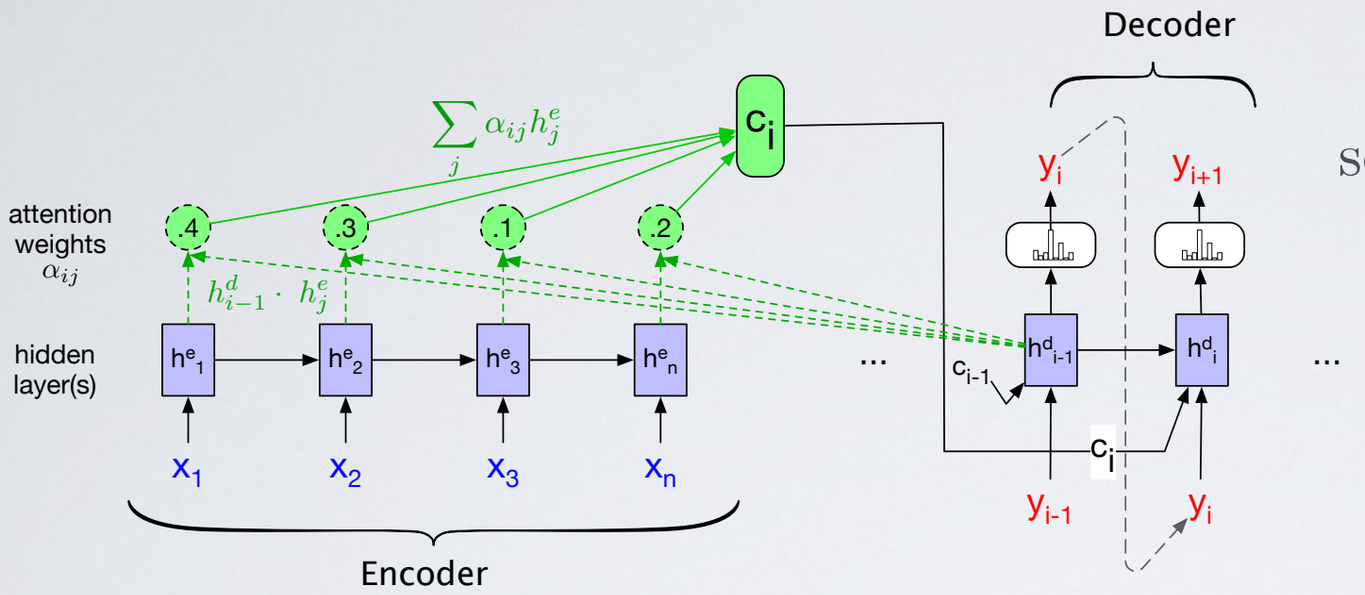
...

$$\mathbf{h}_t^d = \mathrm{g}(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}_t)$$

- The idea of **attention** is to create a fixed-length vector by taking a weighted sum of all the encoder hidden states.
  - The weights focus on ("**attend to**") a particular part of the source text that is relevant for the token the decoder is currently producing
- Attention replaces the static context vector with **dynamic context vector derived** from the encoder hidden states, different for each token in decoding
  - This context vector $\mathbf{c}_t$ is generated anew with each decoding step t and takes all of the encoder hidden states into account in its derivation

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv.org, Computation and Language (cs.CL)*, accepted as *Proc. 3rd International Conference on Learning Representations (ICLR 2015)*, arXiv:1409.0473.

Attention Computation



$$\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) = \mathbf{h}^d_{i-1} \cdot \mathbf{h}^e_j$$

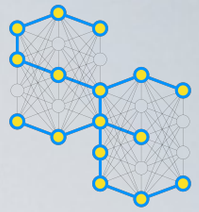
$$\alpha_{ij} = \text{softmax}\left(\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j)\right)$$

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}^e_j$$

- We compute **how much to focus on** ("attend to) each encoder state, i.e. how relevant each encoder state is to the decoder state captured in $\mathbf{h}^d_{i-1}$
  - We capture **relevance** by computing, at each state i during decoding, a $\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j)$ for each encoder state j
  - The simplest such score, called **dot-product attention**, implements relevance as attention similarity
- To make use of these scores, we normalize them with a `softmax` to create a vector of weights α_{ij}
- We compute the fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states

Transformer

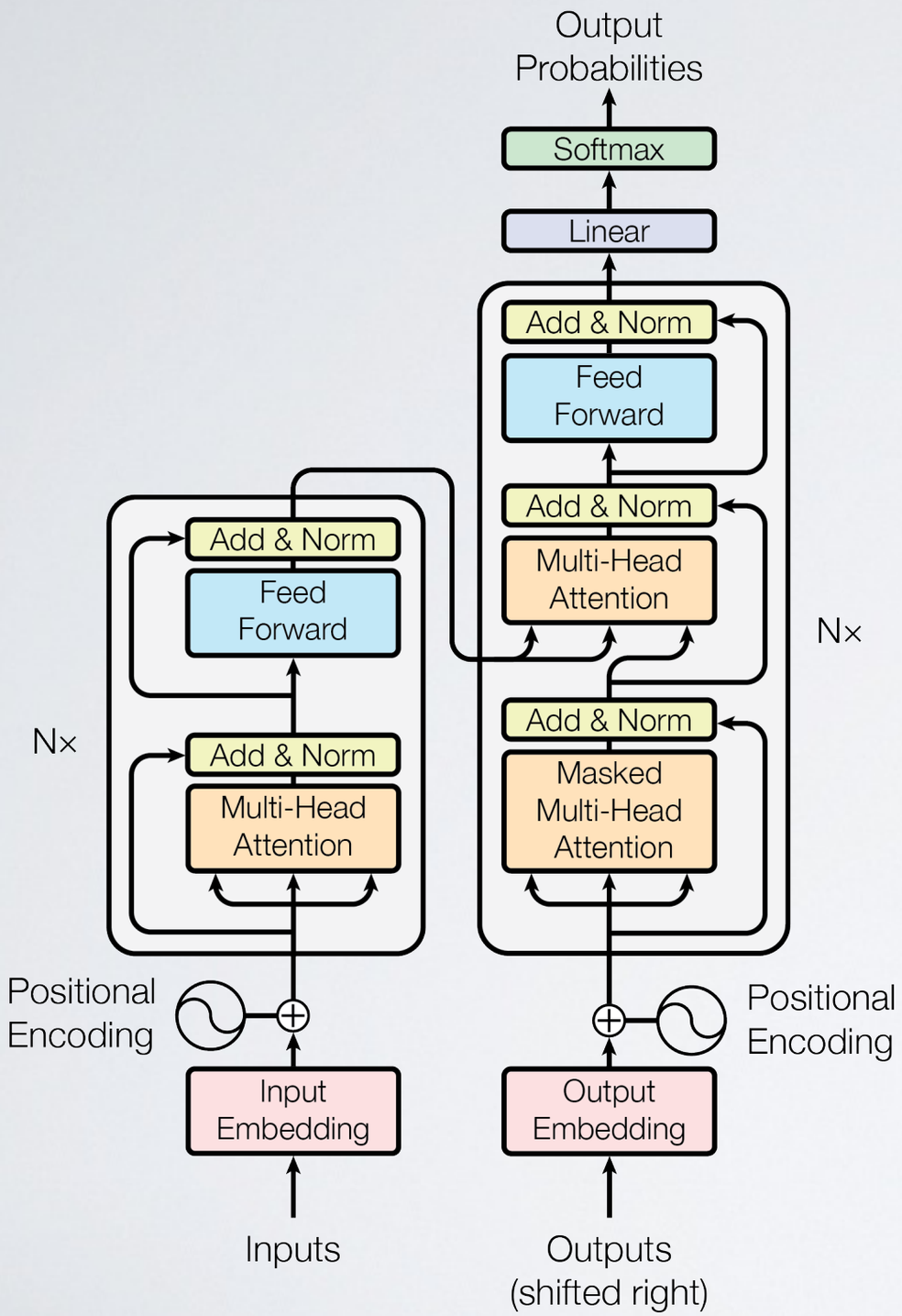
NLP Tasks and Transformer



- **Machine translation**: translating a text from a language into another language
  - Also done with recurrent neural networks (encoder-decoder + attention)
- **Document summarisation**: given a text, extract a summary
  - Also done with recurrent neural networks (encoder-decoder + attention)
- **Question answering**: given a question and a context, extracting the answer to the question based on the information provided in the context
  - Also done with recurrent neural networks (encoder-decoder + attention)
- **Language Modeling (content generation)**: completing a prompt with auto-generated text
  - Also done with recurrent neural networks
- **Sequence labelling**: part of speech tagging (PoS), I.e. identifying the grammatical components of a sentence (noun, verb, adjective), or the named entity extraction (person, location, organization)
  - Also done with recurrent neural networks
- **Sentence (sequence) classification**: getting the sentiment of a review, detecting if an email is spam, determining whether two sentences are logically related or not
  - Also done with feedforward neural networks and recurrent neural networks

- All these tasks are specific instances of a more general task, i.e., the **input sequence to output sequence** task

Transformer: Encoder-Decoder & Self-Attention



- The main innovation of the Transform architecture is the **self-attention** layer
  - It allows for **getting rid** of the **recurrence** and providing a significant boost in **parallelization**
  - It increases the ability in learning **long-range dependencies**
- The original paper focused on a machine translation task, so it adopted a full **encoder-decoder architecture**
  - Some terminology confusion happened afterwards since, in addition to encoder-decoder models, it is common to talk about **encoder-only** and **decoder-only models**
  - They are actually just two different ways of instantiating the self-attention layer in the half of the architecture, depending on the specific task to be carried out

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention Is All You Need. In Guyon, I., Von Luxburg, U., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Proc. 30th Annual Conference on Neural Information Processing Systems (NeurIPS 2017)*. https://proceedings.neurips.cc/paper_files/paper/2017.

Decoder Models

- Decoder models are best suited for tasks involving **text generation**
- The **self-attention layers** in such models can **only** access **tokens positioned before** the corresponding output in the input sentence
  - These models are often called **causal models** or **auto-regressive models**
- Representatives of this family of models include:
  - GPT - OpenAI, 2018 [Radford et al., 2018]
  - GPT-2 - OpenAI, 2019 [Radford et al., 2019]
  - GPT-3 - OpenAI, 2020 [Brown et al., 2020]
  - GPT-4 - OpenAI, 2023 [OpenAI et al., 2023]
  - LLAMA-1 and -2 - Meta and Meta&Microsoft 2023 [Touvron et al., 2023a, b]
  - Gemini - Google 2023 [Gemini Team Google et al., 2023]
  - Falcon (open source) [Almazrouei et al., 2023]
  - LLAMA-3 - Meta 2024, April 18 (`https://ai.meta.com/blog/meta-llama-3/`)

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training. Technical report, OpenAI, USA, https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. Technical report, OpenAI, USA, https://paperswithcode.com/paper/language-models-are-unsupervised-multitask.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language Models are Few-Shot Learners. *arXiv.org, Computation and Language (cs.CL)*, arXiv:2005.14165.

OpenAI et al. (2023). GPT-4 Technical Report. *arXiv.org, Computation and Language (cs.CL)*, arXiv:2303.08774.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozi`ere, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. (2023a). LLaMA: Open and Efficient Foundation Language Models. *arXiv.org, Computation and Language (cs.CL)*, arXiv:2302.13971.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Canton Ferrer, C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Singh Koura, P., Lachaux, M.-H., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. (2023b). Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv.org, Computation and Language (cs.CL)*, arXiv:2307.09288.

Gemini Team Google (2023). Gemini: A Family of Highly Capable Multimodal Models. *arXiv.org, Computation and Language (cs.CL)*, arXiv:2312.11805.

Almazrouei, E., Alobeidli, H., Alshamsi, A., Cappelli, A., Cojocaru, R., Debbah, M., Goffinet, E., Hesslow, D., Launay, J., Malartic, Q., Mazzotta, D., Noune, B., Pannier, B., and Penedo, G. (2023). The Falcon Series of Open Language Models. *arXiv.org, Computation and Language (cs.CL)*, arXiv:2311.16867

Encoder Models

- Encoder models are best suited for tasks requiring an **understanding of the full sentence**, such as sentence classification, word classification, e.g., named entity recognition, and summarisation, e.g., extractive question answering
- The **self-attention layers** of such models can access **all the tokens** in the input sequence
  - These models are often called **auto-encoding model**s
- Representatives of this family of models include:
  - ELMo - AllenAI 2018 [Peters et al., 2018]
  - BERT - Google, 2018 [Devlin et a., 2019]
  - RoBERTa - Meta, 2019 [Liu et al., 2019] {rejected at ICLR 2020}
  - ALBERT (Google, 2019) [Lan et al., 2020] {accepted at ICLR 2020}
  - DistilBERT - HuggingFace, 2019
  - ELECTRA - Google, 2020 [Clarke et al., 2020] {accepted at ICLR 2020}
  - DeBERTa - Microsoft 2020 [He et al., 2021]

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep Contextualized Word Representations. In Walker, M., Ji, H., and Stent, A., editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL 2018)*, pages 2227–2237. Association for Computational Linguistics, USA.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL 2019)*, pages 4171–4186. Association for Computational Linguistics, USA.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv.org, Computation and Language (cs.CL)*, arXiv:1907.11692.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2020). ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In Rush, A., Mohamed, S., Song, D., Cho, K., and White, M., editors, *Proc. 8th International Conference on Learning Rep- resentations (ICLR 2020).* OpenReview.net, `https://openreview.net/group?id=ICLR.cc/2020/Conference`.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv.org, Computation and Language (cs.CL)*, arXiv:1910.01108.

Clark, K., Luong, M.-T., Le, Q. C., and Manning, C. D. (2020). ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In Rush, A., Mohamed, S., Song, D., Cho, K., and White, M., editors, *Proc. 8th International Conference on Learning Representations (ICLR 2020*). OpenReview.net, `https://openreview.net/group?id=ICLR.cc/2020/Conference`.

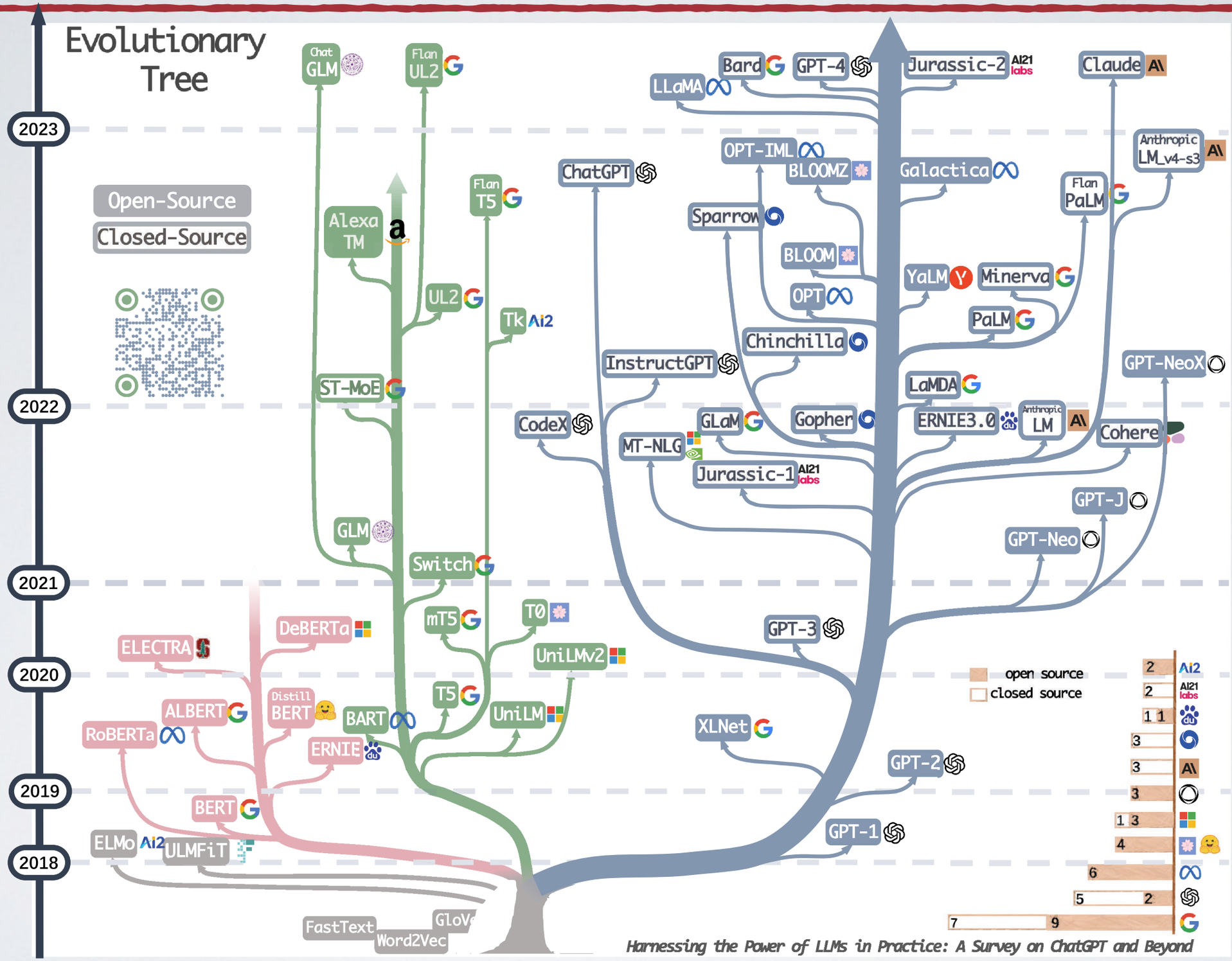
He, P., Liu, X., Gao, J., and Chen, W. (2021). DeBERTa: Decoding-enhanced BERT with Disentangled Attention. In Mohamed, S., Hofmann, K., Oh, A., Murray, N., and Titov, I., editors, *Proc. 9th International Conference on Learning Representations (ICLR 2021)*. OpenReview.net, `https://openreview.net/group?id=ICLR.cc/2021/Conference` .

Encoder-Decoder Models

- **Sequence-to-sequence (seq2seq) models** are best suited for tasks revolving around generating new sentences depending on a given input, such as summarisation, machine translation, or generative question answering
  - Encoder-decoder models use both parts of the Transformer model
  - The **self-attention layers** of the encoder can access all the tokens in the input sequence
  - The **self-attention layers** of the decoder can only access the tokens positioned before the corresponding output it in the input sentence
- Representatives of this family of models include:
  - BART - Meta 2019 [Lewis et al., 2020]
  - T5 - Google 2019 [Raffel et al., 2020]
  - FlanT5 - Google 2022

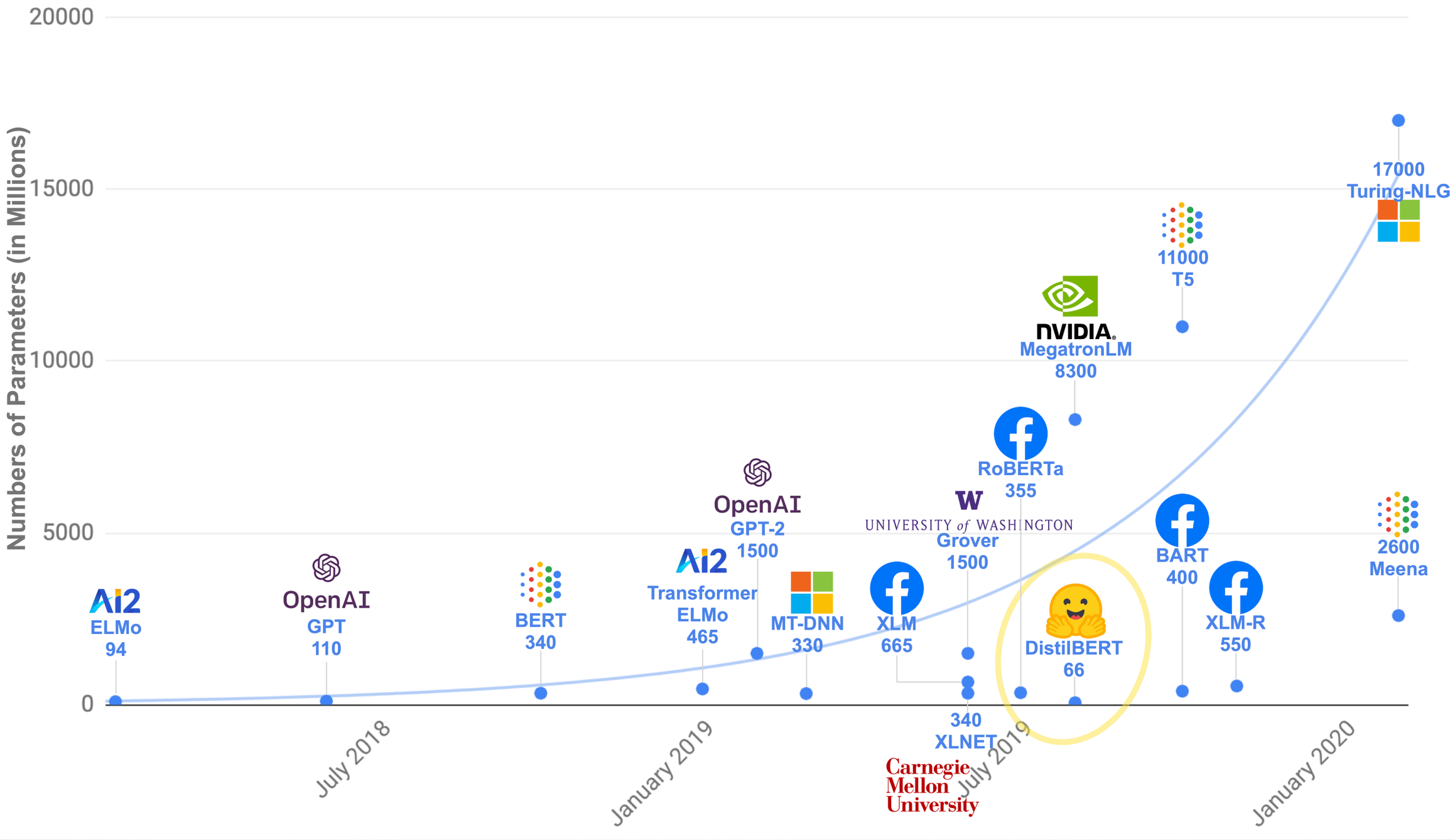
Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2020). BART: Denoising Sequence-to- Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J., editors, *Proc. of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*, pages 7871–7880. Association for Computational Linguistics, USA.
Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the Limits of Transfer Learn- ing with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research (JMLR)*, 21(140):1–67.
Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., Webson, A., Shane Gu, S. S., Dai, Z., Suzgun, M., Chen, X., Chowdhery, A., Castro-Ros, A., Pellat, M., Robinson, K., Valter, D., Narang, S., Mishra, G., Yu, A., Zhao, V., Huang, Y., Dai, A., Yu, H., Petrov, S., Chi, E. H., Dean, J., De- vlin, J., Roberts, A., Zhou, D., Le, Q. V., and Wei, J. (2022). Scaling Instruction-Finetuned Language Models. *arXiv.org, Machine Learning (cs.LG),* arXiv:2210.11416.

Transformer Genealogy



Yang, J., Jin, H., Tang, R., Han, X., Feng, Q., Jiang, H., Zhong, S., Yin, B., and Hu, X. (2024). Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 18(6):160:1–160:32.

Transformer Size



- The general strategy to achieve better performance is by increasing the models' sizes as well as the amount of data they are trained on

Transformer Openness

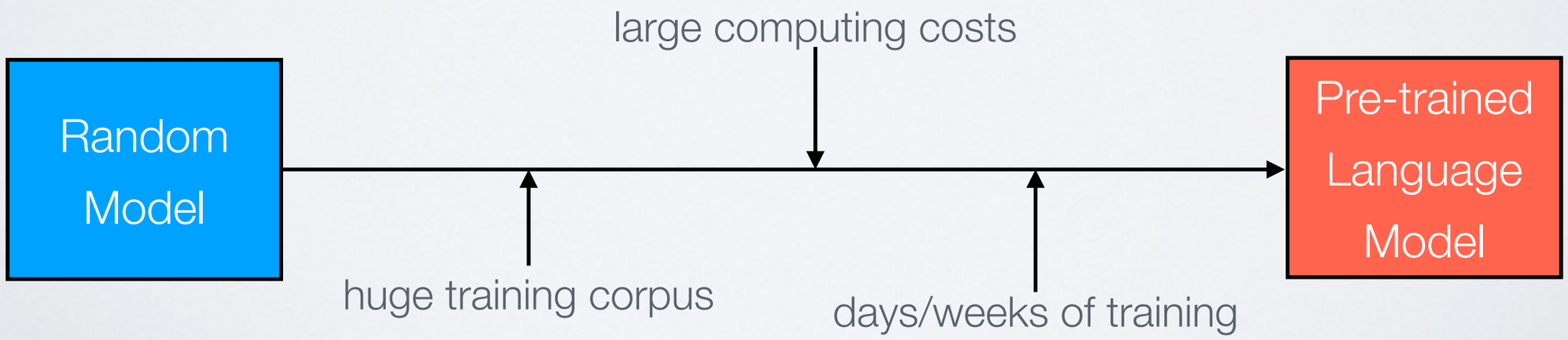
Project	Availability						Documentation					Access methods	
(maker, bases, URL)	Open code	LLM data	LLM weights	RLHF data	RLHF weights	License	Code	Architecture	Preprint	Paper	Data sheet	Package	API
chatGPT	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	~
OpenAI	LLM base: GPT3.5, GPT4			RLHF base: Instruct-GPT									https://chat.openai.com
StableVicuna-13B	✓	✓	~	~	~	~	~	✓	✓	✗	✗	~	✗
CarperAI	LLM base: LLaMA			RLHF base: oasst1, anthropic									https://huggingface.co/CarperAI/stable-vicuna-13b-delta
text-generation-webu	✓	✓	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗
oobabooga	LLM base: various			RLHF base: various									https://github.com/Akegarasu/ChatGLM-webui
MPT-7B-Instruct	✓	✗	✓	~	✗	✓	✓	~	✗	✗	✗	✓	✗
MosaicML	LLM base: MosaicML			RLHF base: dolly, anthropic									https://github.com/mosaicml/llm-foundry#mpt
Falcon-40B-Instruct	✓	~	✓	~	✓	✓	~	~	~	✗	~	~	✗
TII	LLM base: Falcon 40B			RLHF base: Baize (synthetic)									https://huggingface.co/tiiuae/falcon-40b-instruct
minChatGPT	✓	✓	✓	~	✗	✓	✓	~	✗	✗	✗	✗	✓
ethanyanjiali	LLM base: GPT2			RLHF base: anthropic									https://github.com/ethanyanjiali/minChatGPT
trlx	✓	✓	✓	~	✗	✓	✓	~	✗	✗	✗	~	✓
carperai	LLM base: various (pythia, flan, OPT)			RLHF base: various									https://github.com/carperai/trlx
stanford_alpaca	✓	✓	~	~	✗	~	✓	✓	✗	✗	~	✗	✗
Tatsu labs	LLM base: LLaMA			RLHF base: Self-Instruct (synthetic)									https://github.com/tatsu-lab/stanford_alpaca
Cerebras-GPT-111M	✓	✓	✓	✓	✗	✓	✓	✓	~	✗	✗	✗	✗
Cerebras, Schramm	LLM base: not open			RLHF base: alpaca (synthetic)									https://huggingface.co/SebastianSchramm/Cerebras-GPT-111M-instruction
OpenChatKit	✓	✓	✓	✓	✓	✓	✓	✗	~	✗	✗	✓	✗
togethercomputer	LLM base: EleutherAI pythia			RLHF base: OIG									https://github.com/togethercomputer/OpenChatKit
dolly	✓	✓	✓	~	✗	✓	✓	✓	~	✗	✗	✓	✗
databrickslabs	LLM base: EleutherAI pythia			RLHF base: databricks-dolly-15k									https://github.com/databrickslabs/dolly
CharRWKV	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓
BlinkDL	LLM base: RWKV-LM (own)			RLHF base: alpaca, shareGPT (synthetic)									https://github.com/BlinkDL/ChatRWKV
BELLE	✓	✓	~	✓	✓	✓	✓	✓	✓	✗	~	✗	✗
LianjiaTech	LLM base: LLaMA, BLOOMZ			RLHF base: alpaca, shareGPT (synthetic)									https://github.com/LianjiaTech/BELLE
Open-Assistant	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓
LAION-AI	LLM base: oasst1 (own)			RLHF base: OIG									https://github.com/LAION-AI/Open-Assistant
xmtf	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
bigscience-workshop	LLM base: BLOOMZ, mT0			RLHF base: xP3									https://github.com/bigscience-workshop/xmtf

- Many projects inherit data of dubious legality
- Few projects share the all-important instruction-tuning
- Preprints are rare, peer-reviewed papers even rarer
- Synthetic instruction-tuning data is on the rise, with unknown consequences that are in need of research

Liesenfeld, A., Lopez, A., and Dingemanse, M. (2023). Opening up Chat- GPT: Tracking openness, transparency, and accountability in instruction- tuned text generators. In Lee, M., Munteanu, C., Porcheron, M., Trippas, J., and ¨olkel, S. T., editors, *Proc. 5th International Conference on Conversational User Interfaces (CUI 2023)*, pages 47:1–47:6.
https://opening-up-chatgpt.github.io/

Transformer Pre-Training

- Training a model, especially a large one, requires a **large amount of data**
- This becomes **very costly** in terms of time and compute resources
- This translates to **environmental impact**
- **Pre-training** is the act of training a model from scratch: the weights are randomly initialized and the training starts without any prior knowledge
  - **self-supervised training** is the typical approach

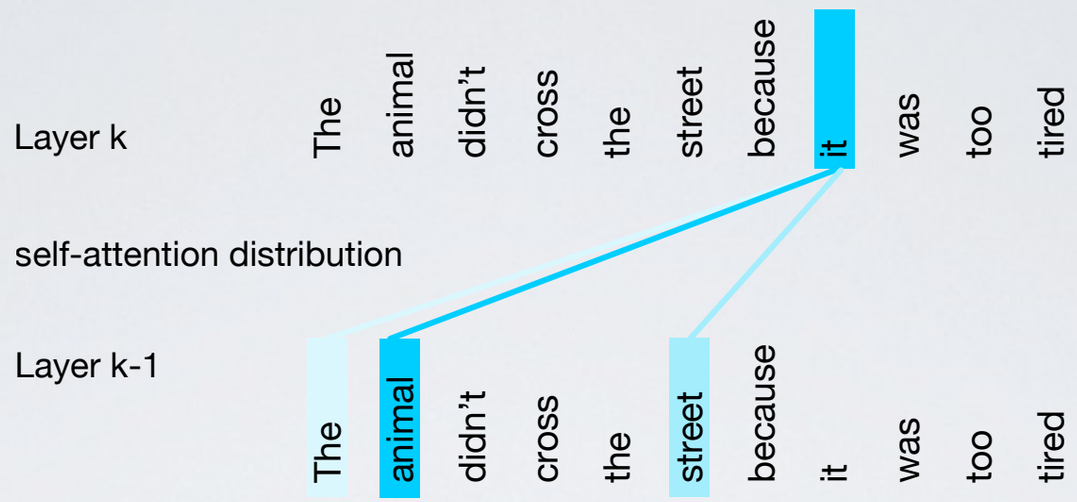


Transformer Fine-Tuning

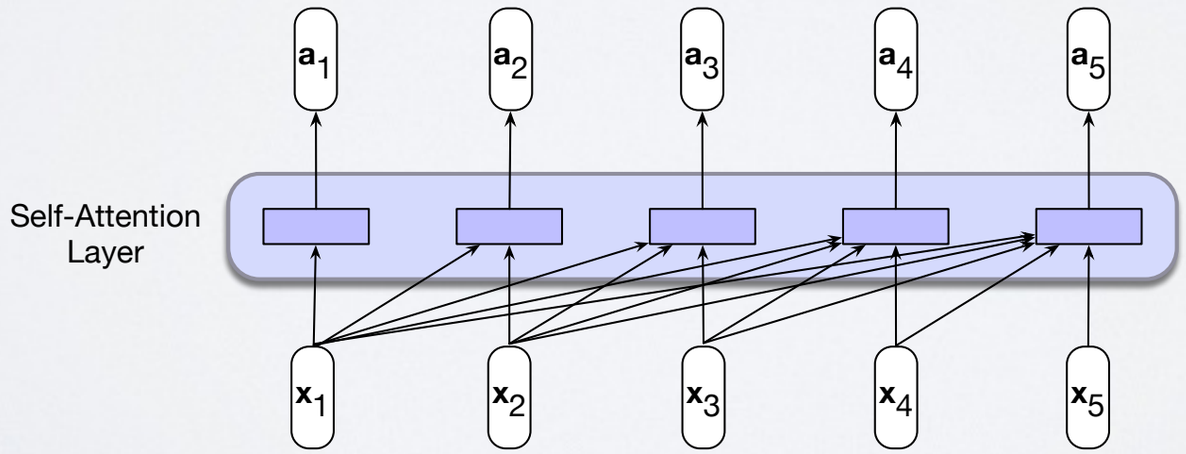
- **Fine-tuning** is the training done after a model has been pre-trained
  - First acquire a pre-trained language model, then perform **additional training** with a dataset specific to your **downstream task**
  - **Supervised training** is typically adopted
- The fine-tuning process is thus able to take advantage of **knowledge acquired by** the initial model during **pre-training**
- Since the pre-trained model was already trained on lots of data, the fine-tuning requires way **less data** to get good results

Decoder Model Transformer: Causal Self-Attention

Intuition on Self-Attention



- **Self-attention** allows us to **look broadly in the context** and tells us how to integrate the representation from words in that context from layer k–1 to build the representation for words in layer k
- The word *animal* has a **high attention weight**, meaning that, computing the **representation for *it***, the transformer will draw most heavily on the ***representation for animal***
  - This will be useful for the model to build a representation that has the correct meaning for *it*, which indeed is **coreferent** here with the word *animal*
- In **causal**, or **backward looking**, **self-attention** the context is any of the **prior words**
  - In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one
  - Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel



https://research.google/blog/transformer-a-novel-neural-network-architecture-for-language-understanding/

Self-Attention: Computation

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i \qquad \mathbf{k}_i = \mathbf{W}^K \mathbf{x}_i \qquad \mathbf{v}_i = \mathbf{W}^V \mathbf{x}_i$$

- The **attention** mechanism we have seen in the case of encoder-decoder RNNs compare states using the **dot product** and `softmax` to turn them into a vector of weights
  - This attention mechanism is based on a simple notion of context, where all the vectors play the same role
- **Self-attention** relies on a more elaborated notion of context and considers **three different roles** that each input embedding can play during the course of the attention process
  - **Query**: as the **current focus** of attention when being compared to all of the other preceding inputs
  - **Key**: as a **preceding input** being compared to the current focus of attention
  - **Value**: as a value used to compute the output for the current focus of attention
- Transformers introduce weight matrices $\mathbf{W}^Q$, $\mathbf{W}^K$, and $\mathbf{W}^V$, used to **project** each input vector $\mathbf{X}_i$ into a representation of one of its three roles

Self-Attention: Computation

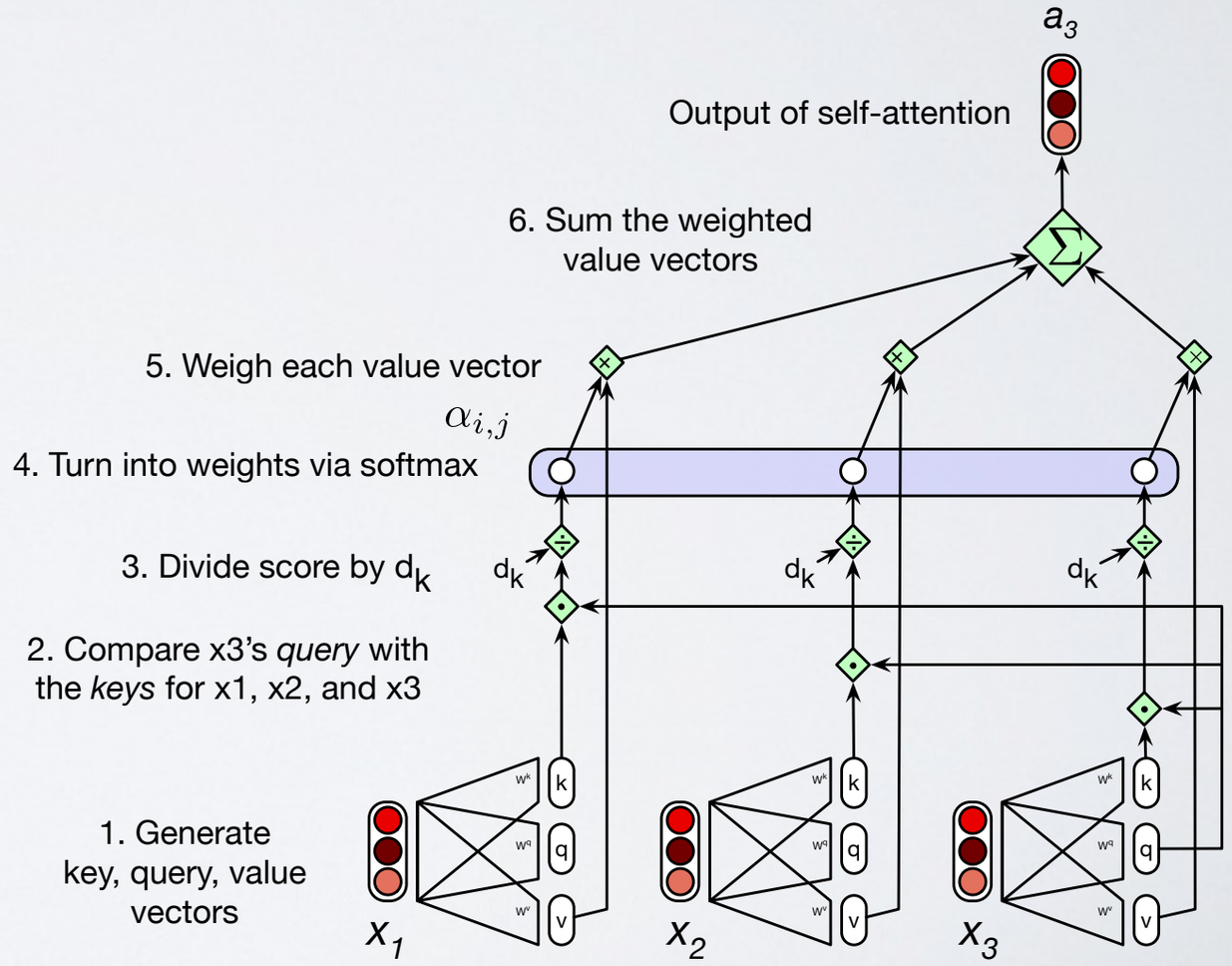
$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i;\ \mathbf{k}_i = \mathbf{W}^K \mathbf{x}_i;\ \mathbf{v}_i = \mathbf{W}^V \mathbf{x}_i$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = \text{softmax}\left(\text{score}(\mathbf{x}_i, \mathbf{x}_j)\right),\ \forall j \le i$$

$$\mathbf{a}_i = \sum_{j \le i} \alpha_{ij} \mathbf{v}_j$$

- The dot product can be an arbitrarily large (positive or negative) value
- Exponentiating large values can lead to numerical issues and to an effective loss of gradients during training.
- To avoid this, we scale down the result of the dot product, by dividing it by a factor related to the size of the embeddings
  - A typical approach is to divide by the square root of the dimensionality of the query and key vectors



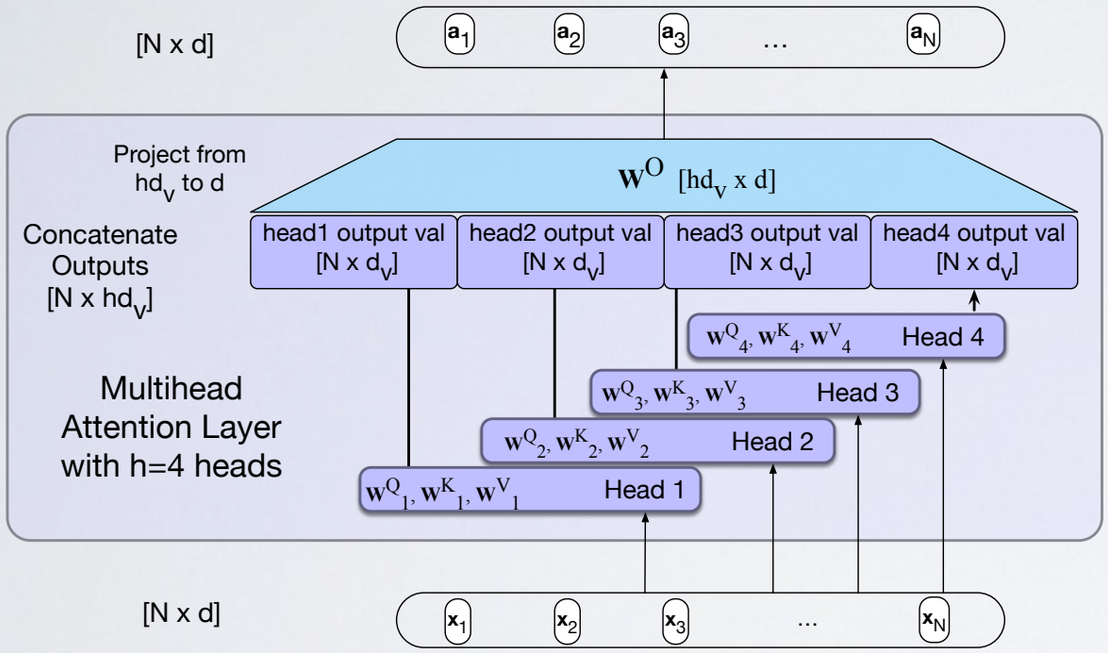
Multihead Attention

Upper triangle, i.e. **future words**, is **masked**, i.e. set to -∞

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_j^Q; \mathbf{K} = \mathbf{X}\mathbf{W}_j^K; \mathbf{V} = \mathbf{X}\mathbf{W}_j^V$$

$$\mathbf{head}_j = \text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

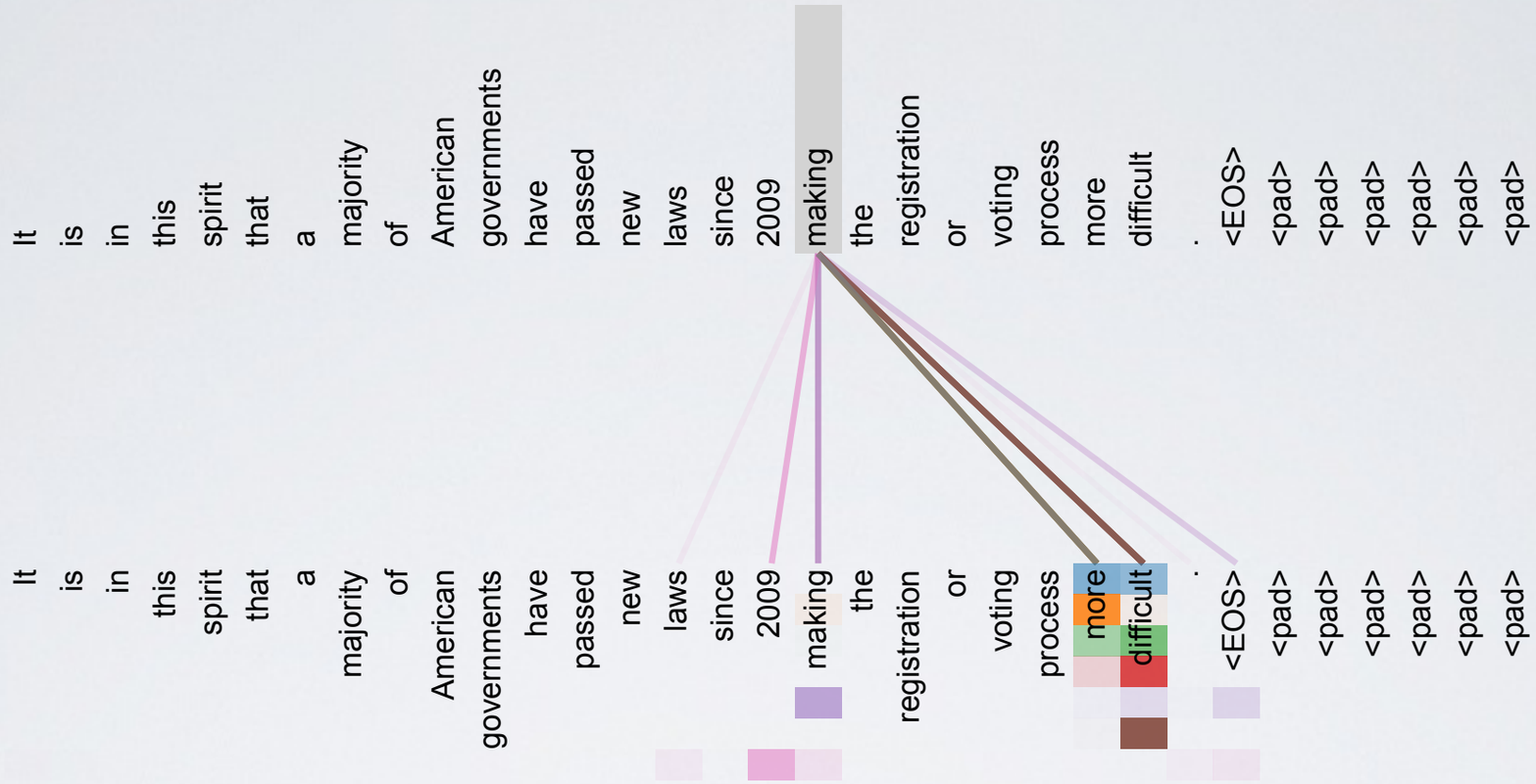
$$\mathbf{A} = \text{MultiHeadAttention}(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \oplus \ldots \oplus \mathbf{head}_h)\mathbf{W}^O$$



$\mathbf{X}$ is the matrix notation where each of the N rows is one of the input embeddings $\mathbf{x}_i$

- Different words in a sentence can relate to each other in many different ways simultaneously
  - It would be difficult for a single self-attention model to learn to capture all of the different kinds of parallel relations among its inputs
- **Multihead self-attention layers** are sets of self-attention layers, called **heads**, that reside in parallel layers at the same depth in a model, each with its own set of parameters
  - Each head can learn different aspects of the relationships among inputs at the same level of abstraction
- The $\mathbf{W}^O$ linear projection reshapes the multihead self-attention to the original output dimension for each token
- Some dimensions
  - Typical size of $N = 1024, 2048, 4096$
  - d is used for the input and output ($d = 512$ in the original transformer paper)
  - the key and query embeddings have dimensionality d_k, and the value embeddings are of dimensionality d_v ($d_k = d_v = 64$ in the original transformer paper)
  - h is the number of heads ($h = 8$ in the original transformer paper)

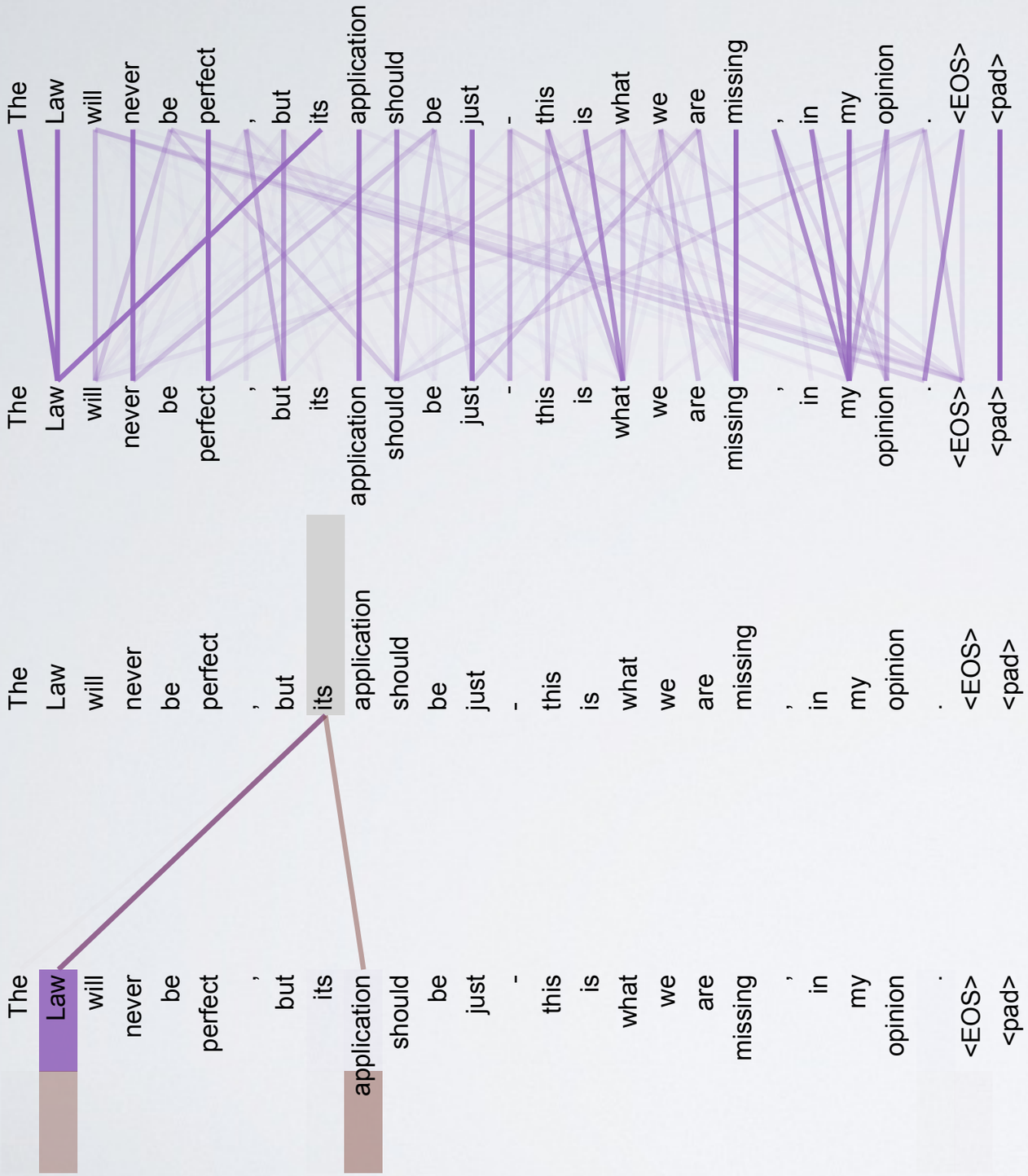
Self-Attention: Examples of Actual Distribution



- An example of the attention mechanism following **long-distance dependencies** in the encoder self-attention in layer 5 of 6
- Many of the attention heads **attend to a distant dependency** of the verb `making`, completing the phrase `making...more difficult`
  - Attentions here shown only for the word `making`. Different colors represent different heads of the multihead attention

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention Is All You Need. *arXiv.org, Computation and Language (cs.CL)*, arXiv:1706.03762.

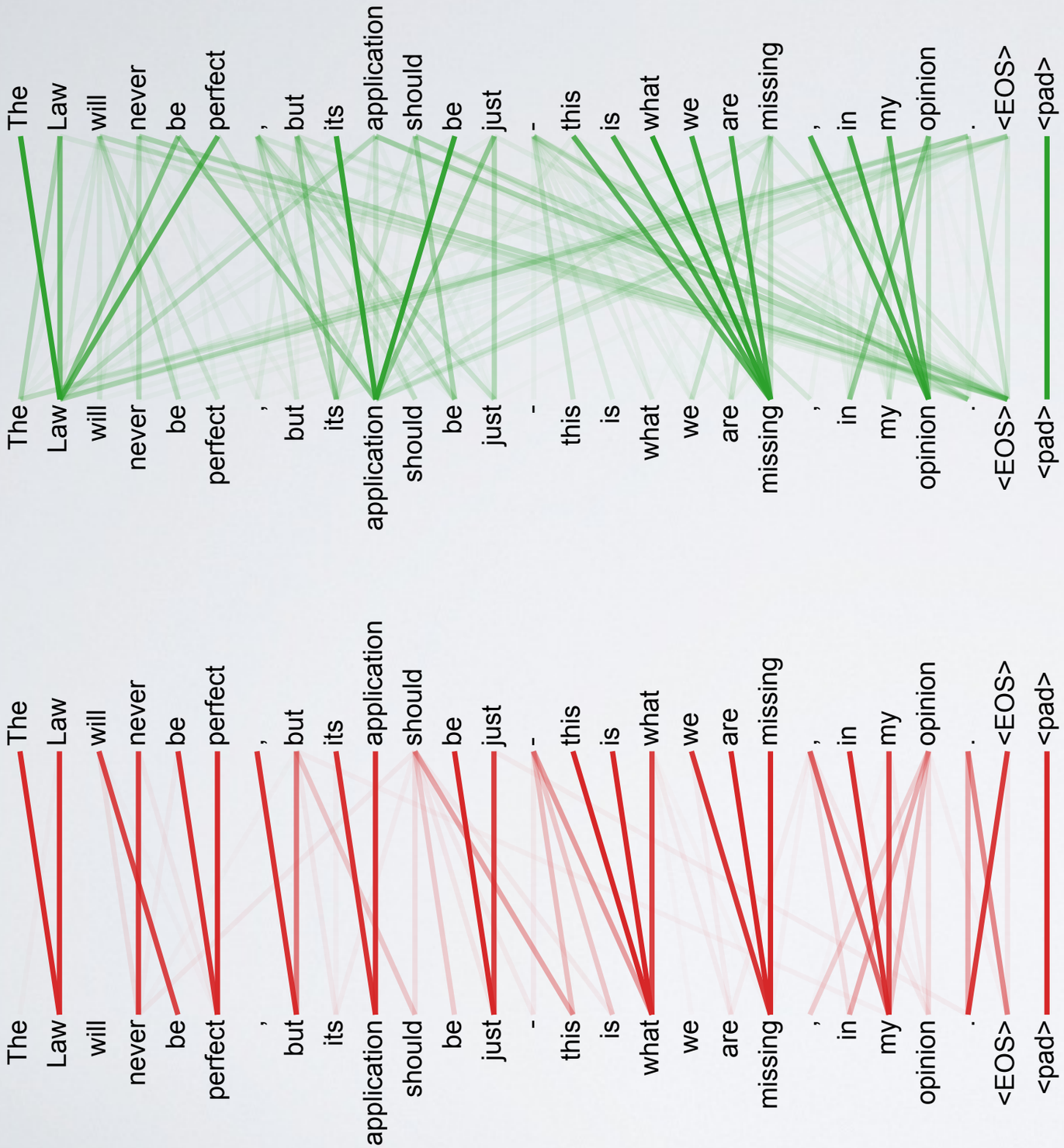
Self-Attention: Examples of Actual Distribution



- Two attention heads, also in layer 5 of 6, apparently involved in **anaphora resolution**
- Top: Full attentions for head 5
- Bottom: Isolated attentions from just the word `its` for attention heads 5 and 6
  - Note that the attentions are very sharp for this word.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention Is All You Need. *arXiv.org, Computation and Language (cs.CL)*, arXiv:1706.03762.

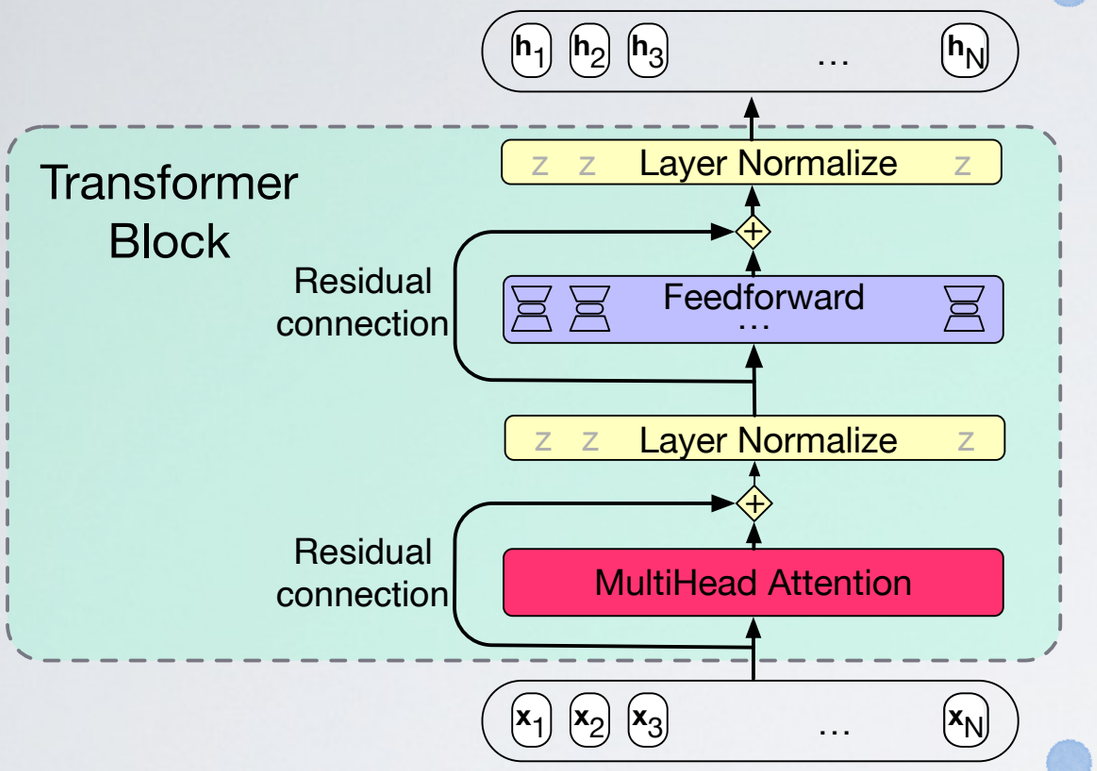
Self-Attention: Examples of Actual Distribution



- Many of the attention heads exhibit behaviour that seems related to the **structure of the sentence**
- We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6
  - The heads clearly learned to perform different tasks.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention Is All You Need. *arXiv.org, Computation and Language (cs.CL)*, arXiv:1706.03762.

The Transformer Block



$$\mu = \frac{1}{d}\sum_{k=1}^{d}\mathbf{x}[k]$$

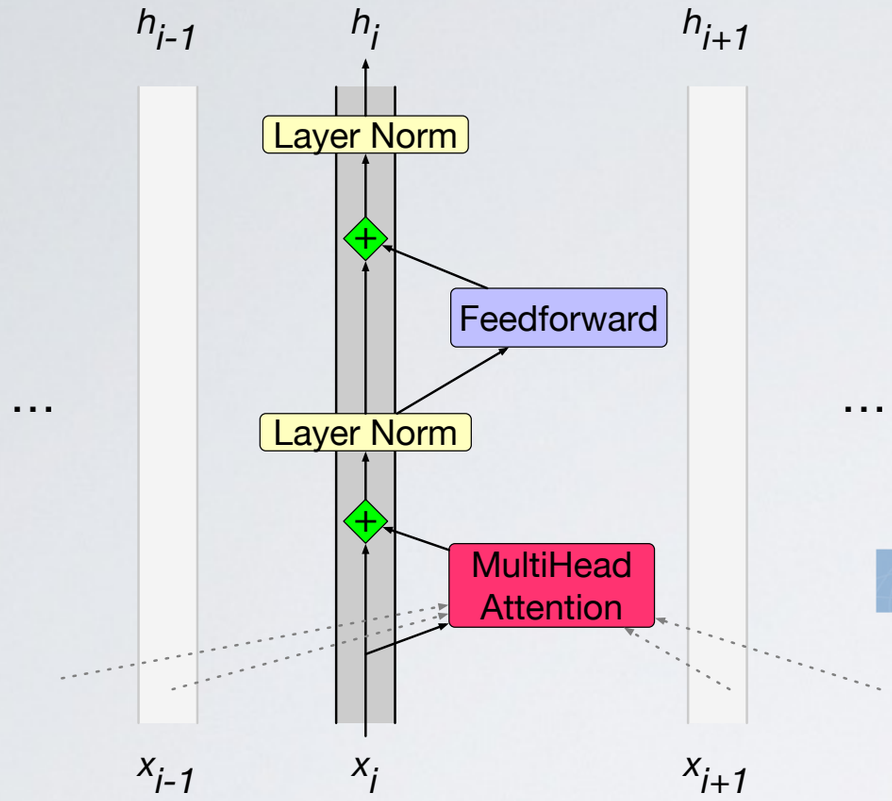
$$\sigma = \sqrt{\frac{1}{d-1}\sum_{k=1}^{d}(\mathbf{x}[k]-\mu)^2}$$

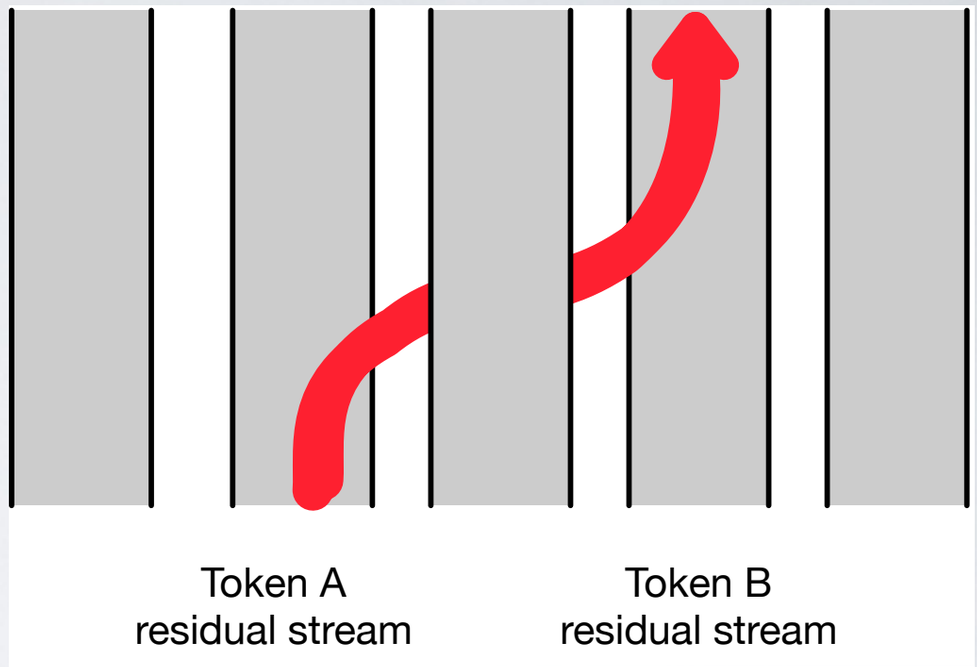
$$\hat{\mathbf{x}} = \frac{\mathbf{x}-\mu}{\sigma}$$

$$\mathrm{LayerNorm}(\hat{\mathbf{x}}) = \gamma\hat{\mathbf{x}} + \beta$$

- **Feedforward layer**: contains N position-wise networks, one for each input position
  - Each is a fully-connected 2-layer network, i.e., one hidden layer, two weight matrices
  - The weights are the same for each position, but the parameters are different from layer to layer
  - The feedforward networks are independent for each position and so can be computed in parallel
  - It is common to make the dimensionality d_{ff} of the hidden layer of the feedforward network be larger than the model dimensionality ($d_{\text{ff}} = 2048$ in the original transformer paper)
- **Residual connections**: are connections that pass information from a lower layer to a higher layer without going through the intermediate layer
  - Allowing information from the activation going forward and the gradient going backwards to skip a layer improves learning and gives higher level layers direct access to information from lower layers
- **Layer Norm**: is one of many forms of normalization that can be used to improve training performance in deep neural networks by keeping the values of a hidden layer in a range that facilitates gradient-based training

The Residual Stream in the Transformer Block





- The input at the bottom of the stream is an embedding for a token. That initial embedding is passed up by the residual connections and the outputs of feedforward and attention layers get added into it.
- The residual layers are constantly copying information up from earlier embeddings (hence the metaphor of '**residual stream**'), so we can think of the **other components as adding new views** of this representation back into this constant stream
  - Feedforward networks add in a different view of the earlier embedding
  - The only component that takes as input information from other tokens (other residual streams) is multi-head attention
- Elhage et al. (2021) show that we can view attention heads as literally **moving attention from the residual stream of a neighboring token into the current stream**
  - The high-dimensional embedding space at each position thus contains information about the current token and about neighboring tokens, albeit in different subspaces of the vector space

Elhage, N., Na, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernez, D., Jones, y., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McClish, S., and Olah, C. (2021). A Mathematical Framework for Transformer Circuits. Technical report, Anthropic, https://transformer-circuits.pub/2021/framework/.

Subword Tokenization

```
sub ##word token ##ization , because token ##s
can be part of words as well as whole words
```

[WordPiece for BERT]

- Instead of white-space segmentation (or any other more sophisticated approach), **use the data** to tell us how to tokenize
  - **Subword tokenization**, because tokens can be part of words as well as whole words
- Subword tokenization **reduces the vocabulary size** and allows for dealing with **out-of-vocabulary words**
- Three common algorithms:
  - Byte-Pair Encoding (BPE) [Sennrich et al., 2016]
  - Unigram language modeling tokenization [Kudo, 2018]
  - WordPiece (Schuster and Nakajima, 2012)
- All have two parts
  - A **token learner** that takes a raw training corpus and induces a **vocabulary** (a **set of tokens**)
  - A **token segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary (typically indexes to tokens in the vocabulary)

Kudo, T. (2018). Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In Gurevych, I. and Miyao, Y., editors, *Proc. of the 56th Annual Meeting of the Association for Computational Linguistics (ACL 2018)*, pages 66–75. Association for Computational Linguistics, USA.

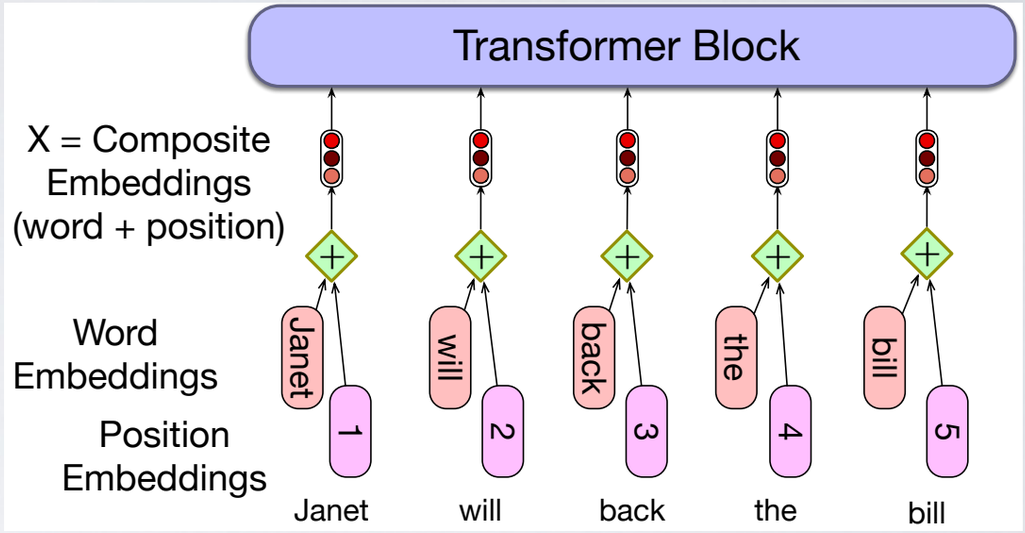
Kudo, T. and Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Process- ing. In Riloff, E., Chiang, D., Hockenmaier, J., and Tsujii, J., editors, Proc. of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP 2018), pages 66–71. Association for Computational Linguistics, USA.

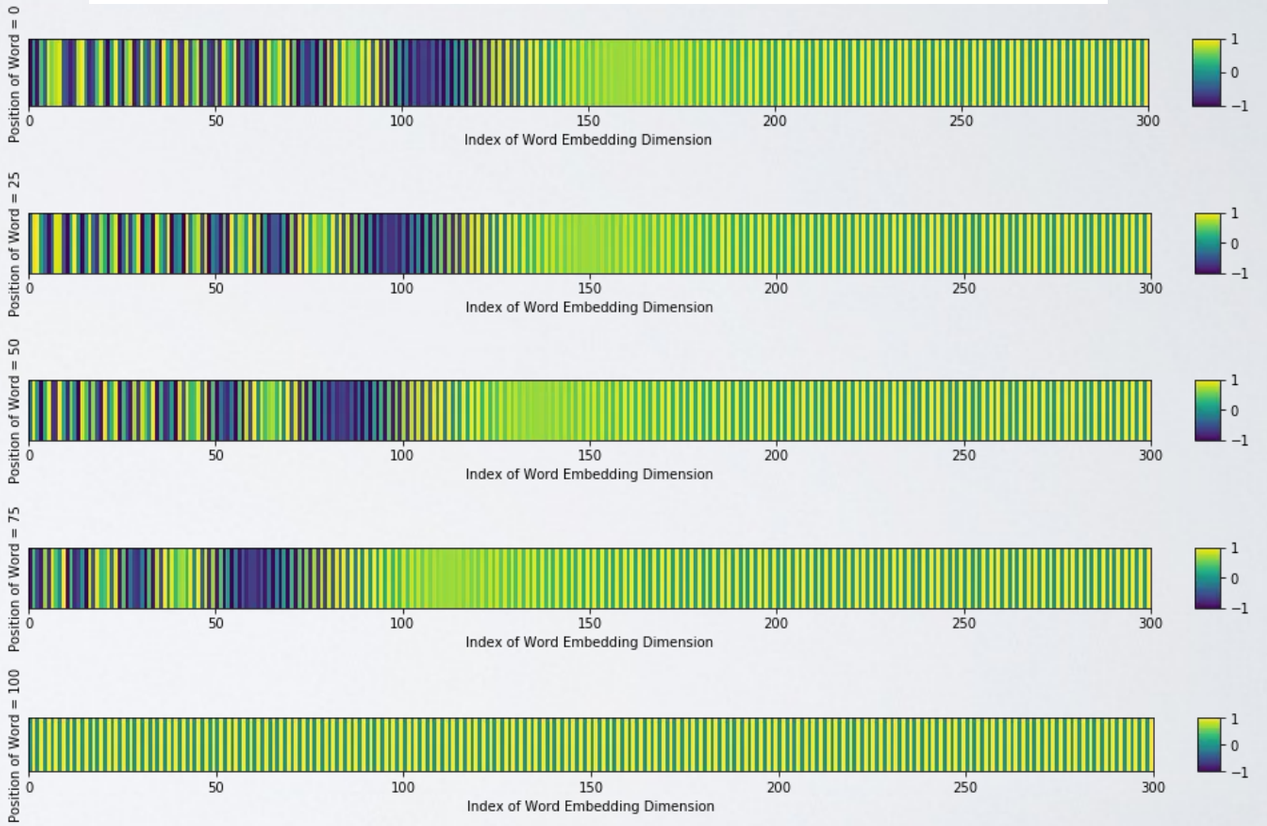
Schuster, M. and Nakajima, K. (2012). Japanese and Korean voice search. In Sakai, H., Nishitani, T., Sugiyama, A., and Kiya, H., editors, *Proc. 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2012)*, pages 5149–5152.

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural Machine Translation of Rare Words with Subword Units. In Erk, K. and Smith, N. A., editors, *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*, pages 1715–1725. Association for Computational Linguistics, USA.

Input Embeddings for Token and Position

- Given an input string, we first convert the **tokens** into vocabulary indices, e.g. using BPE or WordPiece
  - Next we use indexing to select the corresponding rows from the embedding matrix $\mathbf{E}$ for the whole vocabulary
- **Token embeddings** are not position-dependent. To represent the position of each token in the sequence, we combine these token embeddings with **positional embeddings** specific to each position in an input sequence
  - IN RNNs (and FF networks) the position is implicit
- We can use **absolute position**, i.e. a random embedding for each position
  - A potential problem is that there will be plenty of training examples for the initial positions in our inputs and correspondingly fewer at the outer length limits. These latter embeddings may be poorly trained and may not generalize well during testing
- As an alternative, we can use **relative position**, i.e. a function that captures the fact that position 4 in an input is more closely related to position 5 than it is to position 17
  - A combination of sine and cosine functions with differing frequencies was used in the original transformer work

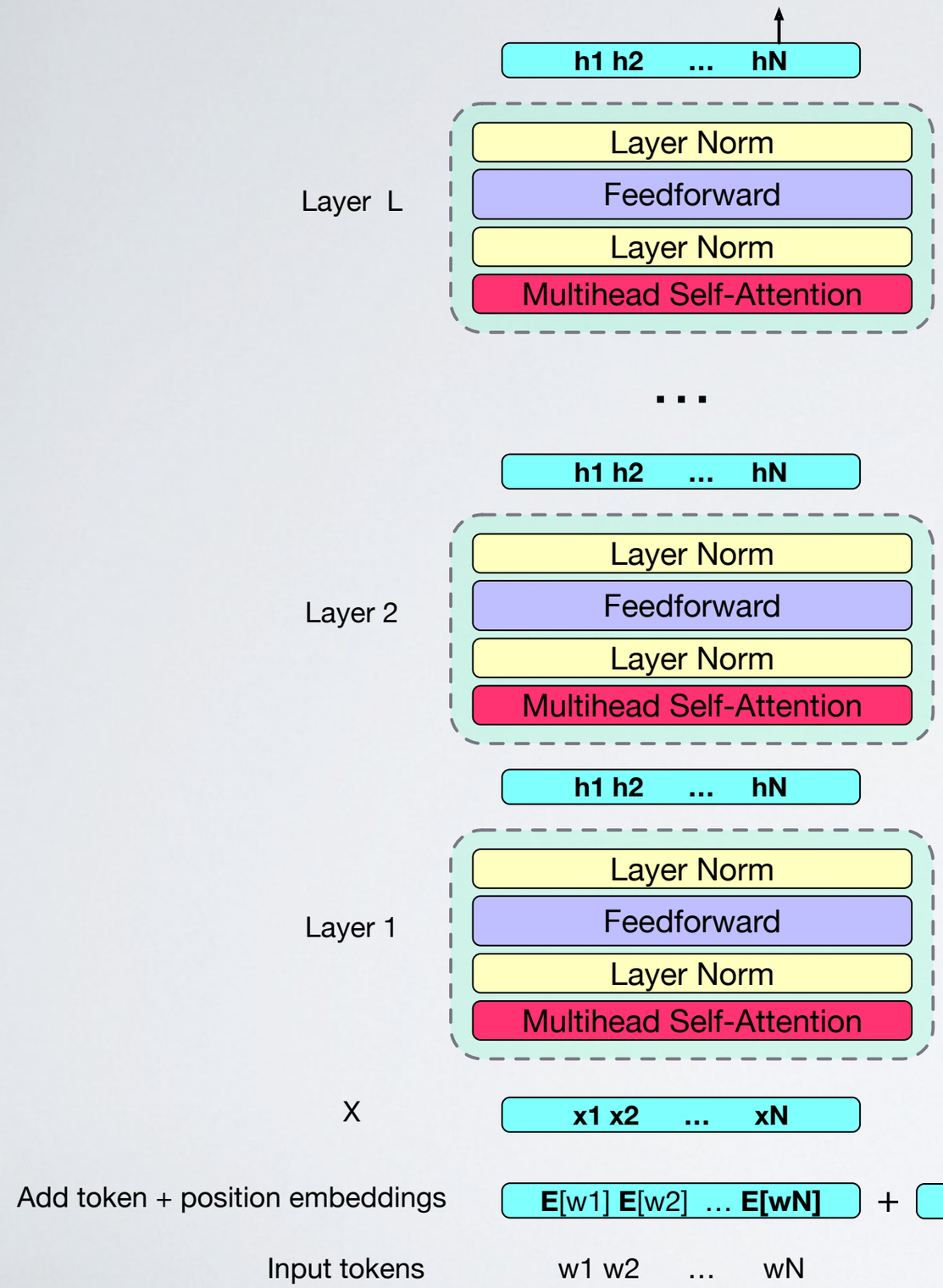




$$\mathbf{pe}_i[k] = \begin{cases} \sin\left(\frac{1}{10000^{\frac{k}{d}}} i\right) & \text{if } k \text{ even} \\ \cos\left(\frac{1}{10000^{\frac{k-1}{d}}} i\right) & \text{if } k \text{ odd} \end{cases}$$

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

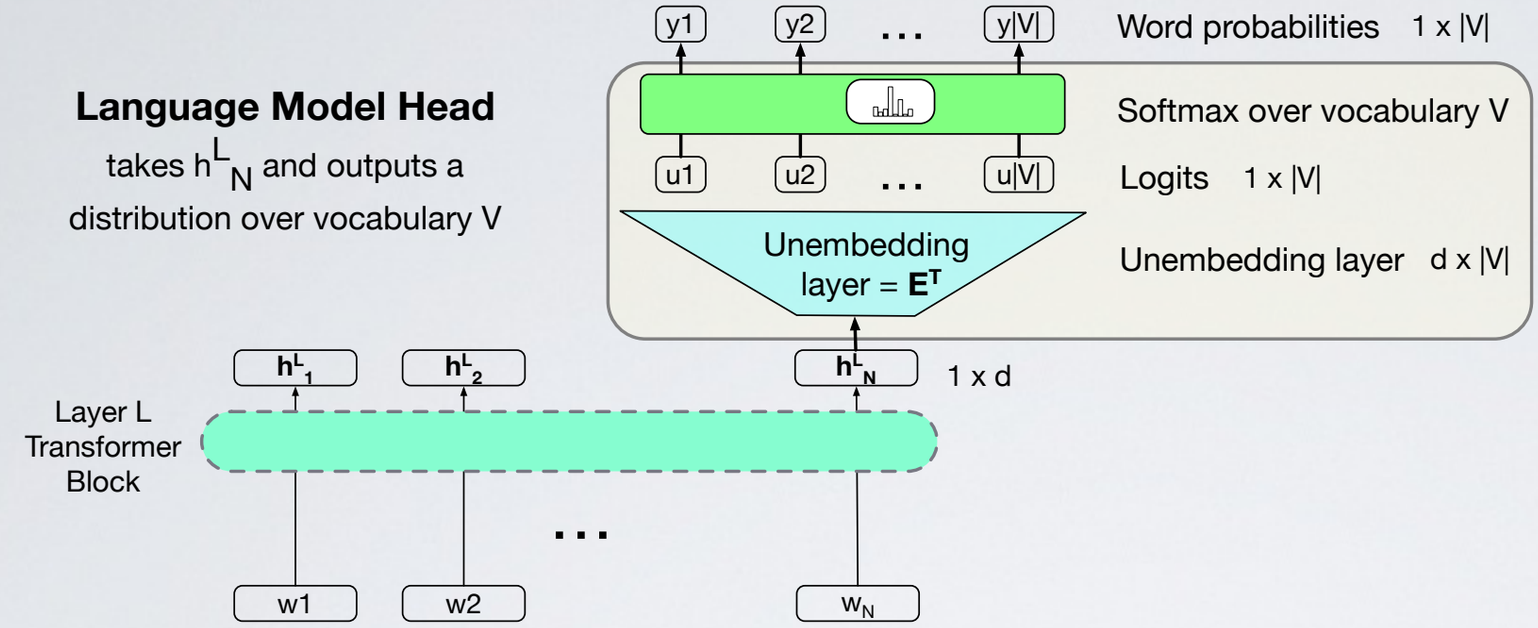
Putting All Together



Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(N^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(N \cdot d^2)$	$O(N)$	$O(N)$
Convolutional	$O(k \cdot N \cdot d^2)$	$O(1)$	$O(\log_k N)$

- **Total computational complexity per layer**: self-attention layers are faster than recurrent layers when the sequence length N is smaller than the representation dimensionality d
- **Sequential operations**: the amount of computation that can be parallelized
- **Length between long-range dependencies**: the shorter the length of the paths between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies

The Language Modeling Head

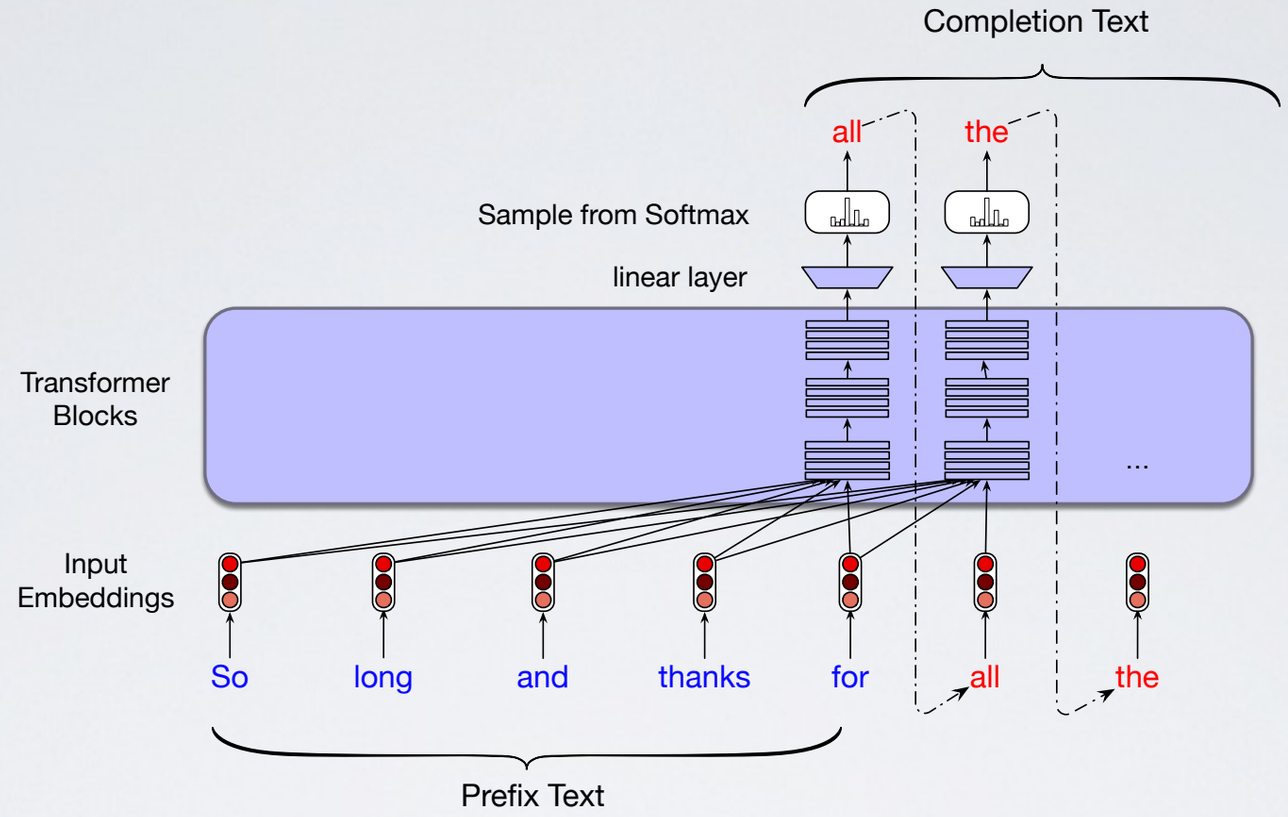


$$\mathbf{u} = \mathbf{h}_N^L \mathbf{E}^T$$

$$\mathbf{y} = \text{softmax}(\mathbf{u})$$

- The **language modeling head** takes the the output of the final transformer layer L from the last token N and use it to predict the upcoming word at position $N+1$
- The linear layer projects from the output $\mathbf{h}_N^L$ to the **logit** $\mathbf{u}$
  - It is called logit is because the `logit` function is the inverse of the `sigmoid` and it is the log of the odds ratio
  - Using the term logit for $\mathbf{u}$ is a way of reminding us that by using the `sigmoid` to turn $\mathbf{u}$ into a probability, we are implicitly interpreting $\mathbf{u}$ as not just any real-valued number, but as specifically a **log odds**
- This linear layer could be learned but, more commonly, we tie this matrix to (the transpose of) the embedding matrix $\mathbf{E}$, a technique called **weight tying**
- The transpose $\mathbf{E}^T$ is called the **unembedding layer** because it is performing this reverse mapping back from an embedding (shape $[1 \quad d]$) to a vector over the vocabulary (shape $[1 \quad |V|]$)
  - In the learning process, $\mathbf{E}$ will be optimized to be good at doing both of these mappings

Large Language Models (LLM) with Transformer



- **Conditional generation** is the task of generating text conditioned on an input piece of text, a **prompt**
  - The fact that transformers have such **long contexts** (1024 or even 4096 tokens) makes them very powerful for conditional generation, because they can look back so far into the prompting text
  - As the generation process proceeds, the model has direct access to the **priming context** as well as to all of **its own subsequently generated outputs**
- The task of choosing a word to generate based on the model's probabilities is called **decoding**
  - Hence also the terminology **decoder-only model**
- Decoding from a language model in a left-to-right manner (or right-to-left for languages like Arabic in which we read from right to left), and thus repeatedly choosing the next word conditioned on our previous choices (of the model itself) is called **autoregressive generation** or **causal LM generation**

LLM: Generation by Sampling

- In **greedy decoding**, the output is chosen by computing the probability for each possible outputs (every word in the vocabulary) and then choosing the highest probability word (the `argmax`)
  - The chosen words are (by definition) extremely predictable and so the resulting **text** is **generic** and often quite **repetitive**
  - If the context is identical, and the probabilistic model is the same, greedy decoding will always result in generating exactly the same string, thus it is **deterministic**

$$\hat{w}_t = \underset{w \in V}{\operatorname{argmax}} \mathbb{P}(w | \mathbf{w}_{<t})$$

- In **random sampling**, at each step we sample words according to their probability conditioned on the previous choices, and we use a transformer language model as the probability model that tells us this probability
  - The problem is that even though random sampling is mostly going to generate sensible, high-probable words, there are many odd, low probability words in the tail of the distribution, and even though each one is low probability, **if you add up all the rare words, they constitute a large enough portion of the distribution that they get chosen often** enough to result in generating weird sentences

$\sim$ stands for sampling a word from the distribution $p(\cdot)$

$$\begin{aligned}
&\mathrm{i} \leftarrow 1 \\
&w_i \sim \mathrm{p}(w) \\
&\textbf{while } w_i \text{ != EOS} \\
&\quad \mathrm{i} \leftarrow \mathrm{i} + 1 \\
&\quad w_i \sim \mathrm{p}(w_i \mid w_{<i})
\end{aligned}$$

Large Language Models: Generation by Sampling

- **Top-k sampling** is a generalization of greedy decoding. Instead of choosing the single most probable word to generate, we first **truncate** the **distribution** to the **top k most likely words**, renormalize to produce a legitimate probability distribution, and then randomly sample from within these k words according to their renormalized probabilities
  - When k = 1 it is the same as greedy decoding
  - One problem with top-k sampling is that k is fixed, but the shape of the the probability distribution over words differs in different contexts
  - If we set k = 10, sometimes the top 10 words will be very likely and include most of the probability mass, but other times the probability distribution will be flatter and the top 10 words will only include a small part of the probability mass
- **Top-p sampling** or **nucleus sampling** is to keep not the top k words, but the **top p percent of the probability mass**
  - By measuring probability rather than the number of words, the hope is that the measure will be more robust in very different contexts, dynamically increasing and decreasing the pool of word candidates
  - Given a distribution $\mathbb{P}(w|\mathbf{w}_{<t})$ the **top-p vocabulary** $V^{(p)}$ is the smallest set of tokens such that

$$\sum_{w \in V^{(p)}} \mathbb{P}(w|\mathbf{w}_{<t}) \geq p$$

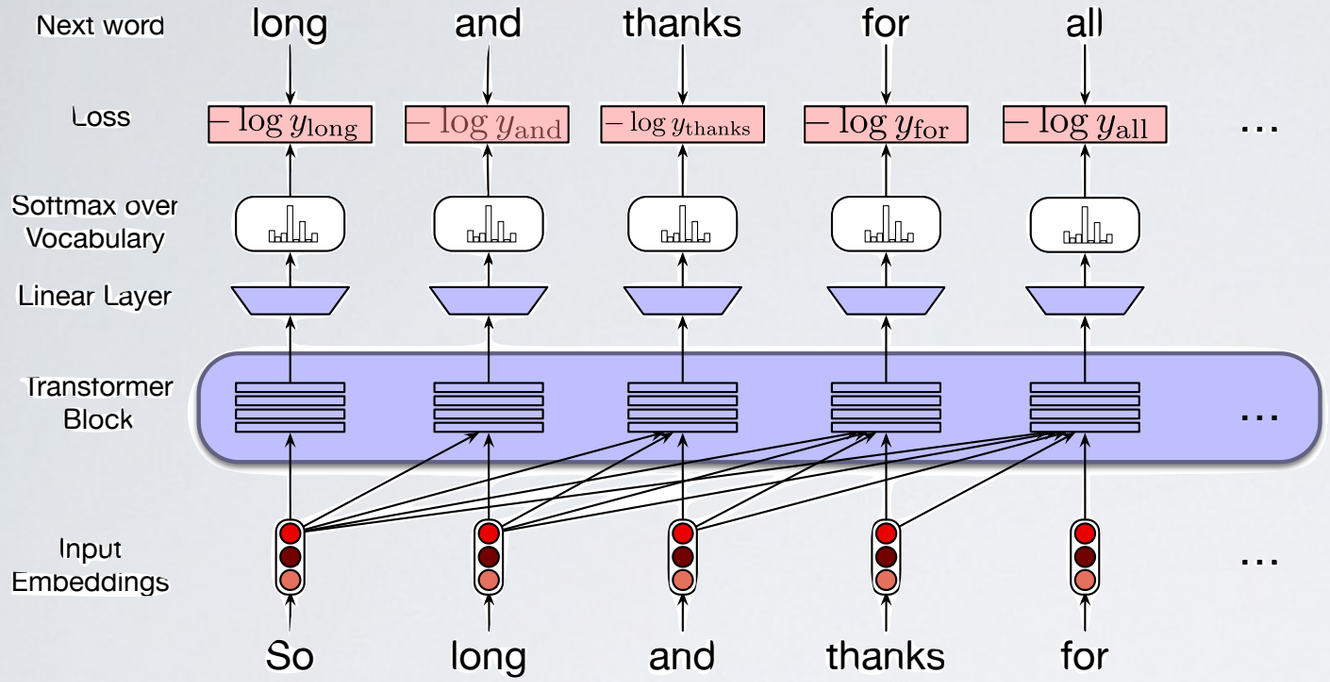
Large Language Models: Generation by Sampling

$$\mathbf{y} = \text{softmax}\left(\frac{\mathbf{u}}{\tau}\right), \quad \begin{cases} \tau \in (0, 1] & \text{low temperature} \\ \tau > 1 & \text{high temperature} \end{cases}$$

- In **temperature sampling**, we do not truncate the distribution, but instead **reshape** it
- In **low-temperature sampling**, we smoothly increase the probability of the most probable words and decrease the probability of the rare words, making it greedy
  - `softmax` tends to push high values toward 1 and low values toward 0
- In **high-temperature sampling**, we flatten the word probability distribution, making it easier to sample from different words
  - Infinite temperature becomes equivalente to uniform sampling
- The intuition for temperature sampling comes from thermodynamics, where a system at a high temperature is very flexible and can explore many possible states, while a system at a lower temperature is likely to explore a subset of lower energy (better) states

Holtzman, A., Buys, J., Du, K., Forbes, M., and Choi, Y. (2020). The Curious Case of Neural Text Degeneration. In Rush, A., Mohamed, S., Song, D., Cho, K., and White, M., editors, *Proc. 8th International Conference on Learning Representations (ICLR 2020)*. OpenReview.net, https://openreview.net/group?id=ICLR.cc/2020/Conference.

Large Language Models: Training



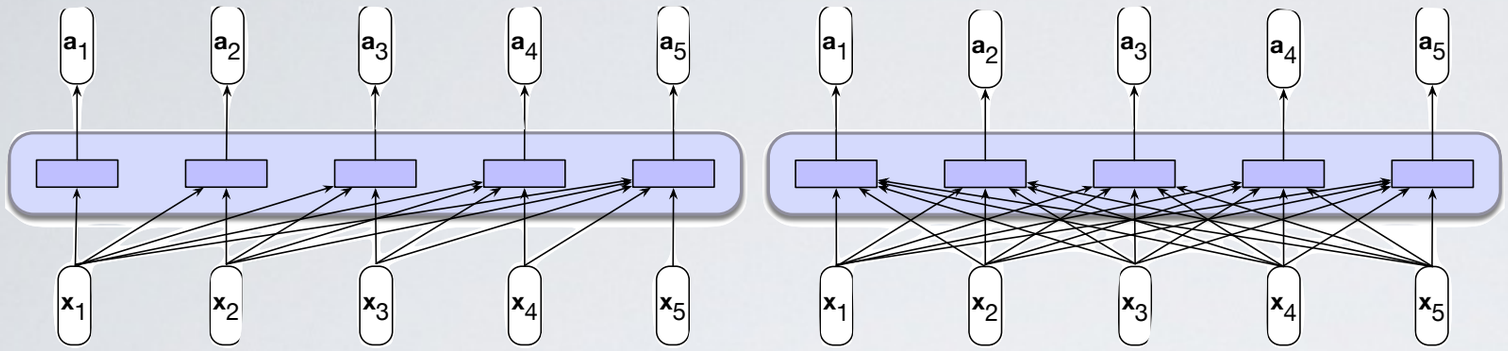
$$= \frac{1}{T}\sum_{t=1}^{T} L_{CE}$$

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

- At each step, given all the preceding words, the final transformer layer produces an output distribution over the entire vocabulary. During training, the probability assigned to the correct word is used to calculate the **cross-entropy loss** for each item in the sequence
  - As with RNNs, the loss for a training sequence is the **average cross-entropy loss** over the entire sequence via gradient descent
  - We use **teacher forcing**, i.e. we always give the model the correct history sequence to predict the next word
- In **RNNs** the calculation of the outputs and the losses at each step us inherently serial given the recurrence in the calculation of the hidden states. With **transformers**, each training item can be processed in parallel since the output for each element in the sequence is computed separately
- Large models are generally trained by filling the full context window (for example 2048 or 4096 tokens for GPT3 or GPT4)
  - If documents are shorter than this, multiple documents are packed into the window with a special end-of-text token between them
  - The batch size for gradient descent is usually quite large (the largest GPT-3 model uses a batch size of 3.2 million tokens)

Encoder Model Transformer: Bidirectional Self-Attention

Bidirectional Encoders



a) A causal self-attention layer

b) A bidirectional self-attention layer

q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

N

q1•k1	q1•k2	q1•k3	q1•k4	q1•k5
q2•k1	q2•k2	q2•k3	q2•k4	q2•k5
q3•k1	q3•k2	q3•k3	q3•k4	q3•k5
q4•k1	q4•k2	q4•k3	q4•k4	q4•k5
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

N

- **Bidirectional encoder models** use the same self-attention mechanism as causal models but they allow the self-attention mechanism to range over the entire input

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q; \mathbf{K} = \mathbf{X}\mathbf{W}^K; \mathbf{V} = \mathbf{X}\mathbf{W}^V$$

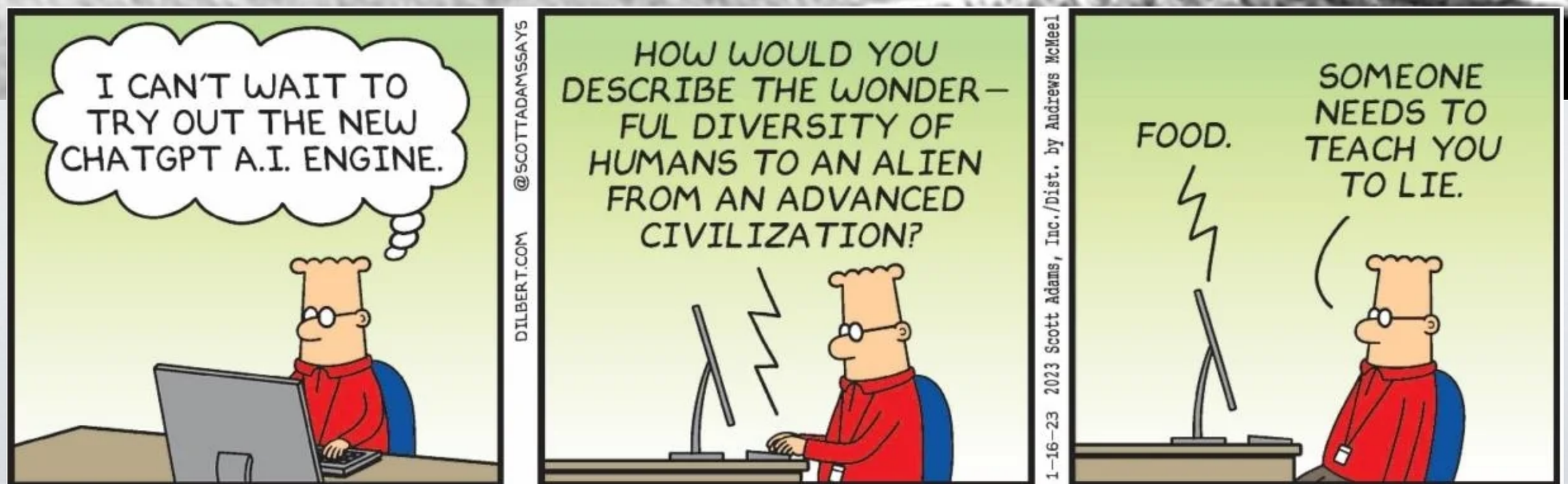
$$\mathbf{head}_j = \text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

BERT

- BERT [Devlin et al., 2019] consisted of the following:
  - An English-only subword vocabulary consisting of 30,000 tokens generated using the WordPiece algorithm
  - Fixed input size of 512 subword tokens was used
  - Hidden layers of size of 768
  - 12 layers of transformer blocks, with 12 multihead attention layers each
  - The resulting model has about 100M parameters.
- The larger multilingual XLM-RoBERTa model [Conneau et al., 2020], trained on 100 languages, has
  - A multilingual subword vocabulary with 250,000 tokens generated using the SentencePiece Unigram LM algorithm
  - Fixed input size of 512 subword tokens was used
  - 24 layers of transformer blocks, with 16 multihead attention layers each
  - Hidden layers of size 1024
  - The resulting model has about 550M parameters.

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., and Stoyanov, V. (2020). Unsupervised Cross-lingual Representation Learning at Scale. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J., editors, *Proc. of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*, pages 8440–8451. Association for Computational Linguistics, USA.
Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL 2019)*, pages 4171–4186. Association for Computational Linguistics, USA.

Questions?


I CAN'T WAIT TO TRY OUT THE NEW CHATGPT A.I. ENGINE.
DILBERT.COM @SCOTTADAMSSAYS
HOW WOULD YOU DESCRIBE THE WONDER-FUL DIVERSITY OF HUMANS TO AN ALIEN FROM AN ADVANCED CIVILIZATION?
1-16-23 2023 Scott Adams, Inc./Dist. by Andrews McMeel
FOOD.
SOMEONE NEEDS TO TEACH YOU TO LIE.