JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Building Our First Neural LM

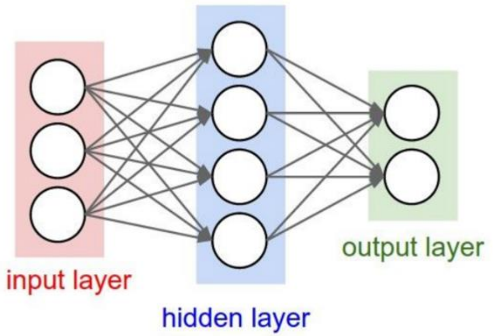
CSCI 601-471/671 (NLP: Self-Supervised Models)

https://self-supervised.cs.jhu.edu/sp2025/

Logistics Reminders

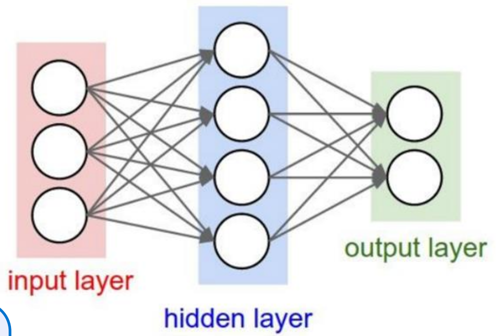
- **Quiz 1:** next Tuesday
  - During the class (~1:15 mins)
  - All on paper
  - Closed-book (no formula sheet)
  - Content: everything we discuss before the class (before this slide)

Recap: Neural Nets



- A powerful function-approximation tool.
- Can be trained efficiently via Backpropagation.
- Out focus here: how to use NNs for language modeling.

Big Picture: Language Modeling + NNs



How do MLPs are for Language modeling?

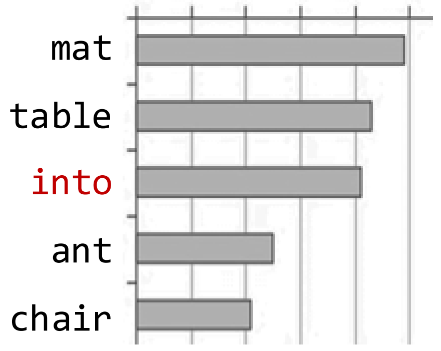
Remember NNs expect numbers. How do you feed a string to neural net?

$$\underbrace{f(x, \Theta)}$$

and our problems turning | into

context words in window of size 4 | target word

Probs over vocabulary



Building First Neural LMs

1. Fixed-window neural language models
2. Atomic units of language

Chapter goal: Get more comfortable with thinking about the role of neural networks in modeling distribution of language.

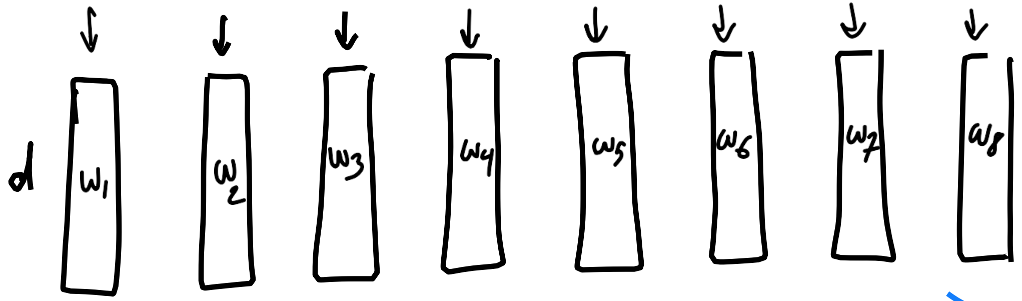
Feeding Text to Neural LMs

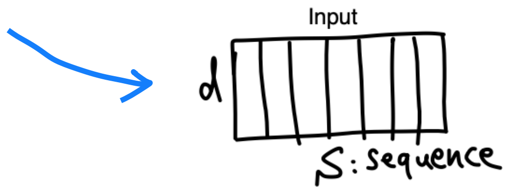
Feeding Text to Neural Nets

- Neural Nets expect numbers.
- How do you turn numbers into numbers?

Feeding Text to Neural Nets

```
['Hello', ',', 'world', '!', "How's", 'it', 'going', '?']
```

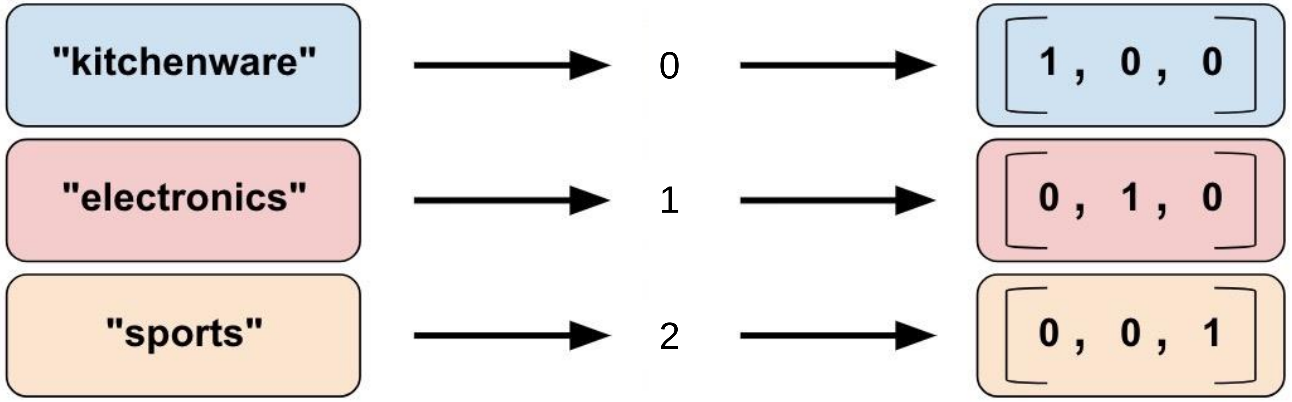




- Associate each word with a randomly initialized vector.
- Pass the vector as input to the model.
- One can initialize these vectors with more informative values (e.g. Word2Vec).
  - Not used in practice.

Feeding Text to Neural Net: In Practice

- In practice this is implemented in this way:
  1. Turn each word into a unique index
  2. Map each index into a one-hot vector



Feeding Text to Neural Net: In Practice

- In practice this is implemented in this way:
  1. Turn each word into a unique index
  2. Map each index into a one-hot vector
  3. Lookup the corresponding word embedding via matrix multiplication

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 8 & 2 & 1 & 9 \\ 6 & 5 & 4 & 0 \\ 7 & 1 & 6 & 2 \\ 1 & 3 & 5 & 8 \\ 0 & 4 & 9 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 & 8 \end{bmatrix}$$

One-hot vector

Embedding Weight Matrix

Hidden layer output

Question: what is the size of this embedding matrix?

Note, this embedding matrix is a trainable parameter of the model.

Feeding Text to Neural Net: PyTorch

Initialize a random embedding matrix

Indices corresponding to input units (tokens)

Embeddings corresponding to the inputs

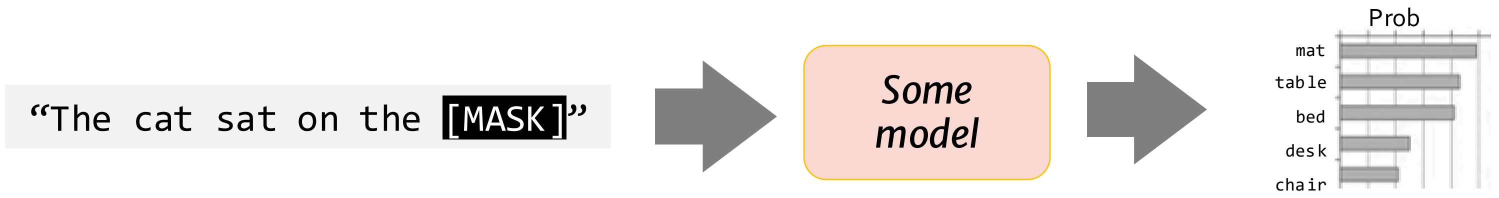
```
# an Embedding module containing 10 tensors of size 3
n, d = 10, 3
embedding = nn.Embedding(n, d)
# a batch of 2 samples of 4 indices each
input = torch.LongTensor([[1, 2, 4, 5], [4, 3, 2, 9]])
embedding(input)
tensor([[[-0.0251, -1.6902,  0.7172],
         [-0.6431,  0.0748,  0.6969],
         [ 1.4970,  1.3448, -0.9685],
         [-0.3677, -2.7265, -0.1685]],

        [[ 1.4970,  1.3448, -0.9685],
         [ 0.4362, -0.4004,  0.9400],
         [-0.6431,  0.0748,  0.6969],
```

Fixed-Window
MLP Language Models

Recap: LMs

$$\mathbf{P}(\overbrace{X_t}^{\text{next word}} \mid \overbrace{X_1, \ldots, X_{t-1}}^{\text{context}})$$

- Directly we train models on "conditionals":



Recap: Counting

$$\mathbf{P}(\underbrace{X_t}_{\text{next word}} | \underbrace{X_1, \dots, X_{t-1}}_{\text{context}})$$

How do we estimate these probabilities?
Let's just count!

$$\text{P(mat | the cat sat on the)} = \frac{\text{count(“the cat sat on the mat”)}}{\text{count(“the cat sat on the”)}}$$

Challenge: Increasing n makes sparsity problems worse.
Typically, we can't have n bigger than 5.

Some partial solutions (e.g., smoothing and backoffs) though still an open problem.

Recap Summary

- Language Models (LM): distributions over language

- N-gram: language modeling via counting

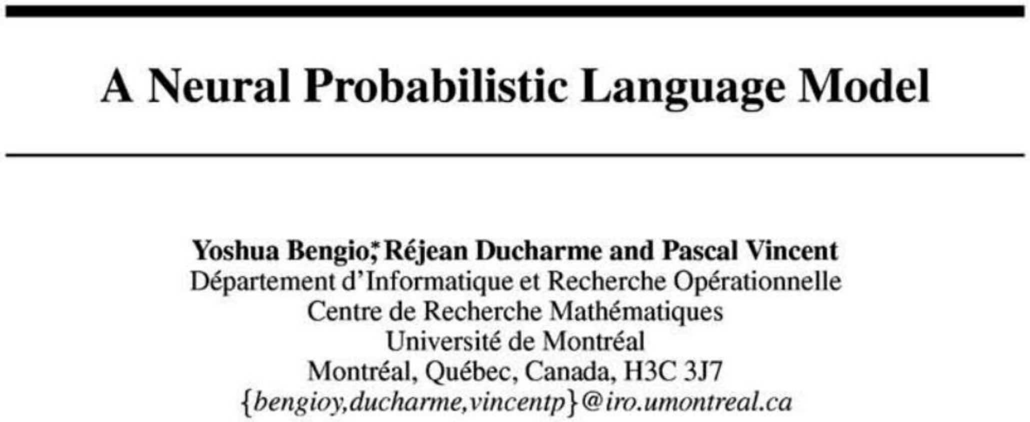
- Challenge with large N's: sparsity problem — many zero counts/probs.
- Challenge with small N's: not very informative and lack of long-range dependencies.

From Counting (N-Gram) to Neural Models

- Probabilistic n-gram models of text generation [Jelinek+ 1980’s, …]
  - Applications: Speech Recognition, Machine Translation

“Shallow” statistical/neural language models (2000’s) [Bengio+ 1999 & 2001, …]

NeurIPS 2000

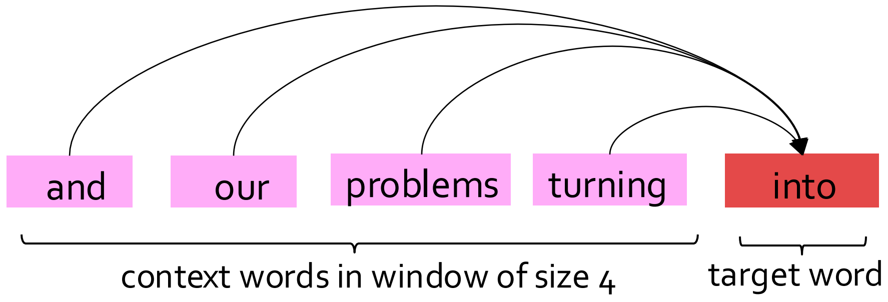


A Neural Probabilistic Language Model

Yoshua Bengio*, Réjean Ducharme and Pascal Vincent
Département d’Informatique et Recherche Opérationnelle
Centre de Recherche Mathématiques
Université de Montréal
Montréal, Québec, Canada, H3C 3J7
{bengioy,ducharme,vincentp}@iro.umontreal.ca

A Fixed-Window Neural LM

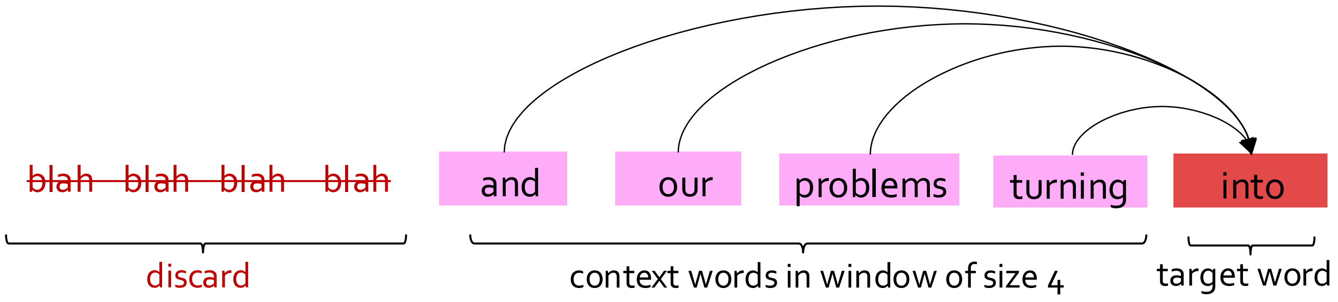
- Given the embeddings of the context, predict the word on the right side.
  - Dropping the right context for simplicity -- not a fundamental limitation.



A Fixed-Window Neural LM

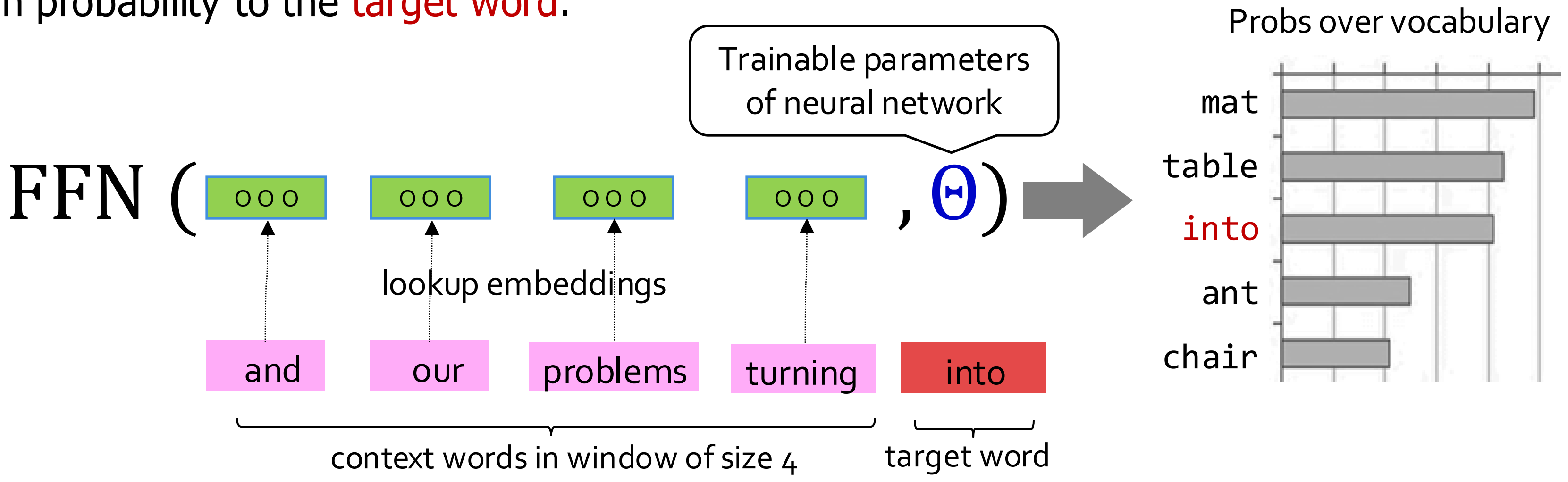
- Given the embeddings of the context, predict the word on the right side.
  - Dropping the right context for simplicity -- not a fundamental limitation.

- Discard anything beyond its context window



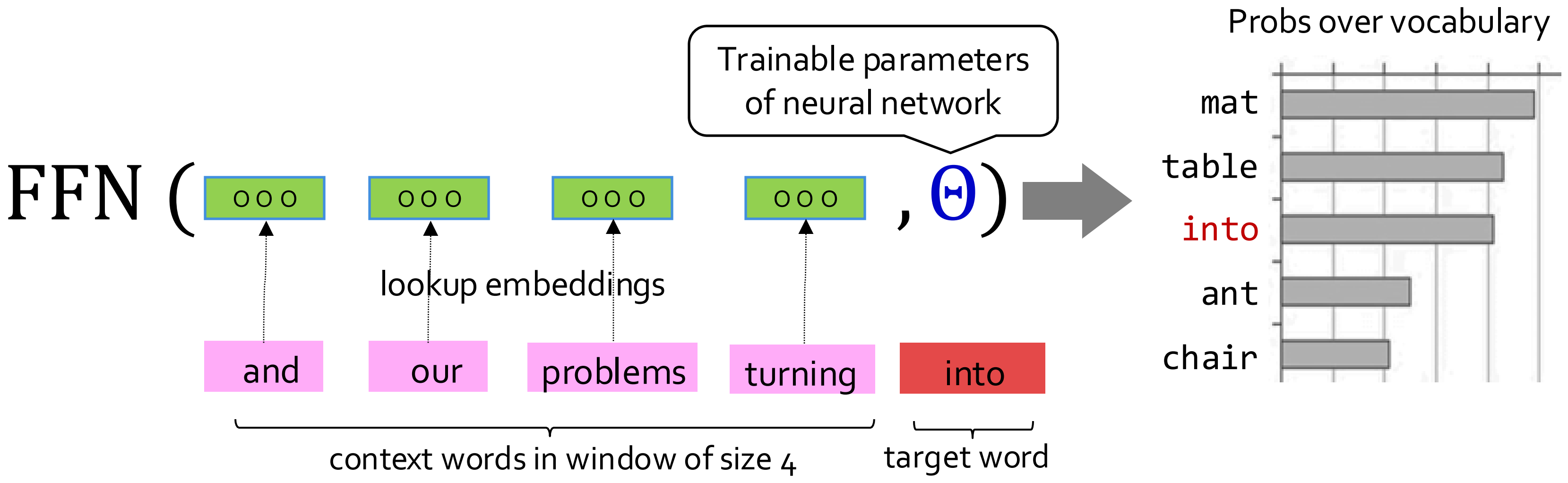
A Fixed-Window Neural LM

- Given the embeddings of the context, predict a target word on the right side.
  - Dropping the right context for simplicity -- not a fundamental limitation.
- Training this model is basically optimizing its parameters Θ such that it assigns high probability to the target word.



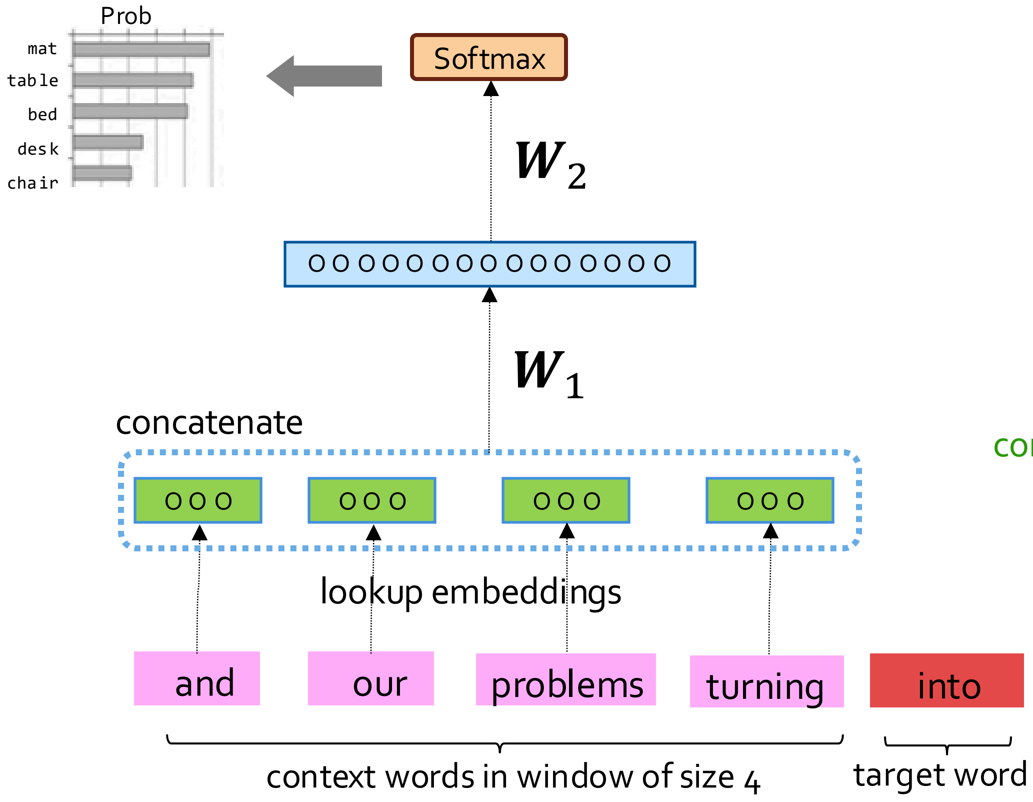
A Fixed-Window Neural LM

- This is actually a pretty good model!
- It will also lay the foundation for the future models (e.g., transformers, ...)
- But first we need to figure out how to train neural networks!


Probs over vocabulary
Trainable parameters of neural network
FFN (ooo ooo ooo ooo , Θ)
mat
table
into
ant
chair
lookup embeddings
and
our
problems
turning
into
context words in window of size 4
target word

[Bengio et al. 2003]

A Fixed-Window Neural LM



output distribution

$$y = \mathrm{softmax}(W_2 h)$$

hidden layer

$$h = f(W_1 x)$$

concatenated word embeddings

$$x = [v_1, v_2, v_3, v_4]$$

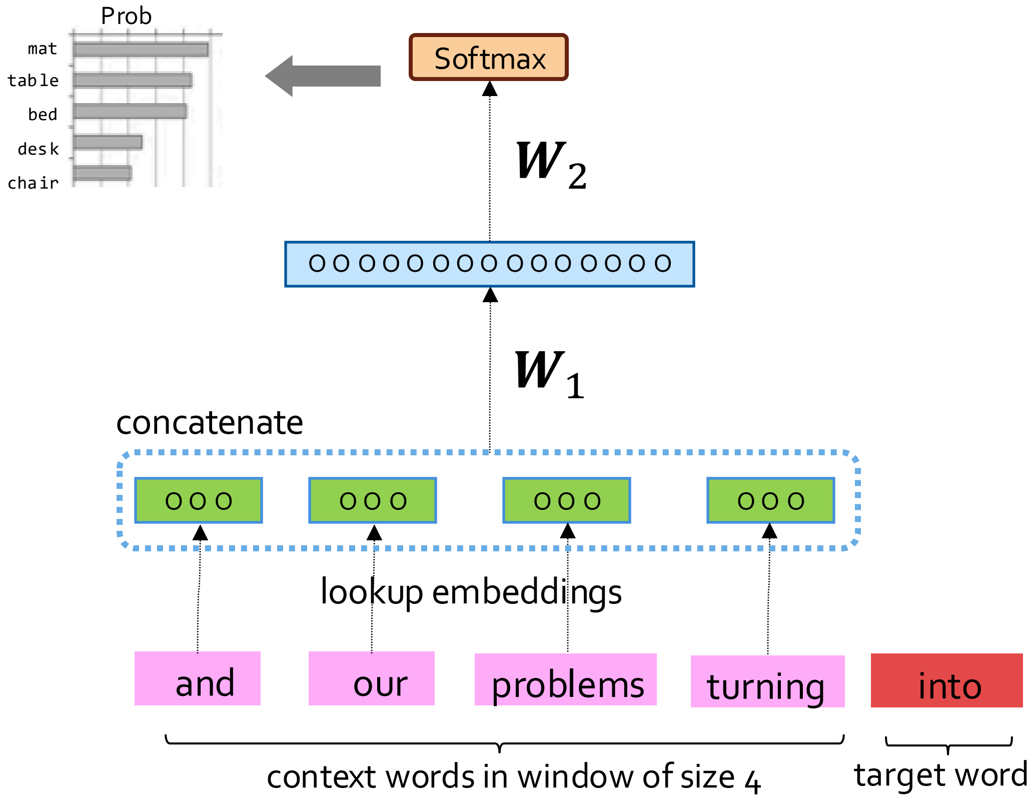
[Bengio et al. 2003]

A Fixed-Window Neural LM: Compared to N-Grams

Improvements over n-gram LM:

- Tackles the sparsity problem
- Model size is O(n) not O(exp(n)) — n being the window size.

	n	valid.	test.
MLP10	6	104	**109**
Back-off KN	3	121	127
Back-off KN	4	113	119
Back-off KN	5	112	**117**



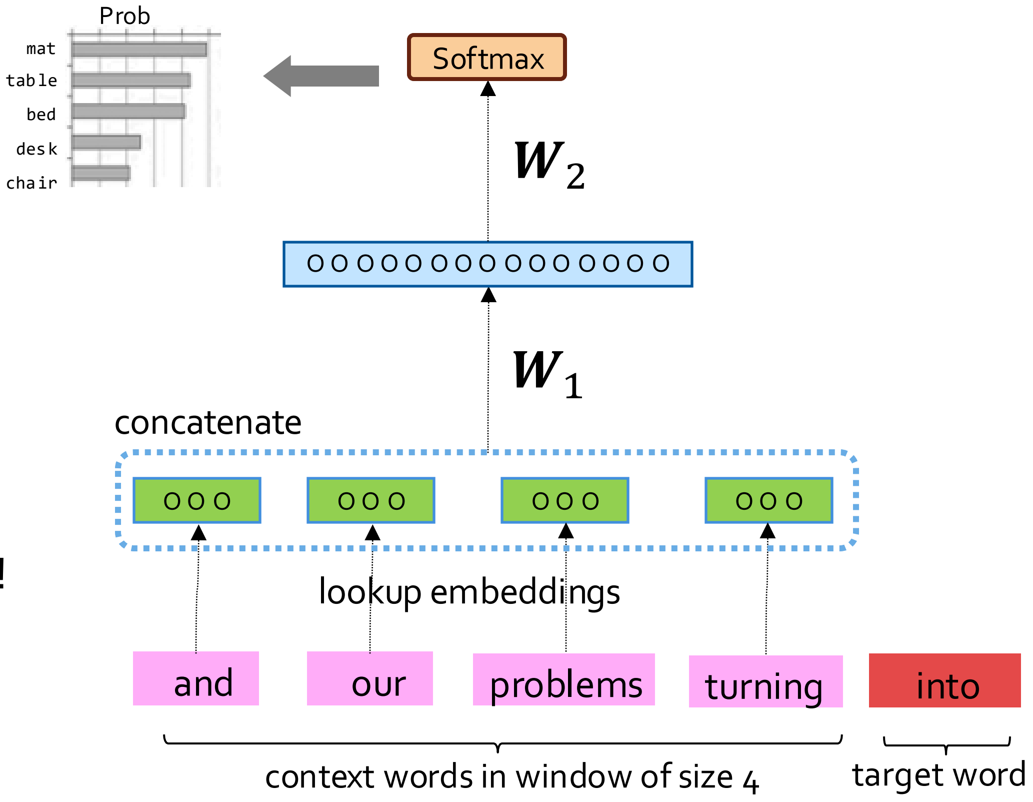
A Fixed-Window Neural LM: Compared to N-Grams

Improvements over n-gram LM:

- Tackles the sparsity problem
- Model size is O(n) not O(exp(n)) — n being the window size.

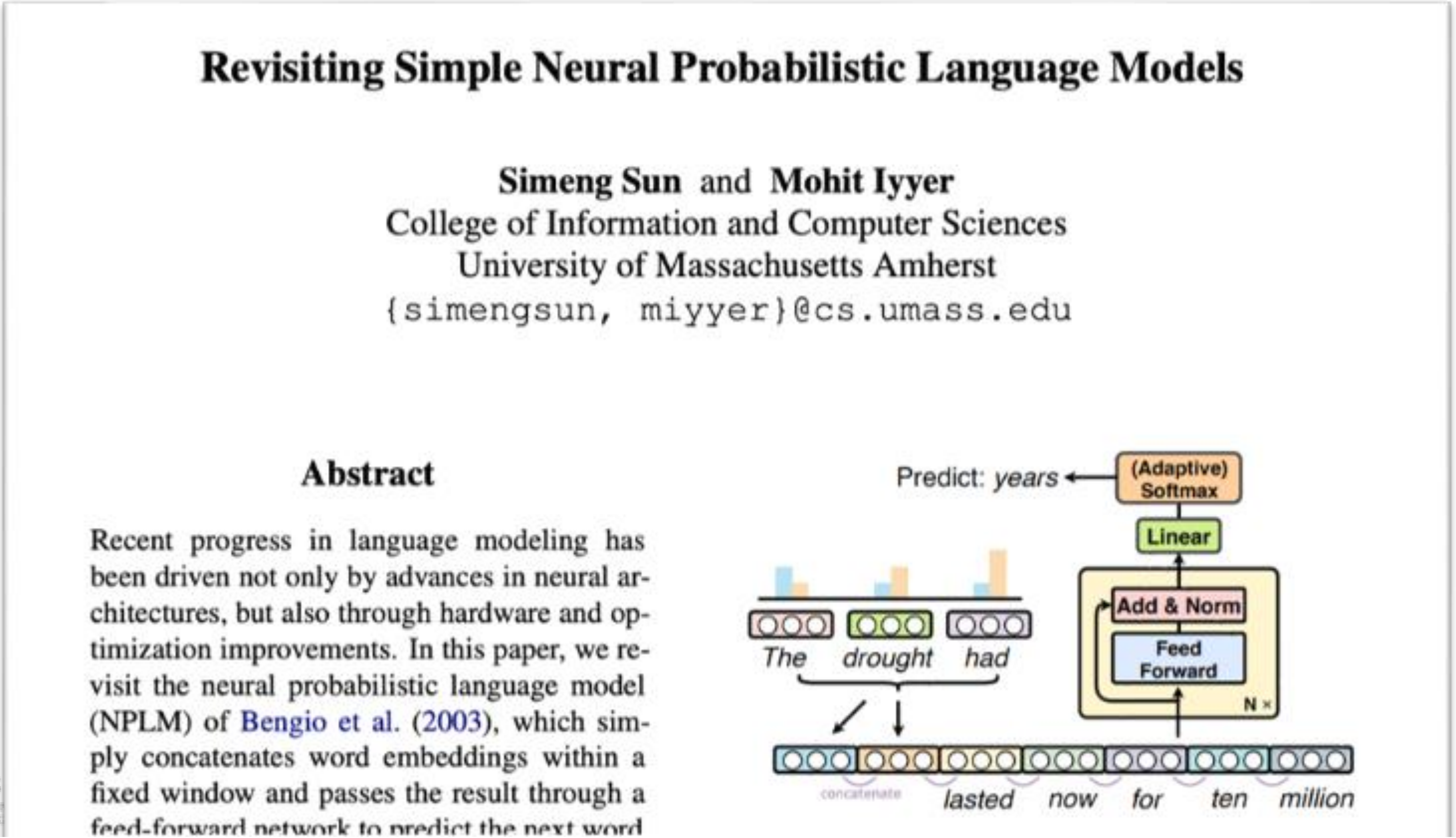
Remaining problems:

- Fixed window is too small
- Enlarging window enlarges $\boldsymbol{W}$ — Window can never be large enough!
- It's not deep enough to capture nuanced contextual meanings



[Bengio et al. 2003, notes from Richard Socher]

A Fixed-Window Neural LM: Going Deeper

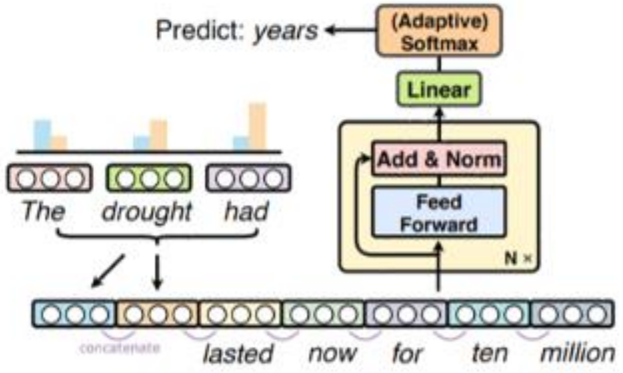


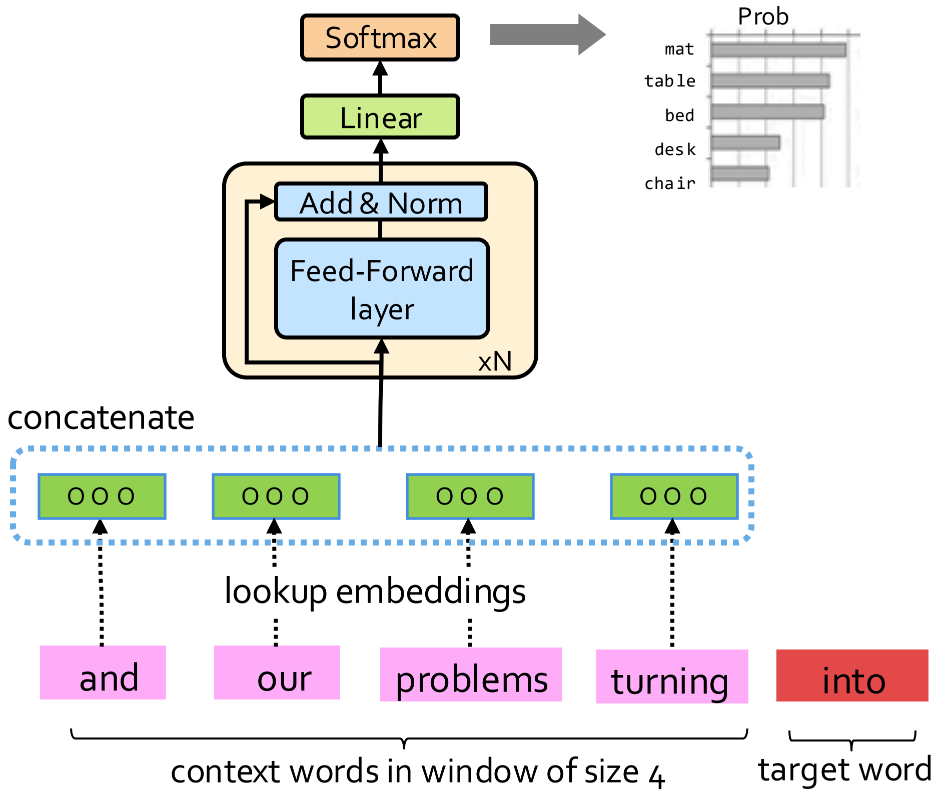
Revisiting Simple Neural Probabilistic Language Models

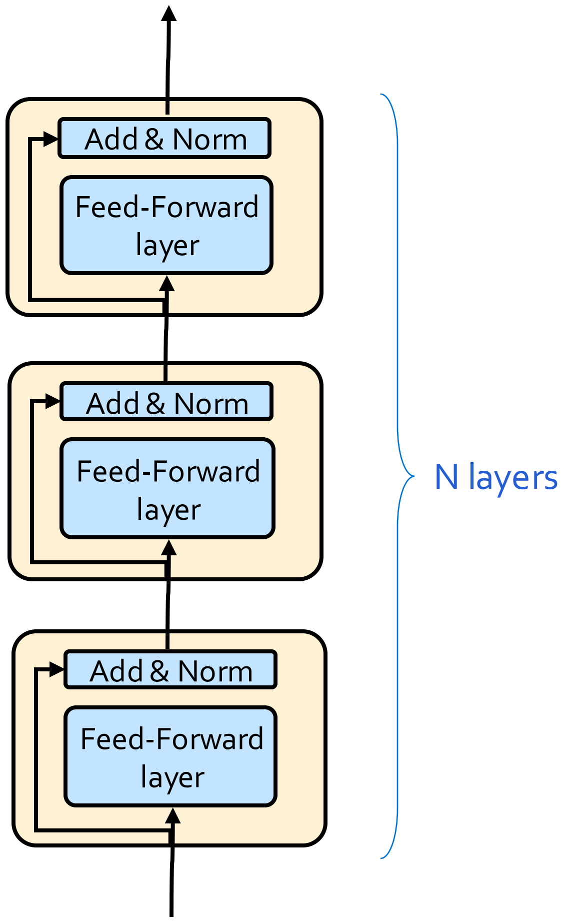
Simeng Sun and **Mohit Iyyer**
College of Information and Computer Sciences
University of Massachusetts Amherst
`{simengsun, miyyer}@cs.umass.edu`

Abstract

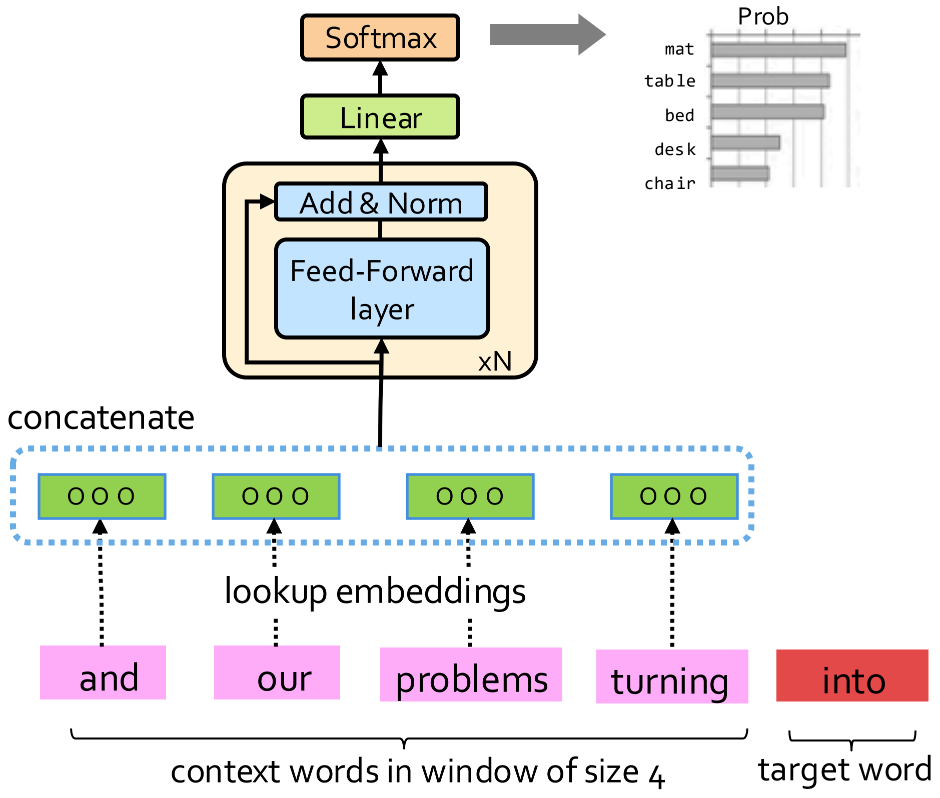
Recent progress in language modeling has been driven not only by advances in neural architectures, but also through hardware and optimization improvements. In this paper, we revisit the neural probabilistic language model (NPLM) of Bengio et al. (2003), which simply concatenates word embeddings within a fixed window and passes the result through a feed-forward network to predict the next word

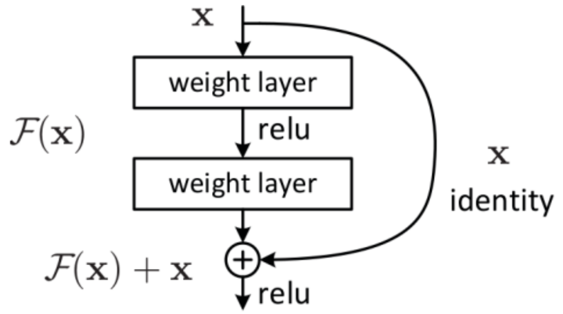






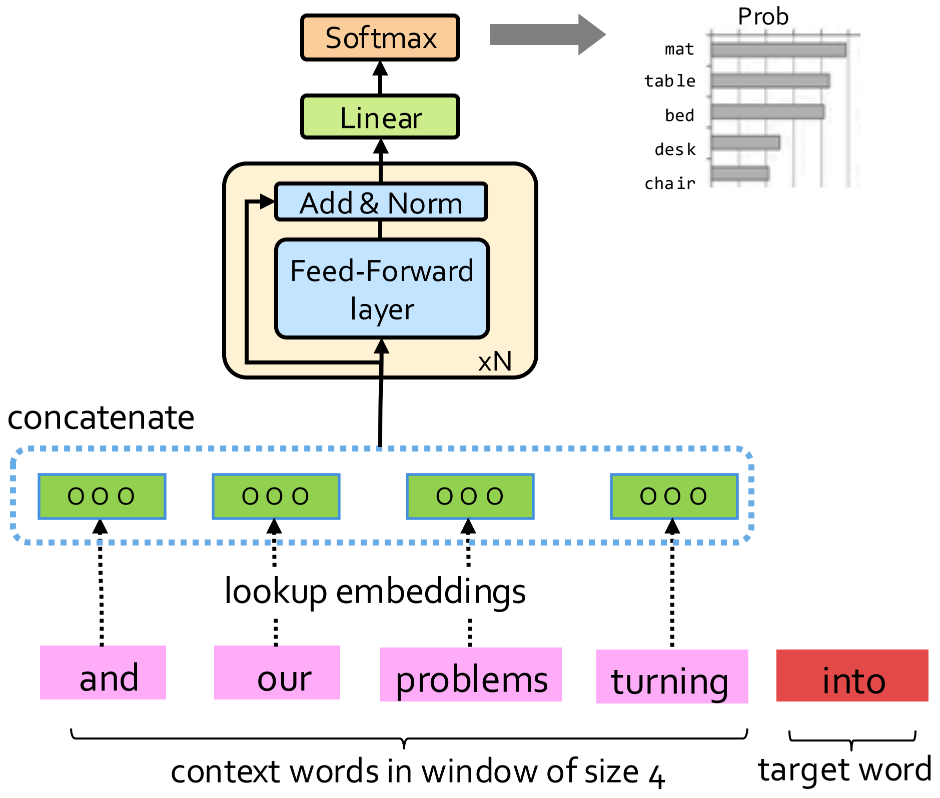
[Sun and Iyyer 2021]





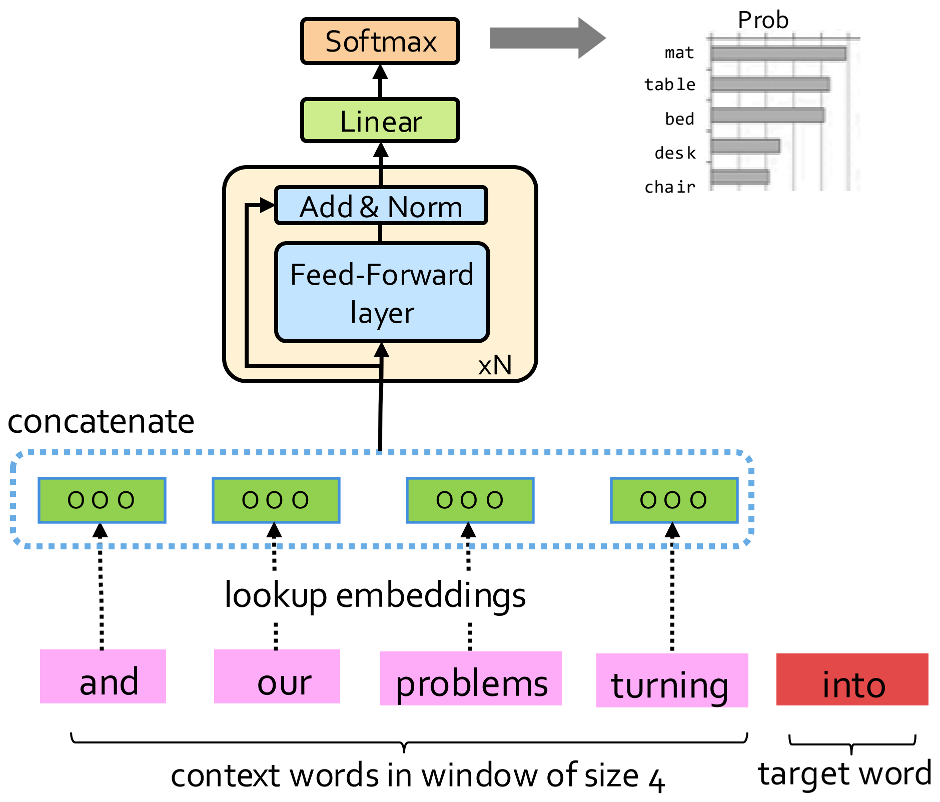
Uses residual connections (He et al. 2016) — "information highways" between layers. (we saw them in the earlier chapter)

[Sun and Iyyer 2021]



Uses layer normalization (Ba et al. 2016) which reduces variance across different data/batches and makes the optimization easier/faster.

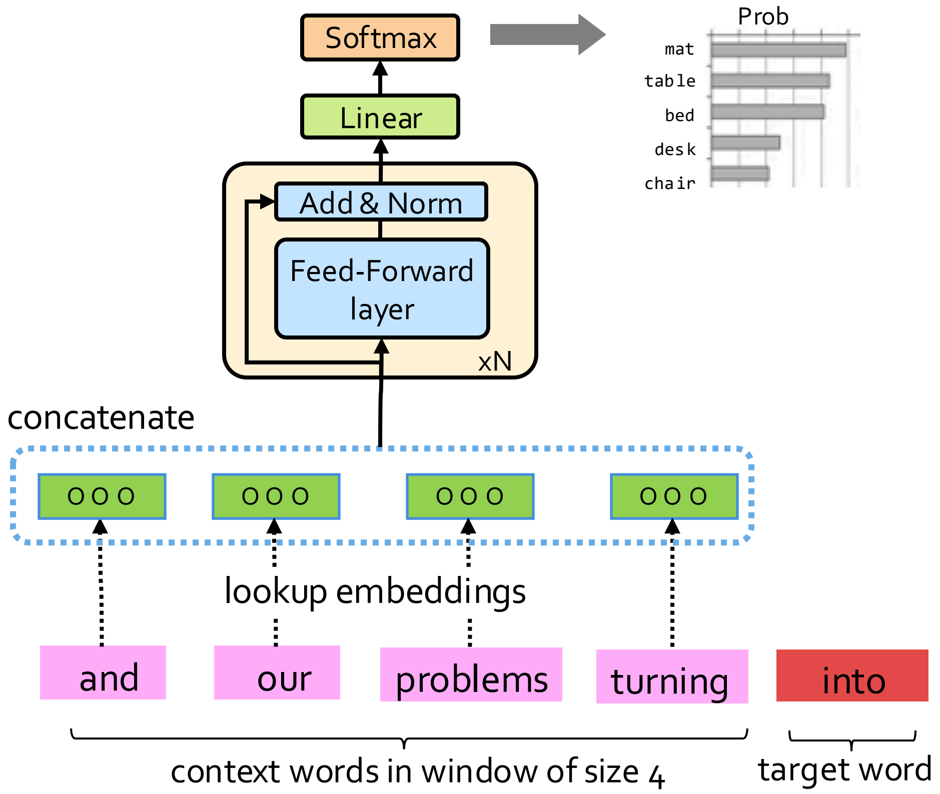
[Sun and Iyyer 2021]

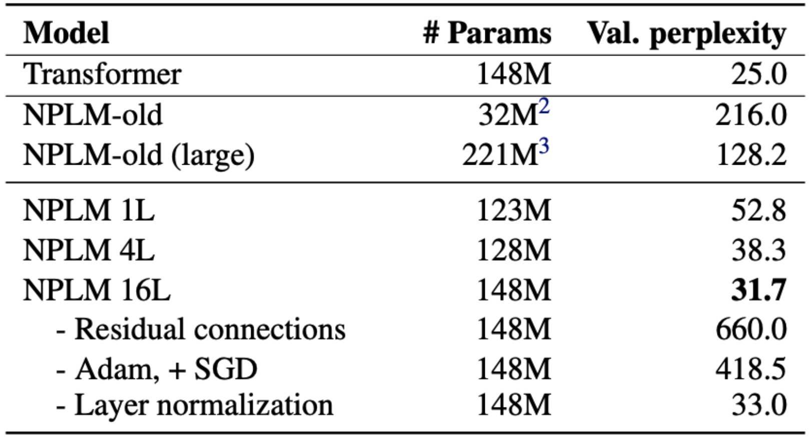


Use “dropout” to avoid overfitting.

Use ADAM optimizer ([Kingma & Ba, 2017]), a variant of Stochastic Gradient Descent.

[Sun and Iyyer 2021]



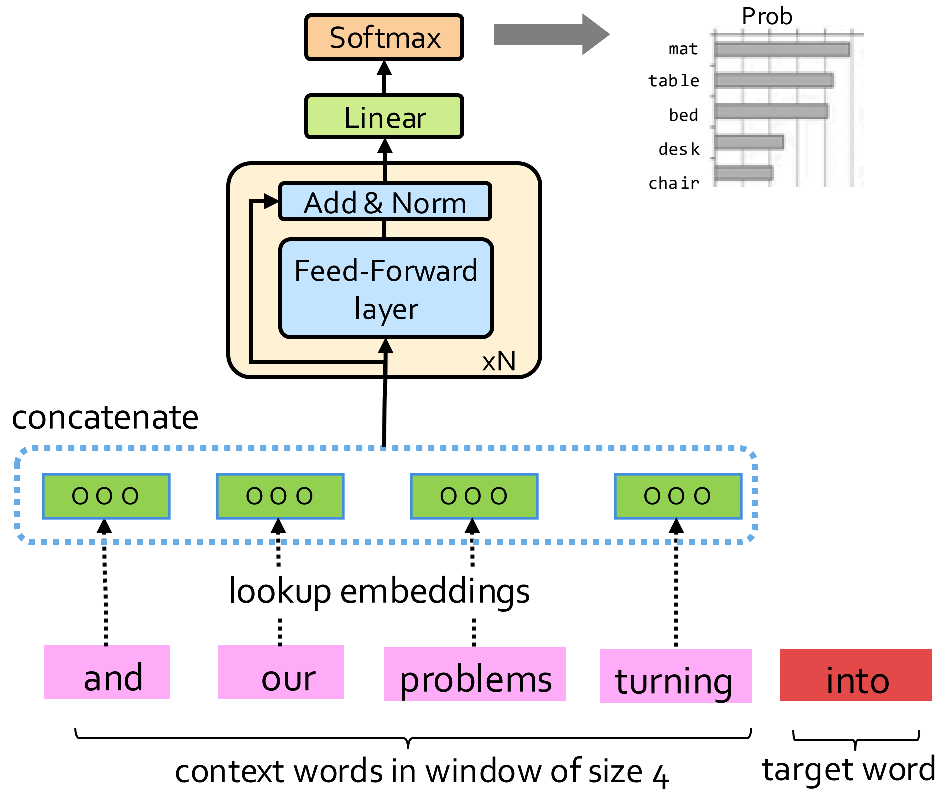


Model	# Params	Val. perplexity
Transformer	148M	25.0
NPLM-old	32M[2]	216.0
NPLM-old (large)	221M[3]	128.2
NPLM 1L	123M	52.8
NPLM 4L	128M	38.3
NPLM 16L	148M	**31.7**
- Residual connections	148M	660.0
- Adam, + SGD	148M	418.5
- Layer normalization	148M	33.0

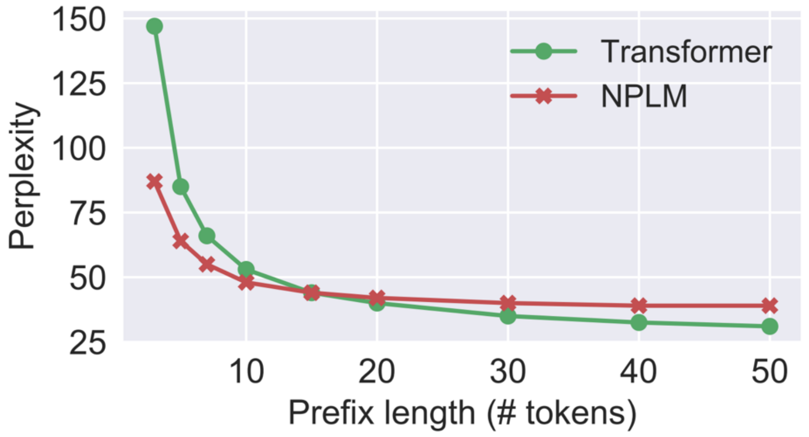
Table 1: NPLM model ablation on WIKITEXT-103.

Takeaways:

- Depth helps
- Residual connections are important
- Adam works (here) better than SGD



Effect of window size:



Fixed-WindowLM (NPLM) is better than the **Transformer** (will see them in 2 weeks!) with short prefixes but worse on longer ones.

[Sun and Iyyer 2021]

What Changed from N-Gram LMs to Neural LMs?

- What is the source of Neural LM's strength?
- Why sparsity is less of an issue for Neural LMs?

- **Answer:** In n-grams, we treat all prefixes independently of each other! (even those that are semantically similar)

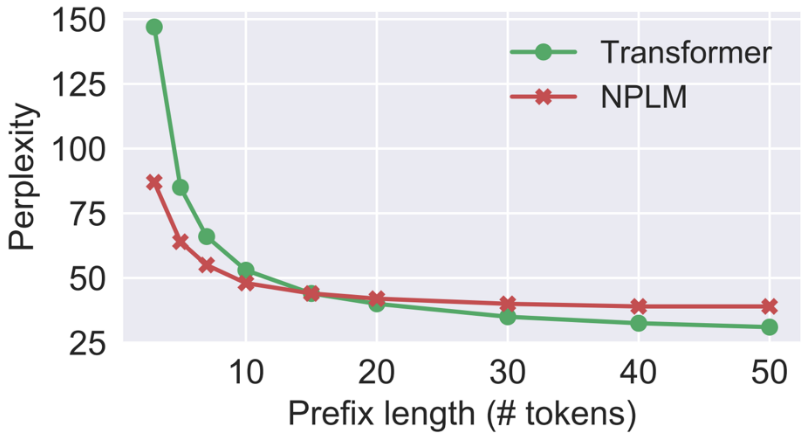
```
students opened their ___
pupils opened their ___
scholars opened their ___
undergraduates opened their ___
students turned the pages of their ___
students attentively perused their ___
...
```

Neural LMs are able to share information across these semantically-similar prefixes and overcome the sparsity issue.

Summary

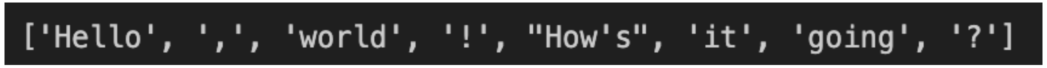
- Language Modeling (LM), a probabilistic model of language
- N-gram models (~1980 to early 2000's)
  - Difficult to scale to large n's
- Fixed-window Neural LM: first of many LMs we will see in this class
  - Stronger than n-gram LMs
  - But still fail at capturing longer contexts

- **Next:** other architectural alternatives.

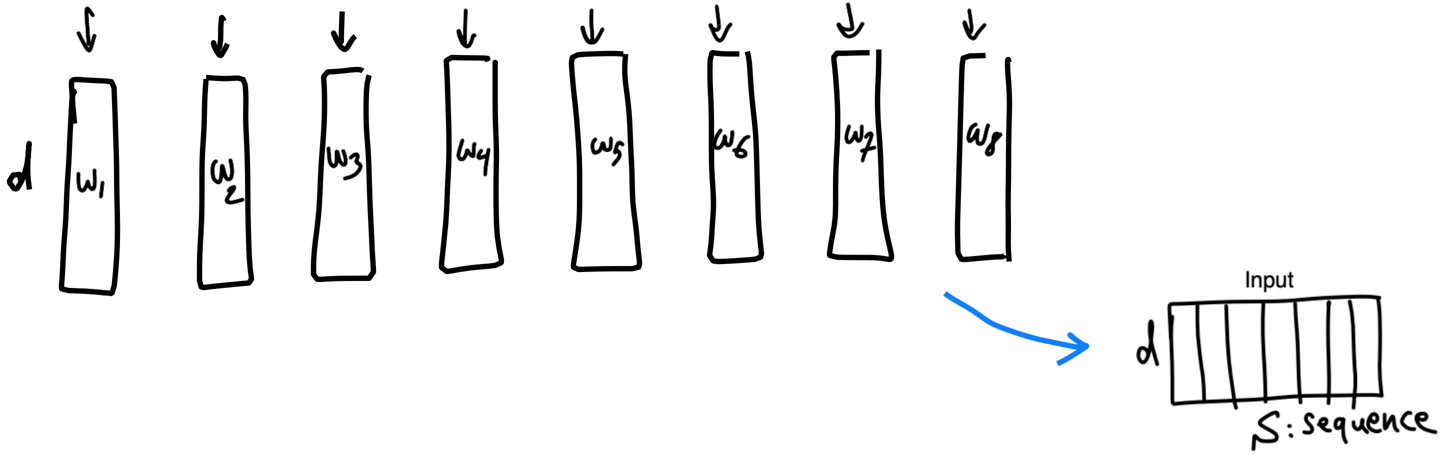


Atomic Units of Language

What is the right level of granularity for breaking up a sentence into vectors?



```
['Hello', ',', 'world', '!', "How's", 'it', 'going', '?']
```



The cat sat on the mat.

The cat sat on the mat.

words split based on white space?

```
BOS, The, cat, sat, on, the, mat, ., EOS
```

characters?

```
BOS, T, h, e, SPACE, c, a, t, SPACE, s, …
```

bytes??!

```
01100001011100000011100000011011000110010101100001
111000000111000000110110001100101011000010111000 …
```

The cat sat on the mat.

words split based on white space?

BOS

Which one should we use as the atomic building blocks for modeling language?

🤔

chara

BOS

bytes??!

```
011000010111000001110000011011000110010101100001
1110000011100000110110001100101011000010111000 …
```

Cost of Using Word Units

- What happens when we encounter a word at test time that we’ve never seen in our training data?
  - *Loquacious:* Tending to talk a great deal; talkative.
  - *Omnishambles:* A situation that has been mismanaged, due to blunders and miscalculations.
  - *COVID-19:* was unseen until 2020!
  - Acknowleadgement: incorrect spelling of “Acknowledgement”
- What about relevant words?: “dog” vs “dogs”; ”run” vs “running”
- We would need a very large vocabulary to capture common words in a language.
  - Very large vocabulary size makes training difficult
- What happens with words that we haven’t seen before?
  - With word level tokenization, we have no way of understanding an unseen word!
  - Also, not all languages have spaces between words like English!

Cost of Using Character Units

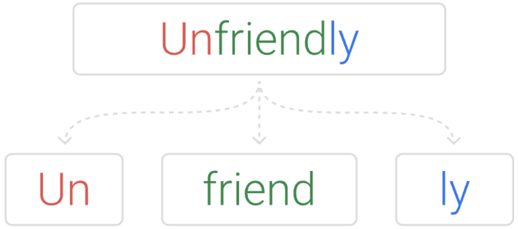
- What if we use characters?
- **Pro:**
  - (1) small vocabulary, just the number of unique characters in the training data.
  - (2) fewer out-of-vocabulary tokens
- **Cost:** much longer input sequences
  - As we discussed, modeling long-range dependences is very challenging.
  - Representing long sequences is computationally costly.

a	→	1
b	→	2
c	→	3
d	→	4
e	→	5
f	→	6
g	→	7
…	→	…
1	→	27
2	→	28
3	→	29
…	→	…
!	→	37
…		…
à	→	256

the	→	1
of	→	2
and	→	3
to	→	4
in	→	5
was	→	6
the	→	7
is	→	8
for	→	9
as	→	10
on	→	11
with	→	12
that	→	13
…		…
malapropism	→	170,000

Subword Tokenization: A Middle Ground

- Breaks words into smaller units that are indicative of their morphological construction.
  - Developed for machine translation (Sennrich et al. 2016)
- Subword tokenization is the best of both worlds
  - Common words are preserved in the vocabulary
  - Less common words are broken down into sub-words
  - This handles the problem of unseen words and large vocabulary size



- Dominantly used in modern language models (BERT, T5, GPT, …)
- Relies on a simple algorithm called Byte Pair Encoding (Gage, 1994)

[Improving Neural Machine Translation Models with Monolingual Data, Sennrich et al. 2016]

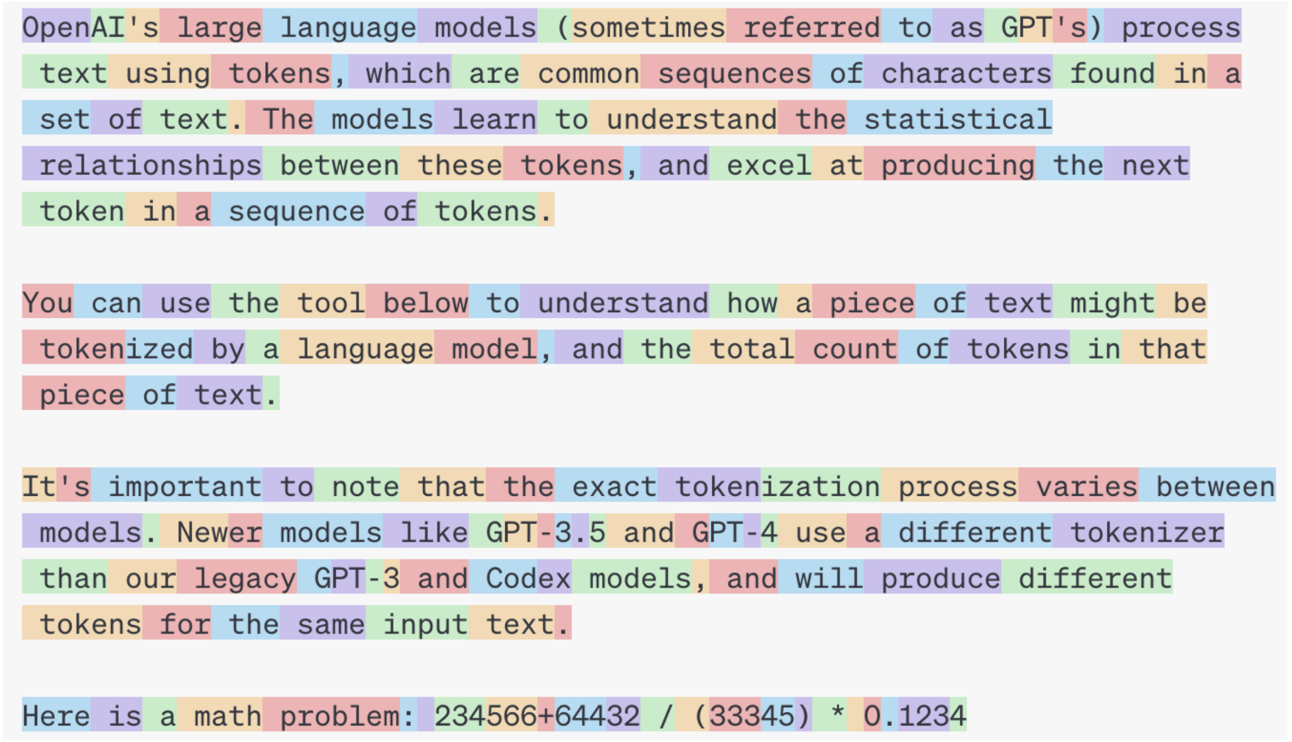
[A new algorithm for data compression, Gage 1994]

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
sequence = "Using a Transformer network is simple"
print(tokenizer.tokenize(sequence))

['Using', 'a', ‘Transform', '##er', 'network', 'is', 'simple’]
```

```
print(tokenizer.convert_tokens_to_ids(tokens))

[7993, 170, 13809, 23763, 2443, 1110, 3014]
```

```
tokenizer = AutoTokenizer.from_pretrained("albert-base-v1")
sequence = "Using a Transformer network is simple"
print(tokenizer.tokenize(sequence))

[‘_using', ‘_a', ‘_transform', 'er', ‘_network', ‘_is', ‘_simple’]
```

GPT4's Tokenizer



OpenAI's large language models (sometimes referred to as GPT's) process text using tokens, which are common sequences of characters found in a set of text. The models learn to understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens.

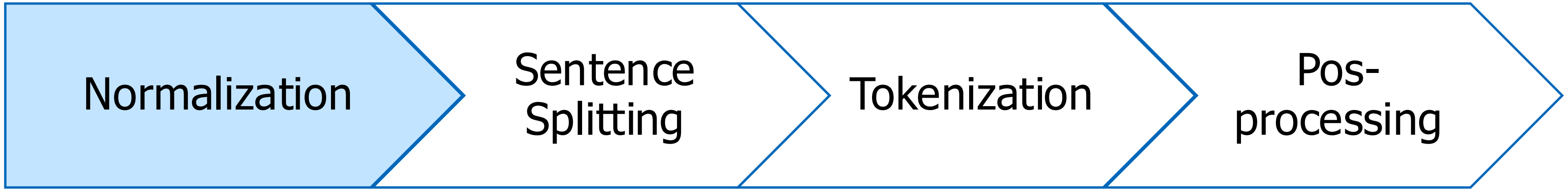
You can use the tool below to understand how a piece of text might be tokenized by a language model, and the total count of tokens in that piece of text.

It's important to note that the exact tokenization process varies between models. Newer models like GPT-3.5 and GPT-4 use a different tokenizer than our legacy GPT-3 and Codex models, and will produce different tokens for the same input text.

Here is a math problem: 234566+64432 / (33345) * 0.1234

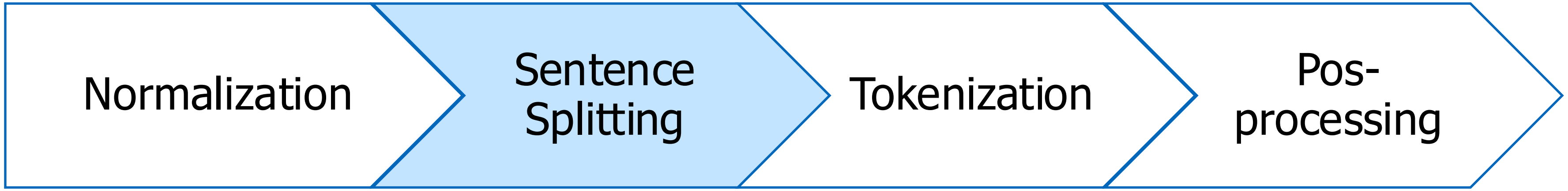
https://platform.openai.com/tokenizer

The Tokenization Pipeline



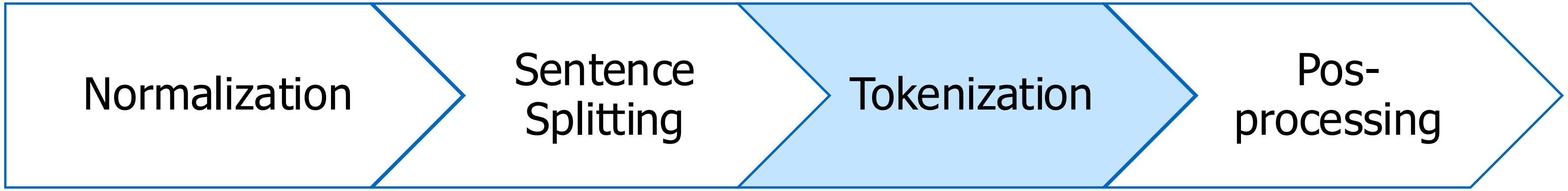
- Converts text into a standard format to reduce variability.
  - Strip extra white spaces between words and sentences
  - Removing punctuations (“Hello, world!” → “Hello world”)
  - Unicode normalization,
  - …

The Tokenization Pipeline



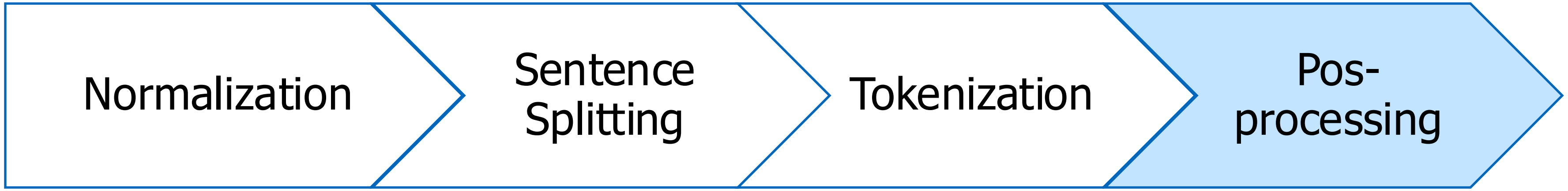
- Divides text into individual sentences.
- Uses punctuation marks (., !, ?) and language-specific rules to identify boundaries.

The Tokenization Pipeline



- Splits sentences into words or subwords.
  - Hello world → [Hello, world]
  - BPE, …. (will discuss this in a second)

The Tokenization Pipeline



- Truncate to match the maximum length of the model
- Pad all sentences in a batch to the same length
- Add special tokens: for example:
  - <UNK> (unknown word)
  - <PAD> (padding for fixed-length sequences)
  - <BOS> (beginning of sentence)
  - <EOS> (end of sentence)
- **Finally,** converts tokens into numerical IDs for model input.
  - Uses a vocabulary (word-to-index mapping).

Byte-pair Encoding (BPE) – An Example

- An algorithm for forming subword tokens based on a collection of raw text.

```
and there are no refueling stations anywhere
one of the city’s more unprincipled real state agents
```

Note that to do this tokenization, I need to learn it by seeing lots of text. (similar to model training)

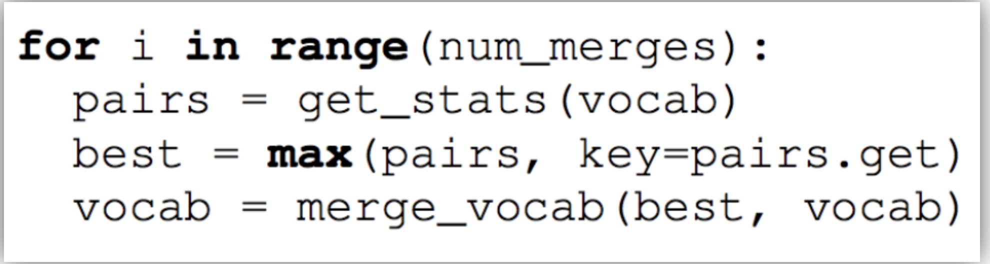
BPE-based tokenization

```
[and, there, are, no, re, ##fueling, stations, anywhere,
one, of, the, city, ’s, more, un, ##princi, ##pled, real,
state, agents]
```

Byte-pair Encoding (BPE) Training

Overview:

- We are given a **large** text corpus of text.
- Start by character-based tokenization.
- Repeatedly merge the most frequent adjacent tokens



```
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
```

- Doing 30k merges => vocabulary of around 30k subwords. Includes many whole words.

[Improving Neural Machine Translation Models with Monolingual Data, Sennrich et al. 2016] [A new algorithm for data compression, Gage 1994]

BPE training: Example

- Form base vocabulary of all characters that occur in the training set.
- *Example:*

  Our (very fascinating🙄) training data: “`jhu jhu jhu hopkins hop hops hops`”
  Base vocab: `h, i, j, k, n, o, p, s, u`
  Tokenized data: `j h u j h u j h u h o p k i n s h o p h o p s h o p s`

Does not show the word separator for simplicity.

BPE training: Example (2)

- Count the frequency of each token pair in the data
- *Example:*

  Our (very fascinating🙄) training data: "`jhu jhu jhu hopkins hop hops hops`"
  Base vocab: `h, i, j, k, n, o, p, s, u`
  Tokenized data: `j h u j h u j h u h o p k i n s h o p h o p s h o p s`
  Token pair frequencies:

  - j + h -> 3
  - h + u -> 3
  - h + o -> 4
  - o + p -> 4
  - p + k -> 1
  - k + i -> 1
  - ….

BPE training: Example (3)

- Choose the pair that occurs more, merge them and add to vocab.
- *Example:*

  Our (very fascinating 🙄) training data: "`jhu jhu jhu hopkins hop hops hops`"
  Base vocab: `h, i, j, k, n, o, p, s, u`
  Tokenized data: `j h u j h u j h u h o p k i n s h o p h o p s h o p s`
  Token pair frequencies:

  - j + h -> 3
  - h + u -> 3
  - h + o -> 4 ⬅
  - o + p -> 4
  - p + k -> 1
  - k + i -> 1
  - ….

BPE training: Example (4)

- Choose the pair that occurs more, merge them and add to vocab.
- *Example:*

Our (very fascinating🙄) training data: “jhu jhu jhu hopkins hop hops hops”
Base vocab: h, i, j, k, n, o, p, s, u, ho ⬅
Tokenized data: j h u j h u j h u h o p k i n s h o p h o p s h o p s
Token pair frequencies:

- j + h -> 3
- h + u -> 3
- h + o -> 4 ⬅
- o + p -> 4
- p + k -> 1
- k + i -> 1
- ….

BPE training: Example (5)

- Retokenize the data
- *Example:*

  Our (very fascinating🙄) training data: "`jhu jhu jhu hopkins hop hops hops`"
  Base vocab: `h, i, j, k, n, o, p, s, u, ho`
  Tokenized data: `j h u j h u j h u ho p k i n s ho p ho p s ho p s` ⬅
  Token pair frequencies:

BPE training: Example (6)

- Count the token pairs and merge the most frequent one
- *Example:*

  Our (very fascinating🙄) training data: "`jhu jhu jhu hopkins hop hops hops`"
  Base vocab: `h, i, j, k, n, o, p, s, u, ho`
  Tokenized data: `j h u j h u j h u ho p k i n s ho p ho p s ho p s`
  Token pair frequencies:
    - j + h -> 3
    - h + u -> 3
    - ho + p -> 4
    - p + k -> 1
    - k + i -> 1
    - i + n -> 1
    - ….

BPE training: Example (7)

- Count the token pairs and merge the most frequent one
- *Example:*

  Our (very fascinating🙄) training data: "`jhu jhu jhu hopkins hop hops hops`"
  Base vocab: `h, i, j, k, n, o, p, s, u, ho`
  Tokenized data: `j h u j h u j h u ho p k i n s ho p ho p s ho p s`
  Token pair frequencies:
  - j + h -> 3
  - h + u -> 3
  - ho + p -> 4 ⬅
  - p + k -> 1
  - k + i -> 1
  - i + n -> 1
  - ….

BPE training: Example (7)

- Count the token pairs and merge the most frequent one
- *Example:*

  Our (very fascinating🙄) training data: "`jhu jhu jhu hopkins hop hops hops`"
  Base vocab: `h, i, j, k, n, o, p, s, u, ho, hop` ⬅
  Tokenized data: `j h u j h u j h u ho p k i n s ho p ho p s ho p s`
  Token pair frequencies:
    - j + h -> 3
    - h + u -> 3
    - ho + p -> 4 ⬅
    - p + k -> 1
    - k + i -> 1
    - i + n -> 1
    - ....

BPE training: Example (7)

- Count the token pairs and merge the most frequent one
- *Example:*

  Our (very fascinating🙄) training data: "`jhu jhu jhu hopkins hop hops hops`"

  Base vocab: `h, i, j, k, n, o, p, s, u, ho, hop` ⬅

  Tokenized data: `j h u j h u j h u hop k i n s hop hop s hop s` ⬅

  Token pair frequencies:

  - j + h -> 3
  - h + u -> 3
  - ho + p -> 4 ⬅
  - p + k -> 1
  - k + i -> 1
  - i + n -> 1
  - ….

BPE training: Example (8)

- Count the token pairs and merge the most frequent one
- *Example:*

  Our (very fascinating🙄) training data: “`jhu jhu jhu hopkins hop hops hops`”
  Base vocab: `h, i, j, k, n, o, p, s, u, ho, hop`
  Tokenized data: `j h u j h u j h u hop k i n s hop hop s hop s`
  Token pair frequencies:
  - j + h -> 3 ⬅
  - h + u -> 3
  - hop + k -> 1
  - hop + s -> 2
  - k + i -> 1
  - i + n -> 1
  - n + s -> 1
  - ….

BPE training: Example (8)

- Count the token pairs and merge the most frequent one
- *Example:*

  Our (very fascinating🙄) training data: "`jhu jhu jhu hopkins hop hops hops`"
  Base vocab: `h, i, j, k, n, o, p, s, u, ho, hop, jh` ⬅
  Tokenized data: `j h u j h u j h u hop k i n s hop hop s hop s`
  Token pair frequencies:
  - j + h -> 3 ⬅
  - h + u -> 3
  - hop + k -> 1
  - hop + s -> 2
  - k + i -> 1
  - i + n -> 1
  - n + s -> 1
  - ….

BPE training: Example (8)

- Count the token pairs and merge the most frequent one
- *Example:*

  Our (very fascinating🙄) training data: "`jhu jhu jhu hopkins hop hops hops`"

  Base vocab: `h, i, j, k, n, o, p, s, u, ho, hop, jh` ⬅

  Tokenized data: `jh u jh u jh u hop k i n s hop hop s hop s` ⬅

  Token pair frequencies:

  - j + h -> 3 ⬅
  - h + u -> 3
  - hop + k -> 1
  - hop + s -> 2
  - k + i -> 1
  - i + n -> 1
  - n + s -> 1
  - ….

BPE training: Example (8)

- Count the token pairs and merge the most frequent one
- *Example:*

  Our (very fascinating 🙄) training data: "`jhu jhu jhu hopkins hop hops hops`"
  Base vocab: `h, i, j, k, n, o, p, s, u, ho, hop, jh`
  Tokenized data: `jh u jh u jh u hop k i n s hop hop s hop s`
  Token pair frequencies:

  - j h+u -> 3 ⬅
  - hop + k -> 1
  - hop + s -> 2
  - k + i -> 1
  - i + n -> 1
  - n + s -> 1
  - ....

BPE training: Example (8)

- Count the token pairs and merge the most frequent one
- *Example:*

Our (very fascinating🙄) training data: “`jhu jhu jhu hopkins hop hops hops`”
Base vocab: `h, i, j, k, n, o, p, s, u, ho, hop, jh, jhu` ⬅
Tokenized data: `jh u jh u jh u hop k i n s hop hop s hop s`
Token pair frequencies:

- j h+u -> 3 ⬅
- hop + k -> 1
- hop + s -> 2
- k + i -> 1
- i + n -> 1
- n + s -> 1
- ….

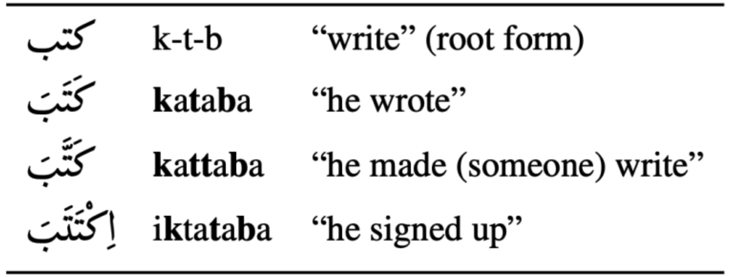
BPE training: Example (8)

- Count the token pairs and merge the most frequent one
- *Example:*

  Our (very fascinating🙄) training data: "jhu jhu jhu hopkins hop hops hops"
  Base vocab: h, i, j, k, n, o, p, s, u, ho, hop, jh, jhu ⬅
  Tokenized data: jhu jhu jhu hop k i n s hop hop s hop s ⬅
  Token pair frequencies:
  - j h+u -> 3 ⬅
  - hop + k -> 1
  - hop + s -> 2
  - k + i -> 1
  - i + n -> 1
  - n + s -> 1
  - ….

Limitations of Subwords

- Loss of whole word semantics
  - E.g., “Understand” -> [“Under”, “stand”] -Doesn’t mean “stand beneath”!
- Language Dependency: Even though subwords helps in multiple-languages it may favor the structure of one language vs the other
  - Hard to apply to languages with non-concatenative (e.g., Arabic) morphology



كتب	k-t-b	“write” (root form)
كَتَبَ	**kataba**	“he wrote”
كَتَّبَ	**kattaba**	“he made (someone) write”
إِكْتَتَبَ	**iktataba**	“he signed up”

Table 1: Non-concatenative morphology in Arabic.[4] The root contains only consonants; when conjugating, vowels, and sometimes consonants, are interleaved with the root. The root is not separable from its inflection via any contiguous split.

Clark et al., 2021, “CANINE”

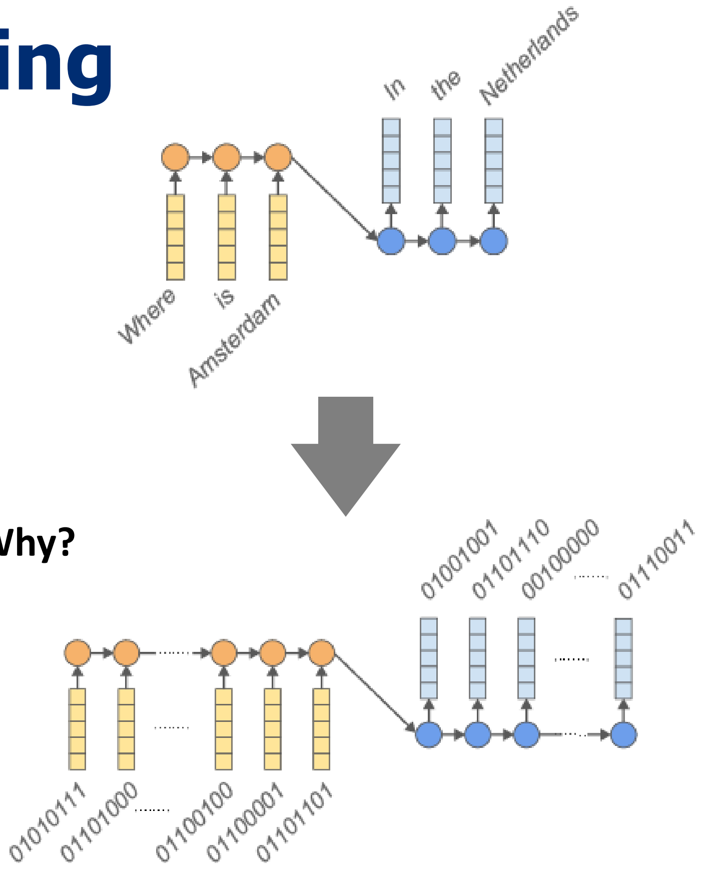
Alternatives: WordPieces

- WordPiece (Schuster & Nakajima, ICASSP 2012): merge by likelihood as measured by language model, not by frequency
  - While voc size < target:
    1. Build a language model over your corpus
    2. Merge tokens that lead to highest improvement in LM perplexity
- Issues: What LM to use? How to make it tractable?

[Shuster & Nakajima 2012; Wu et al. 2016]

Alternatives: Byte Encoding

- Use byte representation of words
  - E.g., `H -> 01010111`
- Vocabulary size: 2^8=256
- **Limitation:**
  - Makes the sequence length 4 to 5x longer
  - At test time it is also slower to generate sentences. **Why?**
    - Need to generate one character at a time



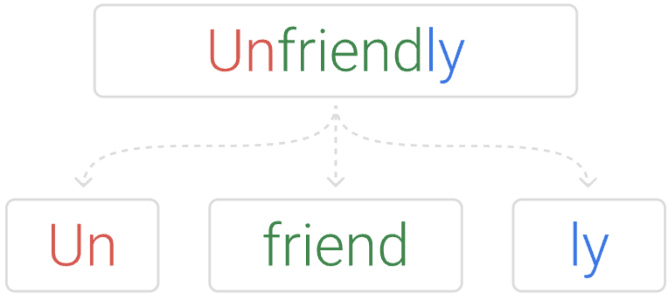
[Byte-level machine reading across morphologically varied languages, Kenter et al. 2018;
ByT5: Towards a Token-Free Future with Pre-trained Byte-to-Byte Models, Xue at al. 2021, and several others]

Summary

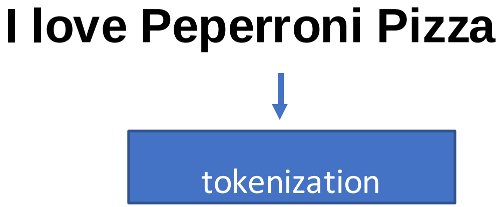
- **Fundamental question:** what should be the atomic unit of representation?

- **Words:** too coarse
- **Characters:** too small

- **Subwords:**
  - A useful representational choice for language.
  - Capture language morphology

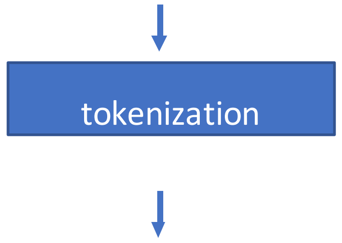


Recap: input pipeline



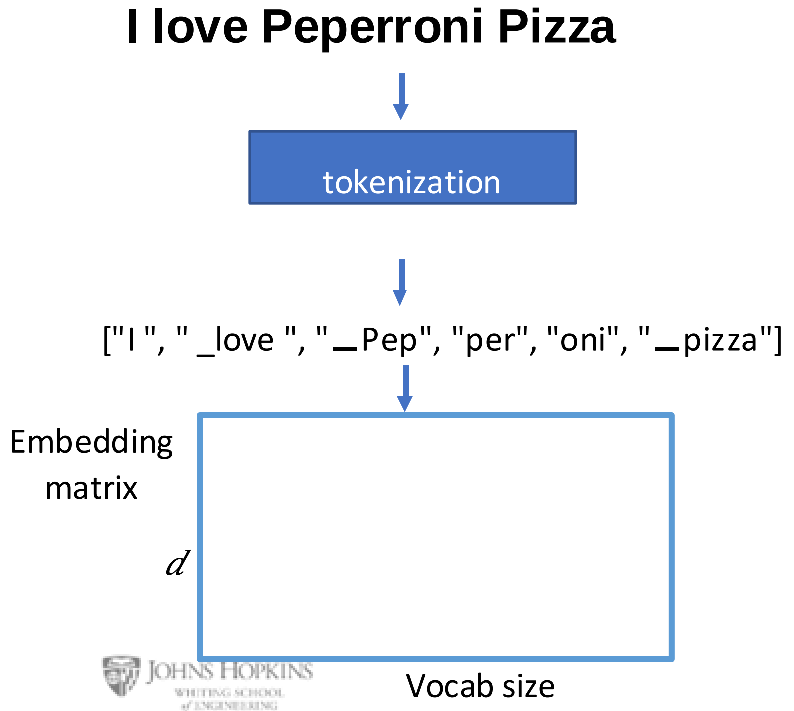
Recap: input pipeline

I love Peperroni Pizza



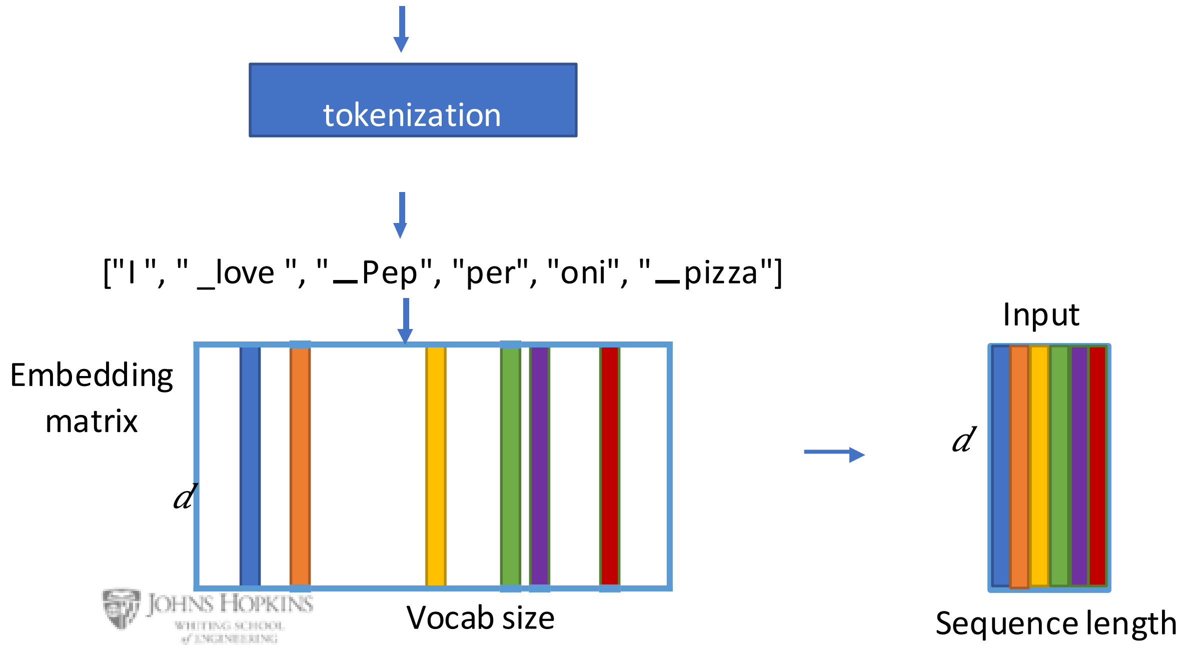
["I ", " _love ", "▁Pep", "per", "oni", "▁pizza"]

Recap: input pipeline

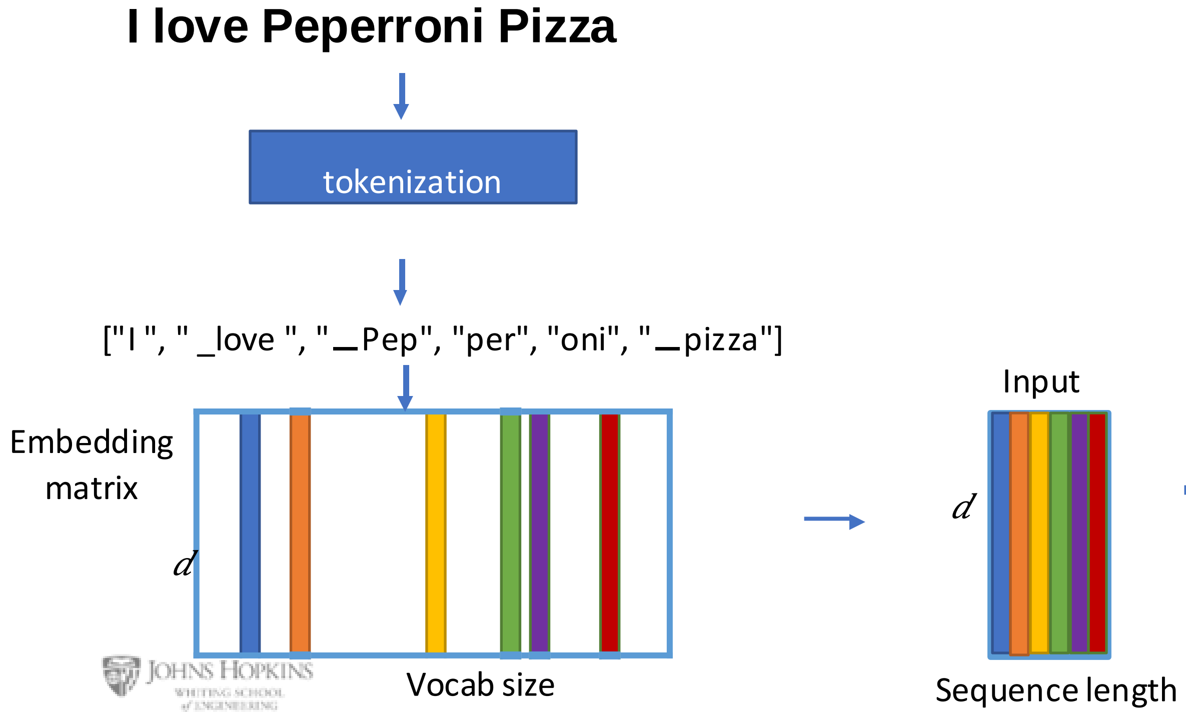


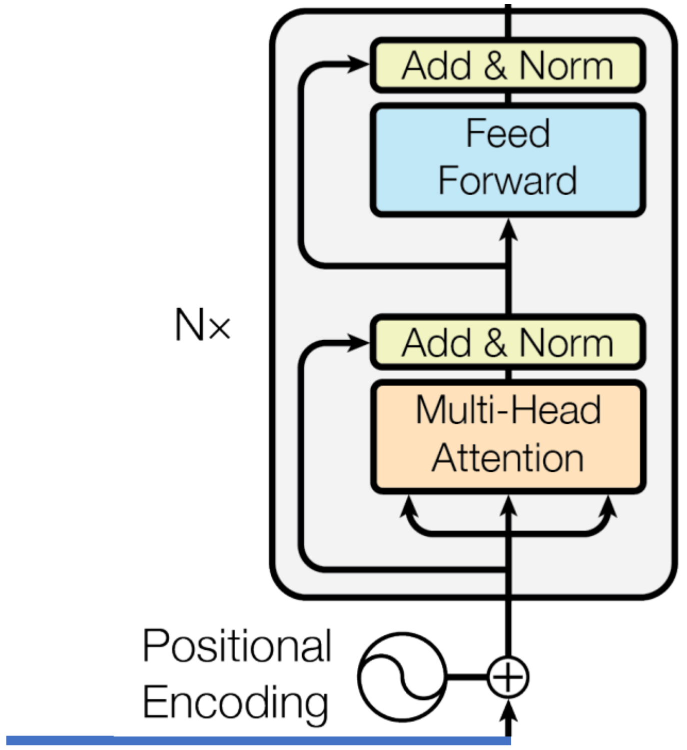
Recap: input pipeline

I love Peperroni Pizza



Recap: input pipeline





Johns Hopkins
Whiting School of Engineering