# Fixed-Window/Feedforward Language Models

CSCI 601 471/671
NLP: Self-Supervised Models

https://self-supervised.cs.jhu.edu/sp2023/

JOHNS HOPKINS UNIVERSITY

[Slide credit: Mohit Iyyer, Chris Manning, and many others ]

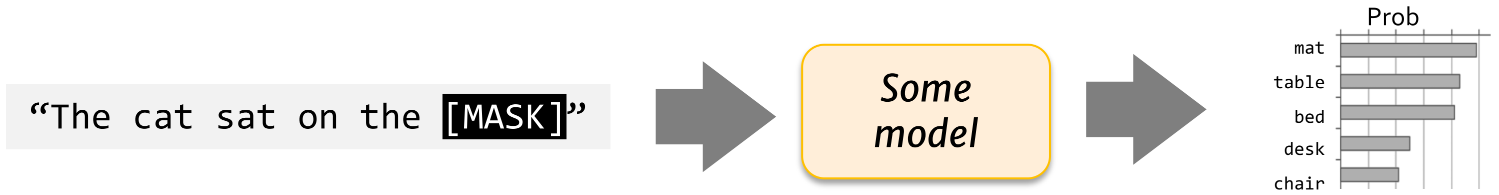
## Logistics

- HW2 grades are up!
  - Min: 50
  - Max 115
  - Median: 104
- HW4 is released!
- Please continue to give us feedback if you see any potential typos, odd phrasings, etc.
- Office hour update: Starting from today I will have **two** office hours
  - Both Tuesday and Thursday immediately after the class.

## Recap: LMs

- Directly we train models on "conditionals":

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



## Recap: LMs as Implicit Joint Distribution of Language

- Though implicitly we are learning the full distribution over the language:
  - Remember the chain rule: $\mathbf{P}(X_1, \dots, X_t) = \mathrm{P}(X_1) \prod_{i=1}^{t} \mathrm{P}(X_i \mid X_1, X_2 \dots, X_i)$

- Language Modeling $\triangleq$ learning prob distribution over language sequence.

$$\mathbf{P}(X_t | X_1, \ldots, X_{t-1})$$

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: N-gram Language Models

- **Terminology:** *n*-gram is a chunk of *n* consecutive words:
  - unigrams: “cat”, “mat”, “sat”, ...
  - bigrams: “the cat”, “cat sat”, “sat on”, ...
  - trigrams: “the cat sat”, “cat sat on”, “sat on the”, ...
  - four-grams: “the cat sat on”, “cat sat on the”, “sat on the mat”, ...

- *n*-gram language model:

$$\mathrm{P}(X_t \mid X_1, \ldots, X_{t-1}) \approx \mathrm{P}(X_t \mid \overbrace{X_{t-n+1}, \ldots, X_{t-1}}^{n-1 \text{ elements}})$$

## Chapter Plan

1. Language modeling: definitions and history
2. Language modeling with counting
3. Measuring language modeling quality
4. Language Modeling with feed-forward networks

## Generation from N-Gram Models

- You can build a simple **tri**gram Language Model over a 1.7 million words corpus in a few seconds on your laptop*

```
today the ___
```

get probability distribution

```
company   0.153
bank      0.153
price     0.077
italian   0.039
emirate   0.039
...
```

Sparsity problem: not much granularity in the probability distribution

Otherwise, seems reasonable!

* Try for yourself: https://nlpforhackers.io/language-models/

[adopted from Chris Manning]

## Generation from N-Gram Models

- Now we can sample from this mode:

`today the ___`

get probability distribution

| | |
|---|---|
| company | 0.153 |
| bank | 0.153 |
| price | 0.077 |
| italian | 0.039 |
| emirate | 0.039 |
| ... | |

<u>Sparsity problem</u>: not much granularity in the probability distribution

Otherwise, seems reasonable!

* Try for yourself: https://nlpforhackers.io/language-models/

[adopted from Chris Manning]

## Generation from N-Gram Models

- Now we can sample from this mode:

$$\texttt{today}\ \overbrace{\texttt{the price}}^{\text{condition on this}}\ \underline{\qquad\qquad}$$

get probability distribution

| | |
|---|---|
| `of` | `0.308` |
| `for` | `0.050` |
| `it` | `0.046` |
| `to` | `0.046` |
| `is` | `0.031` |
| `...` | |

Sparsity problem: not much granularity in the probability distribution

Otherwise, seems reasonable!

* Try for yourself: https://nlpforhackers.io/language-models/

[adopted from Chris Manning]

## Generation from N-Gram Models

- Now we can sample from this mode:

$$\texttt{today the } \overbrace{\texttt{price of}}^{\text{condition on this}} \texttt{ \_\_}$$

get probability distribution

| | |
|---|---|
| the | 0.072 |
| 18 | 0.043 |
| oil | 0.043 |
| its | 0.036 |
| gold | 0.018 |
| ... | |

<u>Sparsity problem</u>: not much granularity in the probability distribution

Otherwise, seems reasonable!

* Try for yourself: https://nlpforhackers.io/language-models/

[adopted from Chris Manning]

## N-Gram Models in Practice

- Now we can sample from this mode:

```
today the price of gold per ton , while production of shoe
lasts and shoe industry , the bank intervened just after it
considered and rejected an imf demand to rebuild depleted
european stocks , sept 30 end primary 76 cts a share .
```

Surprisingly grammatical!

But quite incoherent! To improve coherence, one may consider increasing larger than 3-grams, but that would worsen the sparsity problem!

* Try for yourself: https://nlpforhackers.io/language-models/

[adopted from Chris Manning]

## Scaling N-Grams

- We can extend to trigrams, 4-grams, 5-grams, but soon we will hit the sparsity limitations.
- In general, this is an insufficient model of language because language has long-distance dependencies:

"The computer which I had just put into the machine room on the fifth floor crashed."

# Chapter Plan

1. Language modeling: definitions and history
2. Language modeling with counting
3. Measuring language modeling quality
4. Language Modeling with feed-forward networks

## How Good is Our Language Model?

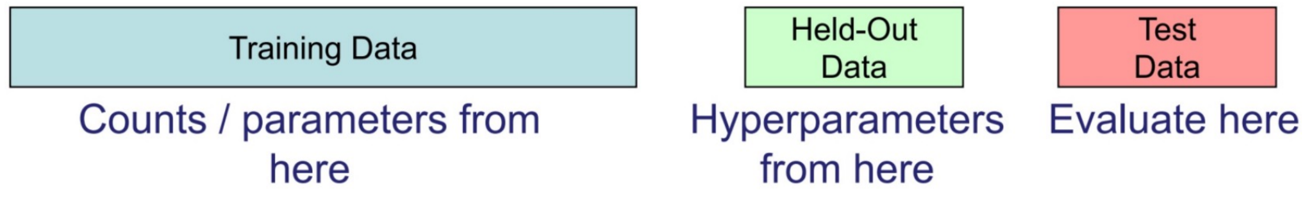
## Evaluating Language Models

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences
  - Than “ungrammatical” or “rarely observed” sentences?
- We test the model’s performance on data we haven’t seen.

# Evaluating Language Models

Setup:

- Train it on a suitable training documents.
- Evaluate their predictions on different, unseen documents.
- An evaluation metric tells us how well our model does on the test set.



## Evaluating Language Models: Example

Setup:

- Train it on a suitable training documents.
- Evaluate their predictions on different, unseen documents.
- An evaluation metric tells us how well our model does on the test set.

Example: I use a bunch of New York Times articles to build a bigram probability table

A good language model should assign a high probability to held-out text!

Now I'm going to evaluate the probability of some heldout data using our bigram table

train →

count("on the mat")

eval →

## Be Careful About Data Leakage!

**Advice from a grandpa👴:**

- Don't allow test sentences leak into training set.
- Otherwise, you will assign it an artificially high probability (=cheating).

Example: I use a bunch of New York Times articles to build a bigram probability table

train $\longrightarrow$

A good language model should assign a high probability to held-out text!

count("on the mat")

eval $\longrightarrow$

Now I'm going to evaluate the probability of some heldout data using our bigram table

# Evaluating Language Models: Intrinsic vs Extrinsic

- Intrinsic: measure how good we are at modeling language
- Extrinsic: build a new language model, use it for some task (MT, ASR, etc.)

Example: I use a bunch of New York Times articles to build a bigram probability table

Now I'm going to evaluate the probability of some heldout data using our bigram table

$$\begin{tikzcd} \text{NYT} \arrow[r, "\text{train}"] & \text{count(“on the mat”)} \arrow[r, "\text{eval}"] \arrow[u, "\text{extrinsic eval}"] & \text{Washington Post} \\ & \text{Google Translate} & \end{tikzcd}$$

## Evaluation Metric for Language Modeling: Perplexity

- **Perplexity** is the inverse probability of the test set, normalized by the number of words:

$$\mathrm{ppl}(w_1, \ldots, w_n) = \mathbf{P}(w_1, w_2, \ldots, w_n)^{-\frac{1}{n}}$$

- A measure of predictive quality of a language model.
- Minimizing perplexity is the same as maximizing probability

## Evaluation Metric for Language Modeling: Perplexity

- **Perplexity** is the inverse probability of the test set, normalized by the number of words:

$$\text{ppl}(w_1, \dots, w_n) = \mathbf{P}(w_1, w_2, \dots, w_n)^{-\frac{1}{n}}$$

- **Quiz:** let's suppose we have a sentence $w_1, \dots, w_n$ and it's fixed. Our model will correctly guess each word with probability 1/5. What is perplexity of our model?

$$\text{ppl}(w_1, \dots, w_n) = \left((1/5)^n\right)^{-\frac{1}{n}} = 5$$

**Intuition:** the model is indecisive among 5 choices.

## Evaluation Metric for Language Modeling: Perplexity

- **Perplexity** is the inverse probability of the test set, normalized by the number of words:

$$\text{ppl}(w_1, \dots, w_n) = \mathbf{P}(w_1, w_2, \dots, w_n)^{-\frac{1}{n}}$$

$$= \sqrt[n]{\frac{1}{\mathbf{P}(w_1, w_2, \dots, w_n)}}$$

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{\mathbf{P}(w_i | w_{<i})}}$$

(the chain rule)

## Evaluation Metric for Language Modeling: Perplexity

- **Perplexity** for n-grams:

$$\text{ppl}(w_1, \dots, w_n) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{\mathbf{P}(w_i | w_{<i})}}$$

- Bi-grams (2nd order Markov assumption):

$$\text{ppl}(w_1, \dots, w_n) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{\mathbf{P}(w_i | w_{i-1})}}$$

- Tri-grams (3rd order Markov assumption):

$$\text{ppl}(w_1, \dots, w_n) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{\mathbf{P}(w_i | w_{i-1}, w_{i-2})}}$$

## Evaluation Metric for Language Modeling: Perplexity

- In practice, we prefer to use log-probabilities
- We can rewrite perplexity formula in terms of log-probs:

$$\mathrm{ppl}(w_1, \ldots, w_n) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{\mathbf{P}(w_i | w_{<i})}} \qquad = 2^{\log \sqrt[n]{\prod_{i=1}^{n} \frac{1}{\mathbf{P}(w_i | w_{<i})}}}$$

$$\mathrm{ppl}(w_1, \ldots, w_n) = 2^H, \text{ where } H = -\frac{1}{n} \sum_{i=1}^{n} \log_2 \mathbf{P}(w_i | w_1, \ldots, w_{i-1})$$

## Evaluation Metric for Language Modeling: Perplexity

- In practice, we prefer to use log-probabilities
- We can rewrite perplexity formula in terms of log-probs:

$$\text{ppl}(w_1, \ldots, w_n) = 2^H, \text{ where } H = -\frac{1}{n}\sum_{i=1}^{n} \log_2 \mathbf{P}(w_i | w_1, \ldots, w_{i-1})$$

```
# getting loss using cross entropy
loss = F.cross_entropy(output, target)

# calculating perplexity
perplexity = torch.exp(loss)

print('Loss:', loss, 'PP:', perplexity)
```

Can be interpreted as cross-entropy between LM prob and language prob

## Evaluation Metric for Language Modeling: Perplexity

- In practice, we prefer to use log-probabilities
- We can rewrite perplexity formula in terms of log-probs:

$$\text{ppl}(w_1, \dots, w_n) = 2^H, \text{ where } H = -\frac{1}{n}\sum_{i=1}^{n} \log_2 \mathbf{P}(w_i | w_1, \dots, w_{i-1})$$

- **Quiz:** let's suppose we have a sentence $w_1, \dots, w_n$ and it's fixed. Our model will correctly guess each word with probability 1/5. What is perplexity of our model?

$$H = -\frac{1}{n}\left[\log_2\left(\frac{1}{5}\right) + \cdots + \log_2\left(\frac{1}{5}\right)\right] = -\log\left(\frac{1}{5}\right) \Rightarrow \text{ppl}(\text{D}) = 5$$

## Evaluation Metric for Language Modeling: Edge Cases

$$\text{ppl}(w_1, \ldots, w_n) = 2^H, \text{ where } H = -\frac{1}{n}\sum_{i=1}^{n} \log_2 \mathbf{P}(w_i | w_1, \ldots, w_{i-1})$$

- If $P(.)$ uninformative:

$$\forall w \in V: \mathbf{P}(w|w_{1:i-1}) = \frac{1}{|V|} \Rightarrow \text{ppl}(D) = 2^{-\frac{1}{\mathrm{n}} n \log_2 \frac{1}{|V|}} = |V|$$

- If $P(.)$ is exact:

$$\exists w \in V: \mathbf{P}(w|w_{1:i-1}) = 1 \Rightarrow \text{ppl}(D) = 2^{-\frac{1}{\mathrm{n}} n \log_2 1} = 1$$

*Perplexity is a measure of model's uncertainty about next word (aka "average branching factor")*

*Perplexity ranges between 1 and |V|.*

## Lower perplexity == Better Model

- Training 38 million words, test 1.5 million words, Wall Street Journal

| N-gram Order | Unigram | Bigram | Trigram |
|---|---|---|---|
| Perplexity | 962 | 170 | 109 |

[Jurafsky & Martin: https://web.stanford.edu/~jurafsky/slp3/3.pdf]

## How Should One Deal With Zeros?

$$\mathrm{ppl}(w_1, \dots, w_n) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{\mathbf{P}(w_i | w_{<i})}}$$

- If $\mathbf{P}(w_i | w_{<i}) = 0$, ppl would go 🤯 !! (division by zero)

## How Should One Deal With Zeros?

**Training set:**

```
… denied the allegations
… denied the reports
… denied the claims
… denied the request
```

**Test set:**

```
… denied the offer
… denied the load
```

$$\mathbf{P}(\text{offer} \mid \text{denied the}) = 0$$

[Dan Jurafsky]

## How Should One Deal With Zeros?

- Actually, how common are zero-probabilities? 🤔
- **Example:** Shakespeare as text corpus
  - n=884,647 tokens (the length of Shakespeare writing),
  - |V|=29,066 (the size vocab used by Shakespear)
  - Shakespeare produced ~300,000 bigrams
  - Out of |V|^2= 844 million possible bigrams (some of them don't make sense, but ok!)
- So, 99.96% of the possible bigrams are never seen (have zero entries in the table)

[Mohit Iyyer]

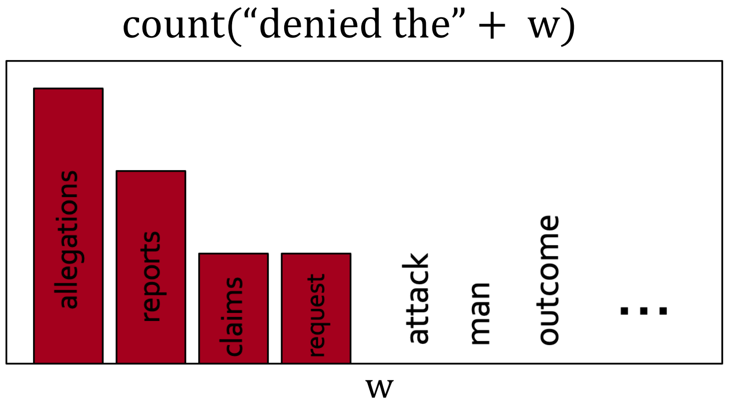
## How Should One Deal With Zeros? Smoothing

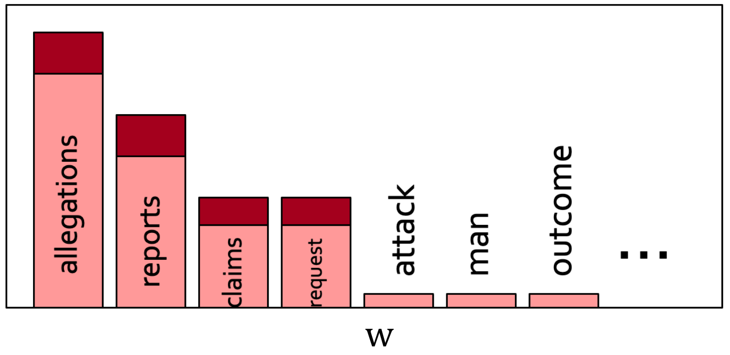
- When we have sparse statistics:

3 allegations, 2 reports,
1 claims, 1 request = 7 total

- Steal probability mass to generalize better

2.5 allegations, 1.5 reports,
0.5 claims, 0.5 request, 2 other = 7 total





[Dan Klein]

## Summary Thus Far

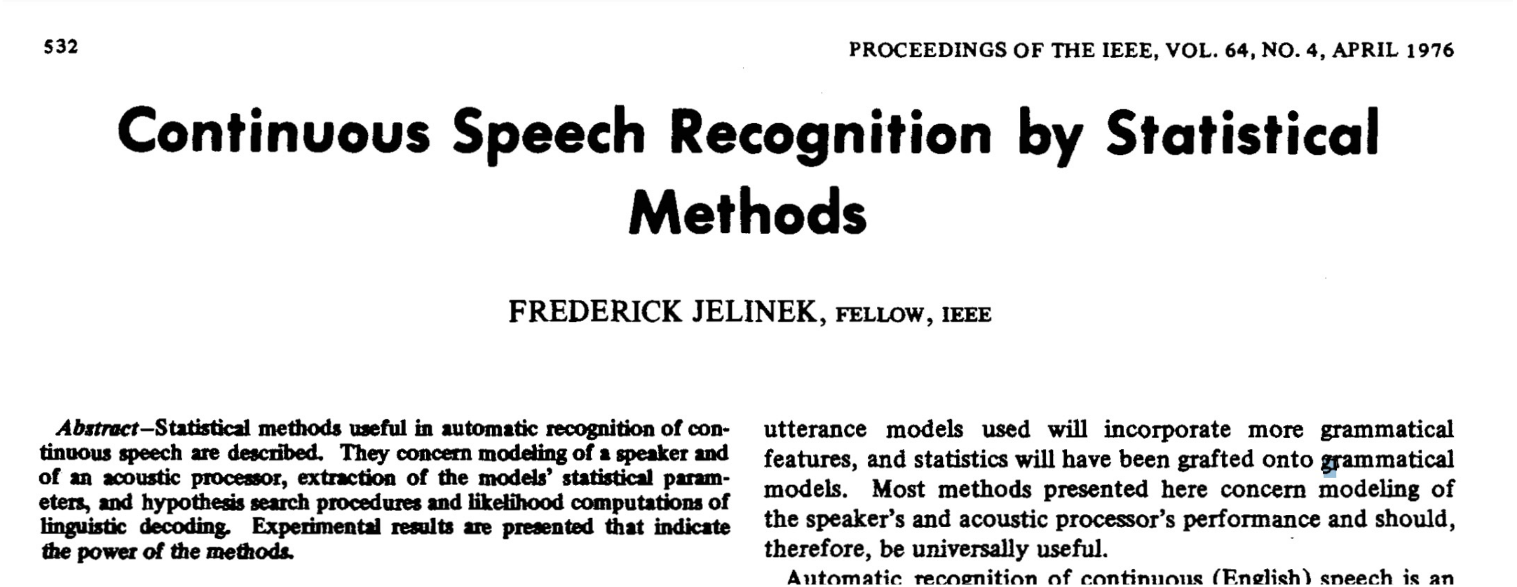
- Language Models (LM): distributions over language

- N-gram: language modeling via counting
  - Models size O(exp(n)) — not good 😔

- 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.

# N-Gram Language Models, A Historical Highlight

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



532 PROCEEDINGS OF THE IEEE, VOL. 64, NO. 4, APRIL 1976

**Continuous Speech Recognition by Statistical Methods**

FREDERICK JELINEK, FELLOW, IEEE

*Abstract*—Statistical methods useful in automatic recognition of continuous speech are described. They concern modeling of a speaker and of an acoustic processor, extraction of the models' statistical parameters, and hypothesis search procedures and likelihood computations of linguistic decoding. Experimental results are presented that indicate the power of the methods.

utterance models used will incorporate more grammatical features, and statistics will have been grafted onto grammatical models. Most methods presented here concern modeling of the speaker's and acoustic processor's performance and should, therefore, be universally useful.

Automatic recognition of continuous (English) speech is an

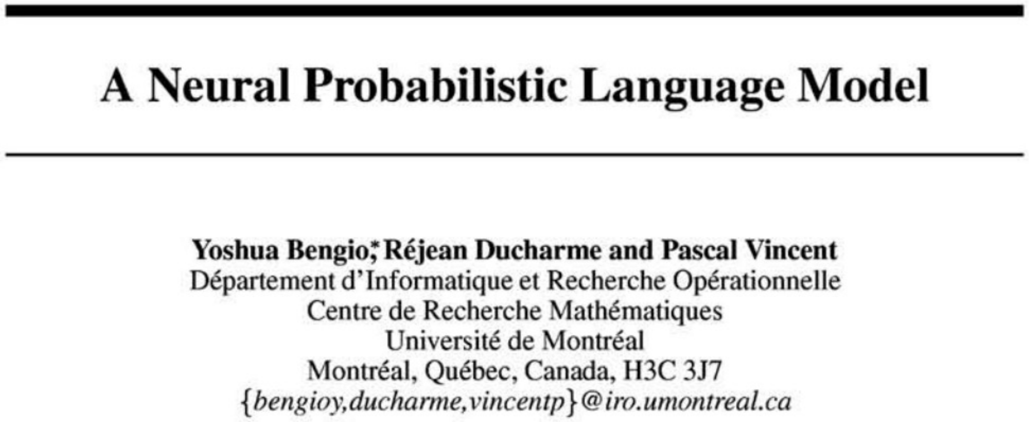
## Chapter Plan

1. Language modeling: definitions and history
2. Language modeling with counting
3. Measuring language modeling quality
4. Language Modeling with feed-forward networks

# 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

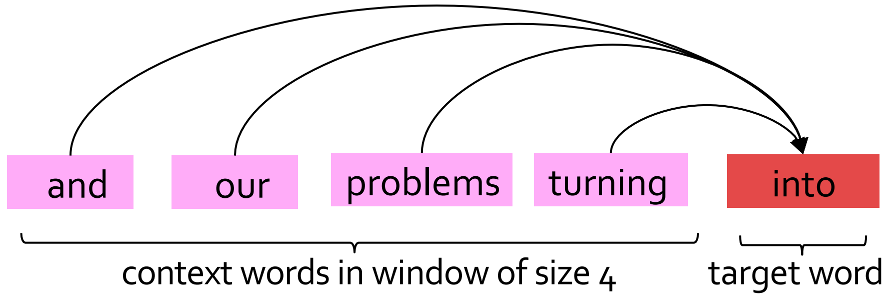


**A Neural Probabilistic Language Model**

**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*

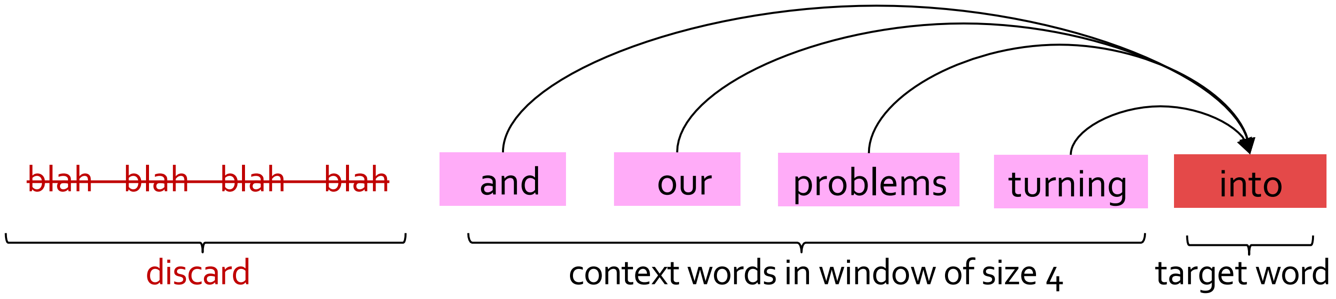
# A Fixed-Window Neural LM

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



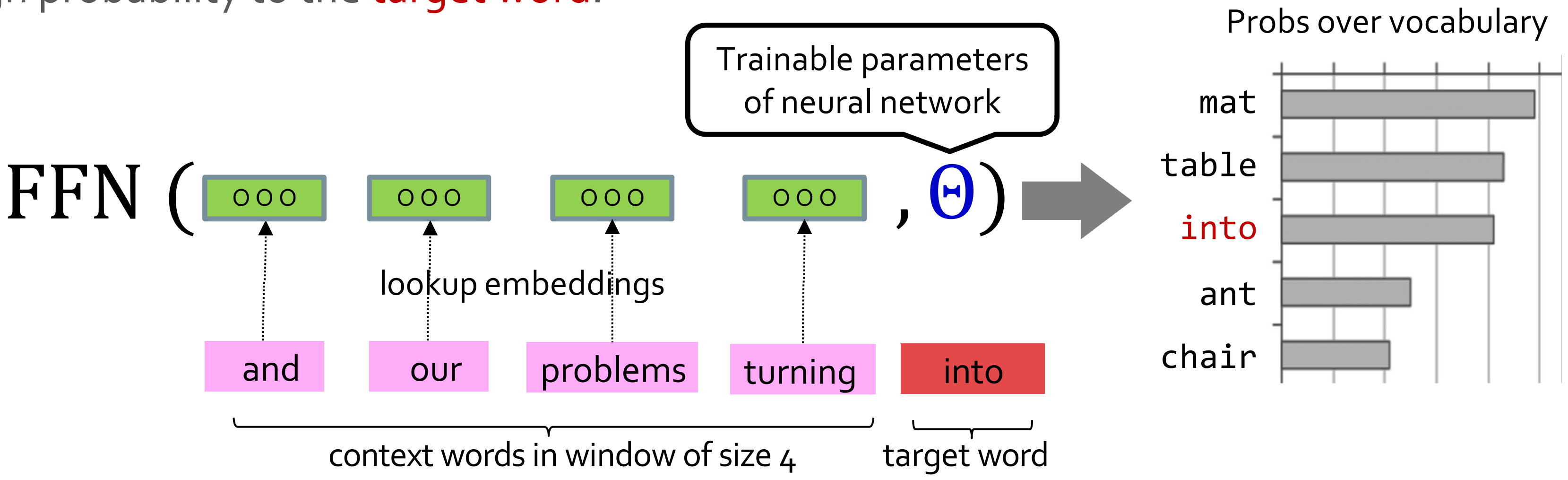
## A Fixed-Window Neural LM

- 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 $\Theta$ such that it assigns high probability to the target word.

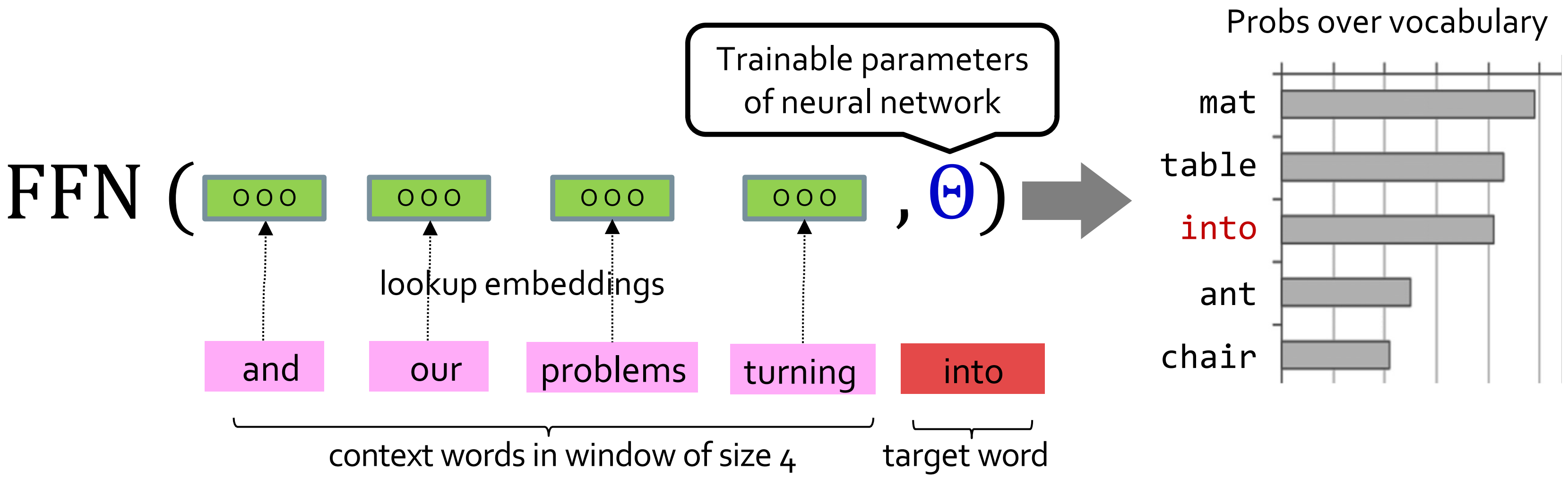


Trainable parameters of neural network
FFN ( ooo ooo ooo ooo , Θ )
lookup embeddings
and
our
problems
turning
into
context words in window of size 4
target word
Probs over vocabulary
mat
table
into
ant
chair

[Bengio et al. 2003]

## A Fixed-Window Neural LM

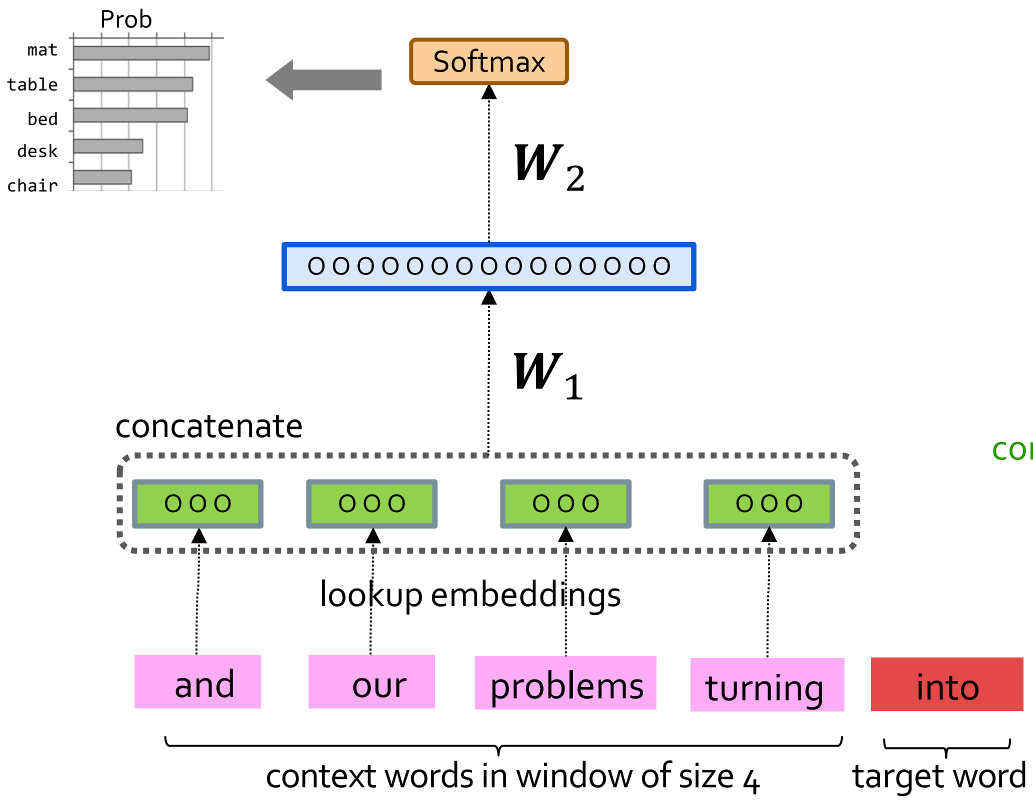
- This is actually a pretty good model!
- It will also lay the foundation for the future models (e.g., transformers, ...)
- But first we need to figure out how to train neural networks!



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

[Bengio et al. 2003]

## A Fixed-Window Neural LM



output distribution

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

hidden layer

$$h = f(W_1 x)$$

concatenated word embeddings

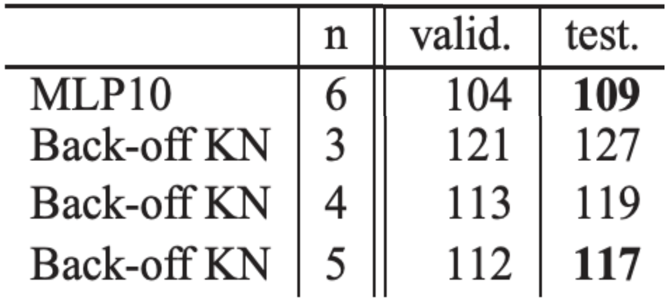
$$x = [v_1, v_2, v_3, v_4]$$

[Bengio et al. 2003]

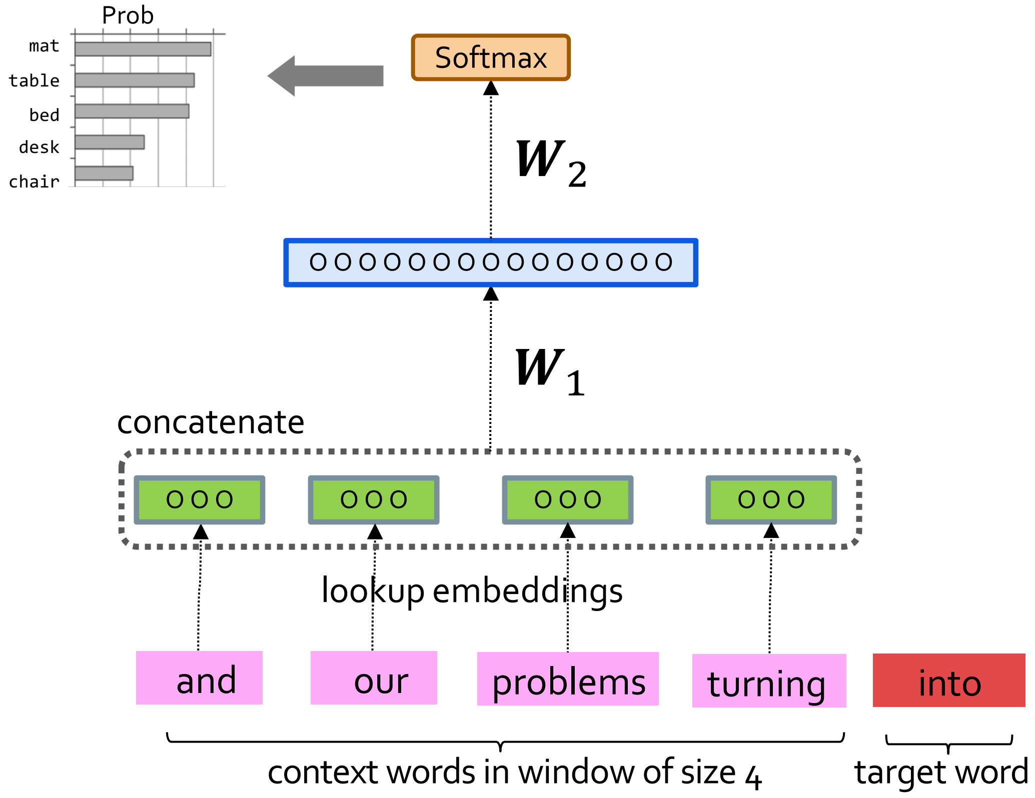
## A Fixed-Window Neural LM: Compared to N-Grams

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** |



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

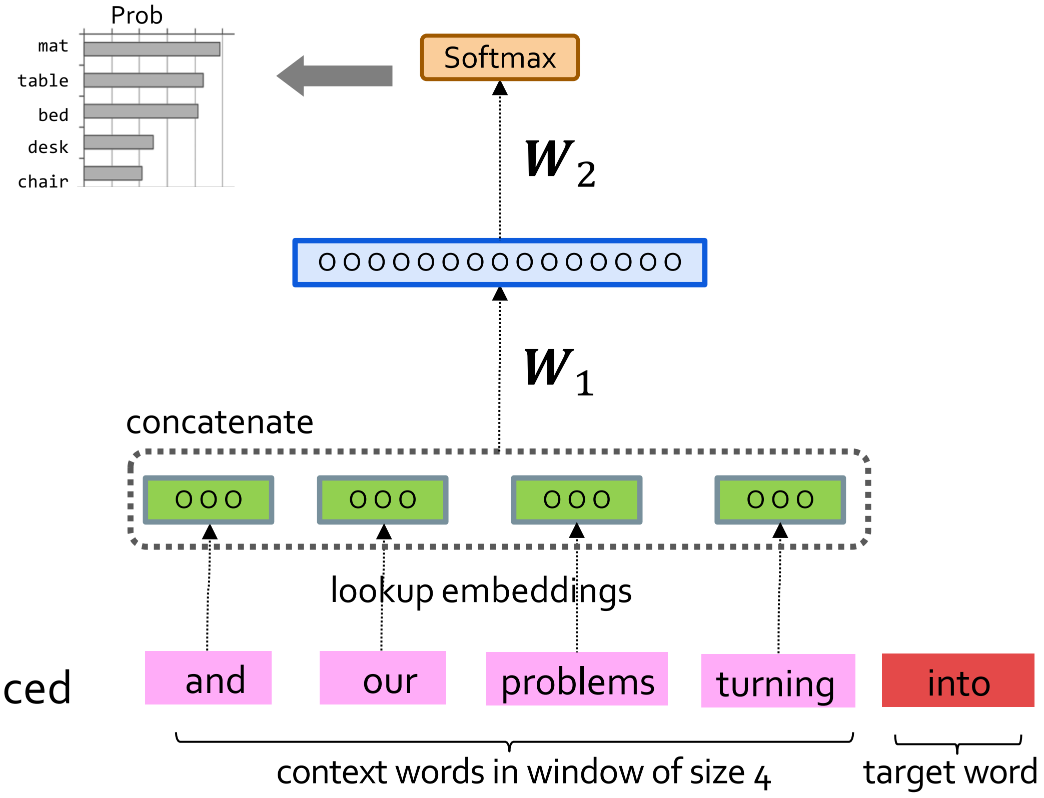
# A Fixed-Window Neural LM: Compared to N-Grams

Improvements over n-gram LM:

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

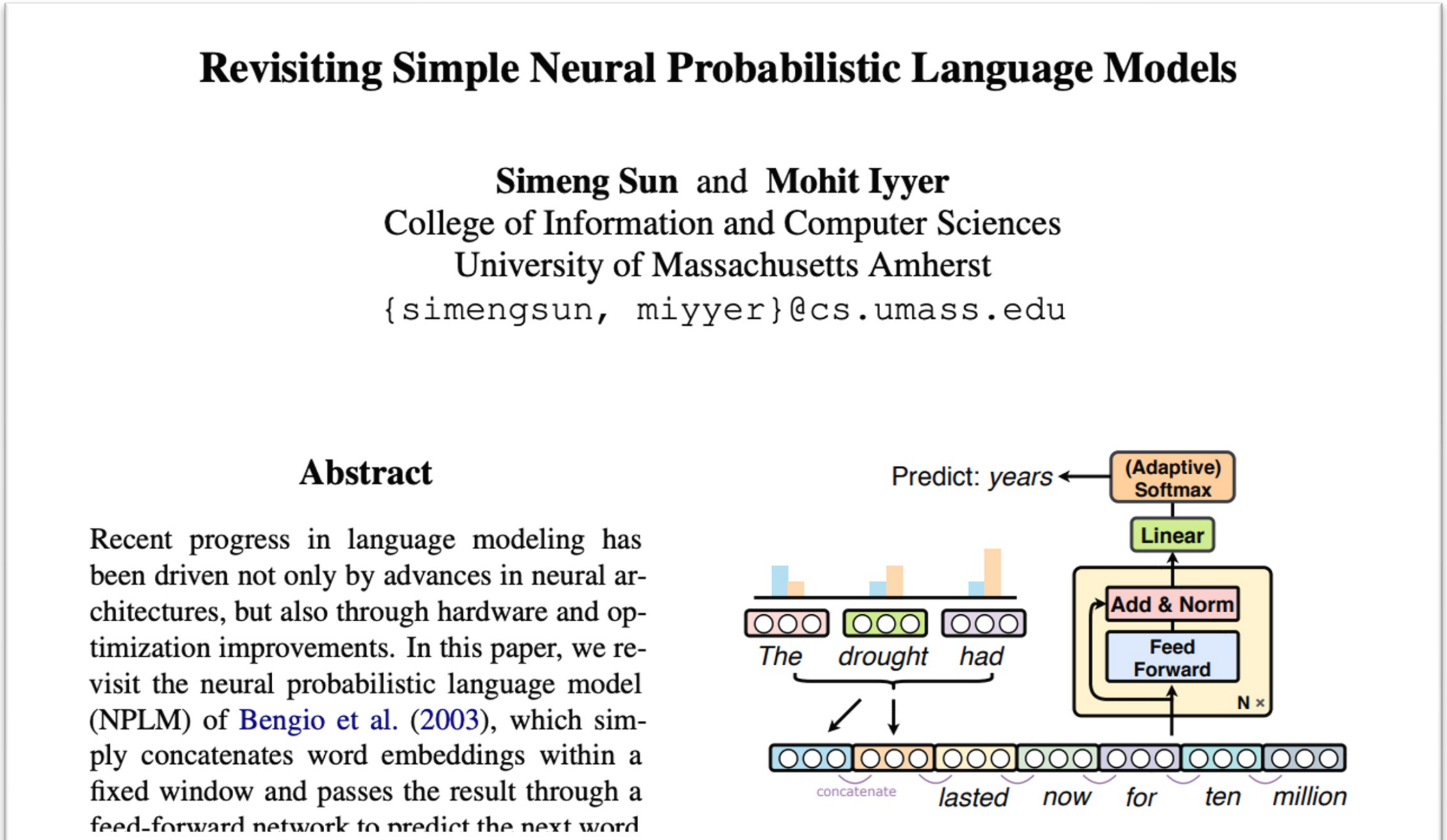
Remaining problems:

- 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

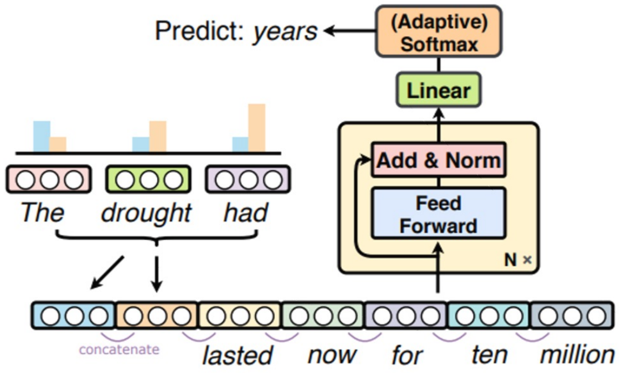


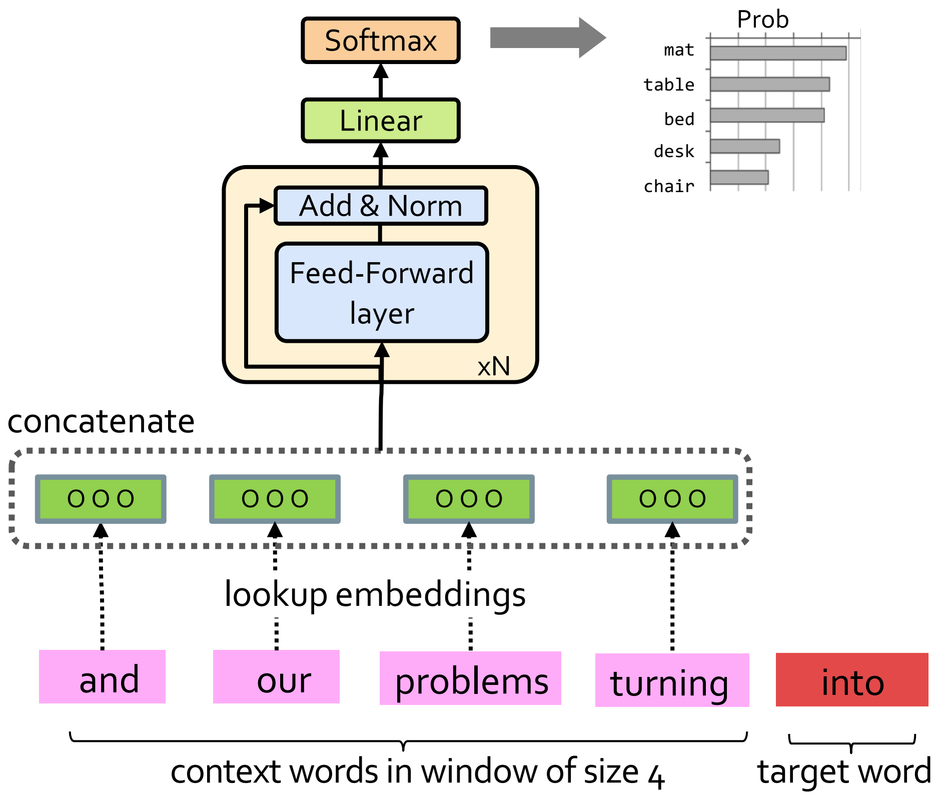
## Revisiting Simple Neural Probabilistic Language Models

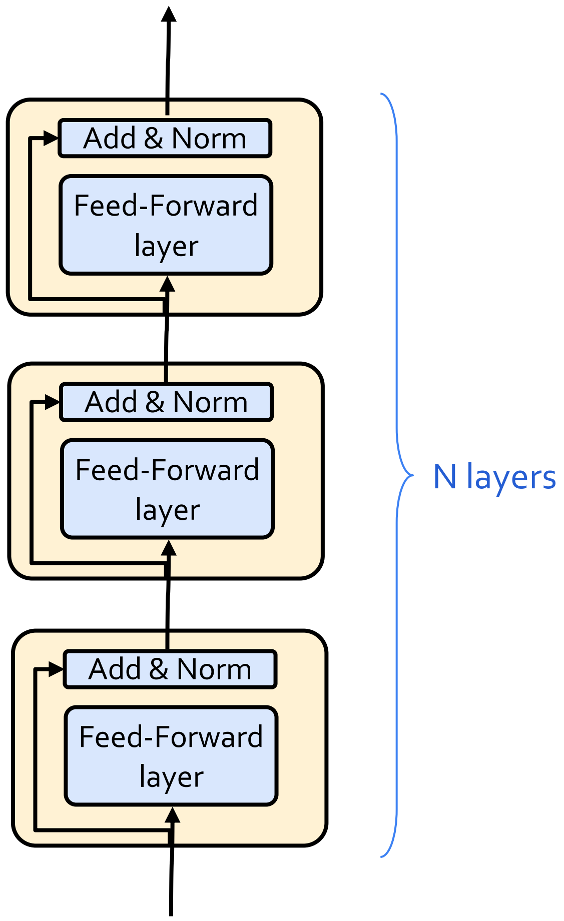
**Simeng Sun** and **Mohit Iyyer**
College of Information and Computer Sciences
University of Massachusetts Amherst
`{simengsun, miyyer}@cs.umass.edu`

### Abstract

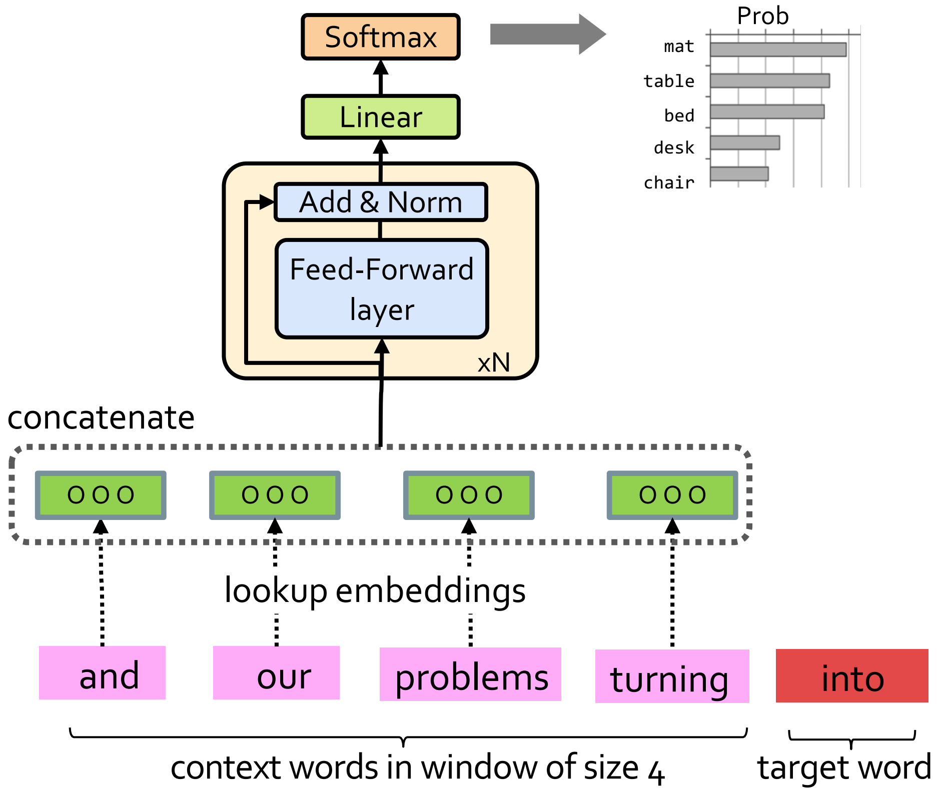
Recent progress in language modeling has been driven not only by advances in neural architectures, but also through hardware and optimization improvements. In this paper, we revisit the neural probabilistic language model (NPLM) of Bengio et al. (2003), which simply concatenates word embeddings within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

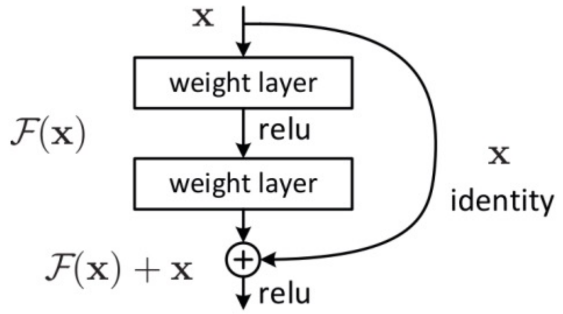






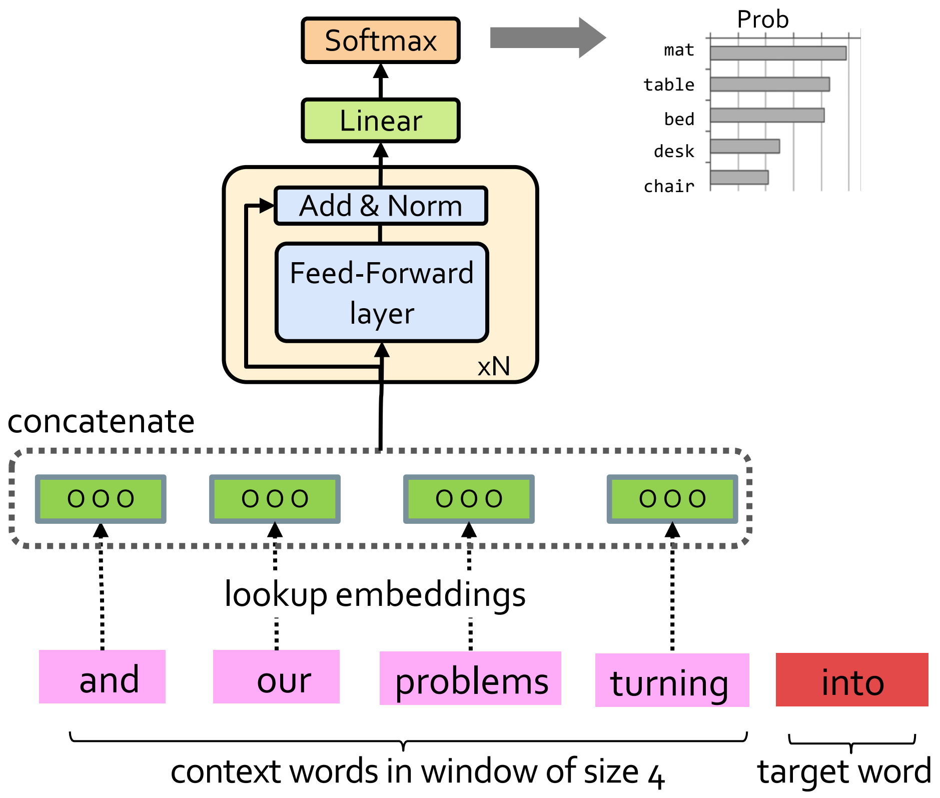
[Sun and Iyyer 2021]




x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
$\mathbf{x}$
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

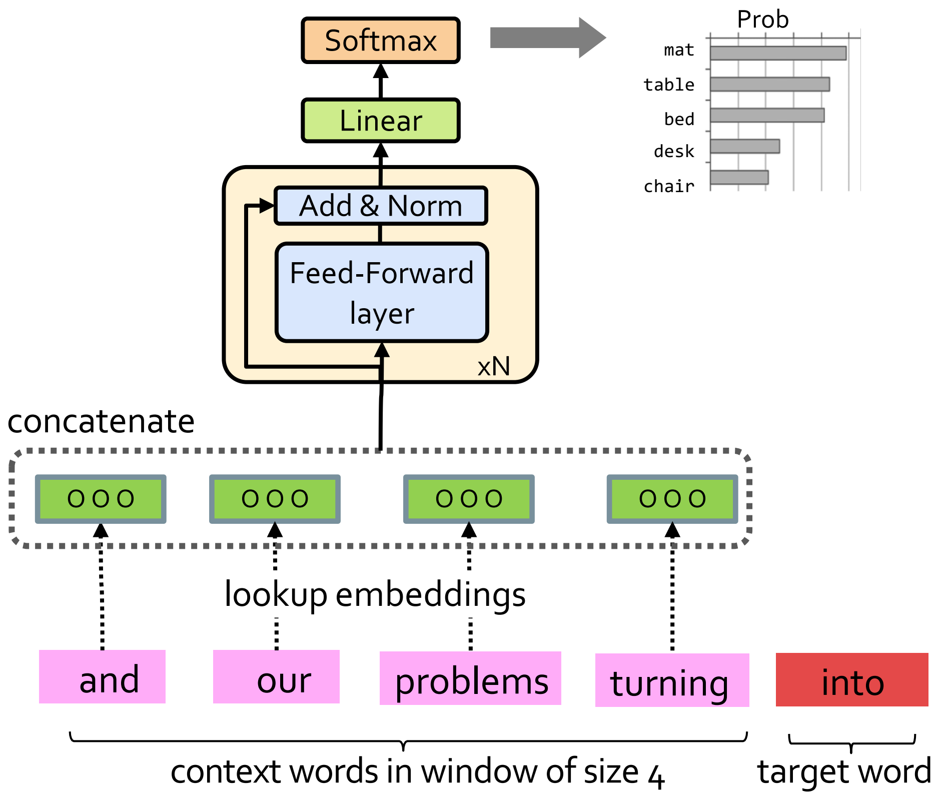
Uses **residual connections** (He et al. 2016) — "information highways" between layers. (we saw them in the earlier chapter)

[Sun and Iyyer 2021]



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

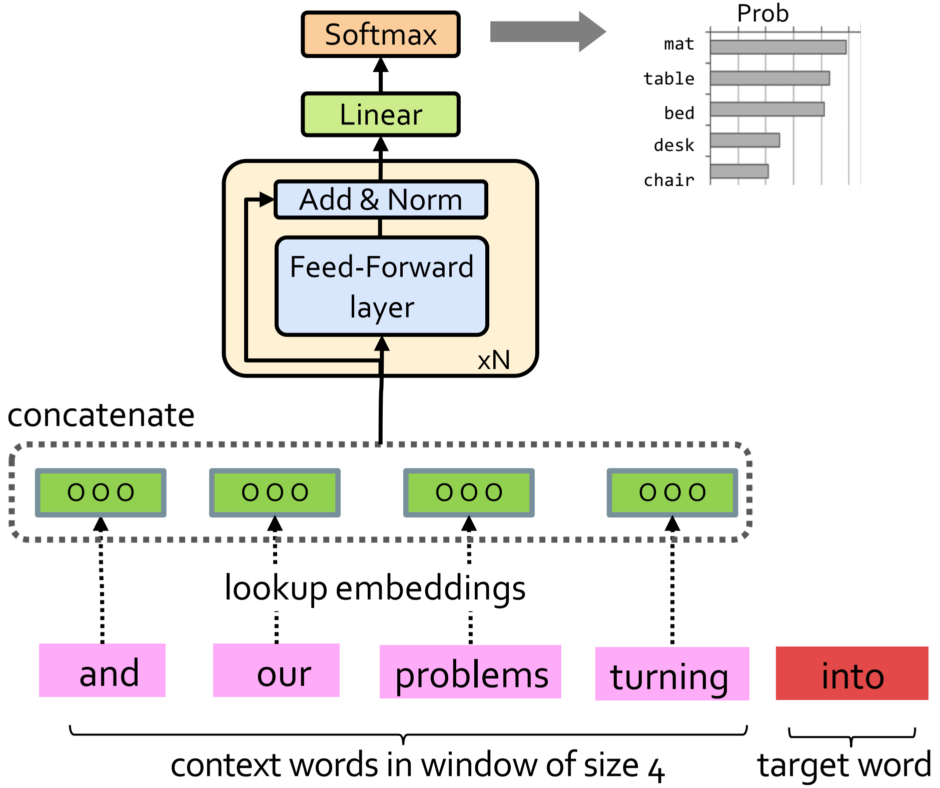
[Sun and Iyyer 2021]



Use “dropout” to avoid overfitting.

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

[Sun and Iyyer 2021]



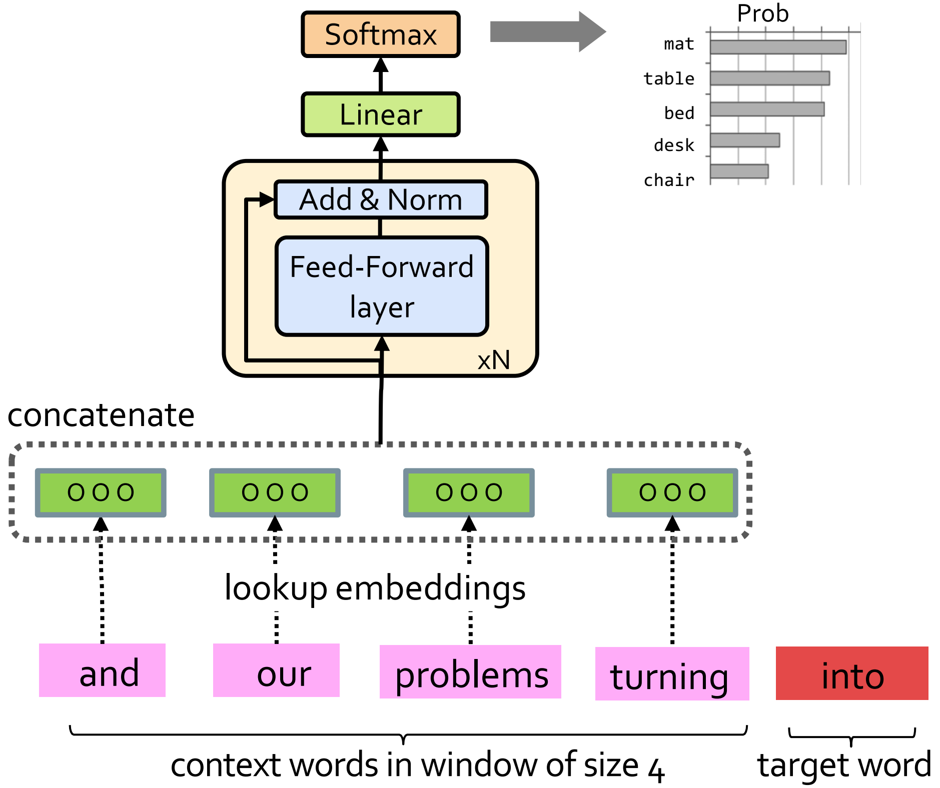
| Model | # Params | Val. perplexity |
|---|---|---|
| Transformer | 148M | 25.0 |
| NPLM-old | 32M[2] | 216.0 |
| NPLM-old (large) | 221M[3] | 128.2 |
| NPLM 1L | 123M | 52.8 |
| NPLM 4L | 128M | 38.3 |
| NPLM 16L | 148M | **31.7** |
| - Residual connections | 148M | 660.0 |
| - Adam, + SGD | 148M |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

Table 1: NPLM model ablation on WIKITEXT-103.

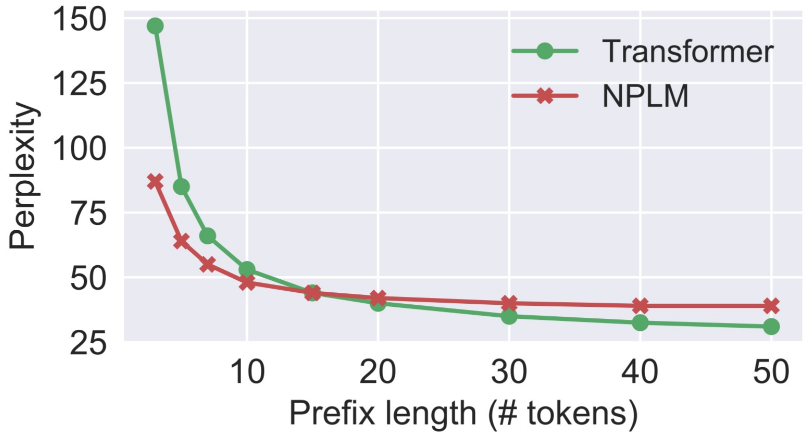
**Takeaways:**

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

[Sun and Iyyer 2021]



## Effect of window size:

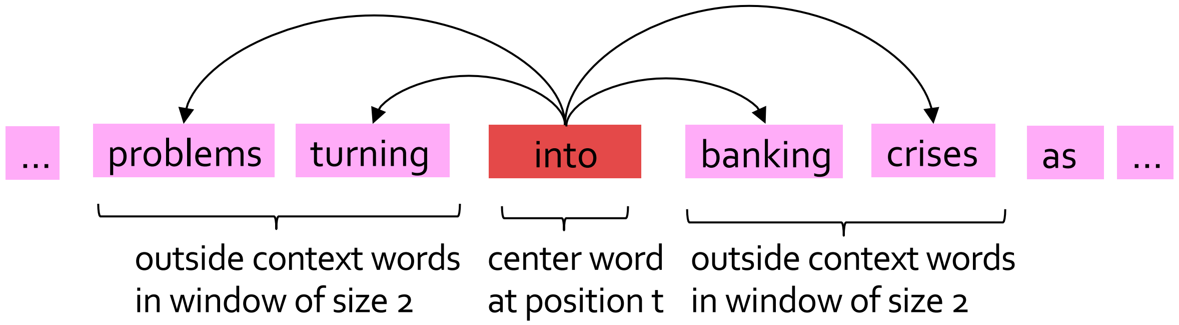


**Fixed-WindowLM (NPLM)** is better than the **Transformer** (will see them in 2 weeks!) with short prefixes but worse on longer ones.

[Sun and Iyyer 2021]

## Flashback to Word2Vec

- Word2Vec objective is similar to the language modeling objective.
- Word2Vec representations are fixed
  - e.g., "play" has one vector regardless of the context.
- In contrast, the meaning representation extracted from more advanced architectures (e.g, FFN-LM) is contextual.


...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2

## Summary

- Language Modeling (LM), a useful predictive objective for language
- Perplexity, a measure of an LM’s predictive ability
- N-gram models (~1980 to early 2000’s),
    - Early instances of LMs
    - Difficult to scale to large window sizes
- FFN-LMs (early and mid-2000’s ),
    - Effective predictive models based on feed-forward networks
    - Difficulty in long-range dependencies

## 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?

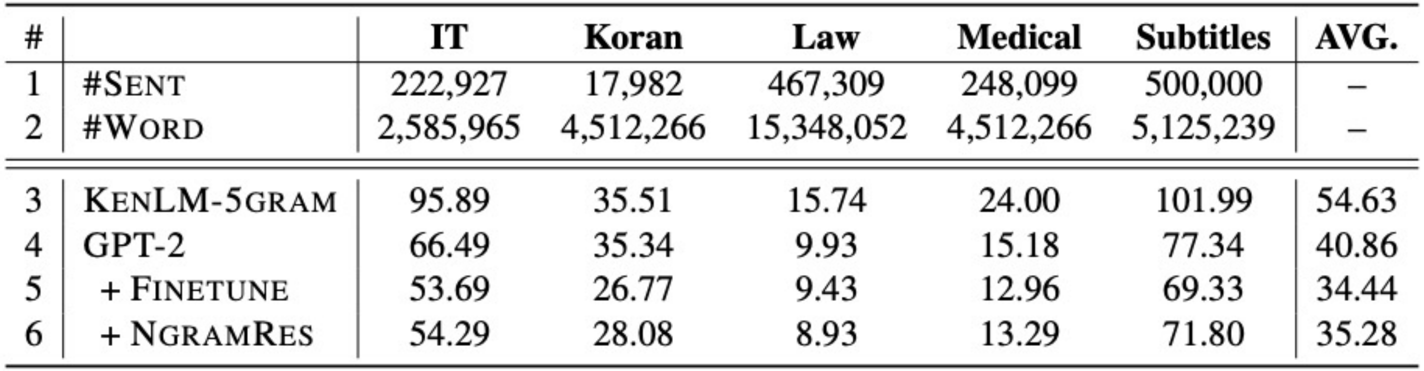
- **Answer:** In n-grams, we treat all prefixes independently of each other! (even those that are semantically similar)

```
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.

## Best of both worlds

- n-grams are easy to compute for small "n".
- Why waste energy to make neural-LMs [re]learn these statistics?
- Li et al. 2022 propose learning neural LMs to fits the residual between an n-gram LM and real-data distribution.
- Allows the neural part to focus on the deeper understanding of language.



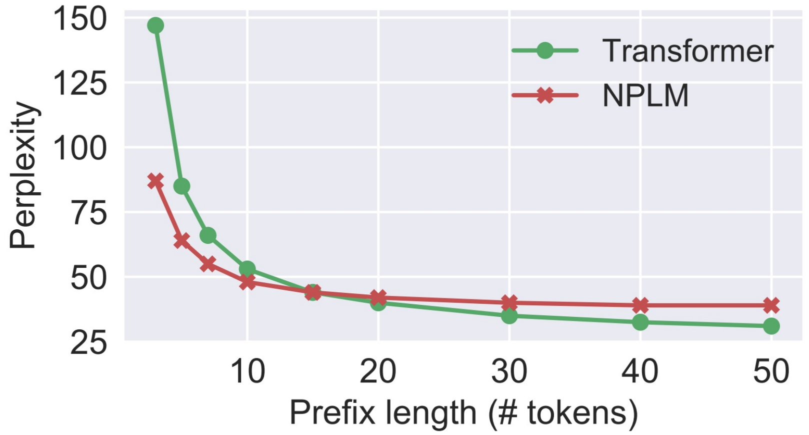
| # | | IT | Koran | Law | Medical | Subtitles | AVG. |
|---|---|---|---|---|---|---|---|
| 1 | #SENT | 222,927 | 17,982 | 467,309 | 248,099 | 500,000 | – |
| 2 | #WORD | 2,585,965 | 4,512,266 | 15,348,052 | 4,512,266 | 5,125,239 | – |
| 3 | KENLM-5GRAM | 95.89 | 35.51 | 15.74 | 24.00 | 101.99 | 54.63 |
| 4 | GPT-2 | 66.49 | 35.34 | 9.93 | 15.18 | 77.34 | 40.86 |
| 5 | + FINETUNE | 53.69 | 26.77 | 9.43 | 12.96 | 69.33 | 34.44 |
| 6 | + NGRAMRES | 54.29 | 28.08 | 8.93 | 13.29 | 71.80 | 35.28 |

Table 1: Test perplexity of five domains. Results in lines 1-2 are the statistical information of each domain. Results

[N-gram Is Back: Residual Learning of Neural Text Generation with n-gram Language Model, Li et al. 2022]

## Summary Thus Far

- Language Models (LM): distributions over language
- N-gram: language modeling via counting
- Neural Language Models: neural networks trained with LM objective
- Fixed-window Neural LM: first of many LMs we will see in this class
- Key question: how to better capture long-range dependencies?


Perplexity
Prefix length (# tokens)
Transformer
NPLM