JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Language Models: Background

Overview

- N-gram language models
- Evaluation
- Neural language models
- Pre-trained language models

Johns Hopkins
Whiting School of Engineering

N-Gram Language Models

Probabilistic Language Models

- Goal: Assign a probability to a sentence
- Why?
  - Machine Translation
    - P(high winds tonight) > P(large winds tonight)
  - Spell Correction
    - P(about fifteen minutes from) > P(about fifteen minuets from)
  - Speech Recognition
    - P(I saw a van) >> P(eyes awe of an)
  - \+ many other tasks

Probabilistic Language Model

- Goal: compute the probability of a sentence or sequence of words:
  - $P(W) = P(w_1, w_2, w_3, w_4, w_5 \dots w_n)$

- Related task: probability of an upcoming word:
  - $P(w_5 | w_1, w_2, w_3, w_4)$

- A model that computes either of these:
  - $P(W)$ or $P(w_n | w_1, w_2, w_3, \dots, w_{n-1})$ is called a language model.

How to compute P(W)

- How to compute this joint probability:
  - P(its, water, is, so, transparent, that)

- Intuition: let’s rely on the Chain Rule of Probability

How to compute P(W)

- Recall the definition of conditional probabilities
  - $P(B|A) = \frac{P(A,B)}{P(A)}$
  - Then we can rewrite: $P(A,B) = \; P(A)P(B|A)$

- The Chain Rule in General
  - $P(x_1, x_2, x_3, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \dots P(x_n|x_1 \dots x_{n=1})$

The Chain Rule applied to compute joint probability of words in sentence

$$P(w_1, w_2, \ldots, w_n) = \prod P(w_i | w_1 w_2 \ldots w_{i-1})$$

- P("its water is so transparent") = P(its) × P(water|its) × P(is|its water) × P(so|its water is) × P(transparent|its water is so)

- How do we estimate these probabilities?

How to estimate these probabilities?

- Try 1: count and divide?
  - $P(transparent \mid its\ water\ is\ so) = \frac{Count(its\ water\ is\ so\ transparent)}{Count(its\ water\ is\ so)}$

- Too many possible sentences!
- We'll never see enough data for estimating these

Markov Assumption

- Simplifying assumption:
  - $P(transparent\ | its\ water\ is\ so) \approx P(transparent\ | its\ water)$

- More generally:
  - $P(w_1, w_2, \dots, w_n) \approx \prod_i P(w_i | w_{i-k} \dots w_{i-1})$

- Unigram model: $P(w_1, w_2, \dots, w_n) \approx \prod_i P(w_i)$
- Bigram model: $P(w_1, w_2, \dots, w_n) \approx \prod_i P(w_i | w_{i-1})$
- Trigram, 4-gram, 5-gram etc.
  - In general, this is insufficient since language has long-term dependencies, but we can often get away with it

Estimating bi-gram probabilities

- Bigram model: $P(w_1, w_2, \ldots, w_n) \approx \prod_i P(w_i|w_{i-1})$
- Maximum likelihood estimate
  - $P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

Practical considerations

- Typically put everything into log space (avoid underflow and adding is faster than multiplying)
- What do we do about rare words? We might have word combinations we never saw in the training set (that we used to estimate probabilities)
  - Smoothing, backoff, interpolation
- There can be LOTS of n-grams
  - Pruning (only store probabilities for frequent ones)
  - Efficient data structures

Evaluation

Extrinsic (in-vivo) Evaluation

- To compare models A and B:
  - Put each model in a real task: Machine Translation, speech recognition, etc.
  - Run the task, get a score for A and for B
    - How many words translated correctly
    - How many words transcribed correctly
  - Compare accuracy for A and B
- Disadvantages:
  - Expensive, time-consuming
  - Doesn't always generalize to other applications

Intrinsic (in-vitro) evaluation

- Perplexity
  - Directly measures language model performance at predicting words
  - Single general metric for language models
  - Doesn't necessarily correspond with real application performance
  - Useful for large language models (LLMs) as well as n-grams
- Data setup:
  - Train model (e.g. estimate probabilities) on training set
  - Compute perplexity on held-out test set

Perplexity: Intuition

- A good LM is one that assigns a higher probability to the next word that actually occurs
- "Its water is so ________"
  - Model A: transparent: 0.3, blue: 0.3, orange: 0.01, red: 0.02
  - Model B: transparent: 0.01, blue: 0.01, orange: 0.01, red: 0.9

- Generalize to all words: best LM assigns high probability to the entire test set
- When comparing two LMs, A and B
  - We compute P_A(test set) and P_B(test set)
  - The better LM will give a higher probability to (=be less surprised by) the test set than the other LM

Perplexity

- Probability depends on size of test set
  - Probability gets smaller the longer the text
  - Better: a metric that is **per-word**, normalized by length
- Perplexity is the inverse probability of the test set, normalized by the number of words

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Perplexity

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

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

(The inverse comes from the original definition of perplexity from cross-entropy rate in information theory)
Probability range is [0,1], perplexity range is [1,∞]
Minimizing perplexity is the same as maximizing probability

Perplexity

- Perplexity for a bigram model

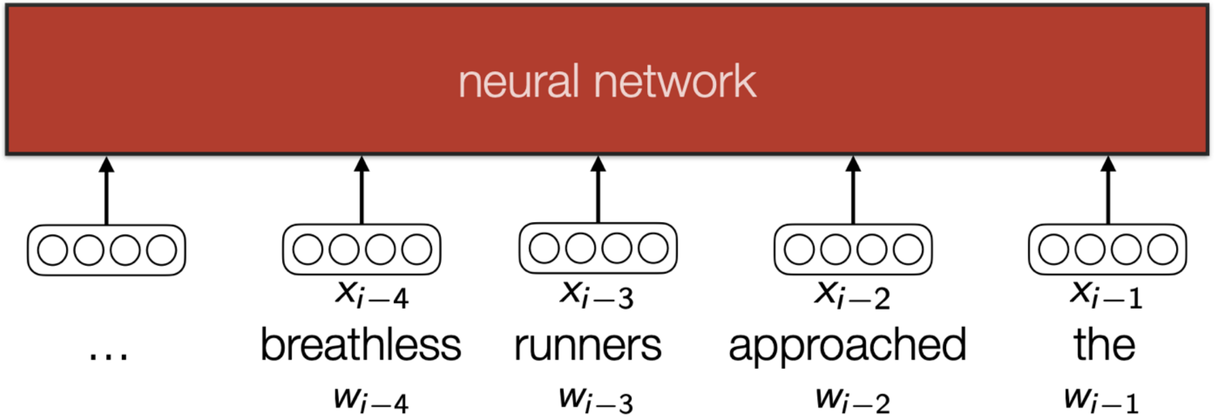
$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Neural Language Models

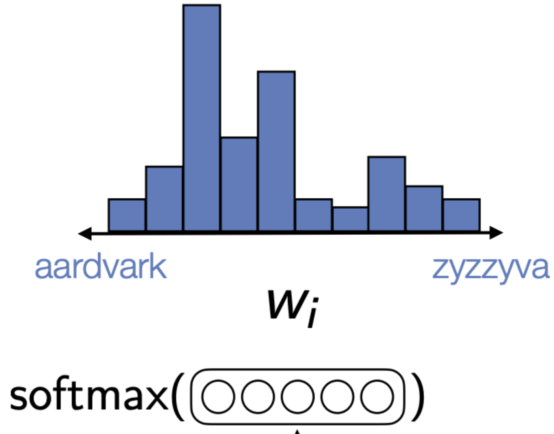
Neural Language Model

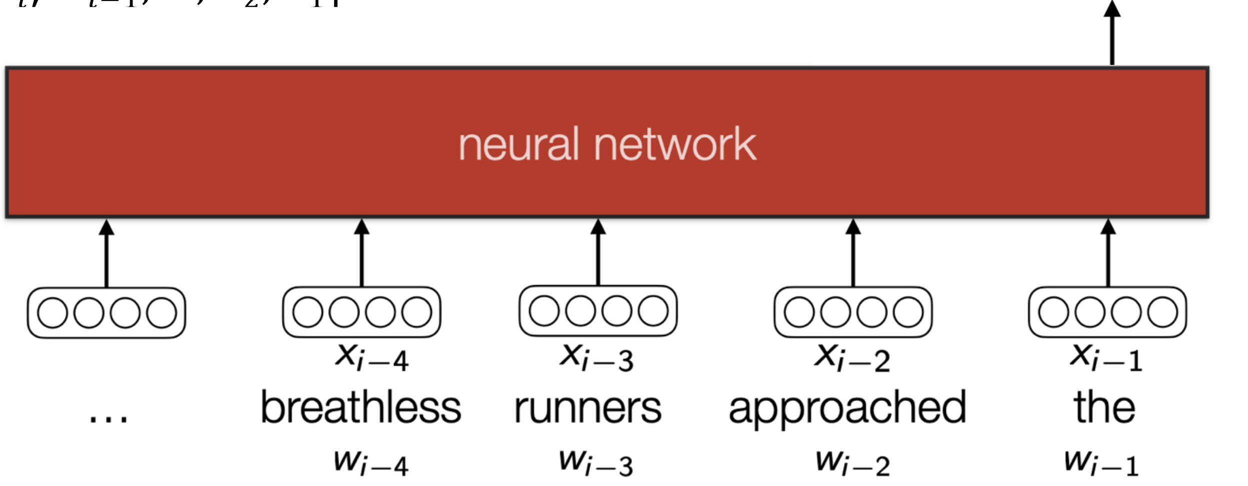
- Don't count, predict
- Input: word embeddings $[x_1, x_2, \ldots x_n]$



Neural Language Model

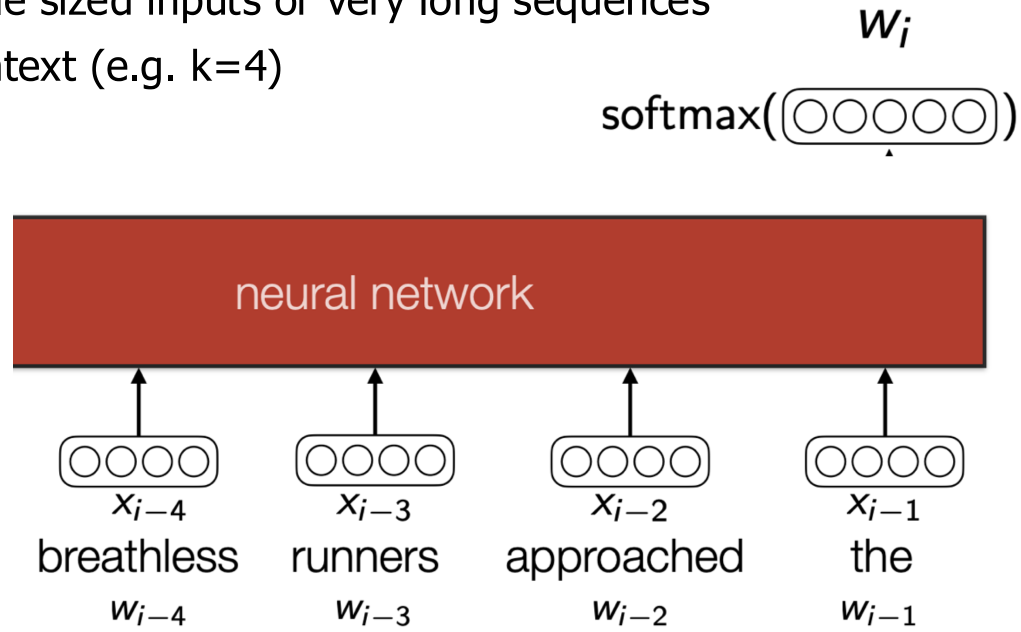
- Don’t count, predict
- Input: word embeddings $[x_1, x_2, \ldots x_n]$
- Output: P$(w_i, w_{i-1}, \ldots, w_2, w_1]$





A feed-forward neural language model

- We can't handle variable sized inputs or very long sequences
- Fix size of previous context (e.g. k=4)

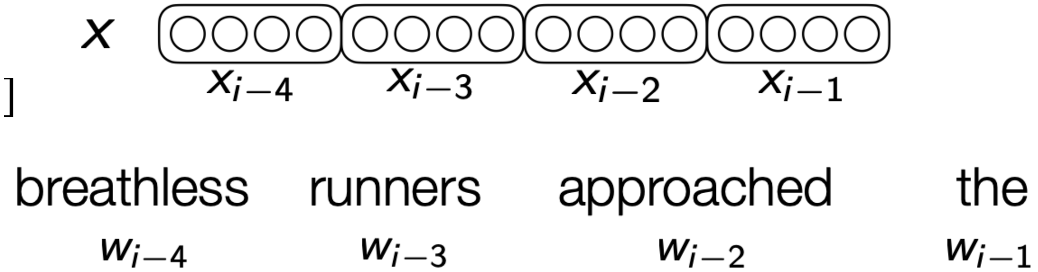


A feed-forward neural language model

Dimensionality

We can start with pretrained embeddings like word2vec or train the embeddings along with the model

Concatenate k word embeddings: $\mathbf{x} = [x_{i-4}; x_{i-3}; x_{i-2}; x_{i-1}]$



$4d_{\text{word embeddings}}$

A feed-forward neural language model

Dimensionality

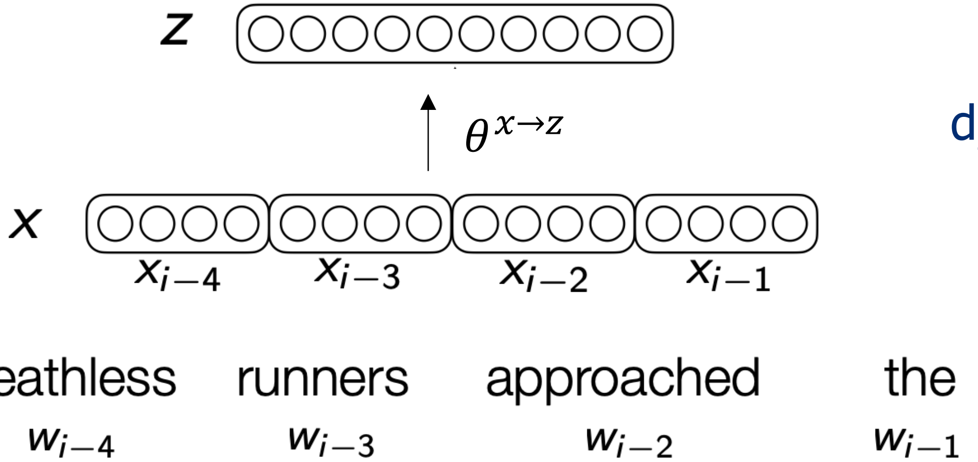
Hidden layer f($\theta^{x \to z}$ **x**)

d_z

d_z x $4d_{\text{word embeddings}}$

Concatenate k word embeddings: **x** = $[x_{i-4}; x_{i-3}; x_{i-2}; x_{i-1}]$

$4d_{\text{word embeddings}}$



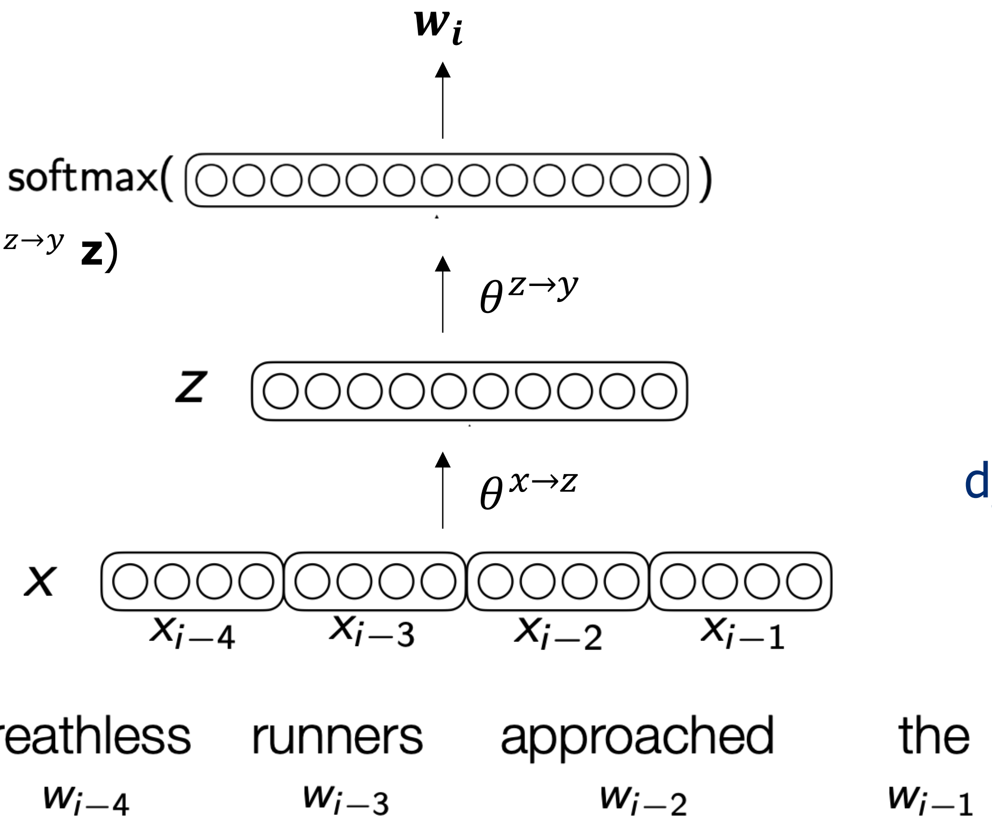
Johns Hopkins Whiting School of Engineering

A feed-forward neural language model

Output layer SoftMax($\theta^{z \to y}$ **z**)

Hidden layer f($\theta^{x \to z}$ **x**)

Concatenate k word embeddings: **x** = $[x_{i-4}; x_{i-3}; x_{i-2}; x_{i-1}]$



Dimensionality

V

V x d_z

d_z

d_z x $4d_{\text{word embeddings}}$

$4d_{\text{word embeddings}}$

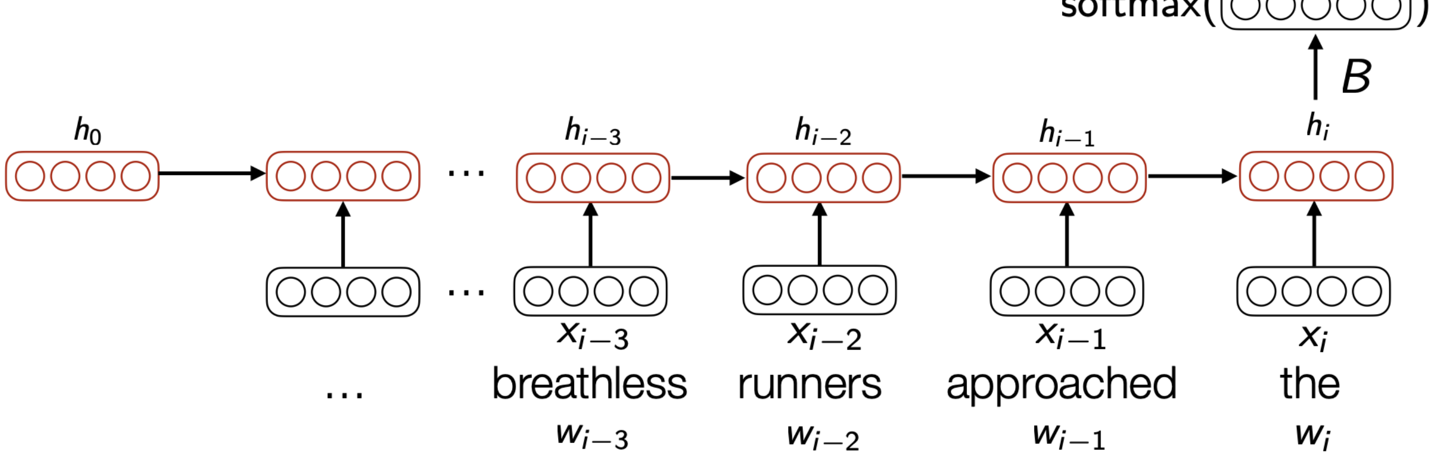
Comparison with n-gram language model

- Improvements:
  - Model size: $O(V)$ instead of $O(V^n)$
  - Lack of sparsity
  - Sharing of representations across words
- Remaining challenges:
  - We still need to truncate context; model size grows linearly with context size

Solutions: *Recurrent* neural network

- Maintain a context vector, h. At each timestep (w_j), compose the context with the
- current word x_j to create a new context for the next timestep:

$$h_j = RNN(x_j, h_{j-1})$$

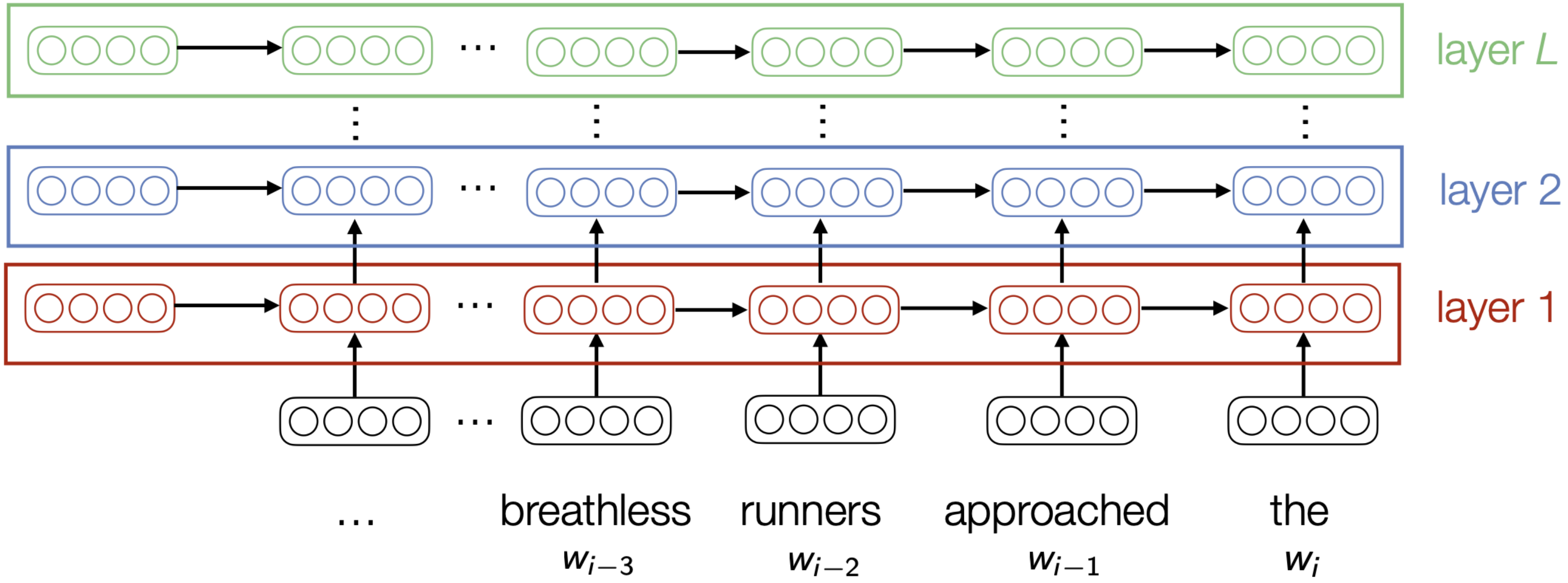


Training tricks

- Same problem with word embeddings, softmax over the vocabulary is expensive → hierarchical softmax
- In theory, we can propagate information over arbitrarily long context → in practice gradient can *vanish* or *explode* → gradient clipping, gating mechanisms
- Overfitting → dropout and regularization

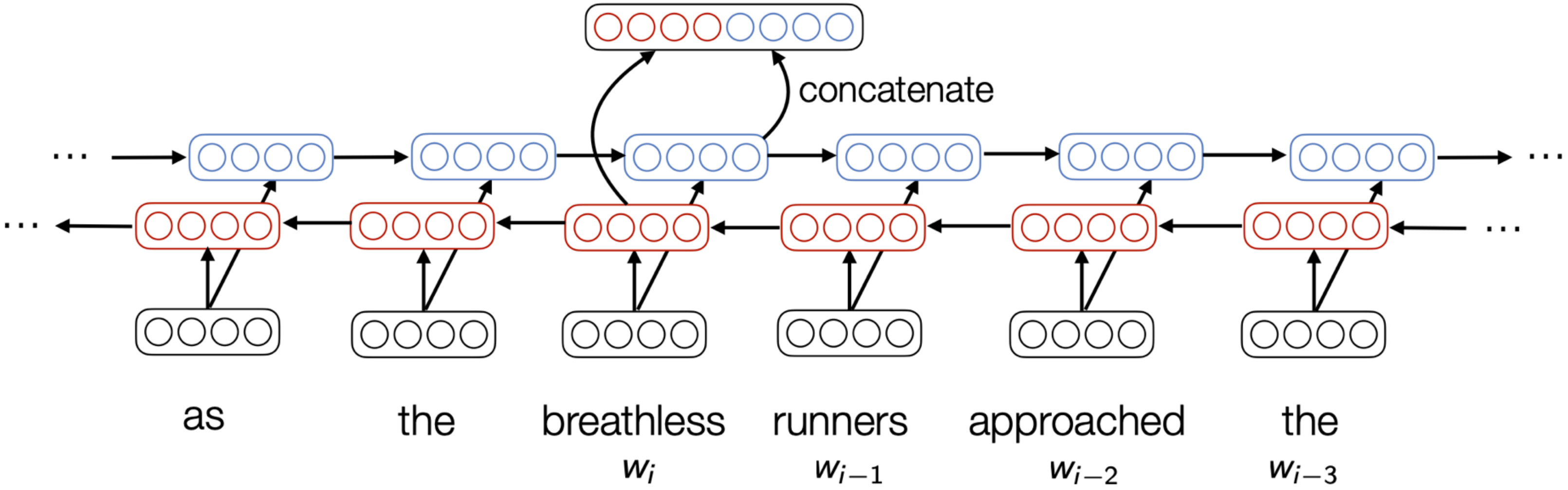
- Other architectures:
  - Long short term memory (LSTM)

Stacking RNNs



Bidirectional RNNs

- In language it's often useful to model *past* and *future* context
- We can run an RNN in the opposite direction (reverse reading order)
- Combining forwards and backwards directions works best



Quiz

1. Which of the following methods might be a reasonable way to handle out-of-vocabulary words in a neural language model?
   A. Replace all out-of-vocabulary words with a special token that has its own word embedding
   B. Use sub-word embeddings, where you can break unknown words into character-level representations if needed
   C. Using additive noise to smooth embeddings for unseen words
   D. A and B
   E. A and C

2. In what circumstances would you guess that a 3-gram language model would perform better than a 4-gram language model?
   A. If your data has complex sentence structures that require modeling long-term dependencies
   B. If your data is likely to frequently recurring bi-grams like “New York”
   C. If your training data is small or from a different domain than your test data
   D. A 4-gram model will always perform better. It will just require more compute / space for storing counts

JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

Pre-trained Language Models

Recall: Word Embeddings

- Key idea: pre-train word embeddings with a self-supervised objective (e.g. CBOW or skip-gram in word2vec)
- Incorporate pre-trained word embeddings into task-specific models

- Problem:
  - Single embedding representation for each word

Recall: Word Embeddings

play =

-1.36	-0.9	0.71	-0.22	0.77	-1.36	-0.72	0.71	0.15	...

- "the new-look **play** area is due to be completed by early spring 2020"
- "gerrymandered congressional districts favor representatives who **play** to the party base"
- "the freshman then completed the three-point **play** for a 66-63 lead"

- Multiple senses get entangled
- Nearest neighbors:
  - playing played Play
  - game plays football
  - games player multiplayer

Contextualized Representations

- Preferred approach: *contextualized* representations where the embedding changes with context

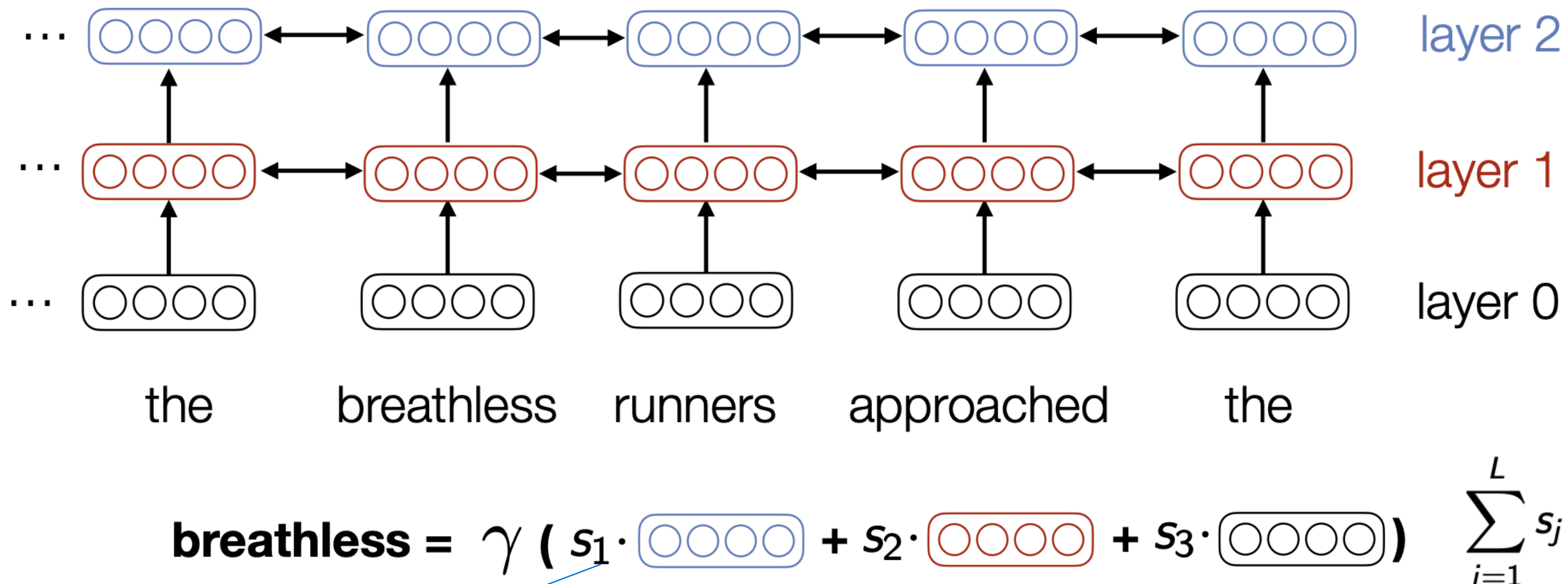
- [But still want to leverage self-supervised training on large data]

- ELMo ("**E**mbeddings from **L**anguage **Mo**del")
  - Use hidden representation from language model
  - (keep middle layers instead of only the embedding layer)

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. NAACL

ELMo: Deep contextualized word embeddings

Stacked bi-directional LSTM



$$\sum_{j=1}^{L} s_j = 1$$

Interpolated weights are task-specific (fine-tuned on supervised data)

ELMo: Deep contextualized word embeddings

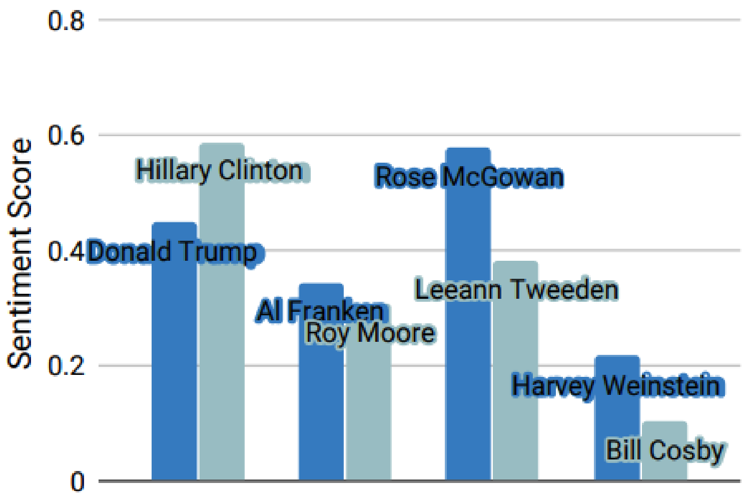
- Adding ELMo to existing state-of-the-art models provides significant improvement on essentially all NLP tasks.

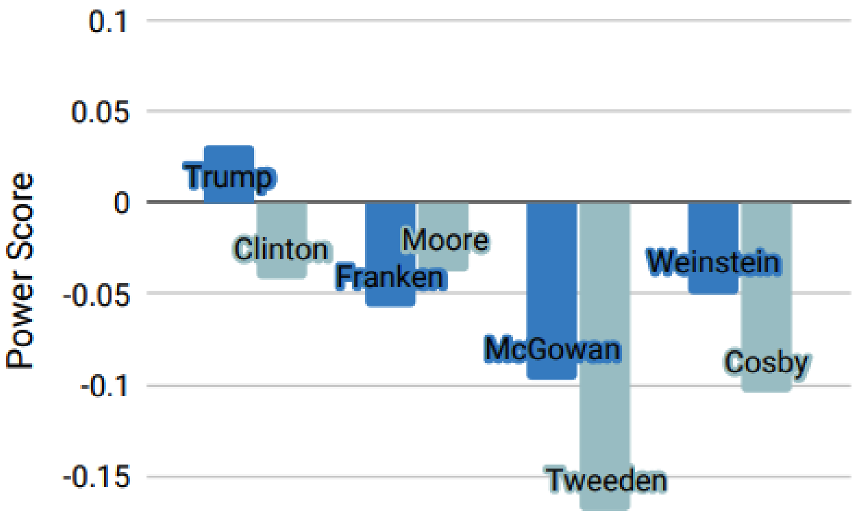
	TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
question answering	SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
natural language inference	SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
semantic role labeling	SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
coreference	Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
named entity recognition	NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
sentiment analysis	SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

How is ELMo useful for social text processing?

- Higher-performance for supervised learning
  - [Mostly eclipsed by future models]
- Adding context to lexicons:
  - Recall connotation frames for power, agency, and sentiment:
    - "The hero **deserves** appellation"
    - "The boy **deserves** punishment"
- Broad approach:
  - Take contextualized embeddings, average them, train a model to predict lexicon score from averaged embeddings
  - Use model to predict lexicon scores in context

Contextual Affective Analysis: A Case Study of People Portrayals in Online #MeToo Stories





Anjalie Field, Gayatri Bhat, and Yulia Tsvetkov. "Contextual Affective Analysis: A Case Study of People Portrayals in Online #MeToo Stories" ICWSM (2019)

How is ELMo useful for social text processing?

- Higher-performance for supervised learning
  - [Mostly eclipsed by future models]
- Adding context to lexicons:
  - “The hero **deserves** appellation”
  - “The boy **deserves** punishment”
- Word embeddings analyses?

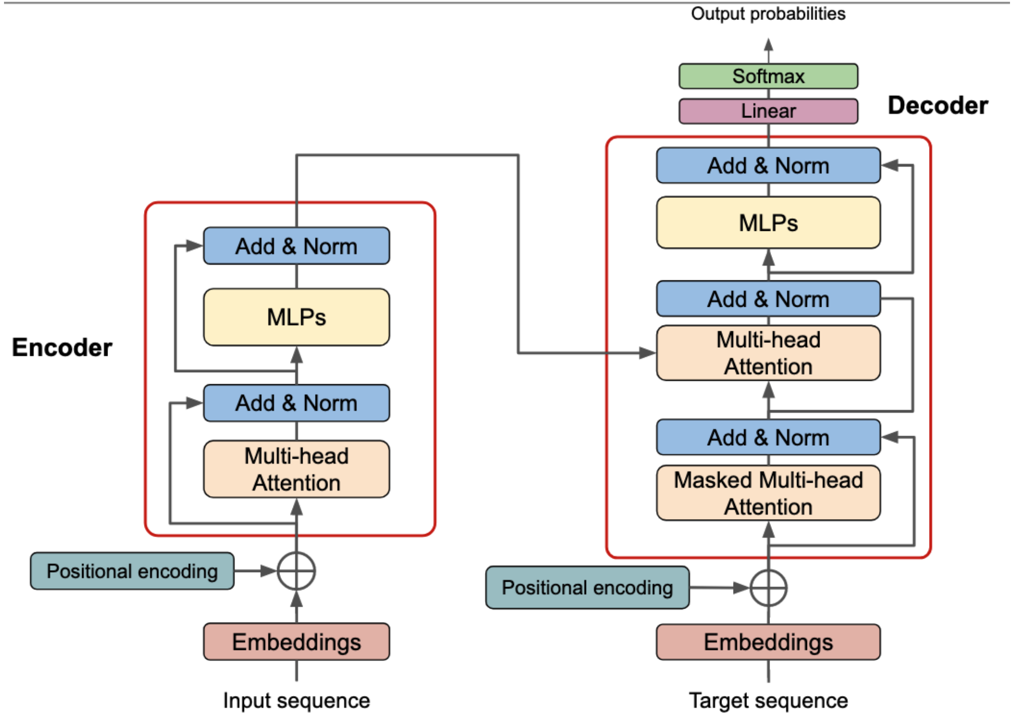
Anjalie Field, Gayatri Bhat, and Yulia Tsvetkov. "Contextual Affective Analysis: A Case Study of People Portrayals in Online #MeToo Stories" ICWSM (2019)

ELMo → BERT


- Key differences:
  - BiLSTM –> Transformer
  - Treat layers as static embeddings → keep entire model and update it during task-specific training
  - Next token prediction → Masked Language Modeling training objective

The Transformer

- Stacks of transformer blocks, each of which is a multilayer network that maps sequences of input vectors $(x_1, \ldots, x_n)$ to sequences of output vectors $(z_1, \ldots, z_n)$ of the same length
- Blocks are made by combining simple linear layers, feedforward networks, and self-attention layers (the key innovation of transformers)



Recap

- N-gram language models
- Evaluation
- Neural language models
- Pre-trained language models (ELMo→BERT)

Next class:

- Masked Language Model use cases in computational social science

Please fill out mid-semester course survey on Piazza!

Acknowledgements

- Slide thanks to Jurafsky&Martin and Strubell&Tsvetkov