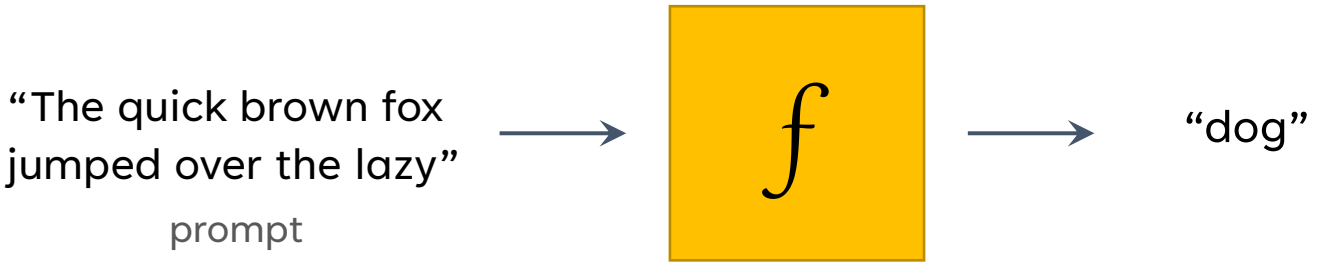
CS 490: NATURAL LANGUAGE PROCESSING

Dan Goldwasser, Abulhair Saparov

Lecture 18: Language Modeling

WHAT IS LANGUAGE MODELING?

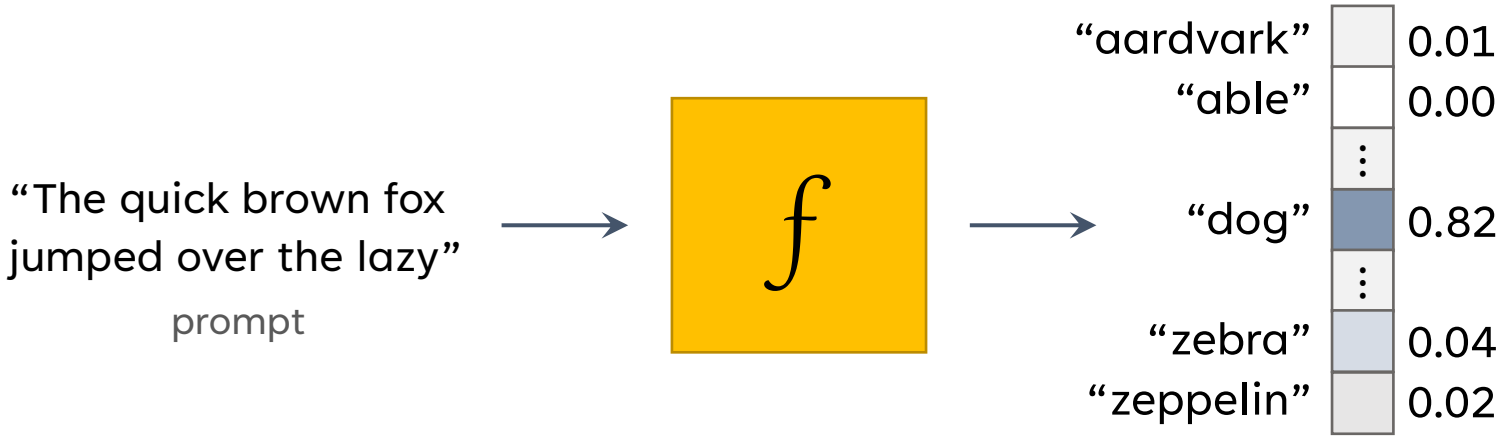
- In simple terms, it is the task of predicting the next word, given the previous *n* words.



- Language modeling is a multi-class classification task.
- Each word in the vocabulary is an output class.

LANGUAGE MODELING

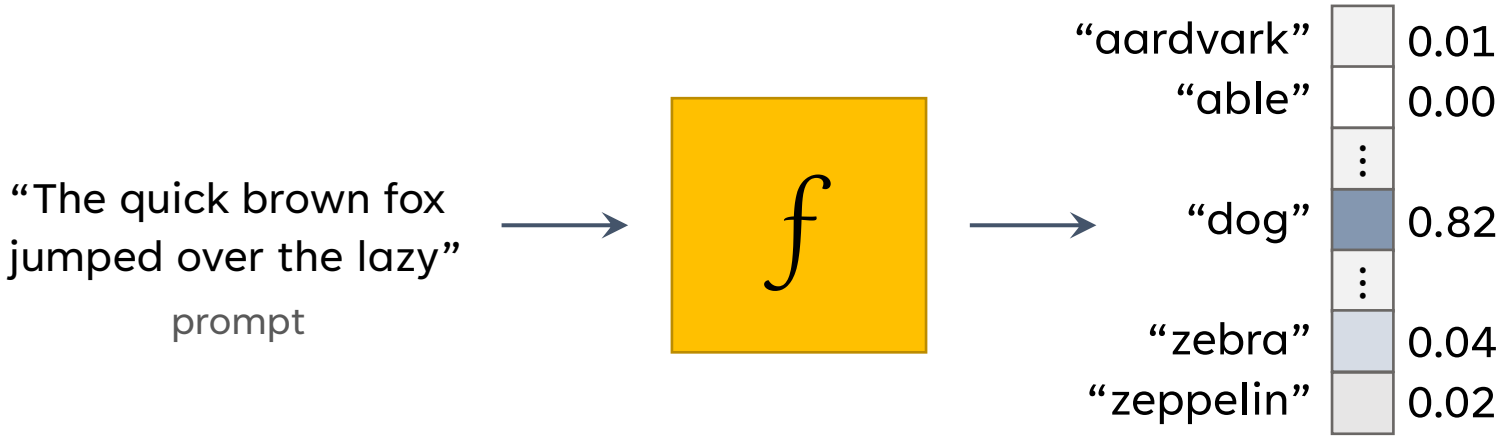
- In simple terms, it is the task of predicting the next word, given the previous *n* words.



- Language modeling is a multi-class classification task.
- Each word in the vocabulary is an output class.
- More precisely: if $x_1, x_2, \ldots, x_n$ is a sequence of words, $f(x_1, \ldots, x_n) = p(x_n \mid x_1, \ldots, x_{n-1})$
- Language is ambiguous, and so language modeling is a probabilistic task.

LANGUAGE MODELING

- What is f?
- It is a machine learning model, trained on many examples of input-output pairs.



- It is easy to find training data:
  - Take any n-word sequence of text (such as from the internet),
  - The first $n-1$ words are the input, and the last word is the output.
- Language modeling is unsupervised/self-supervised.
- But we must be mindful of the training data. There may be inaccuracies or noise.

LANGUAGE MODELING

- What is f?
- It is a machine learning model, trained on many examples of input-output pairs.

- We can use language models to assign probabilities of phrases, or sentences.

$$p(\text{‘The quick fox’}) = p(\text{‘The’})\ p(\text{‘quick’}|\text{‘The’})\ p(\text{‘fox’}|\text{‘The’},\text{‘quick’})$$

- By the chain rule of probability theory:

$$p(x_1, \ldots, x_n) = p(x_n \mid x_1, \ldots, x_{n-1})\ p(x_1, \ldots, x_{n-1})$$
$$= p(x_1)\ p(x_2 \mid x_1)\ p(x_3 \mid x_1, x_2) \ldots p(x_n \mid x_1, \ldots, x_{n-1})$$

- We can use language models to compare the probabilities of different phrases/sentences of arbitrary length:
- p(‘The quick fox’) > p(‘The quick turtle’)
- p(‘The quick fox’) > p(‘The quikc fox’)

LANGUAGE MODELING

- What is f?
- It is a machine learning model, trained on many examples of input-output pairs.

- We can use language models to assign probabilities of phrases, or sentences.

log p('The quick fox') = log p('The') + log p('quick'|'The') + log p('fox'|'The','quick')

- By the chain rule of probability theory:

$$\log p(x_1, \ldots, x_n) = \log p(x_n \mid x_1, \ldots, x_{n-1}) + \log p(x_1, \ldots, x_{n-1})$$
$$= \log p(x_1) + \log p(x_2 \mid x_1) + \log p(x_3 \mid x_1, x_2) + \ldots + \log p(x_n \mid x_1, \ldots, x_{n-1})$$

- We can use language models to compare the probabilities of different phrases/sentences of arbitrary length:
- log p('The quick fox') > log p('The quick turtle')
- log p('The quick fox') > log p('The quikc fox')

OTHER NLP TASKS AS LANGUAGE MODELING

- Many other NLP tasks can be “reduced” into language modeling:
- Spam detection
  - Prompt: “Email: Dear customer, … Question: Is this (a) spam, or (b) not spam? Answer:”
- Machine translation
  - Prompt: “Translate the following into Spanish: ‘The quick brown fox…’ Translation:”
- Sentiment analysis
  - Prompt: “Review: This product was not great… Is this review (a) positive, (b) negative, or (c) neutral? Answer:”
- Question answering
  - Prompt: “Question: Bob has 5 apples. Alice gave 10 apples to Bob. Alice now has 23 apples. How many apples did Alice start with? Answer:”

  …

- Thus, if we can train a model to do well on language modeling, it may be able to perform well on many other NLP tasks.

ML METHODS FOR LANGUAGE MODELING

- What machine learning model can we use to learn f?
- You could use methods we covered in previous lectures:
  - Logistic regression
  - Multi-layer perceptron
  - RNN
  - Transformer
- But language modeling is an old problem.
  - There is a long history of different methods.
  - We will introduce some methods that were specifically designed for this task.
  - Followed by more recent methods.

UNIGRAM MODEL

- If we assume that each word is independent, we obtain a simple model.

$$p(x_n \mid x_1, \ldots, x_{n-1}) = p(x_n) \text{ for all } n$$

- Imagine putting all words of a large corpus in a bag, shuffling their order, and picking one at random.
- How can we estimate the probability of picking a specific word, say "cat"?
- Simple approach:
  - Count the number of times "cat" appears in your training data,
  - Then divide by the total number of words in the data.
- This is called a unigram model.

BIGRAM MODEL

- What are some shortcomings of this model?
- The strong independence assumption causes the model to throw away all word order information.
- What if we instead made a slightly weaker assumption:

$$p(x_n \mid x_1, \ldots, x_{n-1}) = p(x_n \mid x_{n-1}) \text{ for all } n$$

- Each word depends on only the previous word.
- We can extend the counting procedure from the unigram model:
  - For every pair of words in the vocabulary (w_1, w_2), count the number of times w_2 appears after w_1.
  - Then we can estimate: $p(w_n \mid w_{n-1}) = \dfrac{\#\text{ of times } w_n \text{ appeared after } w_{n-1} \text{ in the training set}}{\#\text{ of times } w_{n-1} \text{ appeared in the training set}}$

N-GRAM MODEL

- We can extend this to n-gram models, for arbitrary n.
- Each word depends on only the previous $n - 1$ words.
- We can extend the counting procedure:
  - For every sequence of n words in the vocabulary ($w_1, \ldots, w_n$), count the number of times w_n appears after ($w_1, \ldots, w_{n-1}$).
  - Then we can estimate:

$$p(w_n \mid w_1, \ldots, w_{n-1}) = \frac{\text{\# of times } w_n \text{ appeared after } w_1, \ldots, w_{n-1} \text{ in the training set}}{\text{\# of times } w_1, \ldots, w_{n-1} \text{ appeared in the training set}}$$

SAMPLING FROM LANGUAGE MODELS

- One way to sample from language models is to generate the output word-by-word.
- Suppose we have sampled $w_1, \ldots, w_{n-1}$ so far.
- We compute $\mathrm{p}(w_n = t \mid w_1, \ldots, w_{n-1})$ for all words t in the vocabulary.
- Then we choose the word t with probability $\mathrm{p}(w_n = t \mid w_1, \ldots, w_{n-1})$.

- We then repeat the procedure for the next token.

- There are other sampling schemes:
- Greedy sampling/greedy decoding:
  - At each step, pick the word t that has the highest probability.

TEMPERATURE SAMPLING

- A middle-ground between greedy sampling and standard sampling:
- Choose word t with probability proportional to

$$\exp\{ \log\{p(w_n = t \mid w_1, \ldots, w_{n-1})\} / T \}$$

- T is the temperature parameter.
- If $T = 1$, this is standard sampling.
- At $T = 0$ (at the limit, more precisely), this is greedy sampling.
  - Why?
  - Consider the difference in the log probability of the most likely word and the log probability of the 2nd most likely word.
  - As T gets smaller, then this difference is scaled larger.
  - After renormalizing, the most likely word will reach probability 1, in the limit.

TEMPERATURE SAMPLING

- A middle-ground between greedy sampling and standard sampling:
- Choose word t with probability proportional to

$$\exp\{ \log\{p(w_n = t \mid w_1, \ldots, w_{n-1})\} / T \}$$

- T is the temperature parameter.
- You can also set $T > 1$.
- As T goes to infinity, the term inside the exponent goes to 0.
  - After renormalizing, this results in a uniform distribution.
  - So setting $T > 1$ adds more "randomness" to the language model's predictions.

SAMPLES FROM N-GRAM MODELS

- From language models trained on a Shakespeare corpus:
- Unigram/1-gram:
  - "To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have"
  - "Hill he late speaks; or! a more to leg less first you enter"
- Bigram/2-gram:
  - "Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow."
  - "What means, sir. I confess she? then all sorts, he is trim, captain."

SAMPLES FROM N-GRAM MODELS

- From language models trained on a Shakespeare corpus:
- Trigram/3-gram:
  - "Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done."
  - "This shall forbid it should be branded, if renown made it empty"
- 4-gram:
  - "King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;"
  - "It cannot be but so."

SAMPLES FROM N-GRAM MODELS

- From language models trained on Wall Street Journal text:
- Unigram/1-gram:
  - "Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives"
- Bigram/2-gram:
  - "Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her"
- Trigram/3-gram:
  - "They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions"

EVALUATING LANGUAGE MODELS

- How to measure the performance of a language model (LM)?
- One way is to measure its performance on a downstream task:
  - E.g., apply the LM to a question-answering dataset and measure the accuracy of the answers.
  - (and/or precision, recall, F1-score, etc)
  - Or sentiment analysis, spam detection, document classification, etc.
- This is called extrinsic evaluation.
- Disadvantages:
  - What if we don't have a labeled dataset for the downstream task?
  - Good performance on one downstream task doesn't necessarily transfer to good performance on other tasks.

EVALUATING LANGUAGE MODELS

- How to measure the performance of a language model (LM)?
- The alternative is intrinsic evaluation.
- Where we measure perplexity:
  - Given some text $w_1, \ldots, w_n$, the perplexity of a language model is

customarily base-2

$$\exp\left\{-\frac{1}{n}\sum_{i=1}^{n}\log p(w_i \mid w_1,\ldots,w_{i-1})\right\}$$

  - If the LM assigns high probability to each token, the perplexity will be low.
  - Thus, LMs with lower perplexity are better.
- Disadvantage: Doesn't necessarily correspond to real-world performance.

EVALUATING LANGUAGE MODELS

- How to measure the performance of a language model (LM)?
- The alternative is intrinsic evaluation.
- Where we measure perplexity:
  - Given some text $w_1, \ldots, w_n$, the perplexity of a language model is

$$\exp\left\{-\frac{1}{n}\sum_{i=1}^{n}\log p(w_i \mid w_1,\ldots,w_{i-1})\right\}$$

- The term inside the exponent is related to the cross-entropy!

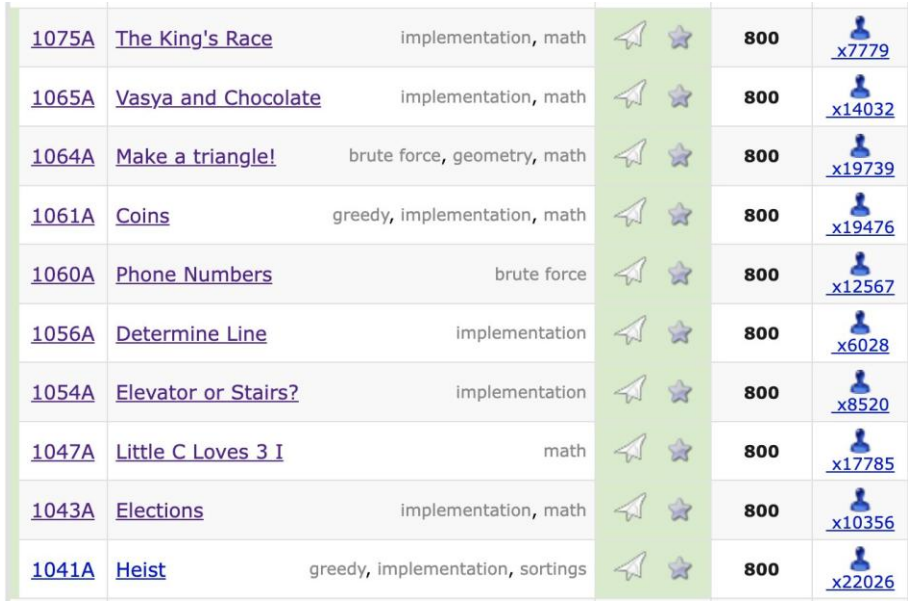
EVALUATING ML MODELS

- When using either intrinsic or extrinsic evaluation, we must be careful to evaluate using unseen/novel test data.
- We must absolutely avoid training the model on test data.
  - This is called *data leakage* or *training data contamination*.
- This will overestimate model performance.
- Bad science!

- As you imagine, this makes evaluating large language models difficult.
- Oftentimes, we don't know the training data.
- Most things on the internet are included in the training data.

DATA LEAKAGE EXAMPLE

- GPT-4 was evaluated on Codeforces programming puzzles.
- Codeforces periodically releases new puzzles.
- GPT-4 scored 10/10 on puzzles released before September 5, 2021:

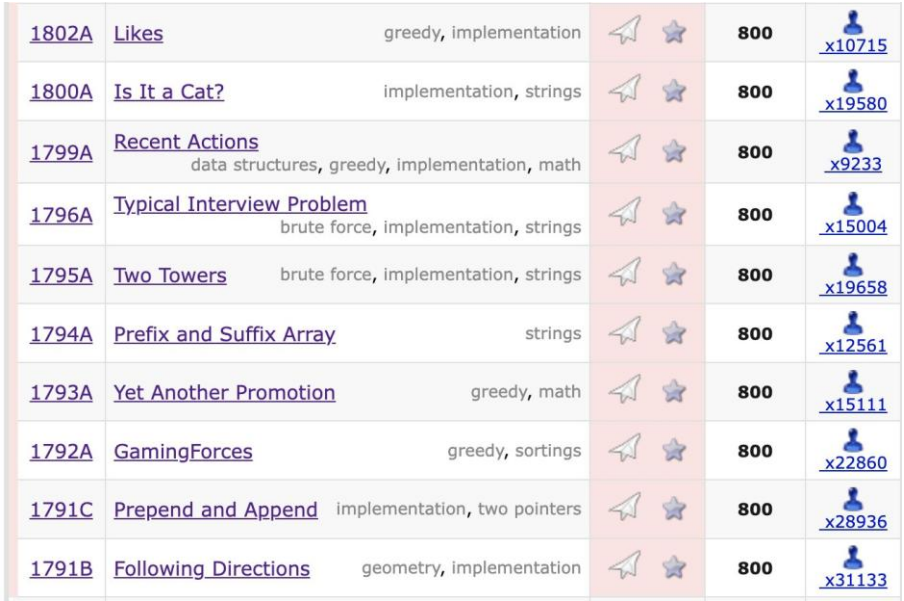


1075A	The King's Race	implementation, math	800	x7779
1065A	Vasya and Chocolate	implementation, math	800	x14032
1064A	Make a triangle!	brute force, geometry, math	800	x19739
1061A	Coins	greedy, implementation, math	800	x19476
1060A	Phone Numbers	brute force	800	x12567
1056A	Determine Line	implementation	800	x6028
1054A	Elevator or Stairs?	implementation	800	x8520
1047A	Little C Loves 3 I	math	800	x17785
1043A	Elections	implementation, math	800	x10356
1041A	Heist	greedy, implementation, sortings	800	x22026

[Horace He, x.com/cHHillee/status/1635790330854526981]

DATA LEAKAGE EXAMPLE

- GPT-4 was evaluated on Codeforces programming puzzles.
- Codeforces periodically releases new puzzles.
- GPT-4 scored 0/10 on puzzles after September 12, 2021:

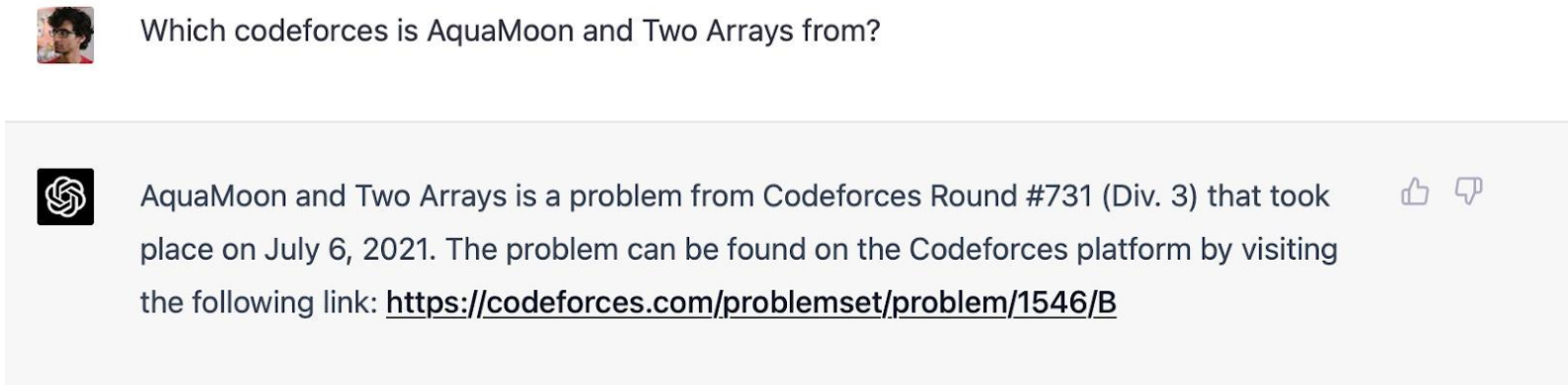


1802A	Likes	greedy, implementation	800	x10715
1800A	Is It a Cat?	implementation, strings	800	x19580
1799A	Recent Actions	data structures, greedy, implementation, math	800	x9233
1796A	Typical Interview Problem	brute force, implementation, strings	800	x15004
1795A	Two Towers	brute force, implementation, strings	800	x19658
1794A	Prefix and Suffix Array	strings	800	x12561
1793A	Yet Another Promotion	greedy, math	800	x15111
1792A	GamingForces	greedy, sortings	800	x22860
1791C	Prepend and Append	implementation, two pointers	800	x28936
1791B	Following Directions	geometry, implementation	800	x31133

[Horace He, x.com/cHHillee/status/1635790330854526981]

DATA LEAKAGE EXAMPLE

- GPT-4 was evaluated on Codeforces programming puzzles.
- Codeforces periodically releases new puzzles.
- Ask GPT-4 directly whether it has memorized a puzzle:



[Narayanan and Kapoor, GPT-4 and professional benchmarks: the wrong answer to the wrong question]

EVALUATING (LARGE) LANGUAGE MODELS

- How to avoid data leakage in LLM evaluation?
- Evaluate with new data.
  - Could be expensive to annotate new data.
- Evaluate on synthetic data.
  - Synthetic data may not accurately reflect real-world settings.
- Keep a private test set.
  - Access to this test set is restricted.
  - Others may submit their model for evaluation.
  - Infeasible to submit a very large model (billions/trillions of parameters).

BACK TO EVALUATING 4-GRAM MODELS

- Consider these samples from the 4-gram model trained on Shakespeare:
  - "King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;"
  - "It cannot be but so."
- These are lines from actual Shakespeare plays.

- How would we evaluate this model?
  - Any ideas?
  - Suggestion: Hold out some plays or lines as test set.
  - What would the perplexity be?

$$\text{perplexity} = \exp\left\{-\frac{1}{n}\sum_{i=1}^{n}\log p(w_i \mid w_1,\ldots,w_{i-1})\right\}$$

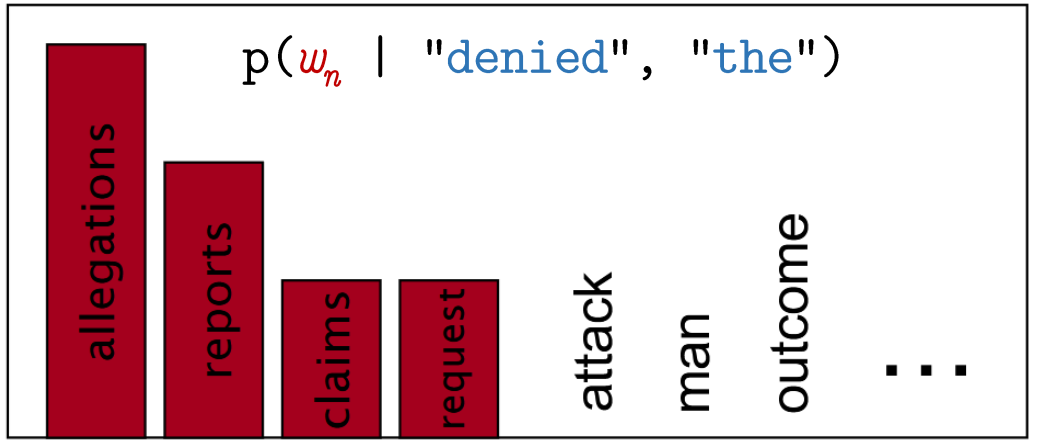
DATA SPARSITY AND OVERFITTING

- Suppose the vocabulary size is V.
- For the 4-gram model, the total number of possible 4-grams is V^4.
- If the training set has N words, there are *at most* N examples of 4-grams.
- So the fraction of 4-grams that are unobserved is *at least* $1 - N/V^4$.
- For the Shakespeare dataset, that is $1 - 10^{-12}$!
- This is a *data sparsity problem*.

- The 4-gram model is prone to *overfitting*.
- It assigns 0 probability to any 4-gram that is not in its training data.

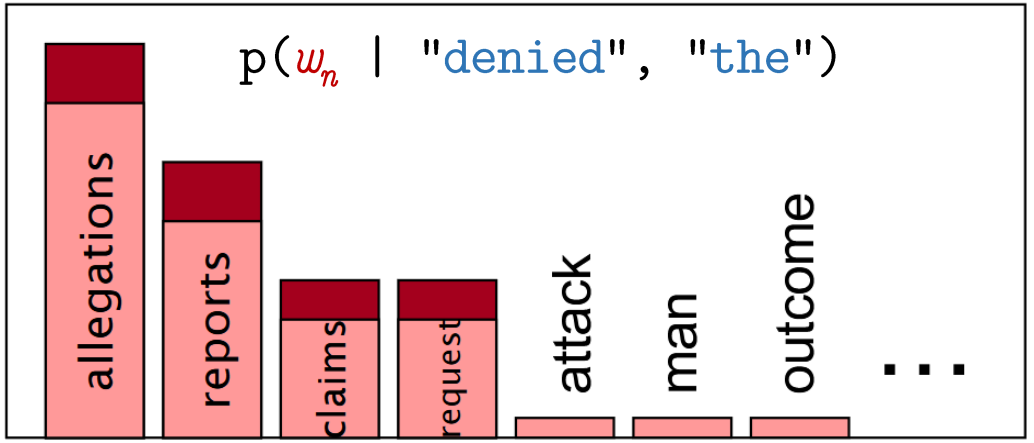
DATA SPARSITY AND OVERFITTING

- How do we resolve the data sparsity issue with n-gram models?
- One idea is called *smoothing*:
- The intuition is to "smooth" out the distribution of the next word, so that no word has probability 0.
- E.g., we have a 3-gram model where we have seen the following phrases in the training data:
  - "denied the allegations" 3 times
  - "denied the reports" 2 times
  - "denied the claims" 1 time
  - "denied the request" 1 time
  - No other instances of "denied the ___"



DATA SPARSITY AND OVERFITTING

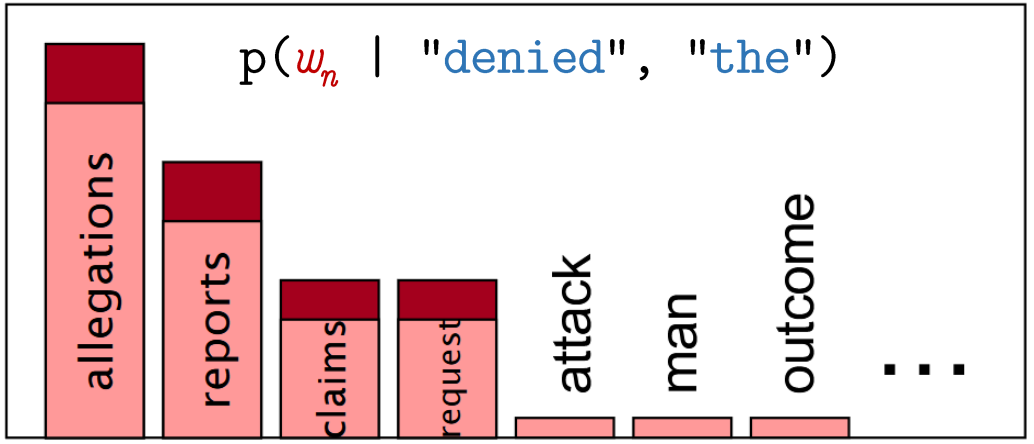
- How do we resolve the data sparsity issue with n-gram models?
- One idea is called *smoothing*:
- The intuition is to "smooth" out the distribution of the next word, so that no word has probability 0.
- E.g., we have a 3-gram model where we have seen the following phrases in the training data:
  - "denied the allegations" 3 times
  - "denied the reports" 2 times
  - "denied the claims" 1 time
  - "denied the request" 1 time
  - No other instances of "denied the ___"



SMOOTHING

- How do we resolve the data sparsity issue with n-gram models?
- Laplace smoothing (also called add-one smoothing):

$$p(w_n \mid w_1, \ldots, w_{n-1}) = \frac{(\text{\# of times } w_n \text{ appeared after } w_1, \ldots, w_{n-1}) + 1}{(\text{\# of times } w_1, \ldots, w_{n-1} \text{ appeared}) + V}$$



SMOOTHING

- How do we resolve the data sparsity issue with n-gram models?
- Laplace smoothing (also called add-one smoothing):

$$p(w_n \mid w_1, \ldots, w_{n-1}) = \frac{(\text{\# of times } w_n \text{ appeared after } w_1, \ldots, w_{n-1}) + 1}{(\text{\# of times } w_1, \ldots, w_{n-1} \text{ appeared}) + V}$$

- This is simple, but doesn't work well in language modeling.
  - Consider the 4-gram model trained on Shakespeare.
  - For almost all 4-grams in the test set, the numerator in the above expression is 1.
- It is useful in other tasks, however.

BACKOFF

- How do we resolve the data sparsity issue with n-gram models?
- Another idea: Simultaneously use multiple n-gram models, with smaller *n*.

$$p(w_n \mid w_1, \ldots, w_{n-1}) = \frac{\#\text{ of times } w_n \text{ appeared after } w_1, \ldots, w_{n-1}}{\#\text{ of times } w_1, \ldots, w_{n-1} \text{ appeared}} \quad \text{if } w_1, \ldots, w_n \text{ occurs in data}$$

$$= \frac{\#\text{ of times } w_n \text{ appeared after } w_2, \ldots, w_{n-1}}{\#\text{ of times } w_2, \ldots, w_{n-1} \text{ appeared}} \quad \text{if } w_2, \ldots, w_n \text{ occurs in data}$$

$$\ldots$$

$$= \frac{\#\text{ of times } w_n \text{ appeared after } w_{n-1}}{\#\text{ of times } w_{n-1} \text{ appeared}} \quad \text{if } w_{n-1}, w_n \text{ occurs in data}$$

$$= \frac{\#\text{ of times } w_n \text{ appears}}{\text{total number of words}} \quad \text{otherwise.}$$

INTERPOLATION

- How do we resolve the data sparsity issue with n-gram models?
- Another idea: Use multiple n-gram models, with interpolation.

$$p(w_n \mid w_1, \ldots, w_{n-1}) = \lambda_n \frac{\#\text{ of times } w_n \text{ appeared after } w_1, \ldots, w_{n-1}}{\#\text{ of times } w_1, \ldots, w_{n-1} \text{ appeared}}$$

$$+ \lambda_{n-1} \frac{\#\text{ of times } w_n \text{ appeared after } w_2, \ldots, w_{n-1}}{\#\text{ of times } w_2, \ldots, w_{n-1} \text{ appeared}}$$

$$\ldots$$

$$+ \lambda_2 \frac{\#\text{ of times } w_n \text{ appeared after } w_{n-1}}{\#\text{ of times } w_{n-1} \text{ appeared}}$$

$$+ \lambda_1 \frac{\#\text{ of times } w_n \text{ appears}}{\text{total number of words}}$$

Require:
$\lambda_1 + \ldots + \lambda_n = 1$

INTERPOLATION

- This type of model is called a mixture model.
- Equivalent to first rolling an n-sided die to choose which n-gram to sample from, and then sampling from the corresponding n-gram model.

$$p(w_n \mid w_1, \ldots, w_{n-1}) = p(w_n \mid w_1, \ldots, w_{n-1}, \text{choose 1-gram})\ p(\text{choose 1-gram}) + \ldots + p(w_n \mid w_1, \ldots, w_{n-1}, \text{choose n-gram})\ p(\text{choose n-gram}),$$

- By the law of total probability.

$$= p(w_n \mid w_1, \ldots, w_{n-1}, \text{choose 1-gram})\ \lambda_1 + \ldots + p(w_n \mid w_1, \ldots, w_{n-1}, \text{choose n-gram})\ \lambda_n.$$

DATA SPARSITY AND OVERFITTING

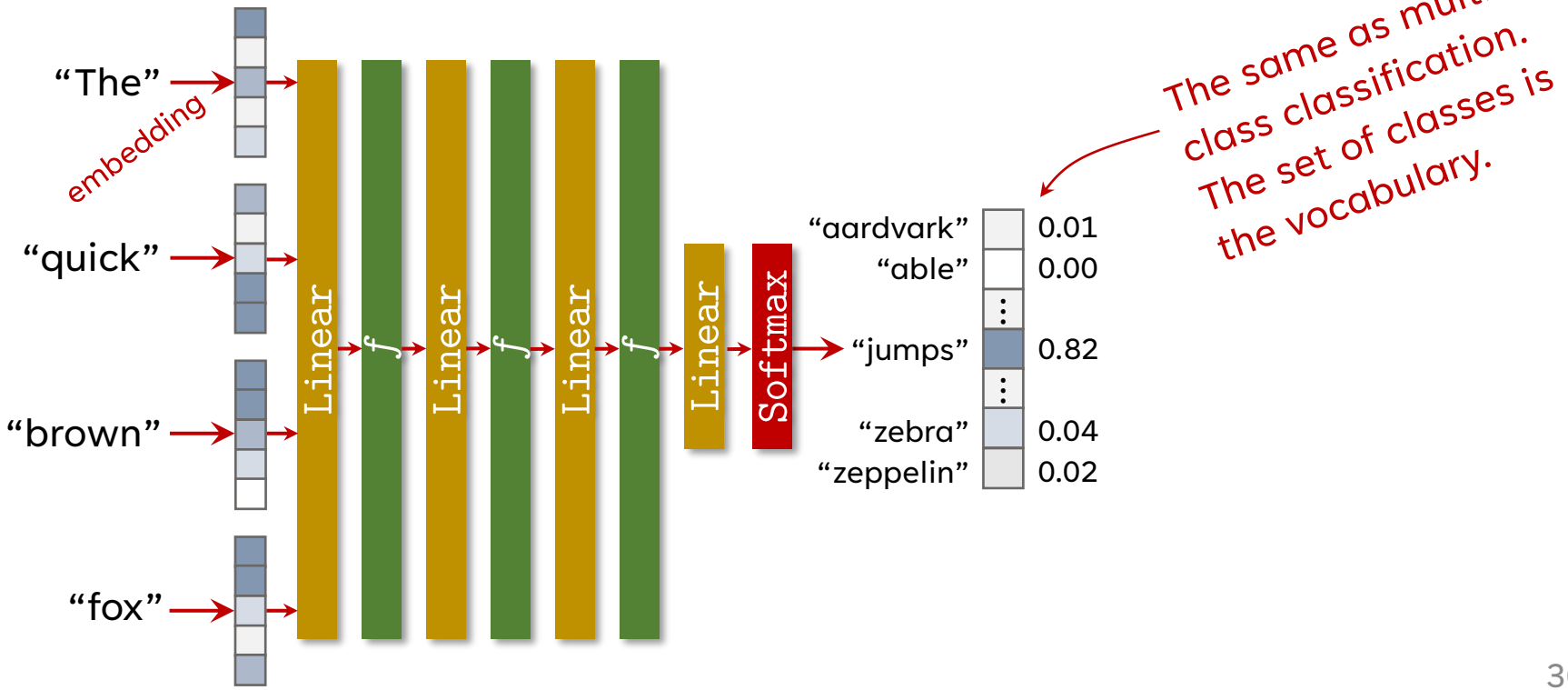
(some NLP history)

- Backoff performs better when combined with smoothing.
  - Kneser-Ney smoothing
  - Interpolated Kneser-Ney
  - Skip n-grams

- Another idea to address the data sparsity issue, is to use a different machine learning model.
  - Perhaps a neural network?
- Smoothing/interpolation superseded by neural language models.

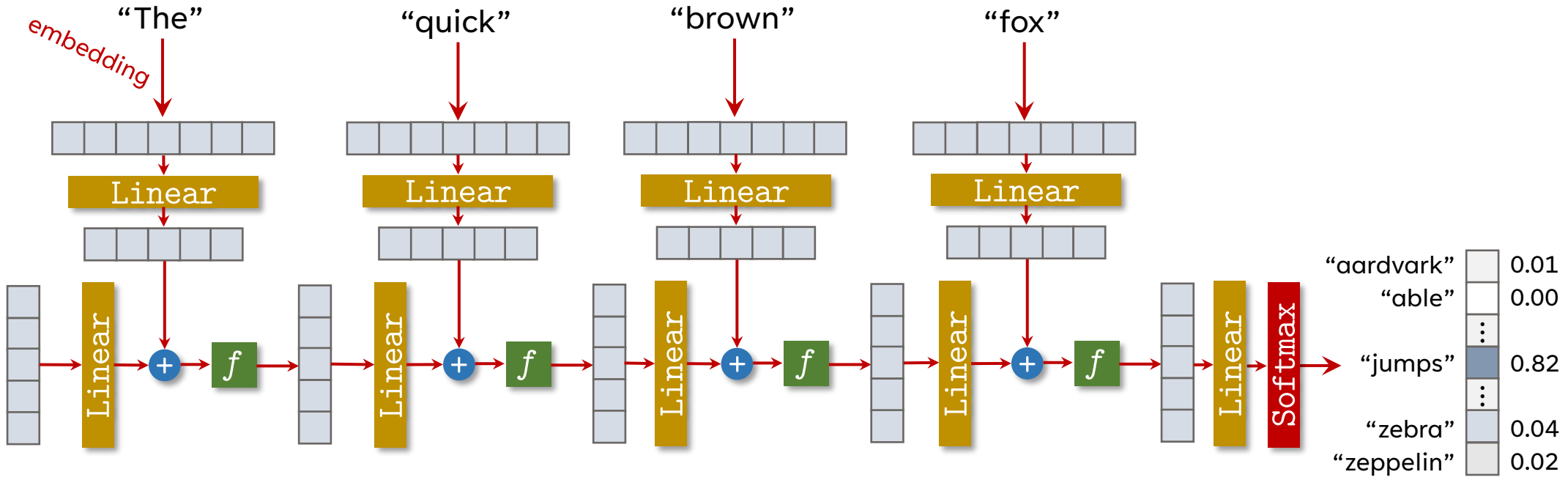
FEEDFORWARD LANGUAGE MODELS

- MLPs can be applied to perform language modeling.
- Predict the next word from the last layer's activations.



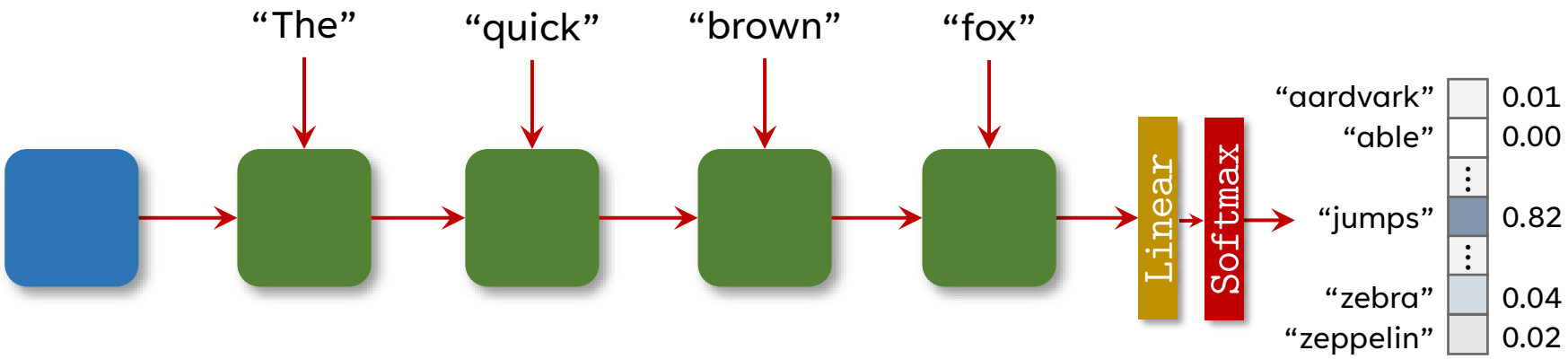
RNN LANGUAGE MODELS

- RNNs can be applied to perform language modeling.
- Each word/token of the input provides an update to the hidden state.
- Predict the next word from the last hidden state vector.



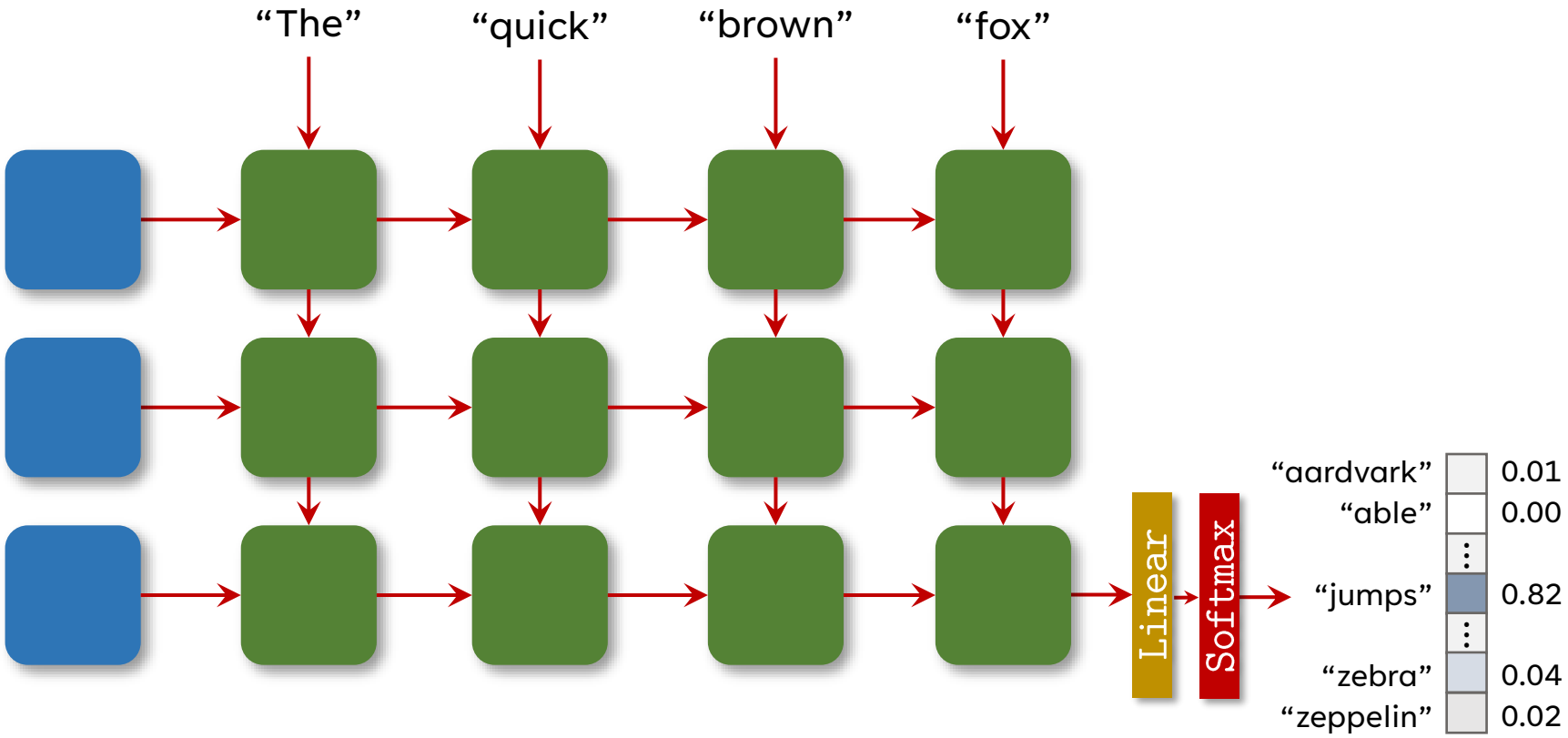
RNN LANGUAGE MODELS

- RNNs can be applied to perform language modeling.
- Each word/token of the input provides an update to the hidden state.
- Predict the next word from the last hidden state vector.



RNN LANGUAGE MODELS

- We can use different RNN architectures for language modeling.



TRAINING NEURAL LANGUAGE MODELS

- How do we train neural LMs?
- Unsurprisingly, we use gradient descent.
- What loss function should we use?
  - The output is a distribution over the next word.
  - Cross-entropy loss is a natural choice.
- Training data is readily available.
  - We can take any piece of text (such as from the internet)
  - And convert it into a training example.

NEURAL LANGUAGE MODEL CAVEATS

- Language models based on MLPs and RNNs run into the same problem as MLPs and RNNs in multi-class classification:
  - MLPs have a very large number of parameters and need a lot of data to train.
    - Same underlying problem as n-gram models with large n.
  - RNNs have difficulty with modeling long-range dependencies.
    - Each hidden state only depends on the previous hidden state
    - And not on any earlier hidden states.
    - This makes RNNs difficult to parallelize.
  - Both run into problems with vanishing/exploding gradients.
- Next time, we will discuss transformer language models.
  - Transformers are the current go-to architecture for large language models (LLMs).

QUESTIONS?