Classical Language Models

Antoine Bosselut

EPFL

Announcements

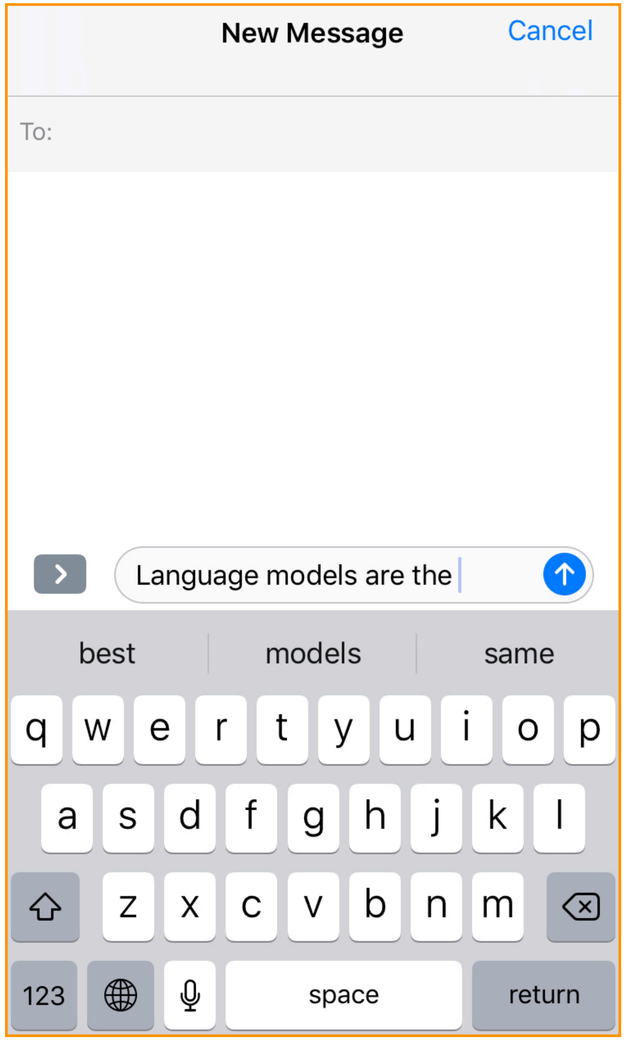
- **Lectures are being recorded and links will be posted to the website**
- **We'll try to get them online in the same week going forward**

Today's Outline

- **Part 1:** Language models introduction, count-based language models, evaluating language models, smoothing
- **Recap:** Neural networks fundamentals
- **Part 2:** Fixed-context Language Models

Some slides adapted from Danqi Chen, Mohit Iyyer, Dan Jurafsky

Language models are ubiquitous



Applications of LMs

- Predicting words is important in many situations
  - **Classical:** Machine translation, text generation

$$P(\text{a } \mathbf{smooth} \text{ finish}) > P(\text{a } \mathbf{flat} \text{ finish})$$

  - **Classical:** Speech recognition/Spell checking

$$P(\text{high school } \mathbf{principal}) > P(\text{high school } \mathbf{principle})$$

  - **Modern:** Information extraction, Question answering, Classification, etc.

Impact on downstream applications

New Approach to Language Modeling Reduces Speech Recognition Errors by Up to 15%

December 13, 2018
Ankur Gandhe

Alexa | Alexa research | Alexa science

What is a language model?

- A language model is a **probabilistic model of a sequence of tokens**
  - How likely is a given phrase/sentence/paragraph/document?

- A sequence is modelled as a joint distribution of tokens $w_1, w_2, w_3, \ldots, w_n$:

$$P(w_1, w_2, w_3, ..., w_n)$$

- Language models estimate this probability!

Chain rule

- How should we compute this joint probability over words in a sequence?

$$P(w_1, w_2, w_3, \ldots, w_N) = P(w_1) \times P(w_2 \mid w_1) \times P(w_3 \mid w_1, w_2) \times \ldots \times P(w_N \mid w_1, w_2, \ldots, w_{N-1})$$

$$P(w_1, w_2, w_3, \ldots, w_N) = \prod_i P(w_i \mid w_1, w_2, \ldots, w_{i-1})$$

Chain rule

- How should we compute this joint probability over words in a sequence?

$$P(w_1, w_2, w_3, \ldots, w_N) = P(w_1) \times P(w_2 \mid w_1) \times P(w_3 \mid w_1, w_2) \times \ldots \times P(w_N \mid w_1, w_2, \ldots, w_{N-1})$$

$$P(w_1, w_2, w_3, \ldots, w_N) = \prod_i P(w_i \mid w_1, w_2, \ldots, w_{i-1})$$

“prefix”

Chain rule

- How should we compute this joint probability over words in a sequence?

$$P(w_1, w_2, w_3, \ldots, w_N) = P(w_1) \times P(w_2 \mid w_1) \times P(w_3 \mid w_1, w_2) \times \ldots \times P(w_N \mid w_1, w_2, \ldots, w_{N-1})$$

$$P(w_1, w_2, w_3, \ldots, w_N) = \prod_i P(w_i \mid w_1, w_2, \ldots, w_{i-1})$$

Sentence: “the cat sat on the mat”

$$P(\text{the cat sat on the mat}) = P(\text{the}) * P(\text{cat}|\text{the}) * P(\text{sat}|\text{the cat}) * P(\text{on}|\text{the cat sat}) * P(\text{the}|\text{the cat sat on}) * P(\text{mat}|\text{the cat sat on the})$$

Chain rule

- How should we compute this joint probability over words in a sequence?

$$P(w_1, w_2, w_3, \ldots, w_N) = P(w_1) \times P(w_2 \mid w_1) \times P(w_3 \mid w_1, w_2) \times \ldots \times P(w_N \mid w_1, w_2, \ldots, w_{N-1})$$

$$P(w_1, w_2, w_3, \ldots, w_N) = \prod_i P(w_i \mid w_1, w_2, \ldots, w_{i-1})$$

→ **“prefix”**

Sentence: “the cat sat on the mat”

$$P(\text{the cat sat on the mat}) = P(\text{the}) * P(\text{cat}|\text{the}) * P(\text{sat}|\text{the cat}) * P(\text{on}|\text{the cat sat}) * P(\text{the}|\text{the cat sat on}) * P(\text{mat}|\text{the cat sat on the})$$

Count-based Modeling

Maximum likelihood estimate (MLE)

$$P(\text{sat}|\text{the cat}) = \frac{\text{count}(\text{the cat sat})}{\text{count}(\text{the cat})}$$

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

$$\vdots$$

- Estimate probabilities by counting occurrences of sequences in a **corpus** of text

Note on Notation:
Word type: unique word in our vocabulary *V*
Token: instance of a word type in our corpus

Question

What's going to be a problem with counting sequences for any prefix?

Count-based Modeling

Maximum likelihood estimate (MLE)

$$P(\text{sat}|\text{the cat}) = \frac{\text{count}(\text{the cat sat})}{\text{count}(\text{the cat})}$$

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

$$\vdots$$

Note on Notation:
Word type: unique word in our vocabulary *V*
Token: instance of a word type in our corpus

- Estimate probabilities by counting occurrences of sequences in a **corpus** of text
- With a vocabulary of size *V*,
  - number of sequences of length n = V^n
- Typical vocabulary ≈ 40000 words
  - even sentences of length <= 11 results in more than $4 * 10^{50}$ sequences!
    (more than the number of atoms in the earth)

Markov assumption

- Use only the recent past to predict the next word
- **Effect:** reduce the number of estimated parameters in exchange for modeling capacity
- 1st order Markov

$$P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{the})$$

- 2nd order Markov

$$P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{on the})$$

k^{th} order Markov

- Consider only the last *k* words for context

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

which implies the probability of a sequence is:

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

(ignore $w_j \quad \forall j < 0$)

n-gram models

Unigram $$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i)$$

Bigram $$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1})$$

n-gram models

Unigram $$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i)$$

Bigram $$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1})$$

and Trigram, 4-gram, and so on.

n-gram models

Unigram $$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i)$$

Bigram $$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1})$$

and Trigram, 4-gram, and so on.

Larger n leads to more accurate and better language models (but also higher costs)*

n-gram models

Unigram $$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i)$$

Bigram $$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1})$$

and Trigram, 4-gram, and so on.

Larger n leads to more accurate and better language models (but also higher costs)*

**Caveat: Assuming infinite data!*

Examples

$$\arg \max_{(w_1,w_2,\ldots,w_n)} P(w_1, w_2, \ldots, w_n) = \arg \max_{(w_1,w_2,\ldots,w_n)} \prod_{i=1}^{n} P(w_i \mid w_{i-k}, \ldots, w_{i-1})$$

Examples

Unigram

release millions See ABC accurate President of Donald Will cheat them a CNN megynkelly experience @ these word out- the

$$\arg \max_{(w_1,w_2,\ldots,w_n)} P(w_1, w_2, \ldots, w_n) = \arg \max_{(w_1,w_2,\ldots,w_n)} \prod_{i=1}^{n} P(w_i \mid w_{i-k}, \ldots, w_{i-1})$$

Examples

Unigram

release millions See ABC accurate President of Donald Will cheat them a CNN megynkelly experience @ these word out- the

Bigram

Thank you believe that @ ABC news, Mississippi tonight and the false editorial I think the great people Bill Clinton . ''

$$\arg \max_{(w_1,w_2,\ldots,w_n)} P(w_1, w_2, \ldots, w_n) = \arg \max_{(w_1,w_2,\ldots,w_n)} \prod_{i=1}^{n} P(w_i \mid w_{i-k}, \ldots, w_{i-1})$$

Examples

Unigram	*release millions See ABC accurate President of Donald Will cheat them a CNN megynkelly experience @ these word out- the*
Bigram	*Thank you believe that @ ABC news, Mississippi tonight and the false editorial I think the great people Bill Clinton . ''*
Trigram	*We are going to MAKE AMERICA GREAT AGAIN! #MakeAmericaGreatAgain https: //t.co/DjkdAzT3WV*

$$\arg \max_{(w_1,w_2,\ldots,w_n)} P(w_1, w_2, \ldots, w_n) = \arg \max_{(w_1,w_2,\ldots,w_n)} \prod_{i=1}^{n} P(w_i \mid w_{i-k}, \ldots, w_{i-1})$$

Examples

Unigram	*release millions See ABC accurate President of Donald Will cheat them a CNN megynkelly experience @ these word out- the*
Bigram	*Thank you believe that @ ABC news, Mississippi tonight and the false editorial I think the great people Bill Clinton . ''*
Trigram	*We are going to MAKE AMERICA GREAT AGAIN! #MakeAmericaGreatAgain https: //t.co/DjkdAzT3WV*

Typical LMs are not sufficient to handle long-range dependencies

"Alice/Bob could not go to work that day because she/he had a doctor's appointment"

Evaluating language models

Evaluating language models

- A good language model should assign **higher probability** to typical, grammatically correct sentences

Evaluating language models

- A good language model should assign **higher probability** to typical, grammatically correct sentences
- Research process:

Evaluating language models

- A good language model should assign **higher probability** to typical, grammatically correct sentences
- Research process:
  - Train parameters on a suitable training corpus

Evaluating language models

- A good language model should assign **higher probability** to typical, grammatically correct sentences
- Research process:
  - Train parameters on a suitable training corpus
    - *Assumption*: observed sentences are probably good sentences

Evaluating language models

- A good language model should assign **higher probability** to typical, grammatically correct sentences
- Research process:
  - Train parameters on a suitable training corpus
    - *Assumption*: observed sentences are probably good sentences
  - Test on *different, unseen* corpus

Evaluating language models

- A good language model should assign **higher probability** to typical, grammatically correct sentences
- Research process:
  - Train parameters on a suitable training corpus
    - *Assumption*: observed sentences are probably good sentences
  - Test on *different, unseen* corpus
    - **Training on any part of test set not acceptable! Why?**

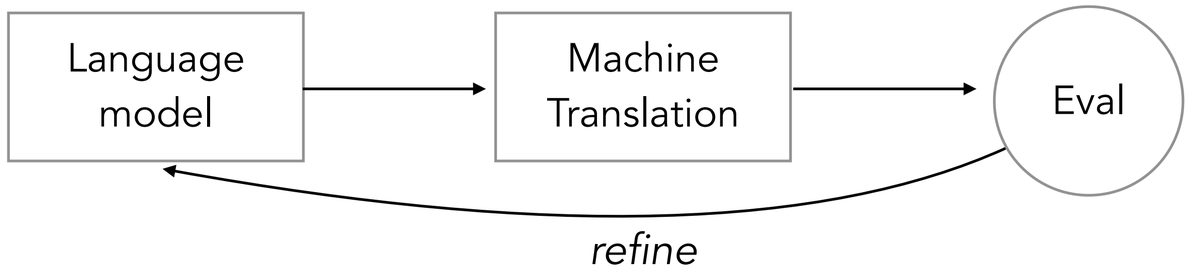
Evaluating language models

- A good language model should assign **higher probability** to typical, grammatically correct sentences
- Research process:
  - Train parameters on a suitable training corpus
    - *Assumption*: observed sentences are probably good sentences
  - Test on *different, unseen* corpus
    - **Training on any part of test set not acceptable! Why?**
  - Evaluation metric

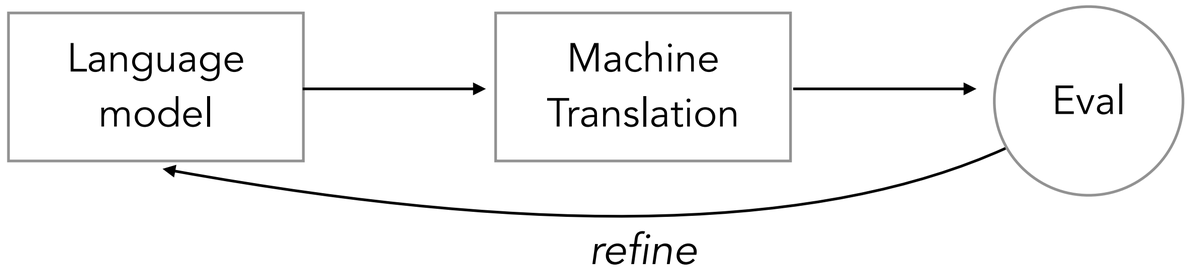
Question

What evaluation metric should we use?

Extrinsic evaluation

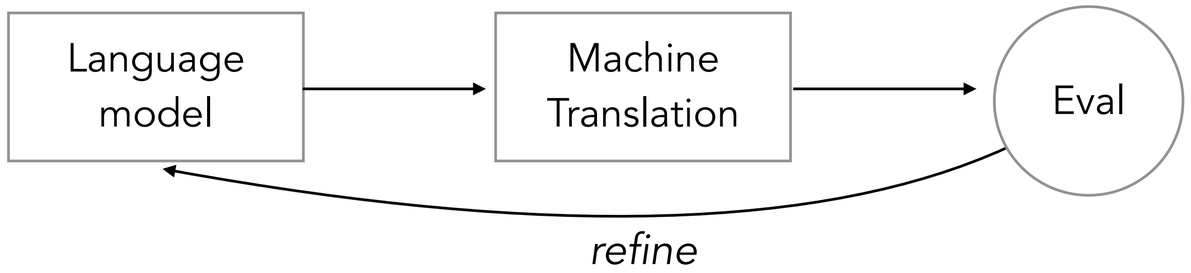


Extrinsic evaluation



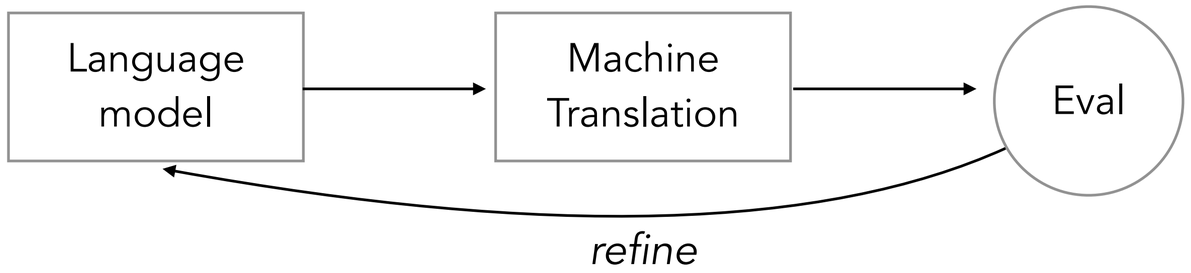
- Train LM —> apply to task —> observe accuracy

Extrinsic evaluation



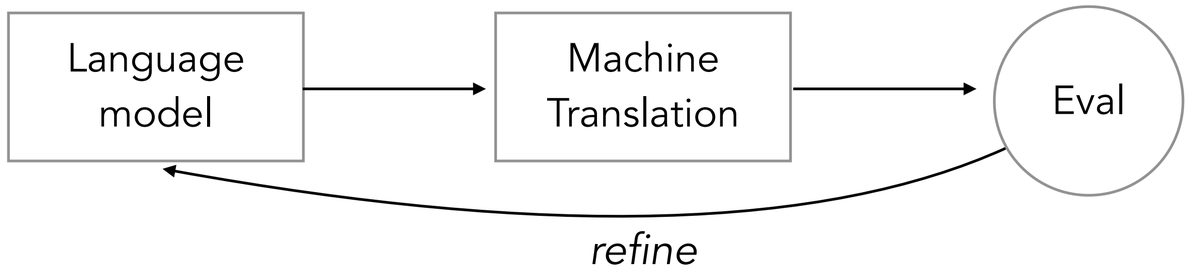
- Train LM —> apply to task —> observe accuracy
- Directly optimized for downstream tasks
  - higher task accuracy —> better model

Extrinsic evaluation



- Train LM —> apply to task —> observe accuracy
- Directly optimized for downstream tasks
  - higher task accuracy —> better model
- Expensive, time consuming

Extrinsic evaluation



- Train LM —> apply to task —> observe accuracy
- Directly optimized for downstream tasks
  - higher task accuracy —> better model
- Expensive, time consuming
- Hard to optimize downstream objective (indirect feedback)

Perplexity (ppl)

Perplexity (ppl)

- Measure of how well a probability distribution (or LM) **predicts** a sample

Perplexity (ppl)

- Measure of how well a probability distribution (or LM) **predicts** a sample
- For a corpus S with sentences $S^1, S^2, \ldots, S^n$

Perplexity (ppl)

- Measure of how well a probability distribution (or LM) **predicts** a sample
- For a corpus S with sentences $S^1, S^2, \ldots, S^n$

$$\text{ppl}(S) = 2^x \quad \text{where} \quad x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

→ Cross-Entropy

Perplexity (ppl)

- Measure of how well a probability distribution (or LM) **predicts** a sample
- For a corpus S with sentences $S^1, S^2, \ldots, S^n$

$$\text{ppl}(S) = 2^x \quad \text{where} \quad x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

→ Cross-Entropy

Perplexity (ppl)

- Measure of how well a probability distribution (or LM) **predicts** a sample
- For a corpus S with sentences $S^1, S^2, \ldots, S^n$

$$\text{ppl}(S) = 2^x \quad \text{where} \quad x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

Cross-Entropy

where W is the total number of words in test corpus

Perplexity (ppl)

- Measure of how well a probability distribution (or LM) **predicts** a sample
- For a corpus S with sentences $S^1, S^2, \ldots, S^n$

$$\text{ppl}(S) = 2^x \quad \text{where} \quad x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

Cross-Entropy

where W is the total number of words in test corpus

- Unigram model:

Perplexity (ppl)

- Measure of how well a probability distribution (or LM) **predicts** a sample
- For a corpus S with sentences $S^1, S^2, \ldots, S^n$

$$\text{ppl}(S) = 2^x \quad \text{where} \quad x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

Cross-Entropy

where W is the total number of words in test corpus

- Unigram model: $$x = -\frac{1}{W}\sum_{i=1}^{n}\sum_{j=1}^{m} log_2 P(w_j^i)$$

Perplexity (ppl)

- Measure of how well a probability distribution (or LM) **predicts** a sample
- For a corpus S with sentences $S^1, S^2, \ldots, S^n$

$$\text{ppl}(S) = 2^x \quad \text{where} \quad x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$ → Cross-Entropy

where W is the total number of words in test corpus

- Unigram model: $$x = -\frac{1}{W}\sum_{i=1}^{n}\sum_{j=1}^{m} log_2 P(w_j^i)$$ (since $P(S) = \prod_j P(w_j)$)

Perplexity (ppl)

- Measure of how well a probability distribution (or LM) **predicts** a sample
- For a corpus S with sentences $S^1, S^2, \ldots, S^n$

$$\text{ppl}(S) = 2^x \quad \text{where} \quad x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$ → Cross-Entropy

  where W is the total number of words in test corpus

- Unigram model: $$x = -\frac{1}{W}\sum_{i=1}^{n}\sum_{j=1}^{m} log_2 P(w_j^i)$$ (since $P(S) = \prod_j P(w_j)$)
- Minimizing perplexity is the same as maximizing probability of corpus $P(S^1 S^2 \ldots S^n)$

Intuition on perplexity

$$\text{ppl}(S) = 2^x \quad \text{where}$$
$$x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

Intuition on perplexity

- If our n-gram model (with vocabulary V) has following probability:

$$\text{ppl}(S) = 2^x \quad \text{where}$$
$$x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

Intuition on perplexity

- If our n-gram model (with vocabulary V) has following probability:

$$P(w_i|w_{i-n},...w_{i-1}) = \frac{1}{|V|} \quad \forall w_i$$

$$\text{ppl}(S) = 2^x \quad \text{where}$$
$$x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

Intuition on perplexity

- If our n-gram model (with vocabulary V) has following probability:

$$P(w_i|w_{i-n},...w_{i-1}) = \frac{1}{|V|} \quad \forall w_i$$

$$\text{ppl}(S) = 2^x \quad \text{where}$$
$$x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

Intuition on perplexity

- If our n-gram model (with vocabulary V) has following probability:

$$P(w_i|w_{i-n},...w_{i-1}) = \frac{1}{|V|} \quad \forall w_i$$

- What is the perplexity of the test corpus?

$$\text{ppl}(S) = 2^x \quad \text{where}$$
$$x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

Intuition on perplexity

- If our n-gram model (with vocabulary V) has following probability:

$$P(w_i|w_{i-n},...w_{i-1}) = \frac{1}{|V|} \quad \forall w_i$$

- What is the perplexity of the test corpus?

$$\text{ppl}(S) = 2^x \quad \text{where}$$
$$x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

Intuition on perplexity

- If our n-gram model (with vocabulary V) has following probability:

$$P(w_i|w_{i-n},...w_{i-1}) = \frac{1}{|V|} \quad \forall w_i$$

- What is the perplexity of the test corpus?

$$ppl = 2^{-\frac{1}{W}W \times \log(\frac{1}{|V|})} = |V|$$

$$\text{ppl}(S) = 2^x \quad \text{where}$$
$$x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

Intuition on perplexity

- If our n-gram model (with vocabulary V) has following probability:

$$P(w_i|w_{i-n},...w_{i-1}) = \frac{1}{|V|} \quad \forall w_i$$

- What is the perplexity of the test corpus?

$$ppl = 2^{-\frac{1}{W}W \times \log(\frac{1}{|V|})} = |V|$$

- **Any word is equally likely at next step!**

$$\text{ppl}(S) = 2^x \quad \text{where}$$
$$x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

Intuition on perplexity

- If our n-gram model (with vocabulary V) has following probability:

$$P(w_i|w_{i-n},...w_{i-1}) = \frac{1}{|V|} \quad \forall w_i$$

- What is the perplexity of the test corpus?

$$ppl = 2^{-\frac{1}{W}W \times \log(\frac{1}{|V|})} = |V|$$

$$\text{ppl}(S) = 2^x \quad \text{where}$$
$$x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

- **Any word is equally likely at next step!**

- ***Intuition**: Perplexity measures a model's uncertainty about the next word*

Intuition on perplexity

- Perplexity is the exponentiated token-level negative log likelihood
  - **Why logs?**

- **Theory:** perplexity measured using a base of 2
  - **Practice**: doesn't matter what the base is
  - **Exercise**: derive why!

$$\text{ppl}(S) = 2^x \quad \text{where}$$
$$x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

Intuition on perplexity

- Perplexity is the exponentiated token-level negative log likelihood
  - **Why logs?**

- **Theory:** perplexity measured using a base of 2
  - **Practice**: doesn't matter what the base is
  - **Exercise**: derive why!

$$\text{ppl}(S) = 2^x \quad \text{where}$$
$$x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

Question: How good is a language model with ppl > |V| ?

Perplexity as a metric

Pros	Cons
Easy to compute	Requires domain match between train and test
standardized	might not correspond to end task optimization
directly useful, easy to use to correct sentences	log 0 undefined
nice theoretical interpretation - matching distributions	can be ‘cheated’ by predicting common tokens
	size of test set matters
	can be sensitive to low prob tokens/sentences

Question

What are major shortcomings of n-gram language models?

Exponential Decay of Counts

- What happens when we scale to longer contexts?

Exponential Decay of Counts

- What happens when we scale to longer contexts?

$P(w|\text{to})$ *to* occurs 1M times in corpus

Exponential Decay of Counts

- What happens when we scale to longer contexts?

$P(w|\text{to})$ *to* occurs 1M times in corpus

$P(w|\text{go to})$ *go to* occurs 50,000 times in corpus

Exponential Decay of Counts

- What happens when we scale to longer contexts?

$P(w|\text{to})$ *to* occurs 1M times in corpus

$P(w|\text{go to})$ *go to* occurs 50,000 times in corpus

$P(w|\text{to go to})$ *go to* occurs 1500 times in corpus

Exponential Decay of Counts

- What happens when we scale to longer contexts?

$P(w|\text{to})$ *to* occurs 1M times in corpus

$P(w|\text{go to})$ *go to* occurs 50,000 times in corpus

$P(w|\text{to go to})$ *go to* occurs 1500 times in corpus

$P(w|\text{want to go to})$ *want to go to*: only 100 occurrences

Exponential Decay of Counts

- What happens when we scale to longer contexts?

$P(w\|\text{to})$	*to* occurs 1M times in corpus
$P(w\|\text{go to})$	*go to* occurs 50,000 times in corpus
$P(w\|\text{to go to})$	*go to* occurs 1500 times in corpus
$P(w\|\text{want to go to})$	*want to go to*: only 100 occurrences

- Probability counts get very sparse, and we often want information from 5+ words away

Not all n-grams observed in training data

Not all n-grams observed in training data

- Test corpus might have some that have zero probability under our model

Not all n-grams observed in training data

- Test corpus might have some that have zero probability under our model
  - Training set: *Google news*

Not all n-grams observed in training data

- Test corpus might have some that have zero probability under our model
  - Training set: *Google news*
  - Test set: *Shakespeare*

Not all n-grams observed in training data

- Test corpus might have some that have zero probability under our model
  - Training set: *Google news*
  - Test set: *Shakespeare*
  - P (affray | voice doth us) = 0 → P(test corpus) = 0

Not all n-grams observed in training data

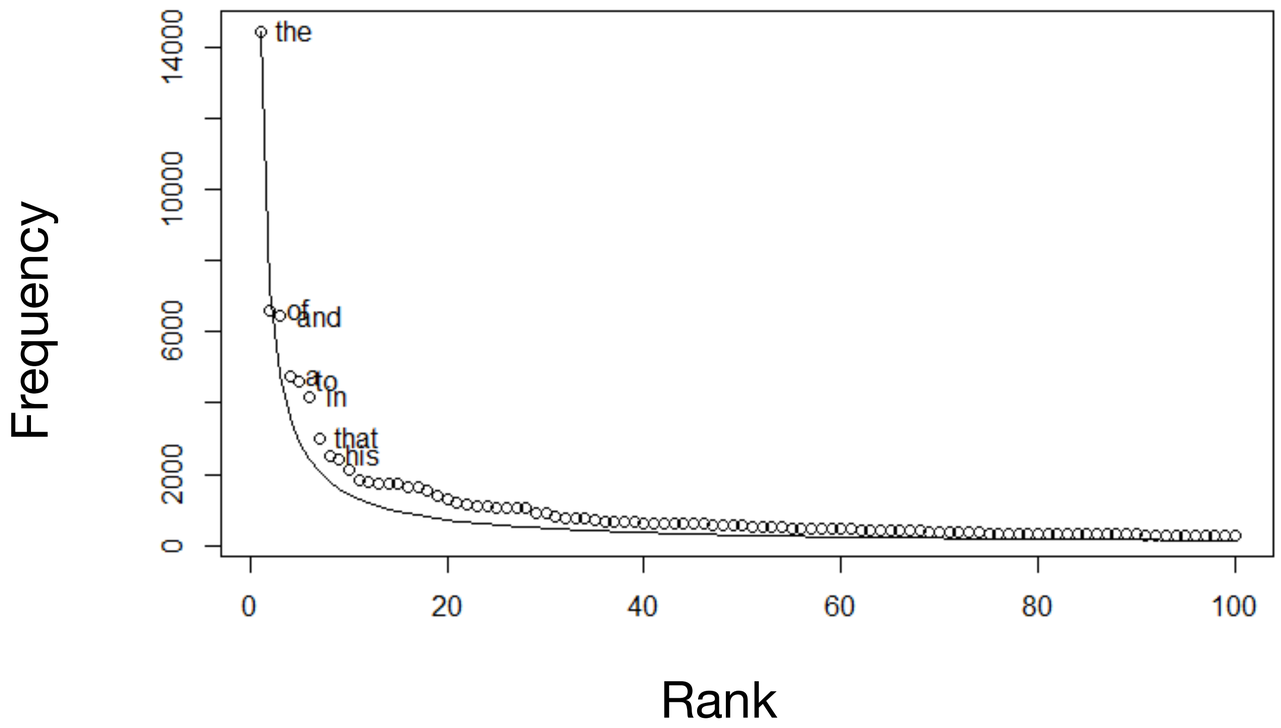
- Test corpus might have some that have zero probability under our model
  - Training set: *Google news*
  - Test set: *Shakespeare*
  - P (affray | voice doth us) = 0 → P(test corpus) = 0
  - **Undefined perplexity**

Fun Fact!

Shakespeare as corpus

- N=884,647 tokens, V=29,066
- Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams.
  - So 99.96% of the possible bigrams were never seen (have zero entries in the table)

Sparsity in language



$$freq \propto \frac{1}{rank}$$

Zipf's Law

- Long tail of infrequent words
- Most finite-size corpora will have this problem.

Question

How might we fix this ?

Smoothing

Smoothing

- Handle sparsity by making sure all probabilities are non-zero in our model

Smoothing

- Handle sparsity by making sure all probabilities are non-zero in our model
  - Additive: Add a small amount to all probabilities

Smoothing

- Handle sparsity by making sure all probabilities are non-zero in our model
  - Additive: Add a small amount to all probabilities
  - Discounting: Redistribute probability mass from observed n-grams to unobserved ones

Smoothing

- Handle sparsity by making sure all probabilities are non-zero in our model
  - Additive: Add a small amount to all probabilities
  - Discounting: Redistribute probability mass from observed n-grams to unobserved ones
  - Back-off: Use lower order n-grams if higher ones are too sparse

Smoothing

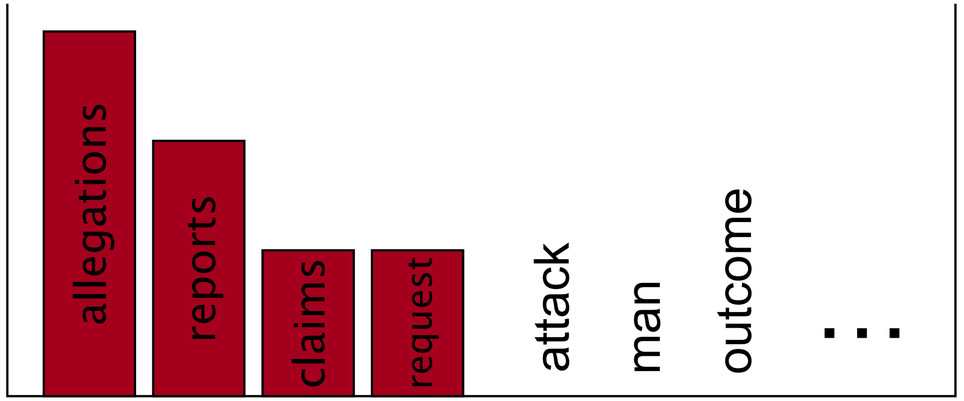
- Handle sparsity by making sure all probabilities are non-zero in our model
  - Additive: Add a small amount to all probabilities
  - Discounting: Redistribute probability mass from observed n-grams to unobserved ones
  - Back-off: Use lower order n-grams if higher ones are too sparse
  - Interpolation: Use a combination of different granularities of n-grams

Smoothing intuition

When we have sparse statistics:

P(w | denied the)
3 allegations
2 reports
1 claims
1 request

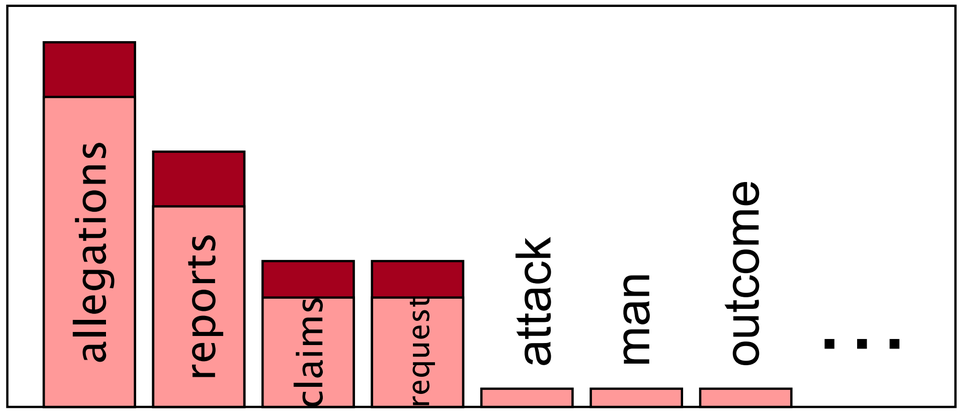
7 total



Steal probability mass to generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other

7 total



Laplace smoothing (add-α)

- **Simplest smoothing heuristic**: add α to all counts and renormalize!
- **Example**: max likelihood estimate for bigrams:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

- After smoothing:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha|V|}$$

Raw bigram counts (Berkeley restaurant corpus)

- Out of 9222 sentences

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Smoothed bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Add 1 to all the entries in the matrix

Smoothed bigram probabilities

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Problem with Laplace smoothing

Raw counts

$$C(w_{n-1}w_n) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \times C(w_{n-1})$$

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Problem with Laplace smoothing

Raw counts

$$C(w_{n-1}w_n) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \times C(w_{n-1})$$

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Reconstituted counts

$$C^*(w_{n-1}w_n) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V} \times C(w_{n-1})$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Problem with Laplace smoothing

Raw counts

$$C(w_{n-1}w_n) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \times C(w_{n-1})$$

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Reconstituted counts

$$C^*(w_{n-1}w_n) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V} \times C(w_{n-1})$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Problem with Laplace smoothing

Raw counts

$$C(w_{n-1}w_n) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \times C(w_{n-1})$$

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Reconstituted counts

$$C^*(w_{n-1}w_n) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V} \times C(w_{n-1})$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Problem with Laplace smoothing

Raw counts

$$C(w_{n-1}w_n) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \times C(w_{n-1})$$

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Reconstituted counts

$$C^*(w_{n-1}w_n) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V} \times C(w_{n-1})$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Discounting

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

$$P_{\text{abs\_discount}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} \text{ if } c(w_{i-1}, w_i) > 0$$

$$\alpha(w_{i-1}) \frac{P(w_i)}{\sum_{w'} P(w')} \text{ for all } w' \text{ s.t. } c(w_{i-1}, w') = 0 \text{ if } c(w_{i-1}, w_i) = 0$$

Unigram probabilities

Discounting

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

- Determine some "mass" to remove from probability estimates

$$P_{\text{abs\_discount}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} \text{ if } c(w_{i-1}, w_i) > 0$$

$$\alpha(w_{i-1}) \frac{P(w_i)}{\sum_{w'} P(w')} \text{ for all } w' \text{ s.t. } c(w_{i-1}, w') = 0 \text{ if } c(w_{i-1}, w_i) = 0$$

Unigram probabilities

Discounting

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

- Determine some "mass" to remove from probability estimates

$$P_{\text{abs\_discount}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} \text{ if } c(w_{i-1}, w_i) > 0$$

$$\alpha(w_{i-1}) \frac{P(w_i)}{\sum_{w'} P(w')} \text{ for all } w' \text{ s.t. } c(w_{i-1}, w') = 0 \text{ if } c(w_{i-1}, w_i) = 0$$

Unigram probabilities

Discounting

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

- Determine some "mass" to remove from probability estimates
- Redistribute mass among unseen n-grams

$$P_{\text{abs\_discount}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} \text{ if } c(w_{i-1}, w_i) > 0$$

$$\alpha(w_{i-1}) \frac{P(w_i)}{\sum_{w'} P(w')} \text{ for all } w' \text{ s.t. } c(w_{i-1}, w') = 0 \text{ if } c(w_{i-1}, w_i) = 0$$

Unigram probabilities

Discounting

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

- Determine some "mass" to remove from probability estimates
- Redistribute mass among unseen n-grams
- Just choose an absolute value to discount (usually <1)

$$P_{\text{abs\_discount}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} \text{ if } c(w_{i-1}, w_i) > 0$$

$$\alpha(w_{i-1}) \frac{P(w_i)}{\sum_{w'} P(w')} \text{ for all } w' \text{ s.t. } c(w_{i-1}, w') = 0 \text{ if } c(w_{i-1}, w_i) = 0$$

Unigram probabilities

Discounting

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

- Determine some "mass" to remove from probability estimates
- Redistribute mass among unseen n-grams
- Just choose an absolute value to discount (usually <1)

$$P_{\text{abs\_discount}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} \text{ if } c(w_{i-1}, w_i) > 0$$

$$\alpha(w_{i-1}) \frac{P(w_i)}{\sum_{w'} P(w')} \text{ for all } w' \text{ s.t. } c(w_{i-1}, w') = 0 \text{ if } c(w_{i-1}, w_i) = 0$$

Unigram probabilities

Absolute Discounting

x	Count(x)	Count$^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Absolute Discounting

- Define Count*(x) = Count(x) - 0.5

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Absolute Discounting

- Define Count*(x) = Count(x) - 0.5
- Missing probability mass:

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Absolute Discounting

- Define Count*(x) = Count(x) - 0.5
- Missing probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_{w} \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Absolute Discounting

- Define Count*(x) = Count(x) - 0.5
- Missing probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_{w} \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

$$\alpha(the) = 10 \times 0.5/48 = 5/48$$

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Absolute Discounting

- Define Count*(x) = Count(x) - 0.5
- Missing probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_{w} \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

$$\alpha(the) = 10 \times 0.5/48 = 5/48$$

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Absolute Discounting

- Define Count*(x) = Count(x) - 0.5
- Missing probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_{w} \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

$$\alpha(the) = 10 \times 0.5/48 = 5/48$$

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Absolute Discounting

- Define Count*(x) = Count(x) - 0.5
- Missing probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_{w} \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

$$\alpha(the) = 10 \times 0.5/48 = 5/48$$

- Divide this mass between words w for which Count(the, w) = 0

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Back-off

Back-off

- Use n-gram if enough evidence, else back off to (n-1)-gram

Back-off

- Use n-gram if enough evidence, else back off to (n-1)-gram

$$
\begin{aligned}
& P_{bo}(w_i \mid w_{i-n+1} \cdots w_{i-1}) \\
&= \begin{cases} d_{w_{i-n+1} \cdots w_i} \dfrac{C(w_{i-n+1} \cdots w_{i-1} w_i)}{C(w_{i-n+1} \cdots w_{i-1})} & \text{if } C(w_{i-n+1} \cdots w_i) > k \\ \alpha_{w_{i-n+1} \cdots w_{i-1}} P_{bo}(w_i \mid w_{i-n+2} \cdots w_{i-1}) & \text{otherwise} \end{cases}
\end{aligned}
$$

- d = amount of discounting
- α = back-off weight

Back-off

- Use n-gram if enough evidence, else back off to (n-1)-gram

$$
\begin{aligned}
&P_{bo}(w_i \mid w_{i-n+1} \cdots w_{i-1}) \\
&= \begin{cases} d_{w_{i-n+1} \cdots w_i} \dfrac{C(w_{i-n+1} \cdots w_{i-1} w_i)}{C(w_{i-n+1} \cdots w_{i-1})} & \text{if } C(w_{i-n+1} \cdots w_i) > k \\ \alpha_{w_{i-n+1} \cdots w_{i-1}} P_{bo}(w_i \mid w_{i-n+2} \cdots w_{i-1}) & \text{otherwise} \end{cases}
\end{aligned}
$$

(Katz back-off)

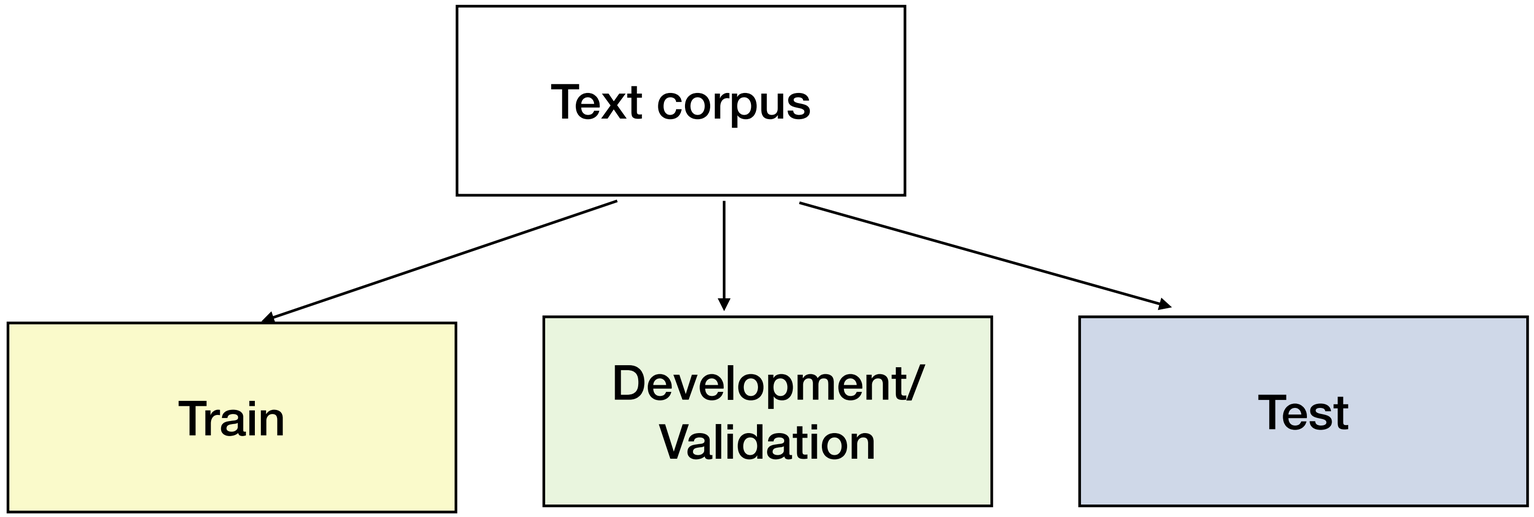
- d = amount of discounting
- α = back-off weight

Linear Interpolation

$$\hat{P}(w_i|w_{i-1}, w_{i-2}) = \lambda_1 P(w_i|w_{i-1}, w_{i-2}) \quad \text{Trigram}$$
$$+\lambda_2 P(w_i|w_{i-1}) \quad \text{Bigram}$$
$$+\lambda_3 P(w_i) \quad \text{Unigram}$$
$$\sum_i \lambda_i = 1$$

- Use a combination of models to estimate probability
- Strong empirical performance

Choosing lambdas



- First, estimate n-gram prob. on training set
- Then, estimate lambdas (hyperparameters) to maximize probability on the held-out development/validation set
- Use best model from above to evaluate on test set

Recap

- *n*-gram language models are simple, effective methods for estimating the probability of sequences
- **Problem #1:** Number of modelable sequences grows exponentially with *n*
- **Problem #2:** Smoothing heuristics must be used to estimate the probability of sequences unseen during training

Language Models: Fixed-context Neural Models

Antoine Bosselut

EPFL

Today's Outline

- **Part 1:** Language models introduction, count-based language models, evaluating language models, smoothing
- **Part 2:** Fixed-context Language Models
- **Additional Slides:** Neural networks recap

Some slides adapted from Danqi Chen, Mohit Iyyer

Question

What's another issue with n-gram language models?

Problem

With n-gram language models, there is no notion of similarity unless sequence overlaps!

We treat all words / prefixes as independent of on another!

students opened their ___

pupils opened their ___

scholars opened their ___

undergraduates opened their ___

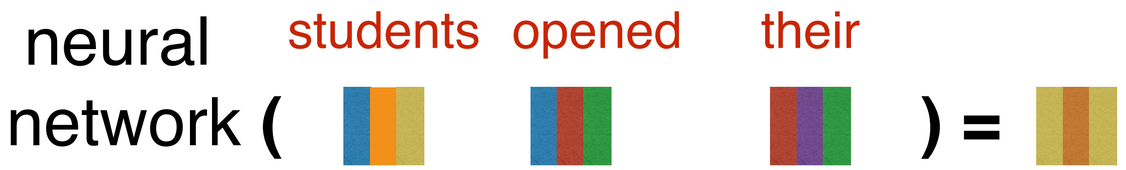
students turned the pages of their ___

students attentively perused their ___

Ideally, we should share information across these same prefixes!

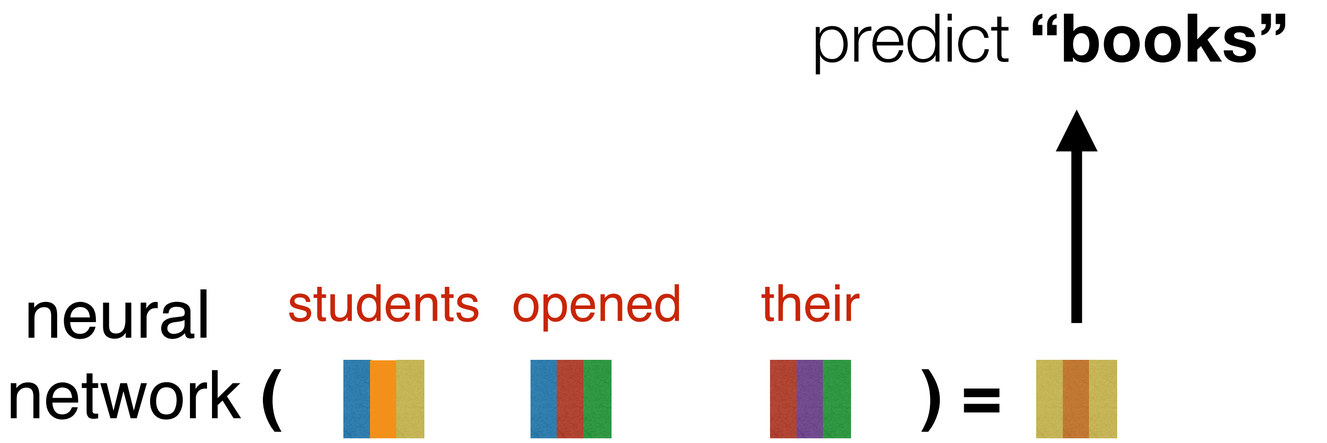
Recall

- neural networks **compose** word embeddings into vectors for natural language phrases, sentences, and documents



Recall

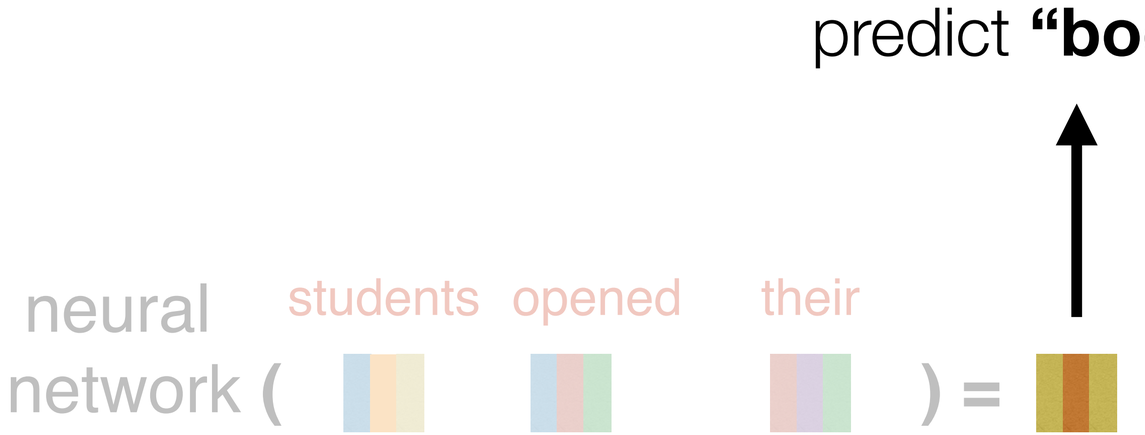
- neural networks **compose** word embeddings into vectors for natural language phrases, sentences, and documents
- Predict the next word from composed prefix representation



How does this happen?

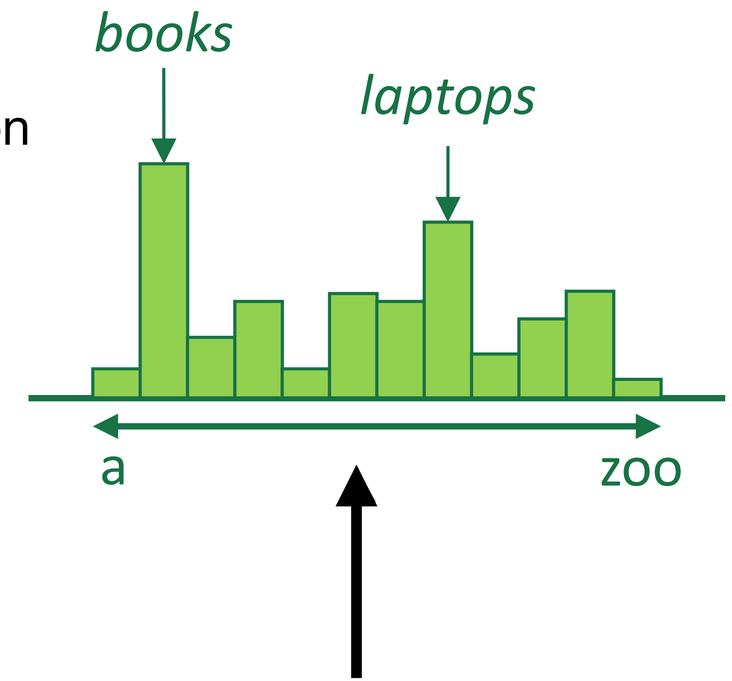
How does this happen?

Softmax layer:
Converts a vector representation into a probability distribution over the entire vocabulary



Softmax

Probability distribution over the entire vocabulary

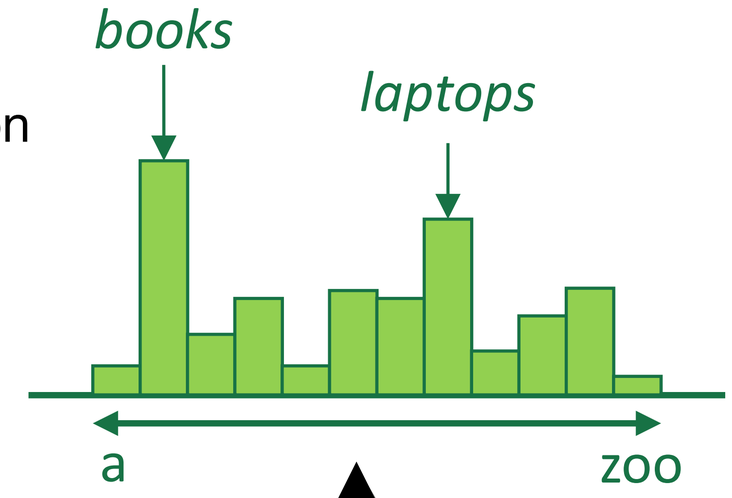


Low-dimensional representation of "students opened their"

Softmax

$P(w_i \mid \text{vector for "students opened their"})$

Probability distribution over the entire vocabulary



Low-dimensional representation of “students opened their”

Let’s say our output vocabulary consists of just four words: “books”, “houses”, “lamps”, and “stamps”.

Low-dimensional representation of “students opened their”

Let’s say our output vocabulary consists of just four words: “books”, “houses”, “lamps”, and “stamps”.

books houses lamps stamps

<0.6, 0.2, 0.1, 0.1>

We want to get a probability distribution over these four words

Low-dimensional representation of “students opened their”

How do we get there?

$$\mathbf{W} = \begin{Bmatrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{Bmatrix}$$

$\boldsymbol{x}$ = <-2.3, 0.9, 5.4>

Here's an example 3-d prefix vector

How do we get there?

$$\mathbf{W} = \begin{Bmatrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{Bmatrix}$$

first, we'll project our 3-d prefix representation to 4-d with a matrix-vector product

$\boldsymbol{x}$ = <-2.3, 0.9, 5.4>

Here's an example 3-d prefix vector

How do we get there?

$$\mathbf{W} = \begin{Bmatrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{Bmatrix}$$

$\boldsymbol{x}$ = <-2.3, 0.9, 5.4>

intuition: each dimension of $\boldsymbol{x}$ corresponds to a *feature* of the prefix

How do we get there?

$$\mathbf{W} = \begin{matrix} \\ \\ \\ \\ \end{matrix} \left\{ \begin{matrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{matrix} \right\} \begin{matrix} \text{books} \\ \text{houses} \\ \text{lamps} \\ \text{stamps} \end{matrix}$$

intuition: each row of **W** contains *feature weights* for a corresponding word in the vocabulary

$$\boldsymbol{x} = \langle -2.3, 0.9, 5.4 \rangle$$

intuition: each dimension of $\boldsymbol{x}$ corresponds to a *feature* of the prefix

$$\mathbf{W}\boldsymbol{x} = \langle 1.8, -11.9, 12.9, -8.9 \rangle$$

How did we compute this? It's just the dot product of each row of **W** with ***x***!

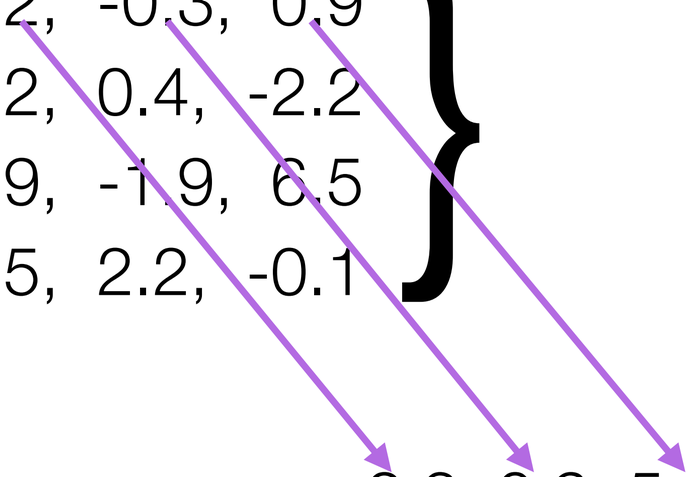
$$\mathbf{W} = \begin{Bmatrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{Bmatrix}$$

$$\boldsymbol{x} = \langle -2.3, 0.9, 5.4 \rangle$$

$\mathbf{W}\boldsymbol{x}$ = <1.8, -11.9, 12.9, -8.9>

How did we compute this? It's just the dot product of each row of **W** with ***x***!

$$\mathbf{W} = \begin{Bmatrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{Bmatrix}$$



$\boldsymbol{x}$ = <-2.3, 0.9, 5.4>

$\mathbf{W}\boldsymbol{x}$ = <1.8, -11.9, 12.9, -8.9>

How did we compute this? It's just the dot product of each row of **W** with ***x***!

$$\mathbf{W} = \begin{Bmatrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{Bmatrix}$$

1.2 * -2.3
+ -0.3 * 0.9
+ 0.9 * 5.4

$\boldsymbol{x}$ = <-2.3, 0.9, 5.4>

Softmax

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

- x is a vector
- x_i is dimension *i* of x
- each dimension *i* of the softmaxed output represents the probability of class *i*

W*x* = <1.8, -1.9, 2.9, -0.9>

softmax(W*x*) = <0.24, 0.006, 0.73, 0.02>

Softmax

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

- x is a vector
- x_i is dimension *i* of x
- each dimension *i* of the softmaxed output represents the probability of class *i*

W*x* = <1.8, -1.9, 2.9, -0.9>

softmax(W*x*) = <0.24, 0.006, 0.73, 0.02>

Softmax will keep popping up, so be sure to understand it!

Recap

- Given a *d*-dimensional vector representation ***x*** of a prefix, we do the following to predict the next word:

1. Project it to a *V*-dimensional vector using a matrix-vector product (a.k.a. a “linear layer”, or a “feedforward layer”), where *V* is the size of the vocabulary

2. Apply the softmax function to transform the resulting vector into a probability distribution

Question

How should we represent the history of the sequence?

Fixed-context Neural Language Models

- P(mat | the cat sat on the) = ?

the

cat

sat

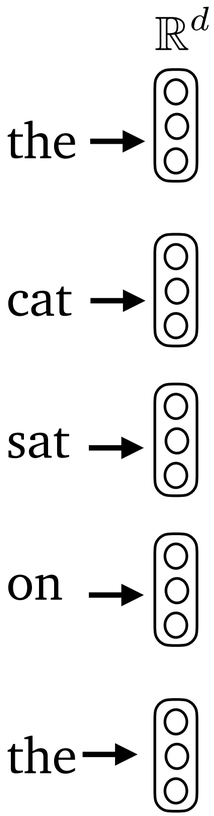
on

the

(Bengio et al., 2003): A Neural Probabilistic Language Model

Fixed-context Neural Language Models

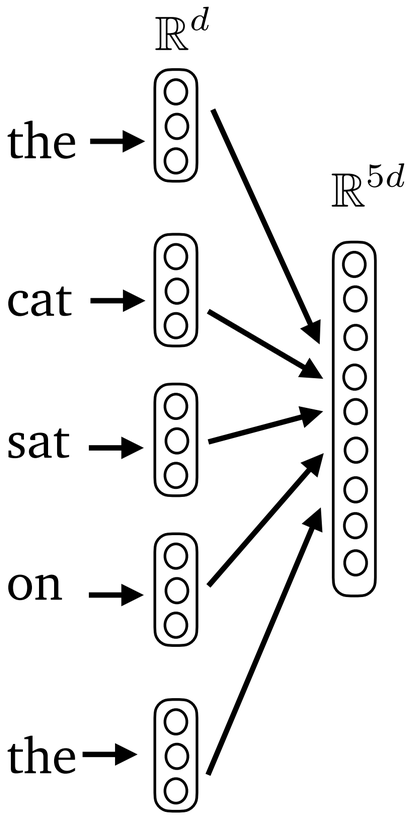
- P(mat | the cat sat on the) = ?



(Bengio et al., 2003): A Neural Probabilistic Language Model

Fixed-context Neural Language Models

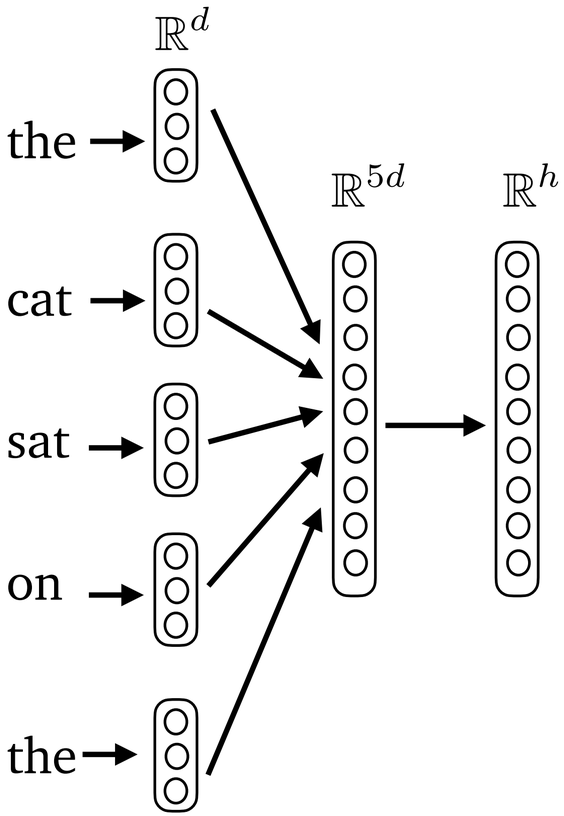
- P(mat | the cat sat on the) = ?



(Bengio et al., 2003): A Neural Probabilistic Language Model

Fixed-context Neural Language Models

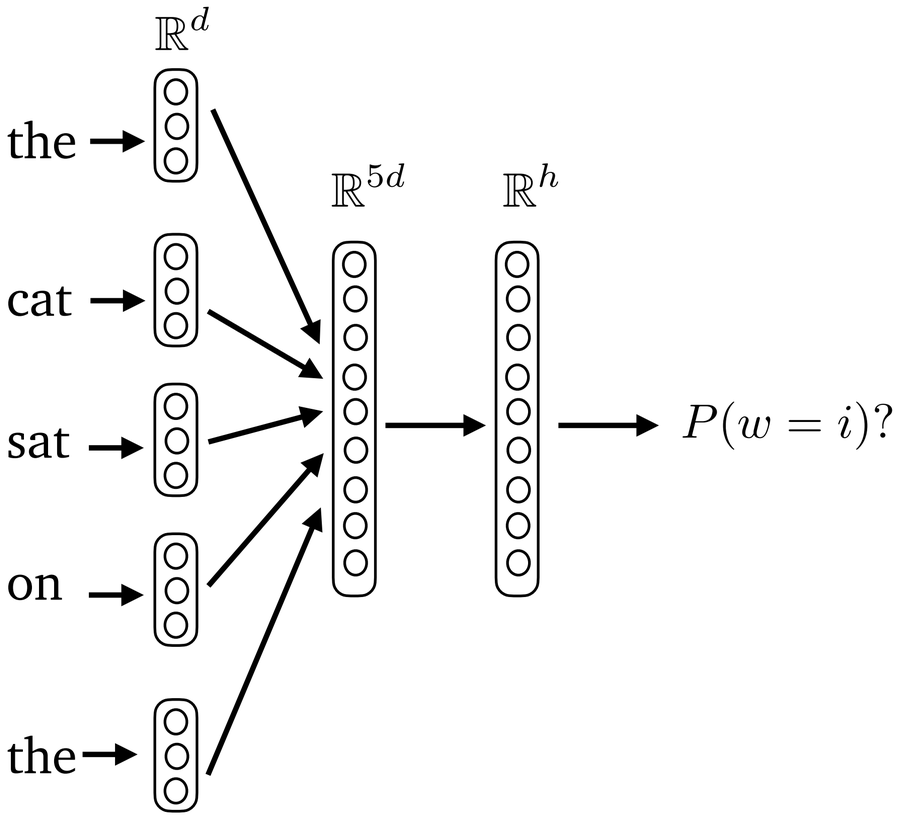
- P(mat | the cat sat on the) = ?



(Bengio et al., 2003): A Neural Probabilistic Language Model

Fixed-context Neural Language Models

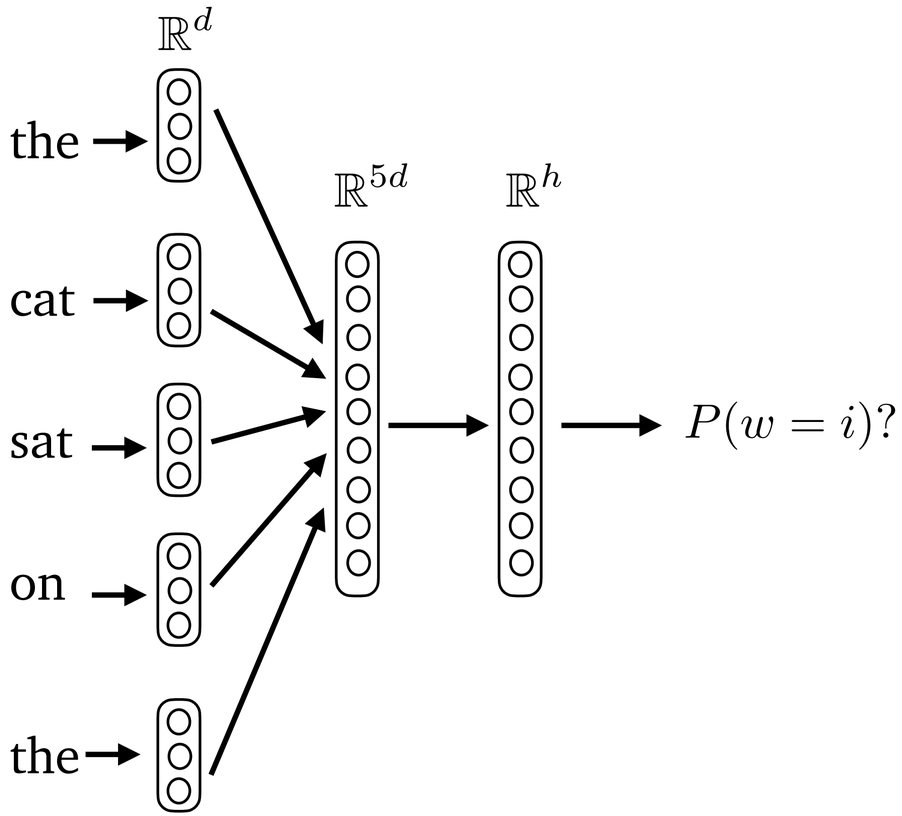
- P(mat | the cat sat on the) = ?



(Bengio et al., 2003): A Neural Probabilistic Language Model

Fixed-context Neural Language Models

- P(mat | the cat sat on the) = ?



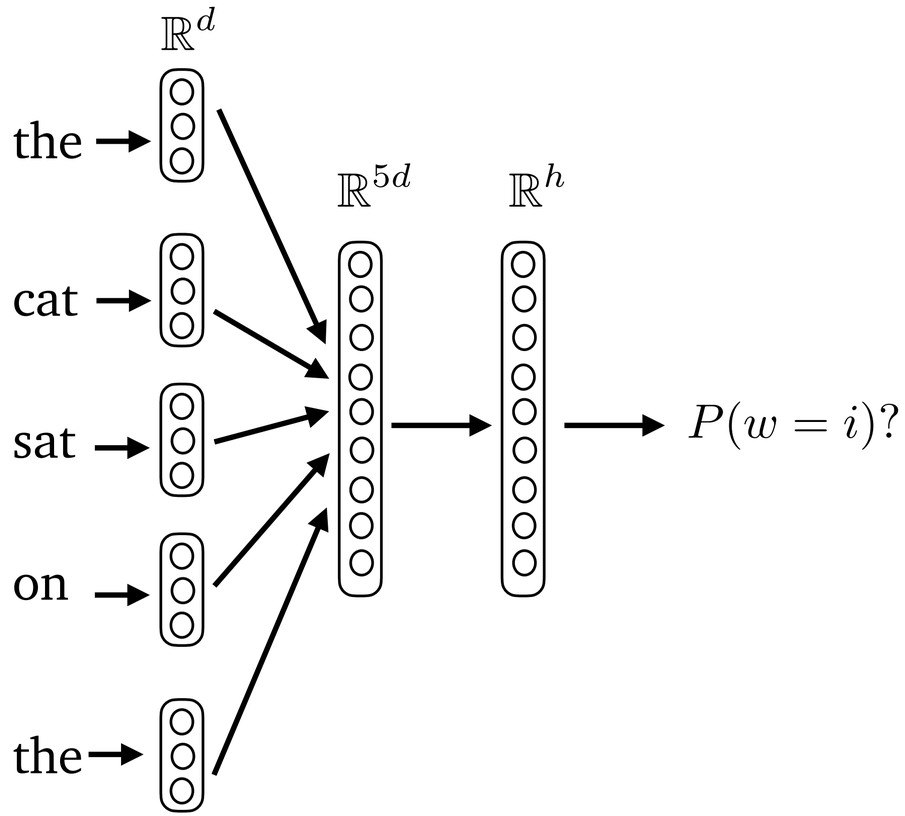
- Input layer (n = 5):

$$\mathbf{x} = [\mathbf{e}_{\text{the}}; \mathbf{e}_{\text{cat}}; \mathbf{e}_{\text{sat}}; \mathbf{e}_{\text{on}}; \mathbf{e}_{\text{the}}] \in \mathbb{R}^{dn}$$

(Bengio et al., 2003): A Neural Probabilistic Language Model

Fixed-context Neural Language Models

- P(mat | the cat sat on the) = ?



- Input layer (n = 5):

$$\mathbf{x} = [\mathbf{e}_{\text{the}}; \mathbf{e}_{\text{cat}}; \mathbf{e}_{\text{sat}}; \mathbf{e}_{\text{on}}; \mathbf{e}_{\text{the}}] \in \mathbb{R}^{dn}$$

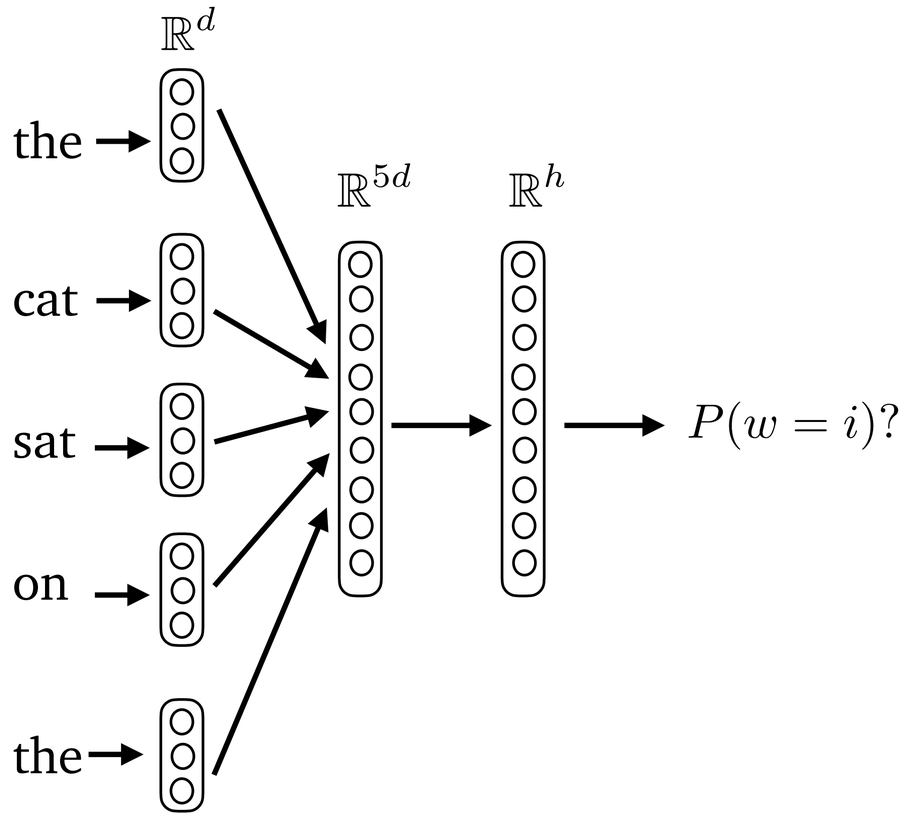
- Hidden layer

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b}) \in \mathbb{R}^h$$

(Bengio et al., 2003): A Neural Probabilistic Language Model

Fixed-context Neural Language Models

- P(mat | the cat sat on the) = ?



- Input layer (n = 5):

$$\mathbf{x} = [\mathbf{e}_{\text{the}}; \mathbf{e}_{\text{cat}}; \mathbf{e}_{\text{sat}}; \mathbf{e}_{\text{on}}; \mathbf{e}_{\text{the}}] \in \mathbb{R}^{dn}$$

- Hidden layer

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b}) \in \mathbb{R}^h$$

- Output layer (softmax)

$$\mathbf{z} = \mathbf{U}\mathbf{h} \in \mathbb{R}^{|V|}$$

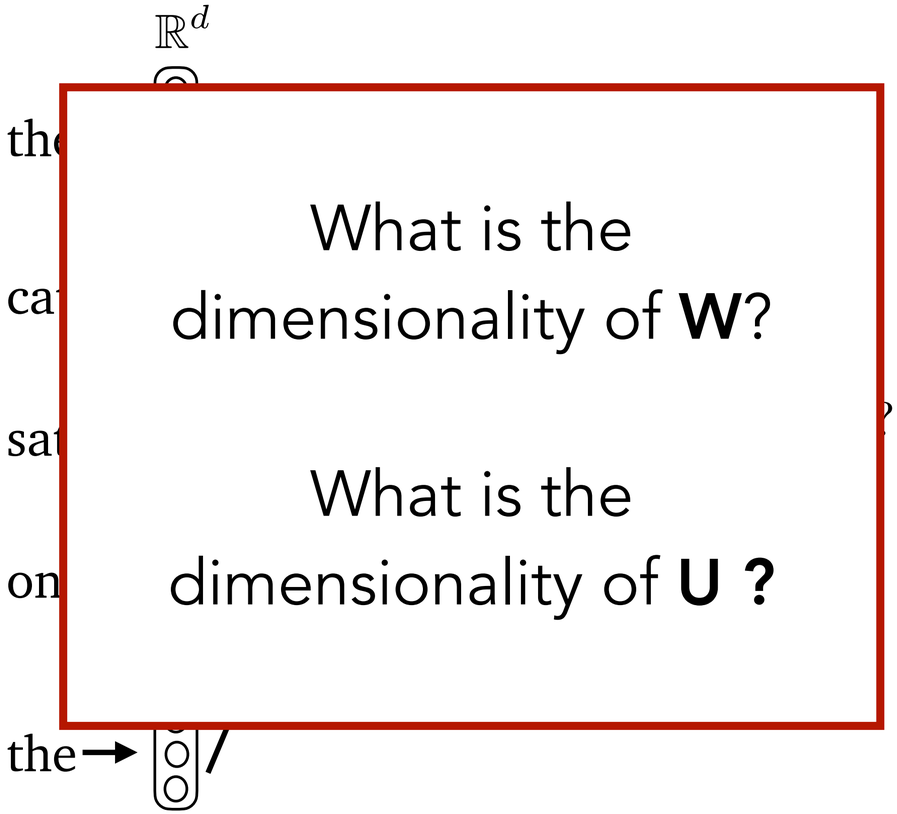
$$P(w = i \mid \text{the cat sat on the})$$

$$= \text{softmax}_i(\mathbf{z}) = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

(Bengio et al., 2003): A Neural Probabilistic Language Model

Fixed-context Neural Language Models

- P(mat | the cat sat on the) = ?



What is the dimensionality of **W**?

What is the dimensionality of **U ?**

- Input layer (n = 5):

$$\mathbf{x} = [\mathbf{e}_{\text{the}}; \mathbf{e}_{\text{cat}}; \mathbf{e}_{\text{sat}}; \mathbf{e}_{\text{on}}; \mathbf{e}_{\text{the}}] \in \mathbb{R}^{dn}$$

- Hidden layer

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b}) \in \mathbb{R}^h$$

- Output layer (softmax)

$$\mathbf{z} = \mathbf{U}\mathbf{h} \in \mathbb{R}^{|V|}$$

$$P(w = i \mid \text{the cat sat on the})$$

$$= \text{softmax}_i(\mathbf{z}) = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

(Bengio et al., 2003): A Neural Probabilistic Language Model

Advantages vs. Disadvantages

- No more sparsity problem
  - All sequences can be estimated with non-zero probability ! **Why?**
- Model size is much smaller!
  - Depends on number of weights in model, not number of sequences!

- Fixed windows are still too small to encode **long-range dependencies**
- Enlarging the window size makes the weight matrix **W** larger (more computationally expensive!)
- Weights in **W** aren't shared across embeddings in the window (computationally inefficient!)

Question

What algorithm did we see last week do fixed context language models resemble the most?

Question

What algorithm did we see last week do fixed context language models resemble the most?

CBOW!

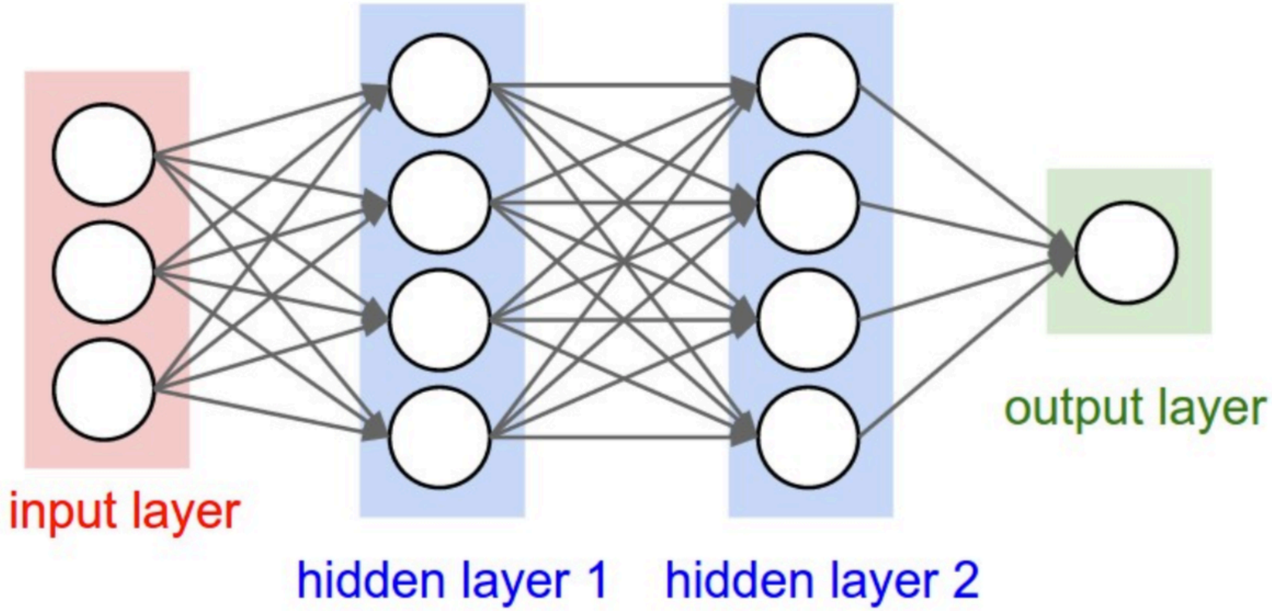
Recap

- Neural language models allow us to *share information* among similar sequences by learning neural representations that similarly represent them
- **Problem:** Fixed context language models can only process a limited window of the word history at a time
- **Tomorrow:** recurrent neural networks

Review:
Feedforward Neural Networks

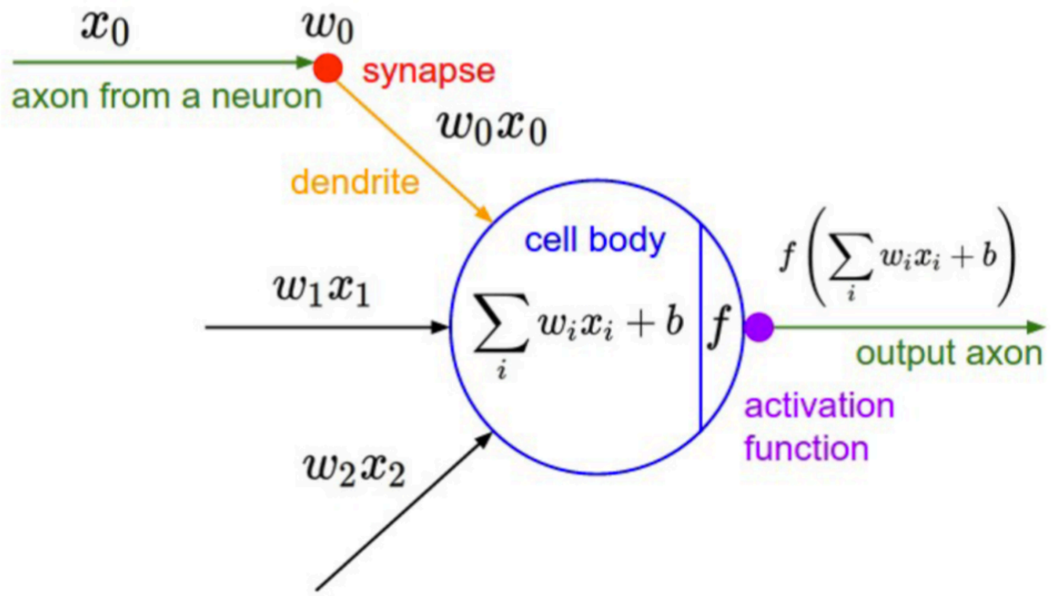
Feed-forward NNs

- Input: $x_1, \ldots, x_d \in \mathbb{R}$
- Output: $y^* \in \{0,1\}$



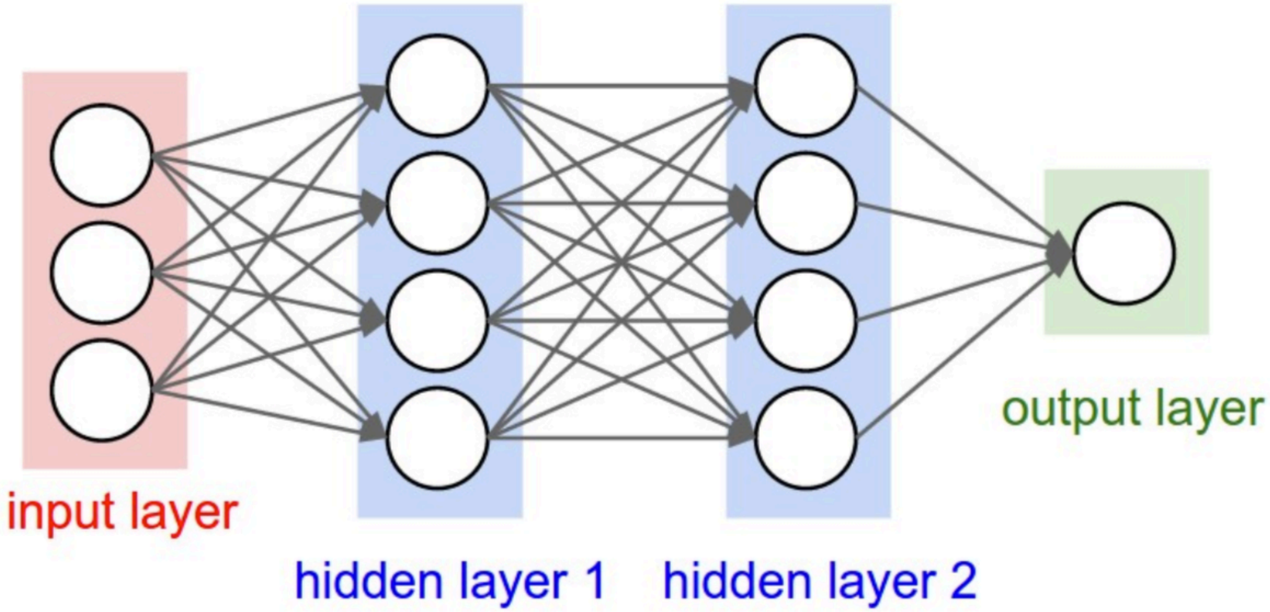
An artificial neuron

- A neuron is a computational unit that has scalar inputs and an output
- Each input has an associated weight.
- The neuron multiples each input by its weight, sums them, applied a **nonlinear activation function** to the result, and passes it to its output.

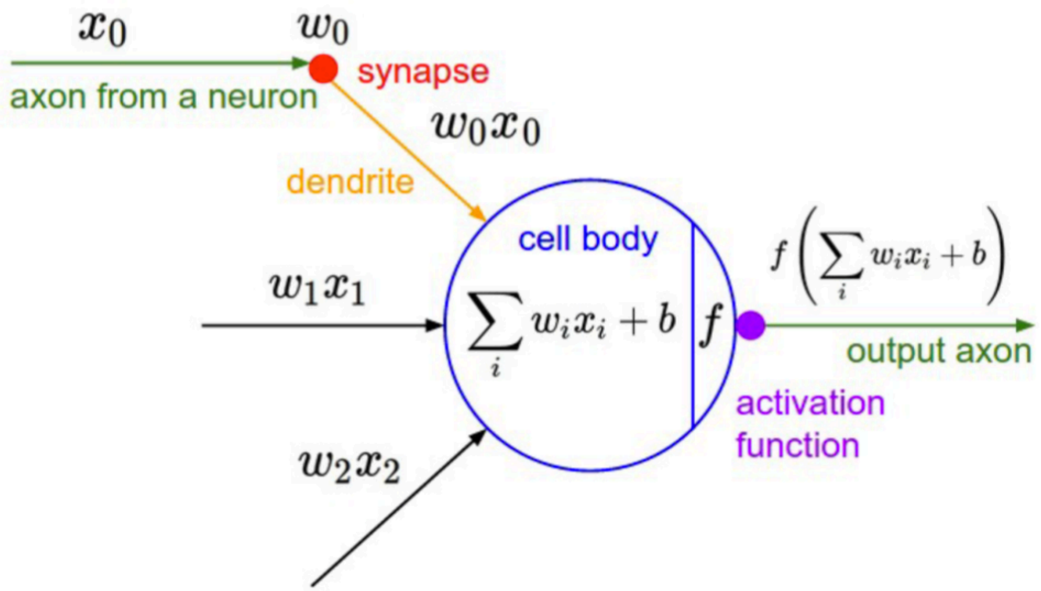


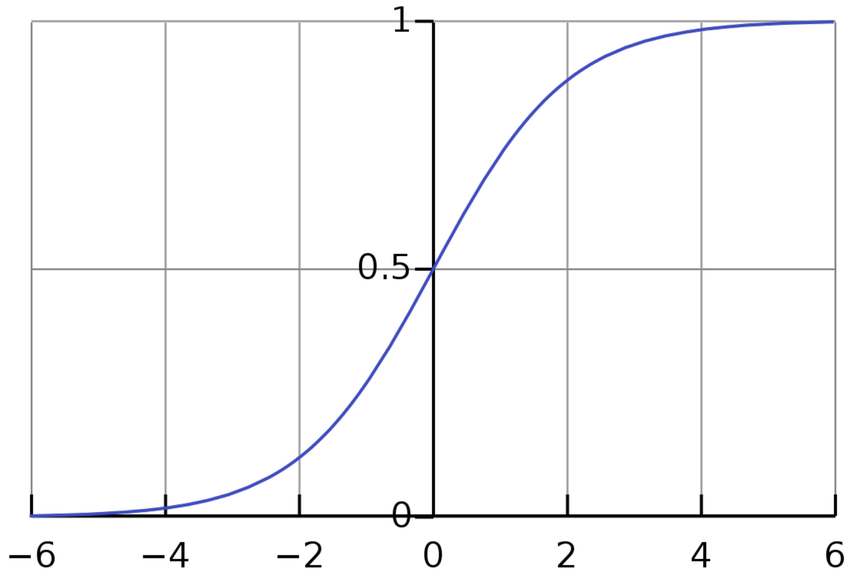
Neural networks

- The neurons are connected to each other, forming a **network**
- The output of a neuron may feed into the inputs of other neurons



A neuron can be a binary logistic regression unit

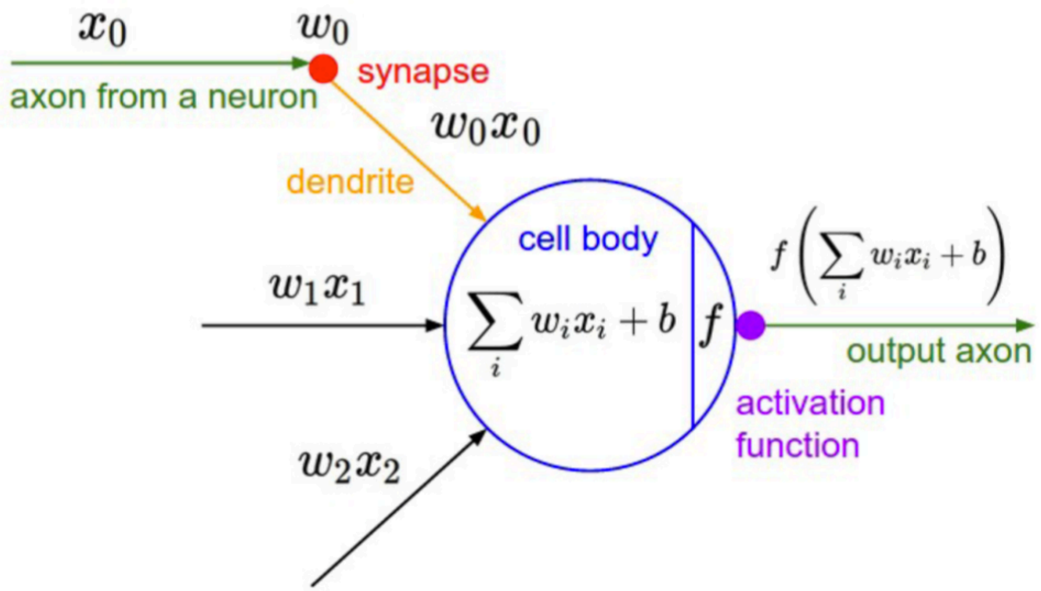


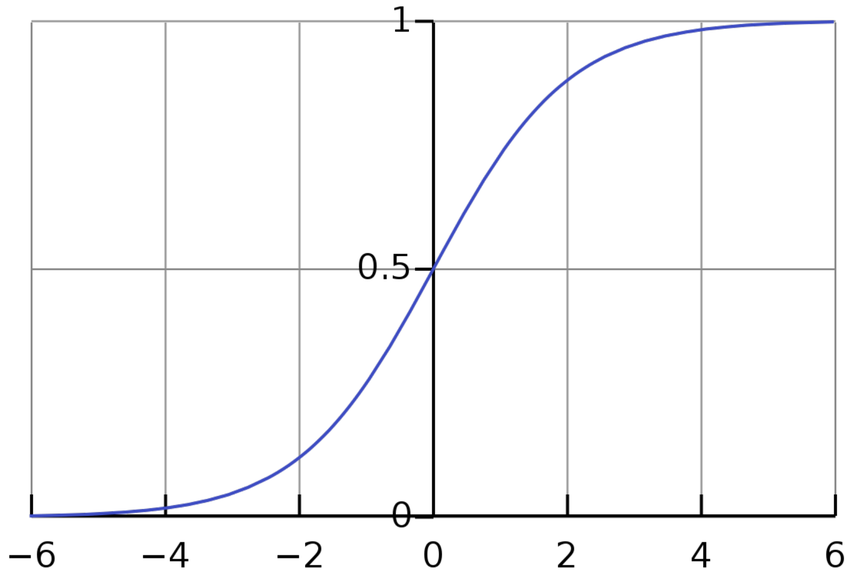


$$h_{\mathbf{w},b}(\mathbf{x}) = f(\mathbf{w}^\intercal \mathbf{x} + b)$$

$$f(z) = \frac{1}{1 + e^{-z}}$$

A neuron can be a binary logistic regression unit



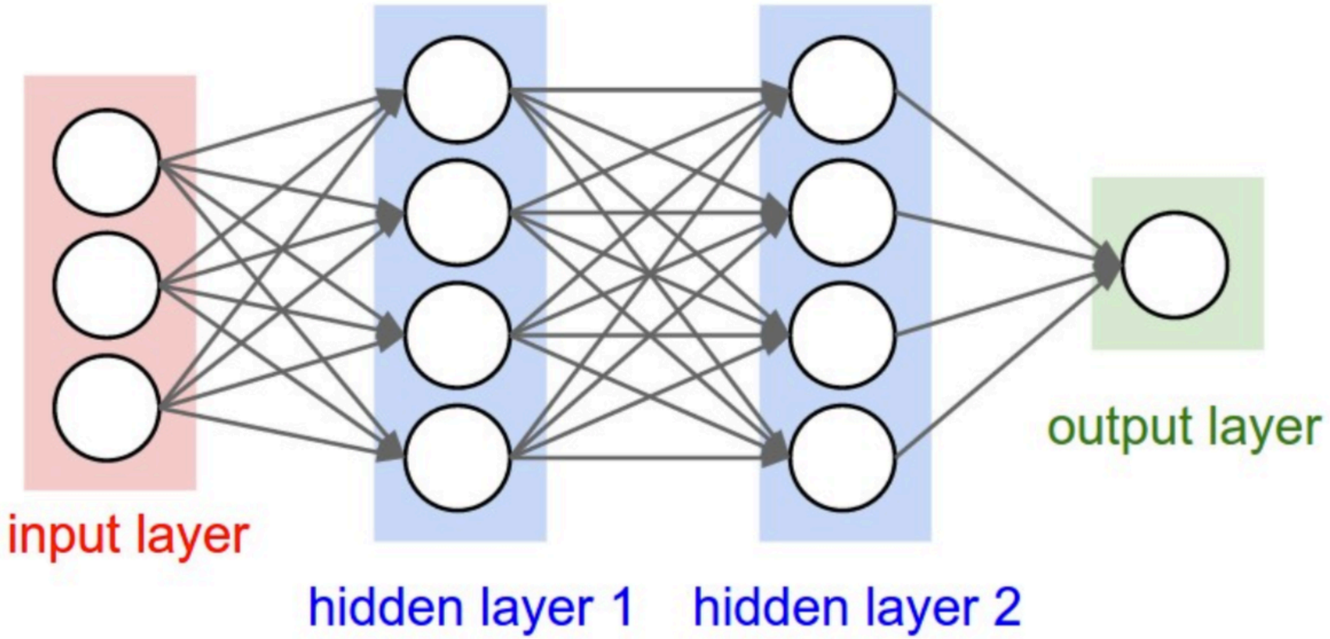


$$h_{\mathbf{w},b}(\mathbf{x}) = f(\mathbf{w}^\intercal \mathbf{x} + b)$$

$$f(z) = \frac{1}{1 + e^{-z}}$$

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

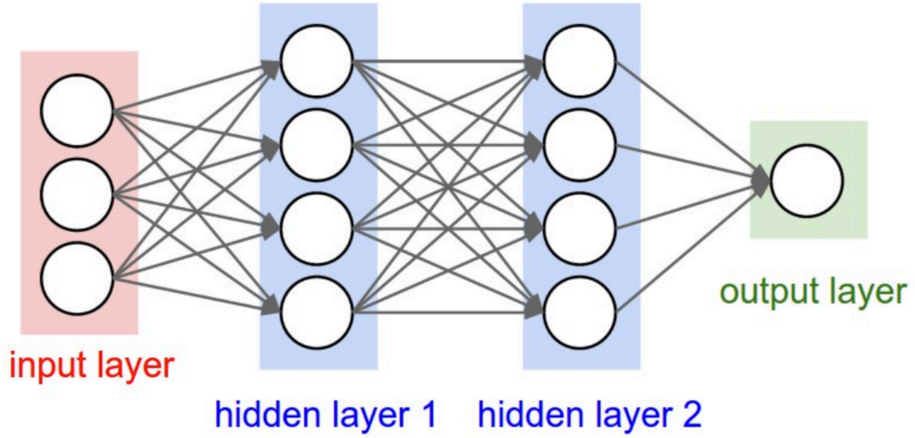
NNs = running several logistic regressions at the same time



If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs, which we can feed into another logistic regression function

Mathematical notations

- Input layer: $x_1, \ldots, x_d$



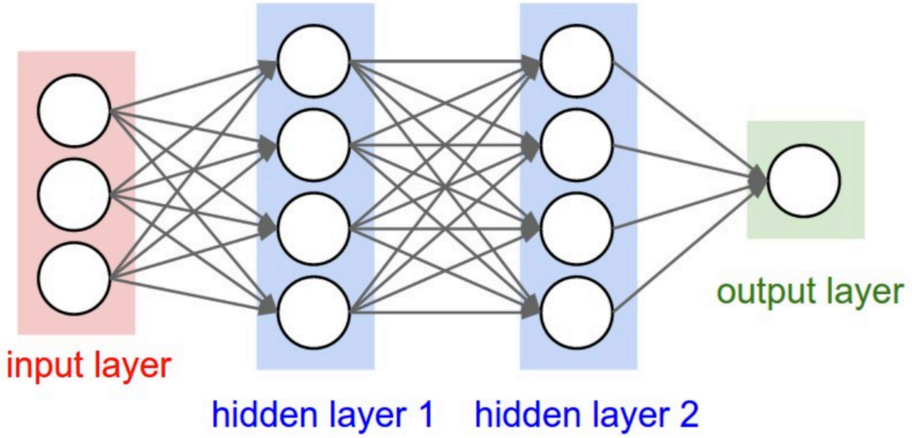
Mathematical notations

- Input layer: $x_1, \ldots, x_d$
- Hidden layer 1: $h_1^{(1)}, h_2^{(1)}, \ldots, h_{d_1}^{(1)}$

$$h_1^{(1)} = f(W_{1,1}^{(1)} + W_{1,2}^{(1)} x_2 + \ldots + W_{1,d}^{(1)} x_d + b_1^{(1)})$$

$$h_2^{(1)} = f(W_{2,1}^{(1)} + W_{2,2}^{(1)} x_2 + \ldots + W_{2,d}^{(1)} x_d + b_2^{(1)})$$

$$\ldots$$



Mathematical notations

- Input layer: $x_1, \ldots, x_d$
- Hidden layer 1: $h_1^{(1)}, h_2^{(1)}, \ldots, h_{d_1}^{(1)}$

$$h_1^{(1)} = f(W_{1,1}^{(1)} + W_{1,2}^{(1)} x_2 + \ldots + W_{1,d}^{(1)} x_d + b_1^{(1)})$$

$$h_2^{(1)} = f(W_{2,1}^{(1)} + W_{2,2}^{(1)} x_2 + \ldots + W_{2,d}^{(1)} x_d + b_2^{(1)})$$

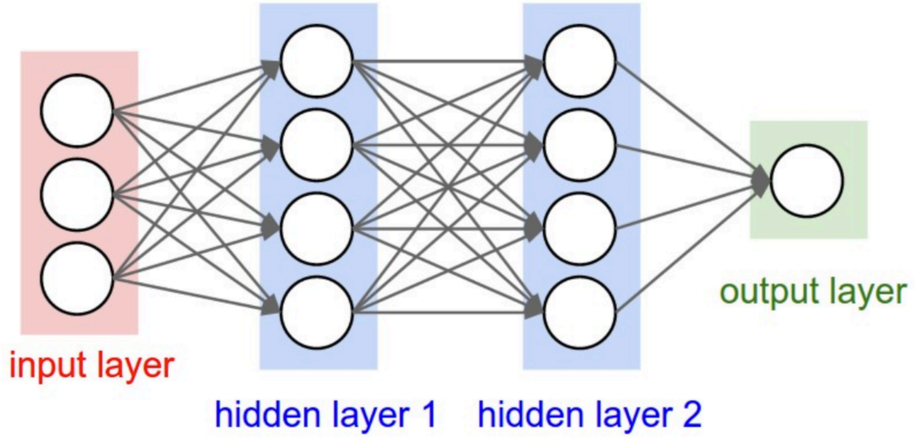
$$\ldots$$

- Hidden layer 2: $h_1^{(2)}, h_2^{(2)}, \ldots, h_{d_2}^{(2)}$

$$h_1^{(2)} = f(W_{1,1}^{(2)} h_1^{(1)} + W_{1,2}^{(2)} h_2^{(1)} + \ldots + W_{1,d_1}^{(2)} h_{d_1}^{(1)} + b_1^{(2)})$$

$$h_2^{(2)} = f(W_{2,1}^{(2)} h_1^{(1)} + W_{2,2}^{(2)} h_2^{(1)} + \ldots + W_{2,d_1}^{(2)} h_{d_1}^{(1)} + b_2^{(2)})$$

$$\ldots$$



Mathematical notations

- Input layer: $x_1, \ldots, x_d$
- Hidden layer 1: $h_1^{(1)}, h_2^{(1)}, \ldots, h_{d_1}^{(1)}$

$$h_1^{(1)} = f(W_{1,1}^{(1)} + W_{1,2}^{(1)} x_2 + \ldots + W_{1,d}^{(1)} x_d + b_1^{(1)})$$

$$h_2^{(1)} = f(W_{2,1}^{(1)} + W_{2,2}^{(1)} x_2 + \ldots + W_{2,d}^{(1)} x_d + b_2^{(1)})$$

$$\ldots$$

- Hidden layer 2: $h_1^{(2)}, h_2^{(2)}, \ldots, h_{d_2}^{(2)}$

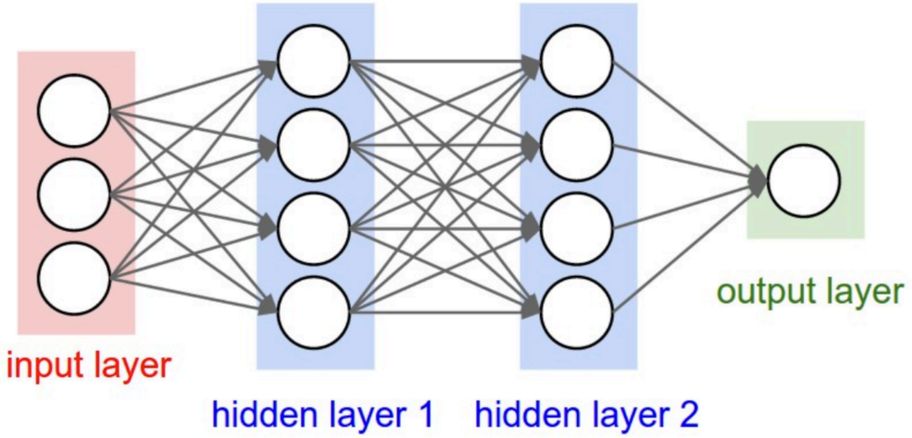
$$h_1^{(2)} = f(W_{1,1}^{(2)} h_1^{(1)} + W_{1,2}^{(2)} h_2^{(1)} + \ldots + W_{1,d_1}^{(2)} h_{d_1}^{(1)} + b_1^{(2)})$$

$$h_2^{(2)} = f(W_{2,1}^{(2)} h_1^{(1)} + W_{2,2}^{(2)} h_2^{(1)} + \ldots + W_{2,d_1}^{(2)} h_{d_1}^{(1)} + b_2^{(2)})$$

$$\ldots$$

- Output layer:

$$y = \sigma(w_1^{(o)} h_1^{(2)} + w_2^{(o)} h_2^{(2)} + \ldots + w_{d_2}^{(o)} h_{d_2}^{(2)} + b^{(o)})$$



Matrix notations

- Input layer: $\mathbf{x} \in \mathbb{R}^d$
- Hidden layer 1:

$$\mathbf{h}_1 = f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \in \mathbb{R}^{d_1}$$

$$\mathbf{W}^{(1)} \in \mathbb{R}^{d_1 \times d}, \mathbf{b}^{(1)} \in \mathbb{R}^{d_1}$$

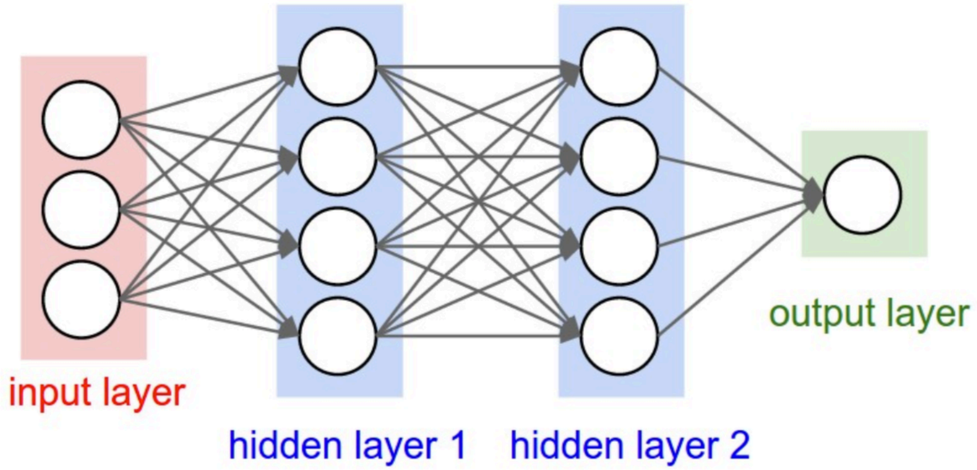
- Hidden layer 2:

$$\mathbf{h}_2 = f(\mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)}) \in \mathbb{R}^{d_2}$$

$$\mathbf{W}^{(2)} \in \mathbb{R}^{d_2 \times d_1}, \mathbf{b}^{(2)} \in \mathbb{R}^{d_2}$$

- Output layer:

$$y = \sigma(\mathbf{w}^{(o)} \cdot \mathbf{h}_2 + b^{(o)})$$



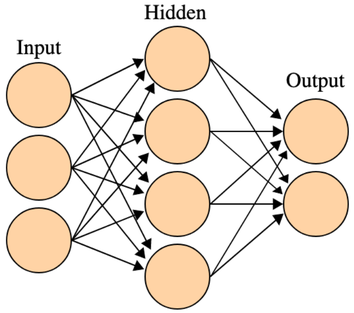
*: f is applied element-wise

$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$

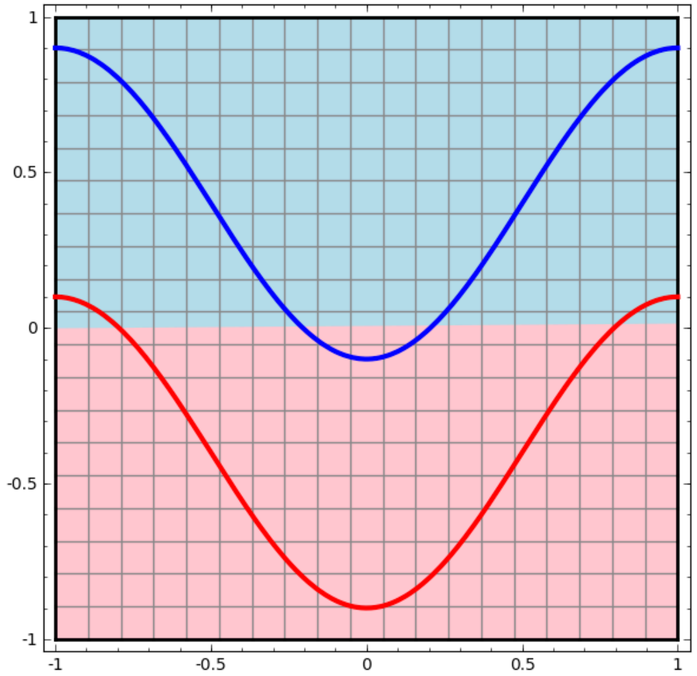
Why non-linearities?

- Non-linearities allow neural networks to learn more complex functions and nonlinear decision boundaries

The capacity of the network increases with more hidden units and more hidden layers

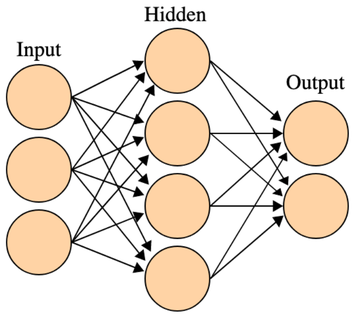


Why non-linearities?

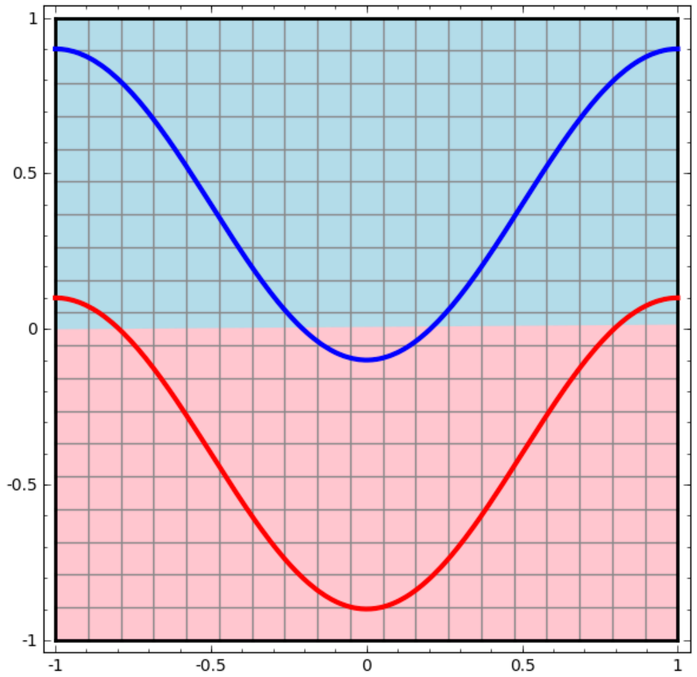


Linear classifier

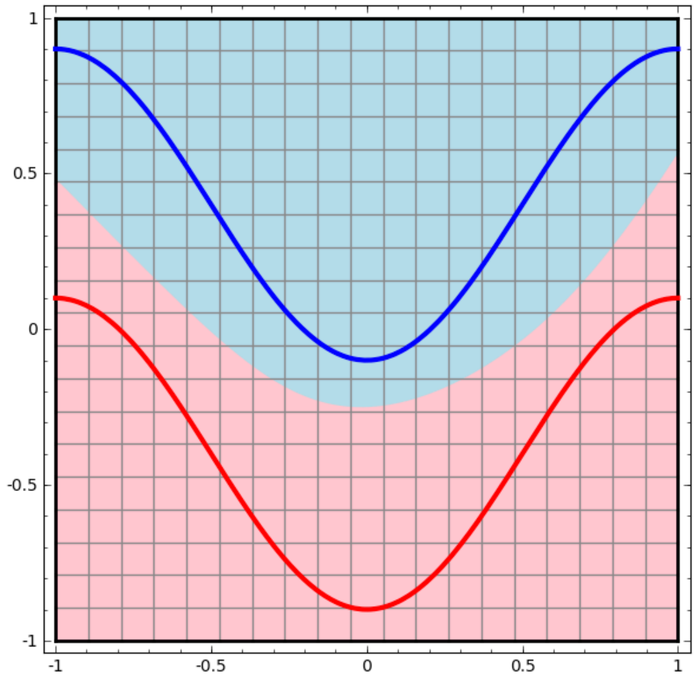
https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/



Why non-linearities?

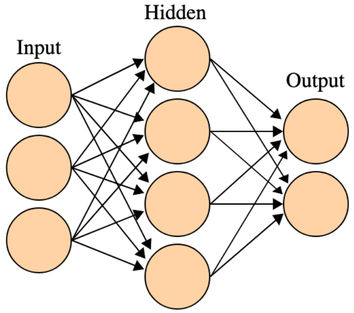


Linear classifier

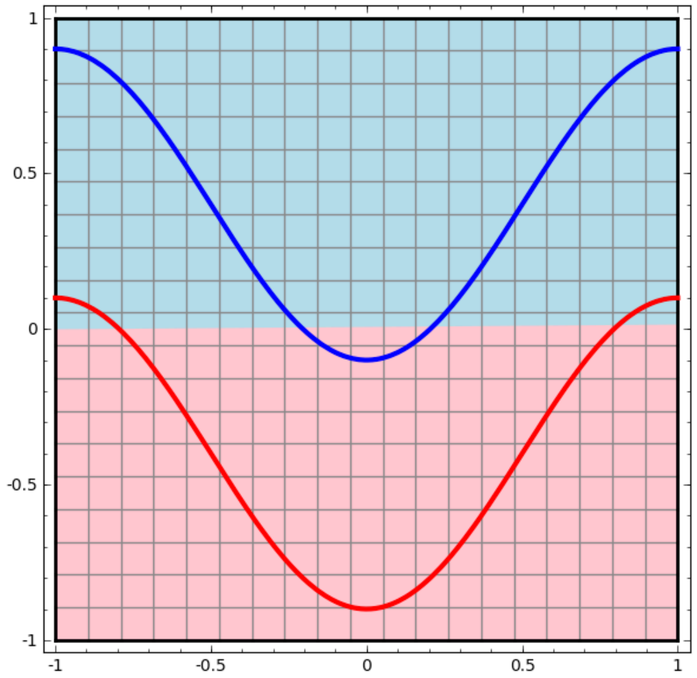


Neural Networks

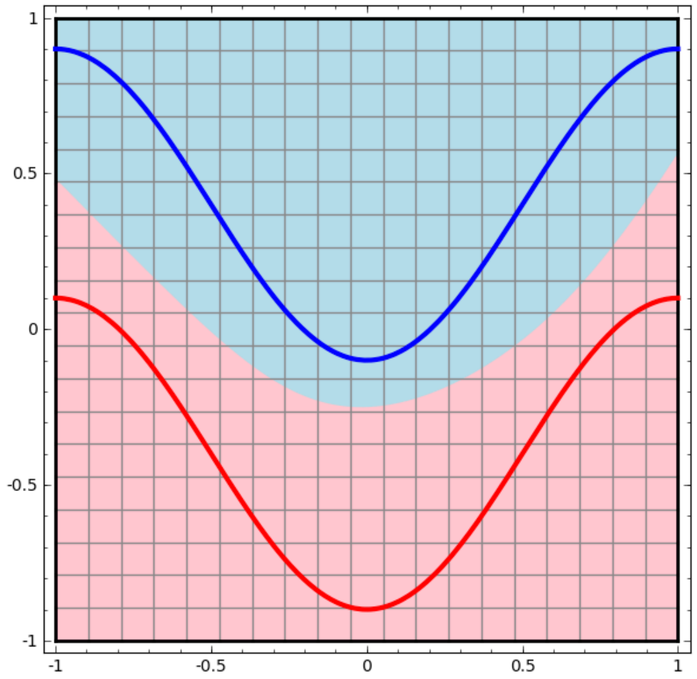
https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/



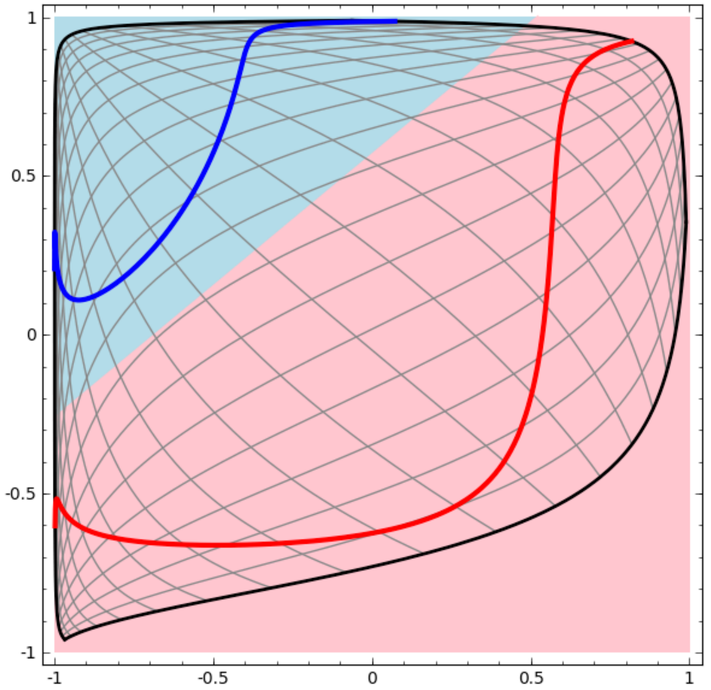
Why non-linearities?



Linear classifier



Neural Networks



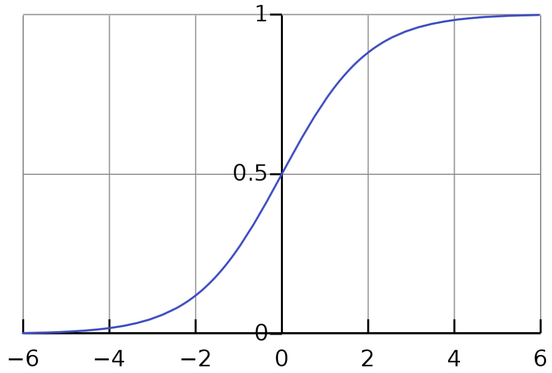
Hidden representations
are linearly separable!

https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Activation functions

Sigmoid

$$f(z) = \frac{1}{1 + e^{-z}}$$

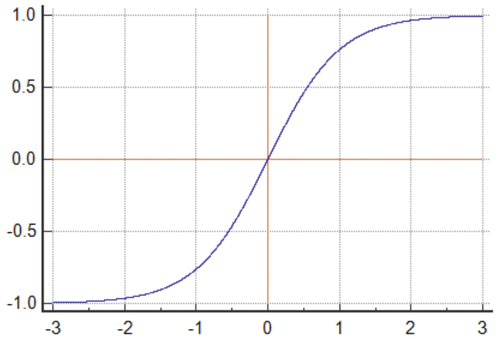


$$f'(z) = f(z) \times (1 - f(z))$$

tanh

(Hyperbolic tangent)

$$f(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

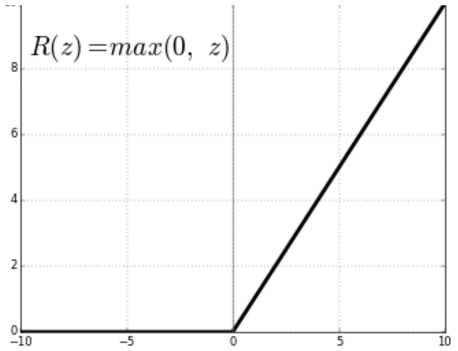


$$f'(z) = 1 - f(z)^2$$

ReLU

(rectified linear unit)

$$f(z) = \max(0, z)$$

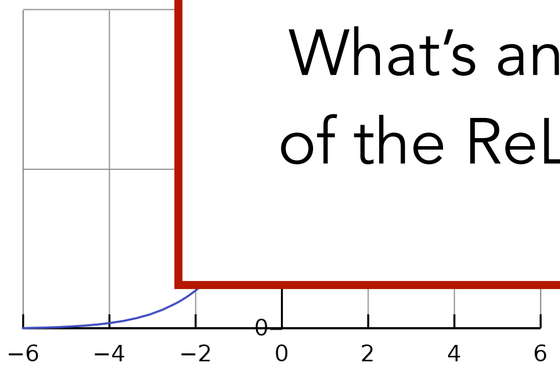


$$f'(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$$

Activation functions

Sigmoid

$$f(z) = \frac{1}{}$$

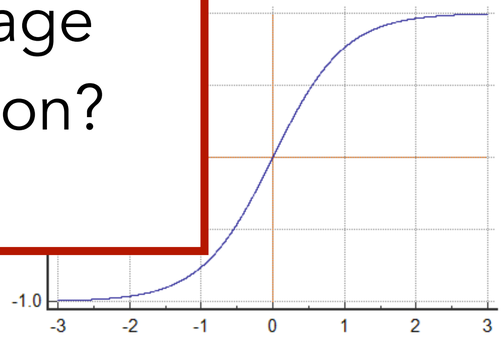


$$f'(z) = f(z) \times (1 - f(z))$$

tanh

(Hyperbolic tangent)

$$\frac{e^{2z} - 1}{e^{2z} + 1}$$

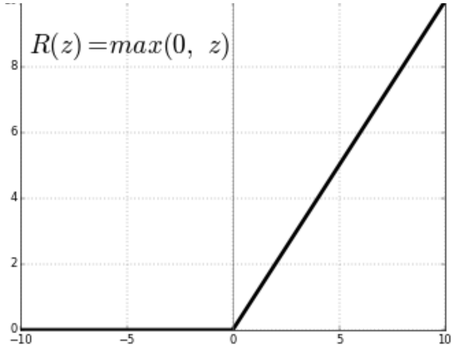


$$f'(z) = 1 - f(z)^2$$

ReLU

(rectified linear unit)

$$f(z) = \max(0, z)$$



$$f'(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$$

What's an advantage of the ReLU function?

How to train neural networks?

How to train neural networks?

Binary classification: $y^* = \{0,1\}$

$$y = \sigma(\mathbf{w}^{(o)} \cdot \mathbf{h}_2 + b^{(o)})$$

$$L_{CE} = -\sum_{i=1}^{n} [y^* \log \hat{y} + (1 - y^*)\log(1 - y)]$$

How to train neural networks?

Binary classification: $y^* = \{0,1\}$

$$y = \sigma(\mathbf{w}^{(o)} \cdot \mathbf{h}_2 + b^{(o)})$$

$$L_{CE} = -\sum_{i=1}^{n} [y^* \log \hat{y} + (1 - y^*)\log(1 - y)]$$

Multi-class classification: $y^* = \{1,2,\ldots,C\}$

$$y = \text{softmax}(\mathbf{W}^{(o)}\mathbf{h}_2 + \mathbf{b}^{(o)}) \qquad \mathbf{W}^{(o)} \in \mathbb{R}^{C \times h}, \mathbf{b}^{(o)} \in \mathbb{R}^{C}$$

- $x = h_2$ in this example

$$L_{CE}(y, y^*) = -\sum_{c=1}^{C} 1\{y^* = c\}\log \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$