The foundations: Ngrams, Semantic Embeddings, Tokenization

Sept 11, 2023
CUNY Graduate Center

What is a language model anyway?

Words and probabilities

"Today is the first day of”

- What do you predict the next word should be?

Words and probabilities

"Today is the first day of”

- the rest of your life
- school
- winter
- table
- etc.

- Language models complete this task by assigning a probability to each possible next word.
- N-gram language models are the simplest.

N-gram language model (Jurafsky & Martin, 2023)

- **Goal**: Predict the probability of a sentence.
- **Why**?
  - Machine Translation
    - P(we must vote) > P(our must vote)
  - Spelling correction
    - P(it’s getting late) > P(it’s geting laet)
  - Speech Recognition
    - P(We went to catch up) > P(We went to ketchup) [for a laugh on Buzzfeed]
  - Predictive Text
    - Emails
  - Question Answering, etc.

N-gram language models

Goal: Predict probability of a sentence.

$$P(W) = P(w1,w2,w3,w4,w5...wn)$$

Subgoal: Predict the probability of of a word, given all the other words

$$P(w5|w1,w2,w3,w4)$$

Conditional Probabilities & The Chain Rule

$$P(B \mid A) = \frac{P(A \cap B)}{P(A)}$$ Rewrites as $$P(A \cap B) = P(A) \cdot P(B \mid A)$$

The probability of A and B (intersection) is the probability of A times the probability of B given A

P("it was the best of times") =

P(it) × P(was|it) × P(the|it was) × P(best|it was the)

× P(of|it was the best) × P(times|it was the best of)

How do we come up with those probabilities?

- Big database with all the sentences possible in the world stored?
  - No! Too many sentences
- Average raw word counts across the language?
  - No! Not enough word pairs
- Magic?

The Markov Assumption

Simplifying assumption: The future depends only on the present.

P(times | it was the best of) ≅ P(times | of) ← Bigram

P(times | it was the best of) ≅ P(times | best of) ← Trigram

Maximum Likelihood Estimation MLE

- All and only the words in the corpus
- Calculate the number of times each word appears in a context
- Return a probability for that word in that context
- The probability is the MLE

Calculating Probabilities

Let’s assume that our entire corpus is the first 4 clauses of *A Tale of Two Cities*

<s>It was the best of times</s>

<s>it was the worst of times</s>

<s>it was the age of wisdom</s>

<s>it was the age of foolishness</s>

Probability Interlude: Notation

$P(y \mid x)$ = The probability of **y** given **x**.

If we know **x**, then what is the probability of **y** happening? (note that x and y are flipped in natural language)

$P(y \mid x)$ will be written $P(x \rightarrow y)$

Probability Interlude: Chain Rule

The chain rule is a way to calculate the joint probabilities of a sequence of events. “What is the probability of all of these things happening?”

Think of events as words.

A sentence is then a sequence of events that all have to happen.

To calculate this probability, multiply the probability of each separate event.

Maximum Likelihood Estimation

$$P(x \to y) = \frac{count(x, y)}{count(x)}$$

Divide the number of times the bigram appears in the corpus by the number of times the first word of the bigram appears in the corpus.

This gives us the probability that y will appear right after x, given that we already know x

The Probabilities

bigram	count	w2/w1	prob
P(<s> → it)	4	4/4	1.0
P(it → was)	4	4/4	1.0
P(was → the)	4	4/4	1.0
P(the → best)	1	1/4	0.25
P(best → of)	1	1/1	1.0
P(of → times)	2	2/4	0.5
P(times → </s>)	2	2/2	1.0

<s>It was the best of times</s>

<s>it was the worst of times</s>

<s>it was the age of wisdom</s>

<s>it was the age of foolishness</s>

The Probabilities

P(<s> it was the best of times</s>)

P(<s> ➔ it) = 4/4 = 1.0

P(it ➔ was) = 4/4 = 1.0

P(was ➔ the) = 4/4 = 1.0

P(the ➔ best) = 1/1 = 1.0

P(the ➔ worst) = 1/1 = 1.0

P(the ➔ age) = 2/4 = .5

P(best ➔ of) = 1/4 = .25

P(worst ➔ of) = 1/4 = .25

P(age ➔ of) = 2/4 = .5

P(of ➔ times) = 2/4 = .5

P(of ➔ wisdom) = 1/4 =.25

P(of ➔ foolishness) = 1/4 = .25

bigram	count	w2/w1	prob
P(<s> ➔ it)	4	4/4	1
P(it ➔ was)	4	4/4	1
P(was ➔ the)	4	4/4	1
P(the ➔ best)	1	1/4	0.25
P(best ➔ of)	1	1/1	1
P(of ➔ times)	2	2/4	0.5
P(times ➔ </s>)	2	2/2	1

$$1.0 * 1.0 * 1.0 * .25 * 1.0 * .5 * 1.0 = .125$$

Trigrams, 4-grams, etc.

Bigrams consider 1 word and predict one word

Trigrams consider 2 words as “given” and predict one word

4-grams consider 3 words as “given” and predict one word

The model is built by finding all the bigrams (or trigrams, 4-grams, etc.) And calculating the probabilities for each group of words.

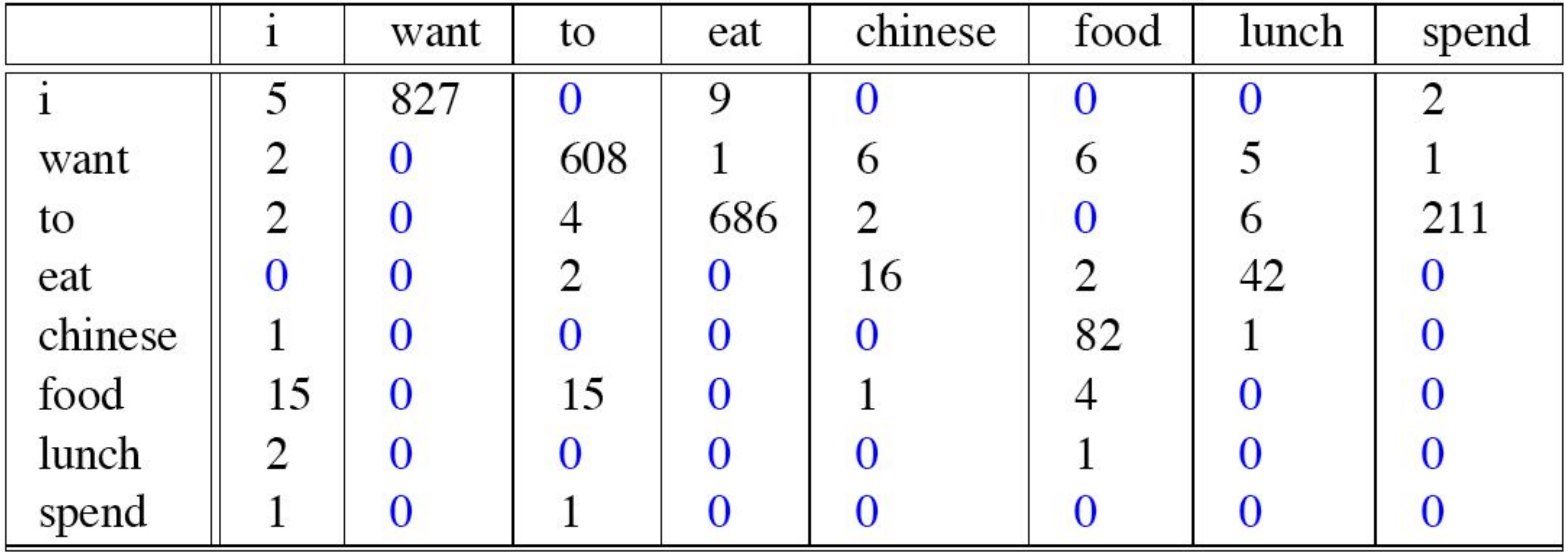
Some degree of randomness is typically encoded so models are not completely deterministic.

More examples: Berkeley Restaurant Project sentences

- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i’m looking for
- tell me about chez panisse
- can you give me a listing of the kinds of food that are available
- i’m looking for a good place to eat breakfast
- when is caffe venezia open during the day

Raw bigram counts

- 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

Raw bigram probabilities

- Normalize by unigrams:
- Result:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Bigram estimates of sentence probabilities

P(<s> I want english food </s>) =

P(I|<s>)

× P(want|I)

× P(english|want)

× P(food|english)

× P(</s>|food)

= .000031

What kinds of knowledge?

- $P(\text{english}|\text{want}) = .0011$
- $P(\text{chinese}|\text{want}) = .0065$
- $P(\text{to}|\text{want}) = .66$
- $P(\text{eat} \mid \text{to}) = .28$
- $P(\text{food} \mid \text{to}) = 0$
- $P(\text{want} \mid \text{spend}) = 0$
- $P(\text{i} \mid \text{<s>}) = .25$

Out of Vocabulary (OOV)

- Ngram models assign 0 probability to ngrams it hasn’t seen.

  It was the mediocre times

- Maybe mediocre hasn’t appeared in your corpus though it’s a perfectly fine word.
- What do we do?

The OOV Probabilities

<s>It was the mediocre times</s>

bigram	**count**	**w2/w1**	**prob**
P(<s> ➜ it)	4	4/4	1
P(it ➜ was)	4	4/4	1
P(was ➜ the)	4	4/4	1
P(the ➜ **mediocre**)	0	0/4	**0**
P(**mediocre** ➜ times)	0	2/0	**∞**
P(times ➜ </s>)	2	2/2	1

<s>It was the best of times</s>

<s>it was the worst of times</s>

<s>it was the age of wisdom</s>

<s>it was the age of foolishness</s>

Add 1 smoothing (Laplace Smoothing)

- Count up all the bigrams
- Add 1 to all of their counts
- All unknown words

The OOV Probabilities

bigram	count	w2/w1	prob
P(<s> → it)	**5**	5/5	1
P(it → was)	**5**	5/5	1
P(was → the)	**5**	5/5	1
P(the → best)	**2**	2/5	0.4
P(best → of)	**2**	2/2	1
P(of → times)	**3**	3/5	0.6
P(times → </s>)	**3**	3/5	1
P(the → <UNK>)	**1**	1/5	.2
P (<UNK> → times)	**1**	5/1	***

<s>It was the <UNK> times</s>

<s>It was the best of times</s>

<s>it was the worst of times</s>

<s>it was the age of wisdom</s>

<s>it was the age of foolishness</s>

*** this is where the math of this model is more than we are going to cover

Other Smoothing options

- Laplace shifts probabilities too much
- Other options (we will not cover)
  - Good-Turing discounting
  - Train an UNK word model (replace certain % of words with <UNK>)
  - Linear combination interpolation

N-grams - the limitations

- Language has long-distance dependencies

The passenger, who the bus driver thought was drunk ate a sandwich.

- N-grams are terrible for this, we need something better

N-grams the uses

Google ngrams [viewer | research]

Speech recognition

Spelling correction

Word Vectors

What do words mean?

- Maybe “see”:
  - lambda x: x sees y
  - perceive with the eyes; discern visually.
  - you see what you want to see
- All of this is useless for a computer

Words and their meanings
Lemmas and their senses

The man had to $\boxed{\text{duck}}$ under the door

The $\boxed{\text{duck}}$ swam in a pond

Senses:

1. Lower the head or body
2. An aquatic bird

Lemma: duck

Lemmas are not synonyms

Duck ≠ Duck

Let’s sit on the **sofa** tonight

Let’s sit on the **couch** tonight

The **story** was written in English

The **novel** was written in Spanish

Synonyms

Want to capture that these words are like each other:

- story/novel
- sofa/couch
- big/large
- sneaker/tennis shoe
-

Synonyms aren’t always synonyms (because lemmas aren’t always lemmas)

My **big** sister ≠ My **large** sister

Word associations

- Books, magazines: similar
- Books, library: related (not similar)

“Library” often appears in the same semantic frame as “books”

- Hot, cold
- Long, short
- Light, dark

Antonyms (opposites) also appear in similar semantic frames

Words are defined by the company they keep

Wittgenstein: **A word is defined by its use in the language.**

The story was written in English

The novel was written in Spanish

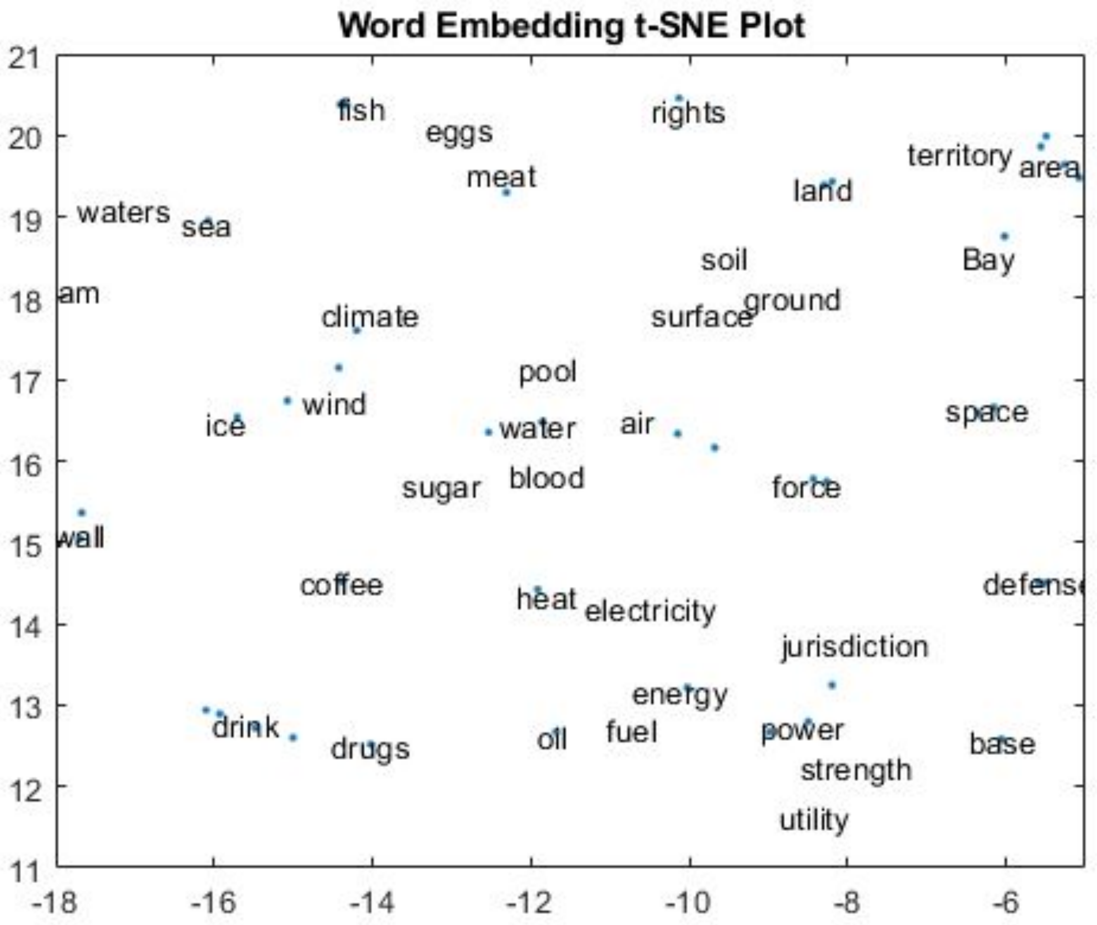
- story/novel ➜ synonyms
- English/Spanish ➜ languages

Defining similar contexts mathematically

- Definition based on the frequency a word appears near other words.
  - “Duck” near discussions of water or flying, probably the animal
  - “Duck” near objects in the built environment, probably the action.
- The words in the corpus create a multidimensional space
  - A vector encoding co-occurrence with other words for “all” words in the corpus

Words as vectors

- Each word is a vector
- 700+ dimensions
- PCA to display as 2D
- Similar words grouped



Fasttext embeddings example from here.

Why vectors?

- We can do vector math

Vector(“King”) — Vector(“Man”)+Vector(“Woman”) = Word(“Queen”)

- Classic text analysis requires matching on **words**.
- Consider sentiment analysis:
    - “delicious” is in the training set
    - “yummy” is unrecognizable to our model
- Matching on **embeddings**
    - [102, 33, 75...]
    - [100, 32, 78...]

A vector example: tf-idf

Term frequency - inverse document frequency

- Measures how important a word is within a document compared to a collection of documents
- Information retrieval
- Sparse (lots of 0’s)

TF-IDF

$$TF = \frac{\text{number of times a word appears in a doc}}{\text{total number of terms in the doc}}$$

$$IDF = log \frac{\text{number of docs in the corpus}}{\text{number of docs in the corpus that contain the term}}$$

$$TF\text{-}IDF = TF \cdot IDF$$

What does this mean?

The importance of a term is high when it occurs a lot in a given document and rarely in others

TF-IDF example

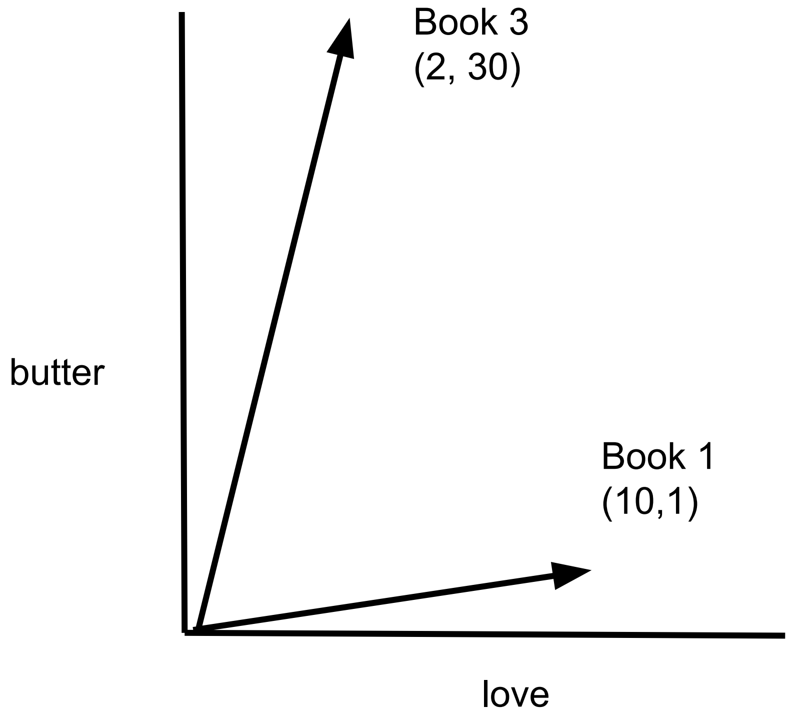
**this is fake data*

	love	technology	butter	total
Book 1	100	0	1	1,000
Book 2	0	50	0	10,000
Book 3	0	1	30	1,000

- Which do you think is the cookbook?

Words as vectors

	love	**technology**	**butter**	**total**
Book 1	100	0	1	1,000
Book 2	0	50	0	10,000
Book 3	0	1	30	1,000



TF-IDF example

$$TF = \frac{\text{number of times a word appears in a doc}}{\text{total number of terms in the doc}}$$

	love	technology	butter	total	TF (tech)
Book 1	100	0	1	1,000	0/1000
Book 2	0	50	0	10,000	50/10000
Book 3	0	1	30	1,000	1/1000

TF-IDF example

$$TF = \frac{\text{number of times a word appears in a doc}}{\text{total number of terms in the doc}}$$

$$IDF = log\,\frac{\text{number of docs in the corpus}}{\text{number of docs in the corpus that contain the term}}$$

	love	**technology**	**butter**	**total**	**TF (tech)**
Book 1	100	0	1	1,000	0/1000
Book 2	0	50	0	10,000	50/10000
Book 3	0	1	30	1,000	1/1000
IDF	log(1/3)	log(2/3)	log(2/3)		

TF-IDF

- Corpus can be a collection of books, tweets, sentences, etc.
- Every term has a relative importance with respect to other terms and to the corpus is general
- Valuable for search, summarization, etc.
- Bad for machine learning because the vectors are so SPARSE (too many 0’s to learn from)

Word2vec

- Better option because much denser

- 2013 developed by Google
- Paradigm shift:
  - Leverages a pre-trained neural network
  - Just 2 layers on the neural network
  - Vector is the weights from the softmax layer
- Considers the key word and a context word
  - CBOW or skip-gram

Word2vec

- Input is ‘one hot vector
- Output is probability that the input word is near “all” the other words.

- There is typically a limit of how many words we care about. Larger models = more words
- That is the vector

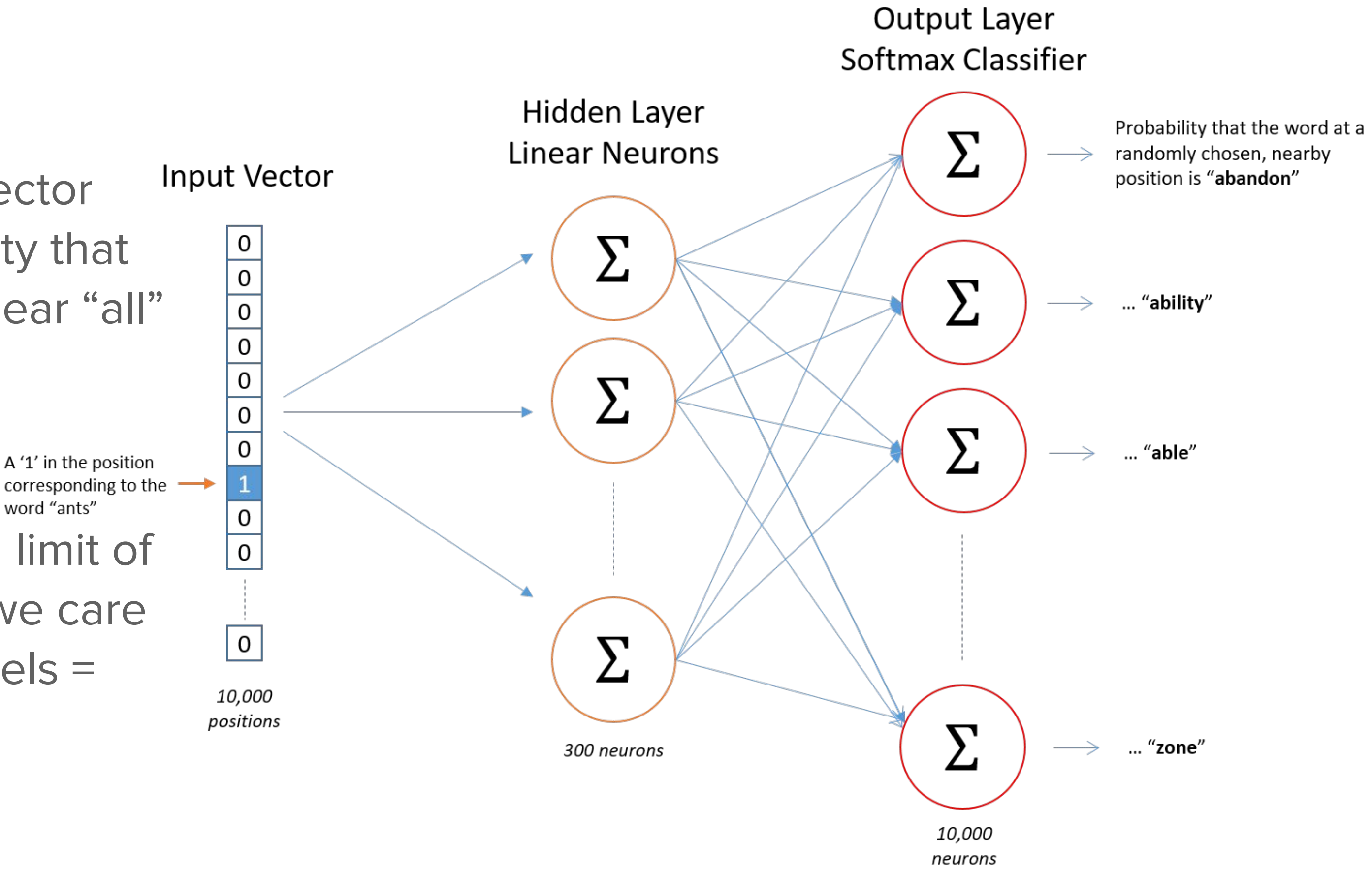


Diagram from here.

Word2vec

- Is the most typical way words are encoded
- Combined with Principal Component Analysis (a dimensionality reduction technique), allows for visualizing relationship between words
- https://projector.tensorflow.org/

Tokenization

Tokenization

- Tokens are word parts. They are not necessarily words.
- **Token-model matching :** whatever tokenizer is used for training the model must be what the model sees for prediction.

Breaking up words into tokens

- White space tokenization
- Linguistically aware tokenization
- Byte-Pair Encoding

White space tokenization

- Literally just breaking on white space
- Lose all morphological information (model ≠ models)
- Every word is a token
- Useful for pre-processing (then encoding word boundaries)

The class was learning about language models and learned how to model words as vectors.

[‘the’, ‘class’, ‘was’, ‘learning’, ‘about’, ‘language’, ‘models’, ‘and’, ‘learned’, ‘how’, ‘to’, ‘model’, ‘words’, ‘as’, ‘vectors’, ‘.’]

Linguistically aware tokenization (Lemmatizing)

- Separate lemmas (root words) from morphology
- Requires a lot of semantic knowledge manually added to model
- No new knowledge discovered
- Not tailored to corpus

The class was learning about language models and learned how to model words as vectors.

[‘the’, ‘class’, ‘was’, ‘learn’, ‘ing’, ‘about’, ‘language’, ‘model’, ‘s’, ‘and’, ‘learn’, ‘ed’, ‘how’, ‘to’, ‘model’, ‘word’, ‘s’, ‘as’, ‘vector’, ‘s’, ‘.’]

Byte Pair Encoding

- Algorithmic approach
- Start with letters and build up
- Find common combinations
- New information can be discovered by finding patterns in the data
- Adaptive to different corpora (non-standard writing)

Here’s a very good video at about 2 mins

https://youtu.be/HEikzVL-lZU

Byte Pair Encoding

1. Make a list of all the letters
2. Go through all the words to find how many times letters appear next to each other.
3. Merge the most common pair into a Byte-Pair.
4. Go through all the words to find the next most common pair (treating the merged BP as one unit.
5. Repeat until a vocab is complete
6. Complete is either a size (i.e., 10,000 tokens) or a number of iterations