

# CSC401 Notes

Jenci Wei

Winter 2024

## Contents

1 Natural Language Computing	3
2 Corpora	4
3 Language Modelling	6
4 Entropy and Decisions	9
5 Features	13
6 Classification	16
7 Neural Models of Language	20
8 Recurrent Neural Networks	27
9 Statistical Machine Traslation	31
10 Neural Machine Translation	35
11 Decoding and Evaluation of Machine Translation	39
12 Transformers	41
13 Large Language Models	46
14 Speech	51
15 Articulatory Phonetics	55
16 Speaker Classification	62
17 Automatic Speech Recognition	67
18 Information Retrieval	75
19 Text Summarization	78
20 Anthropomorphization	80

# 1 Natural Language Computing

Computational Linguistics (CL)

- Include applications such as text classification, automatic translation, automatic speech transcription, etc.

Catigories of Linguistic Knowledge

- **Phonology:** the study of patterns of speech *sounds*, e.g. “read” → /r iy d/
- **Morphology:** how words can be *changed* by inflection or derivation, e.g. “read”, “reads”, “reader”, “reading”
- **Syntax:** the *ordering and structure* between words and phrases, i.e. grammar
- **Semantics:** the study of how *meaning* is created by words and phrases
- **Pragmatics:** the study of meaning *in contexts*, e.g. explanation span, refutation span

Phonological Ambiguity

- Two different things may sound the same
- Ambiguities can often be *resolved* in context, but not always
- Some options may be syntactically invalid, so they can be ruled out

NLP as Artificial Intelligence

- NLP involves *resolving ambiguity* at all levels
  - Reasoning with *linguistic* knowledge (i.e. grammars)
  - Reasoning with *world* knowledge
- Can use numerical parameters (e.g. probabilities) to distinguish competing hypotheses
  - E.g. to determine whether “Google” is a noun or a verb
  - Google can be a noun: “Google makes Android”
  - Google can be a verb: “Google his name”
  - Usually  $\text{Pr}(\text{noun} \mid \text{Google}) > \text{Pr}(\text{verb} \mid \text{Google}) > 0$

## 2 Corpora

Types vs. Tokens

- “*The cat in the hat*”
- First *the* is the type, which is the “kind” of word
- Second *the* is the token, which is the “instance” of the word
- However the distinction between types and tokens is not always clear

Corpora

- **Corpus:** a body of language data of a particular sort (plural: **corpora**)
- The best corpora occur naturally (e.g. newspaper, UN transcripts, etc.)
- Can use corpora to gather statistics
  - More is better
  - Bias may be present
- Sizes of corpora
  - $10^6$  words: tiny
  - $10^9$  words: reasonable
  - $10^{13}$  words: GPT-3
  - $10^{14}$  words: GPT-4
- **Lexicon:** a collection of word-types, like a dictionary (as in the book, not the data structure), but not necessarily with meanings

Frequency Statistics

- (**Term**) **frequency** is defined as

$$TF(w, S) = \# \text{ tokens of } w \text{ in corpus } S$$

- **Relative frequency** is defined as

$$F_S(w) = \frac{TF(w, S)}{|S|}$$

- We have

$$\lim_{|S| \rightarrow \infty} F_S(w) = \Pr(w)$$

- However, when we grow both the corpus  $S$  and the lexicon  $V$ , the average relative frequency converges to 0

$$\lim_{|S|, |V| \rightarrow \infty} \mathbb{E}_w[F_S(w)] = 0$$

because there are more and more infrequent words

- However, the relative frequency itself stabilizes, assuming  $N = |S|$  and  $|V|$  grow independently

$$\log(F_r)_V + \log N \approx H_N - B_N \log \left( \frac{r}{|V|} \right)$$

–  $r$  is the rank of the word

- $F_r$  is the relative frequency of the  $r$ th ranked word
- $H_N \rightarrow 0$  because lowest rank word should occur with relative frequency  $1/N$ ; they are denoted *hapax legomenon*, and are often typos
- When  $B_N \rightarrow B \neq 0$ , then the corpus is said to be **Zipfian**

### Zipf's Law

- **Zipf's law:** the frequency of a word is inversely proportional to its rank
- People imitate relative frequencies of words they hear
- People innovate new words with small, constant probability
- Frequency correlates with several other properties of words
  - Age: frequent words are old
  - Polysemy: frequent words often have many meanings
  - Length: frequent words are short
- Many words occur very infrequently, e.g. the most frequent 256 English word types account for 50% of English tokens

### 3 Language Modelling

#### Statistical Modelling

- Can model language as a die, where each face is a word
- The words do not occur with equal probability
- Words depend on the words that came before them
- However, we cannot consider all of the past words, because the number of possible sentences is too large, and we don't have enough data

#### N-Grams

- **N-gram:** token sequence of length  $N$
- E.g. the fragment “in this sentence is” contains the following 2-grams (bigrams):
  - (in this), (this sentence), (sentence is)
  - The next bigram must start with “is”
- Given the probabilities of N-grams, we can compute the conditional probabilities of possible subsequent words, e.g.
$$\Pr(\text{is the}) > \Pr(\text{is a}) \implies \Pr(\text{the} \mid \text{is}) > \Pr(\text{a} \mid \text{is})$$
  - Based on this, predict “the”

#### Language Model Usage

- Can *score* and *sort* sentences (based on their probability of occurrence)
- Can infer properties about natural language, i.e. syntactically correct sentences are more likely
- Can infer embedding spaces

#### Theorem 3.1 (Chain Rule).

$$\Pr(w_1, w_2, \dots, w_n) = \Pr(w_1) \Pr(w_2 \mid w_1) \cdots \Pr(w_n \mid w_1, w_2, \dots, w_{n-1})$$

#### Word Prediction

- Want to know the probability of the next word given the previous words in a sequence
- Can approximate conditional probabilities by counting occurrences in large corpora of data

$$\Pr(\text{food} \mid \text{I like Chinese}) = \frac{\Pr(\text{I like Chinese food})}{\Pr(\text{I like Chinese} \cdot)} \approx \frac{\text{Count}(\text{I like Chinese food})}{\text{Count}(\text{I like Chinese})}$$

- There are too many possible sentences
- We usually won't have enough data to compute reliable statistics for long prefixes (i.e. numerator and denominator are likely both 0)

#### Markov Assumptions

1. *Limited extent:* assume each observation's dependence on history factors through a short recent history

$$\Pr(w_n \mid w_{1:(n-1)}) \approx \Pr(w_n \mid w_{n-L+1} : (n-1))$$

- E.g. bigrams:  $\Pr(w_n \mid w_{1:(n-1)}) \approx \Pr(w_n \mid w_{n-1})$

2. *Time invariance*: assume the dependence of the observation on the history is the same at all times

### Probability of Sentences

- The probability of a sentence  $s$  is defined as the product of the conditional probabilities of its N-grams:

$$\Pr(s) = \prod_{i=2}^t \Pr(w_i | w_{i-2}w_{i-1}) \quad (\text{trigram})$$

$$\Pr(s) = \prod_{i=1}^t \Pr(w_i | w_{i-1}) \quad (\text{bigram})$$

- N-grams are computationally cheap, and are interpretable

### Evaluating a Language Model

- Want to *quantify* the “goodness” of a model
- 2 general ways of evaluating LMs
  - Extrinsic**: in terms of some external measure, which depends on some task or application
  - Intrinsic**: in terms of properties of the LM itself

#### Extrinsic Evaluation

- The utility of a language model is often determined *in situ* (i.e. in practice)
- E.g. embed LMs  $A$  and  $B$  into a speech recognizer, run speech recognition using each model, and compare recognition rates between the two

#### Intrinsic Evaluation

- First need to estimate the probability of a corpus,  $\Pr(C)$
- This lets us adjust/estimate model parameters to maximize  $\Pr(C)$
- A good model of e.g. the Brown corpus, is one that makes Brown very likely (even if it makes other corpora less likely)
- For a corpus of sentences, we sometimes make the assumption that the sentences are conditionally independent, i.e.  $\Pr(C) = \prod_i \Pr(s_i)$

### Shannon's Method

- Can use a language model to generate random sequences, by sampling the model according to its probability
- We ought to see sequences that are similar to those we used for training
- Unigrams: we just keep picking tokens
  - Give a high probability to odd phrases, e.g.  $\Pr(\text{the the the the}) = \Pr(\text{the})^5 > \Pr(\text{the cat in the hat})$
- Bigrams and above: have *fixed* context once that context has been sampled

### Shakespeare as a Corpus

- 884647 tokens
- Vocabulary size  $V = 29066$

- About 300000 bigram types are produced, out of  $V^2 \approx 845M$  possible bigrams
- 99.96% of possible bigrams were never seen
- If we use a bigram model to determine whether a play is Shakespearean, then whenever the play uses one of those 99.96% of bigrams, the model will assign it a probability of 0, which is bad

### Zero Probability of N-grams

- Some N-grams are just really rare
- If we had more data, perhaps we'd see them
- Can use smoothing, which alters the distribution to be more uniform
  - Move probability mass from “the rich” towards “the poor”

### Add-1 Smoothing

- Add 1 to the count of every N-gram
- However, the probability of some N-grams will be drastically changed, e.g.  $\Pr(\text{to} \mid \text{want})$  went from 0.66 to 0.26
- Results are not great in extrinsic evaluations
- Sometimes  $\sim 90\%$  of the probability mass is spread across unseen events

### Add- $\delta$ Smoothing

- Add  $\delta < 1$  to be less generous
- Choice of  $\delta$  is ad-hoc
- However, by Zipf’s law, unseen words should behave more like *hapax legomena* and frequent words should behave like other frequent words
  - If I keep reading from a corpus, by the time I see a new word like “zasdf”, I will have seen “the” a lot more than once
  - But when we are compensating to the probabilities, we compensate equally to “the” and “zasdf”

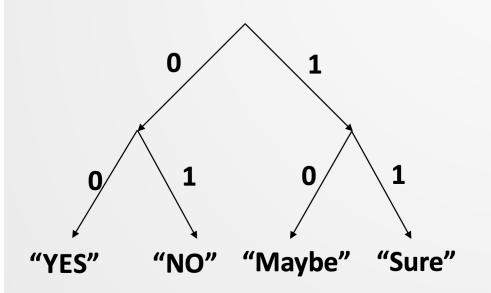
### Good Turing Smoothing

- Estimate the probability of events that occur  $n$  times by some function of the probability of events that occur  $n + 1$  times
- However, when  $\Pr(\text{UofT genius}) = \Pr(\text{UofT brainbox}) = 0$ , and then we smooth bigrams using Good-Turing, we get  $\Pr(\text{genius} \mid \text{UofT}) = \Pr(\text{brainbox} \mid \text{UofT}) > 0$
- But we should expect  $\Pr(\text{genius} \mid \text{UofT}) > \Pr(\text{brainbox} \mid \text{UofT})$  because “genius” is more common than “brainbox”

## 4 Entropy and Decisions

Information

- Assume outcomes are binary
- The minimum, average number of bits needed to encode any outcome is 1 (the bit can be 0 or 1)
- For 4 equiprobable words, we need 2 bits



- In general,  $S(x) = \log_2 \frac{1}{\Pr(x)} = -\log_2 \Pr(x)$
- Information is additive

$$S(x_1 \cdots x_k) = S(x_1) + \cdots + S(x_k)$$

Events With Unequal Information

- Events are not always equally likely
- Surprisal depends on the event, i.e. we are more surprised on unlikely events
- However, we expect to be less surprised on average, because we are fairly sure about the likely events

Entropy

- **Entropy:** the average uncertainty/information/surprisal of a (discrete) random variable  $X$

$$H(X) = \underbrace{\sum_x \Pr(x)}_{\text{Expectation over } X} \log_2 \frac{1}{\Pr(x)}$$

- This is a lower bound on the average number of bits necessary to encode  $X$
- When the probabilities are skewed, there is *less* average uncertainty
  - Flatter distributions  $\implies$  higher entropy  $\implies$  hard to predict
  - Peaky distributions  $\implies$  lower entropy  $\implies$  easy to predict
- Given  $V$  choices, the maximum entropy is  $\log_2 V$ , attained when all choices are equally likely
- The minimum entropy is 0, attained when only one choice is possible
- Can use entropy to measure how predictable language is
- Assume that a LM  $Q$  infinitely generates one word after other:  $X = X_1, X_2, \dots$ 
  - Cannot use entropy because it becomes infinity

- The per-word **entropy rate** is

$$H_{rate}(X) = \lim_{n \rightarrow \infty} \frac{1}{n} H(X_1, \dots, X_n) \leq \log_2 V \quad (*)$$

Joint Entropy

- **Joint entropy:** the *average* amount of information needed to specify multiple variables simultaneously

$$H(X, Y) = \sum_x \sum_y \Pr(x, y) \log_2 \frac{1}{\Pr(x, y)}$$

- Joint entropy is lower than the sum of the individual entropies
  - Information is shared among variables, there are dependencies

Conditional Entropy

- **Conditional entropy:** the *average* amount of information needed to specify one variable given that we know the other

$$H(Y | X) = \sum_{x \in X} \Pr(x) H(Y | X = x)$$

- This is the expectation of  $H(Y | X)$  w.r.t.  $X$

Mutual Information

- **Mutual information:** the *average* amount of information shared between variables

$$I(X; Y) = H(X) - H(X | Y) = H(Y) - H(Y | X) = \sum_{x,y} \Pr(x, y) \log_2 \frac{\Pr(x, y)}{\Pr(x) \Pr(y)}$$

- This is the amount of uncertainty removed in variable  $X$  if we know  $Y$
- If  $X$  and  $Y$  are independent, then there is no mutual information, i.e.  $I(X; Y) = 0$
- $I(X, Y) = H(X) + H(Y) - I(X; Y)$

Entropy in Language

- From (\*), we need to compute  $H(X_1, \dots, X_n)$
- We don't know how to compute  $\Pr(\dots)$
- We can approximate the log terms with our trained LM,  $Q$

Cross-Entropy

- **Cross-entropy** measures the uncertainty of a distribution  $Q$  of samples drawn from  $P$

$$H(X; Q) = \sum_x P(x) \log_2 \frac{1}{Q(x)}$$

- As  $Q$  nears  $P$ , cross-entropy nears entropy
- We can evaluate  $Q$  but not  $P$
- Corpus  $c = x_1, \dots, x_N$  is drawn from  $P$

- Let  $s_1, \dots, s_M$  be  $c$ 's sentences where  $\sum_m |s_m| = N$

$$\begin{aligned}
H_{rate}(X) &\approx \frac{1}{N} H(X_1, \dots, X_N) && \text{Large } N \text{ approximation} \\
&\approx \frac{1}{N} H(X_1, \dots, X_n; Q) && Q \approx P \text{ approximation} \\
&\approx \frac{1}{N} \log_2 \frac{1}{Q(c)} && (**) \\
&\approx \frac{1}{N} \sum_{m=1}^M \log_2 \frac{1}{Q(s_m)}
\end{aligned}$$

- $(**)$  follows from time invariance and ergodicity (i.e. when  $N$  large, the samples can represent the population)
- The last term is **negative log-likelihood (NLL)**
- NLL approaches  $H_{rate}$  as  $N \rightarrow \infty$
- This works well if and only if  $P$  and  $Q$  are “close”

Klback-Leibler Divergence

- **KL-divegence:** the *average log difference* between the distributions  $P$  and  $Q$ , relative to  $Q$ 
  - Also known as **relative entropy**
  - Assume  $0 \log 0 = 0$

$$D_{KL}(P \parallel Q) = \sum_x P(x) \log_2 \frac{P(x)}{Q(x)}$$

- Behaves like a distance
  - $D_{KL}(P \parallel Q) \geq 0$  for all  $P, Q$
  - $D_{KL}(P \parallel Q) = 0$  iff  $P$  and  $Q$  are identical
- But it's not symmetric, i.e.  $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$
- Can relate KL divergence to cross-entropy

$$\begin{aligned}
D_{KL}(P \parallel Q) &= \sum_x P(x) \log_2 \frac{1}{Q(x)} - \sum_x P(x) \log_2 \frac{1}{P(x)} \\
&= H(X; Q) - H(X) \\
&\geq 0
\end{aligned}$$

- This means

$$\begin{aligned}
H_{rate}(X) &\approx \frac{1}{N} H(X_1, \dots, X_N) \\
&\leq \frac{1}{N} H(X_1, \dots, X_N; Q) \\
&\approx NLL(c; Q)
\end{aligned}$$

- Therefore the NLL is an *approximate upper bound* on the entropy rate

Perplexity

- Exponentiated NLL

$$PP(c; Q) = 2^{\frac{1}{N} \sum_{m=1}^M \log_2 \frac{1}{Q(s_m)}} = \left( \prod_{m=1}^M \frac{1}{Q(s_m)} \right)^{\frac{1}{N}}$$

- A uniform  $Q$  over a vocabulary of size  $V$  gives  $PP(c; Q) = V$ 
  - Can be understood as “effective vocabulary size”
- If a LM  $Q$  has lower PP than  $Q'$  (for large  $N$ ), then
  - $Q$  better predicts  $c$
  - $D_{KL}(P || Q) < D_{KL}(P || Q')$
  - $PP(c; Q)$  is a tighter bound on  $2^{H_{rate}(X)}$

### Statistical Test

1. *State* a hypothesis (and choose a test); decide on the null hypothesis  $H_0$
2. *Compute* some test statistics and associated  $p$ -value (such as the  $t$ -statistic)
3. *Reject*  $H_0$  if  $p \leq \alpha$ , otherwise *do not reject*
  - Significance level  $\alpha$  is usually  $\leq 0.05$
  - If we can reject  $H_0$ , then the result is *significant*

### Null Hypothesis and $p$ -Value

- **Null hypothesis**  $H_0$  states that “there is no effect”
  - The negation of what we hope for
- We want to *cast doubt* on the plausibility of  $H_0$ 
  - “It is very unlikely that this measurement would be observed randomly under the  $H_0$ ”
- The  **$p$ -value** is the probability that the measured effect occurs under  $H_0$  by chance

### Parametric Assumptions

- **Parametric tests** make assumptions about the parameter and distribution of RVs (e.g. normally distributed with some fixed variance)
- If such assumptions are untrue,  $H_0$  could be rejected for spurious reasons
- For smaller  $N$ , it must pass tests of normality
- If non-normal, then we must use non-parametric tests, which are less powerful, i.e.  $p$ -values are higher

### Multiple Comparisons

- The more tests we conduct with a statistical test, the more likely we are to accidentally find spurious significance
- If we flip 10 coins 1000 times, sometime we will see 10/10 heads

### Effect Size

- Just because an effect is reliably measured doesn't make it important
- A difference  $\mu_1 = 1$  and  $\mu_2 = 1.00000001$  can be statistically significant
- But do we really care?

## 5 Features

### Features

- **Feature:** a measurable *variable* that is (at least should be) *distinctive* of something we want to model
- We often choose features to *classify* something
- Features are not usually nameable in neural networks
- Often need various, heterogeneous features to adequately model something
- E.g. Alzheimer's disease
  - Excessive pauses  $\mapsto$  proportion of utterance spent in silence
  - Excessive word type repetition  $\mapsto$  entropy of word type usage
  - Simplistic or short sentences  $\mapsto$  number of word tokens in a sentence

### Sentiment Analysis

- Involve detecting
  - Stress or frustration in a conversation
  - Interest, confusion, or preferences (useful to marketers)
  - Deceit, e.g. “Let’s watch Netflix and chill” (actually means ???)
- Complicating factors include sarcasm, and implicitness
- Useful features for sentiment analyzers include trigrams and first-person pronouns

### Preprocessing

- Preparing data to make feature extraction easier or more valid
- E.g. “example,” should be counted as 2 tokens, not 1

### Parts-of-Speech (PoS)

- Words are grouped according to their *distribution* in building grammatical sentences
- **Part-of-speech:** lexical category or morphological class
- Examples
  - Noun, verb, adjective, adverb
  - Preposition (e.g. around, over, under)
  - Pronoun (e.g. I, we, they)
  - Determiner (e.g. the, an, both, either)
  - Conjunction (e.g. and, or, although)

### Content Categories

- Some POSs convey content labels more than function/linguistic structure
- Usually nouns, verbs, adjectives, adverbs
- New content words are continually added, e.g. an app, to google
- Some archaic content words go extinct, e.g. fumigate, frenigerent

## Function Categories

- Some POSs are “glue” that hold others together
- E.g. prepositions, determiners, conjunctions
- Functional POS usually cover a small and fixed number of wordtypes (i.e. a “closed class”)
- Semantics depend on contentful words with which they’re used, e.g. I’m *on* time vs. I’m *on* the bus

## Grammatical Features

- Several grammatical features that can be associated with words, e.g. case, person, number, gender
- These features can restrict other words in a sentence, e.g. usage of “I” restricts the usage of “be” to “am”

## Features of Nouns

- **Proper noun:** named things, e.g. “they’ve killed *Bill*”
- **Common noun:** unnamed things, e.g. “they’ve killed the *bill*”
- **Mass noun:** divisible and uncountable, e.g. *butter* split in two gives two piles of *butter*
- **Count noun:** indivisible and countable, e.g. a *pig* split in two does not give two pigs

## Agreement

- Part-of-speech should match (i.e. agree) in certain ways
- Articles have to agree with the number of their noun
  - E.g. these pretzels are making me thirsty
- Verbs have to agree with their subject (in English)
  - E.g. the dogs eat the gravy (number agreement)
  - E.g. yesterday, all my trouble seemed so far away (tense agreement)

## PoS Tagging

- **Tagging:** the process of assigning a part-of-speech to each word in a sequence
- E.g. using the Penn Treebank tagset
  - The/DT nurse/NN put/VBD the/DT sick/JJ patient/NN to/IN sleep/NN
- Word types can have many parts-of-speech
  - The back/JJ door (adjective)
  - On its back/NN (noun)
  - Win the voters back/RB (adverb)
  - Promise to back/VB you in a fight (verb)
- Need to determine the appropriate tag for a given token in its context
- Tagging is the first step towards many practical purposes
  - Speech synthesis: how to pronounce text
    - \* E.g. I’m conTENT/JJ vs. the CONtent/NN

- \* E.g. I obJECT/VBP vs. the OBject/NN
- Information extraction: help to find names and relations
- Machine translation: help to identify phrase boundaries
- Explainability

### Tagging as Classification

- We have access to a *sequence of observations* and are expected to decide on the best assignment of a *hidden variable*, the PoS

Hidden variable				NN		
				VB		
		VBN		JJ		NN
	PRP	VBD	TO	RB	DT	VB
Observation	she	promised	to	back	the	bill

- We can determine the most likely tag sequence  $t_{1:n}$  by

$$\begin{aligned}
 \arg \max_{t_{1:n}} \Pr(t_{1:n} \mid w_{1:n}) &= \arg \max_{t_{1:n}} \frac{\Pr(w_{1:n} \mid t_{1:n}) \Pr(t_{1:n})}{\Pr(w_{1:n})} && \text{By Bayes' Rule} \\
 &= \arg \max_{t_{1:n}} \Pr(w_{1:n} \mid t_{1:n}) \Pr(t_{1:n}) \\
 &\approx \arg \max_{t_{1:n}} \prod_i^n \Pr(w_i \mid t_i) \Pr(t_i \mid t_{i-1}) && \text{Assuming independence and Markov}
 \end{aligned}$$

- Hidden Markov model
- $\Pr(w_i \mid t_i)$  is the word likelihood probability
  - Compute by counting in a corpus that has already been tagged

$$\Pr(w_i \mid t_i) = \frac{\text{Count}(w_i \text{ tagged as } t_i)}{\text{Count}(t_i)}$$

- $\Pr(t_i \mid t_{i-1})$  is the tag-transition probability

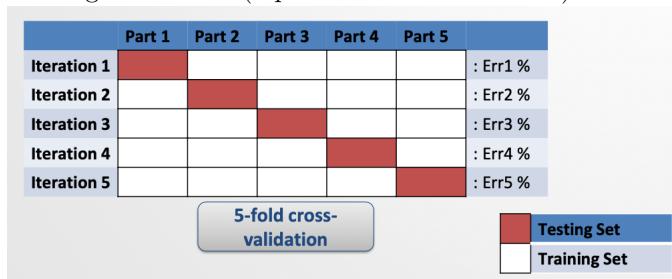
## 6 Classification

General Process

1. Gather a big and relevant training corpus
2. Learn our parameters (i.e. probabilities) from that corpus to build our model
  - Training data consist of 80% to 90% of the available data
  - Often, some subset of this is used as a validation set
3. Once that model is fixed, we use those probabilities to evaluate testing data
  - Testing data are *not* used for training but often come from the same corpus
  - Sometimes, it's important to *partition* speakers/writers so they don't appear in both training and testing

K-Fold Cross Validation

- **K-fold cross-validation:** splitting all data into  $K$  partitions and iteratively testing on each after training on the rest (report mean and variances)

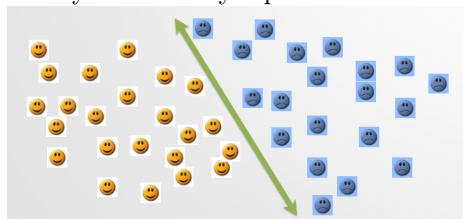


Types of Classifiers

- **Generative** classifiers model the data
  - Parameters set to maximize likelihood of training data
  - We can generate new observations from these
  - E.g. hidden Markov models
- **Discriminative** classifiers emphasize *class boundaries*
  - Parameters set to minimize error on training data
  - E.g. SVM, decision trees

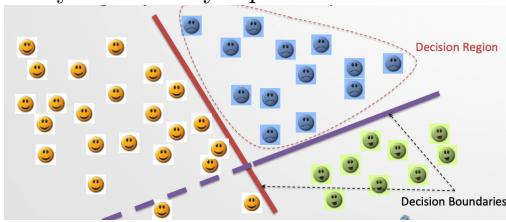
Class Boundaries

- Binary and linearly separable



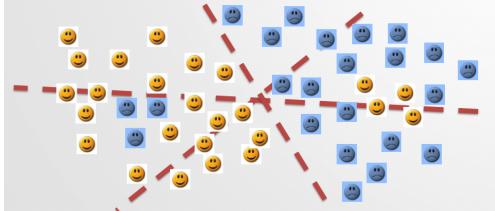
- Extends to higher dimensions, where line becomes hyperplane

- N-ary and linearly separable



– Random guessing gives  $1/N$  accuracy

- Class holes

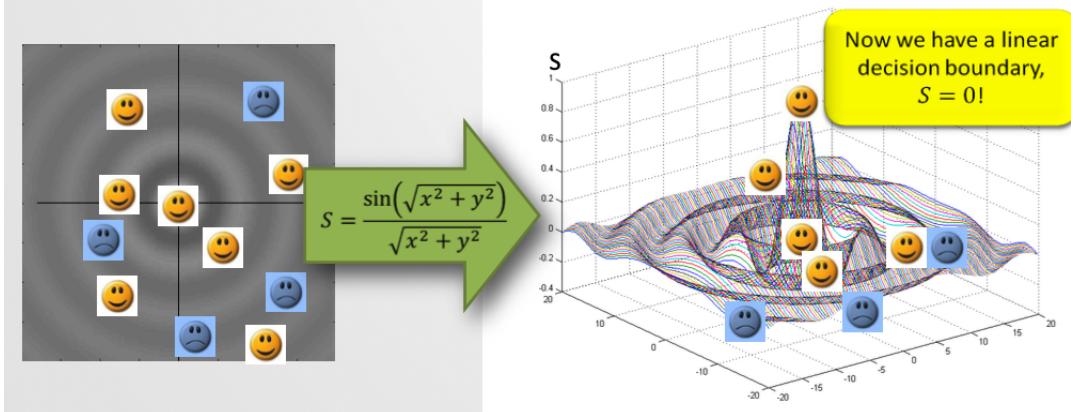


– Impossible to draw any lines through the data to separate the classes

– Could be either real phenomena or random noise

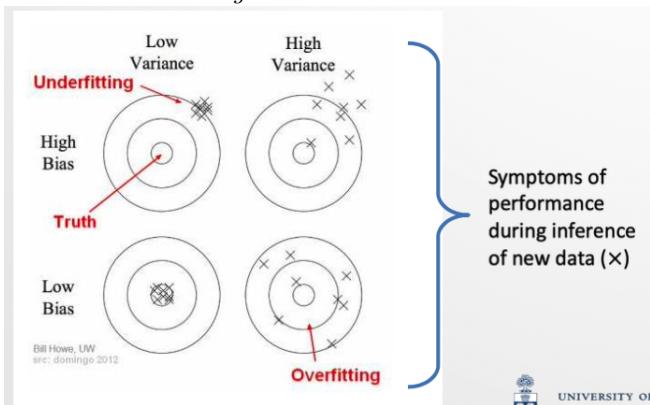
### The Kernel Trick

- Can linearize a non-linear case by moving the data into a higher dimension with a *kernel function*



### Capacity and Over/Underfitting

- Our models should *generalize* to unseen data



## ID3

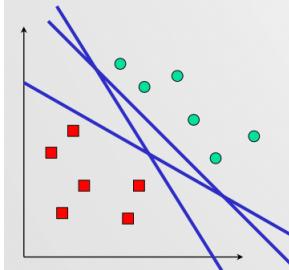
- **ID3 (iterative dichotomiser 3)** is an algorithm to produce decision trees from data
- At each step, we are removing the maximum amount of entropy possible
  - Tend to build *short trees*
- Trains on the whole training set
- Does not have issues with random initialization
- Can *overfit* to training data
- Only 1 attribute is used at a time to make decisions
- Difficult to use continuous data, since many trees need to be generated to see where to break the continuum

## Random Forests

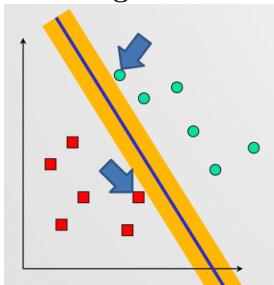
- **Random forests:** *ensemble* classifiers that produce  $K$  decision trees, and output the *mode* class of those trees
- Can support continuous features, and non-binary decisions
- Can perform cross-validation
- The component trees in a random forest must differ
- Sometimes, decision trees are pruned randomly
- Usually, different trees accept different *subset of features*

## Support Vector Machines (SVMs)

- In binary linear classification, two classes are assumed to be separable by a line (or plane)
- Many possible separating planes might exist



- The **margin** is the width by which the boundary could be increased before it hits a training datum



- The **maximum margin linear classifier** is the linear classifier with the maximum margin
- The **support vectors** are the data points against which the margin is pressed
- The bigger the margin, the less sensitive the boundary is to error
- The width of the margin,  $M$ , can be computed by the angle, distance to the origin (of a point  $x$  in the hyperplane) (like a translated subspace), and the distance to the closest points
  - Given an initial guess of the angle and displacement of  $x$ , we can compute whether all data is correctly classified, and the width of the margin
- Update our guess by *quadratic programming*, which is semi-analytic (i.e. utilize both analytical and numerical optimization methods)
- The maximum margin helps SVMs generalize to situations where it's impossible to linearly separate the data
  - Can introduce a parameter that allows us to measure the distance of all data not in their correct “zones”
  - We simultaneously *maximize the margin* while *minimizing the misclassification error*
- Can also use the kernel trick when the data is non-linearly separable
- Binary SVM classifiers can be combined to simulate a multi-category classifier

Naive Bayes and Softmax

- Given features  $\mathbf{x}$ , **Naive Bayes** simply chooses the class  $c \in C$  that maximizes  $\Pr(c | \mathbf{x})$

$$\Pr(c | \mathbf{x}) = \frac{\Pr(\mathbf{x} | c) \Pr(c)}{\Pr(\mathbf{x})}$$

- Equivalently

$$\text{Posterior for class} = \frac{\text{Probability of feature given class} \times \text{Prior for class}}{\text{Probability off feature}}$$

- **Independence assumption:** the features  $x_i$  are conditionally independent given the class  $c$

$$\Pr(c, x_1, \dots, x_D) = \Pr(c) \Pr(x_1 | c) \cdots \Pr(x_D | c)$$

- Denote the prior probability of class  $\Pr(c = 1) = \pi$
- Denote the conditional probability of feature given class  $\Pr(x_j = 1 | c) = \theta_{jc}$
- Assuming  $x \in \mathbb{R}^d$ , learning a linear decision boundary is equivalent to learning  $W \in \mathbb{R}^{C \times d}$

$$\forall c \in C, \mathbf{f}_c = W[c, \dots] \mathbf{x} = \sum_{i=1}^d W[c, i] \mathbf{x}[i]$$

- Since  $\mathbf{f}_c$  can be negative and we want a probability ( $\in [0, 1]$ ), we raise it with an exponent

$$\Pr(y | x) = \frac{\exp(\mathbf{f}_y)}{\sum_{c1}^C \exp(\mathbf{f}_c)}$$

- Very cheap
- Analysis easily extends to probability distributions other than Bernoulli
- Not very accurate due to its “naive” independence assumption

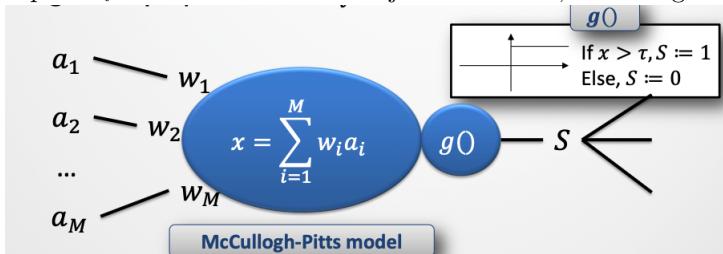
## 7 Neural Models of Language

### Artificial Neural Networks

- **Artificial neural networks (ANNs)** were inspired from neurobiology
- Each unit has many inputs (dendrites), one output (axon)
- The nucleus fires (i.e. send an electric signal along the axon) given input from other neurons
- “Learning” occurs at the synapses that connect neurons, either by amplifying or attenuating signals

### Perceptron

- Each neuron calculates a *weighted sum* of its inputs and compares this to a threshold  $\tau$ 
  - If the sum exceeds the threshold, the neuron fires
  - Inputs  $a_i$  are activations from adjacent neurons, each weighted by a parameter  $w_i$



- Perceptron output is determined by **activation functions**  $g()$ , which can be nonlinear functions of weighted input
  - Popular activation functions include *tanh* and *sigmoid*
  - The sigmoid is defined as  $\sigma(x) = 1/(1 + e^{px})$  and its derivative is easily computable  $\sigma' = \sigma(1 - \sigma)$
- Rectified linear units (ReLUs)
  - $g(x) = \max(0, x)$
  - Biologically plausible and computationally efficient
  - A smooth approximation is the *softplus*  $g(x) = \log(1 + e^x)$ , whose derivative is  $1/(1 + e^{-x})$
- To learn a perceptron, weights are adjusted in proportion to the error (i.e. difference between the desired and actual output)
- Some simple logical functions cannot be learned by a single perceptron, e.g. XOR, which is not linearly separable

### Multi-Layer Perceptron

- Complex functions can be represented by layers of perceptron
- Inputs are passed to the **input layer**
- Activations are propagated through **hidden layers** to the **output layer**
- MLPs are robust to noise

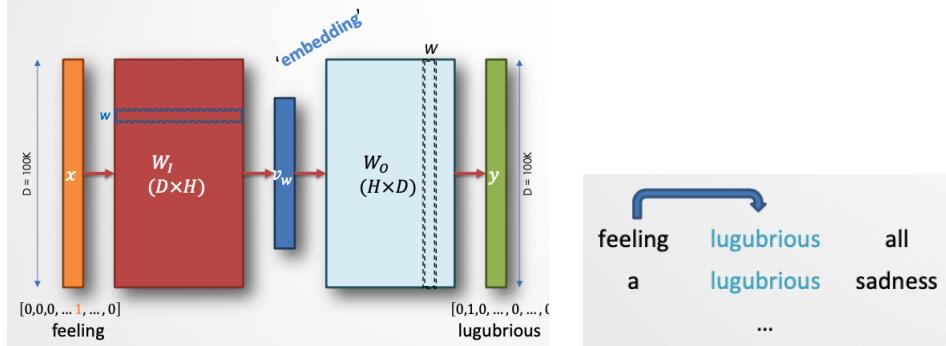
### Words

- Given a corpus with  $D$  unique words (e.g.  $D = 100K$ )

- **One-hot representation:** uniquely assign each word with an index in  $D$ -dimensional vectors
- **Word-feature representation:** assign features to each index in a much denser vector, i.e. dimension  $d \ll D$

Continuous Bag of Words

- 1 Word Context



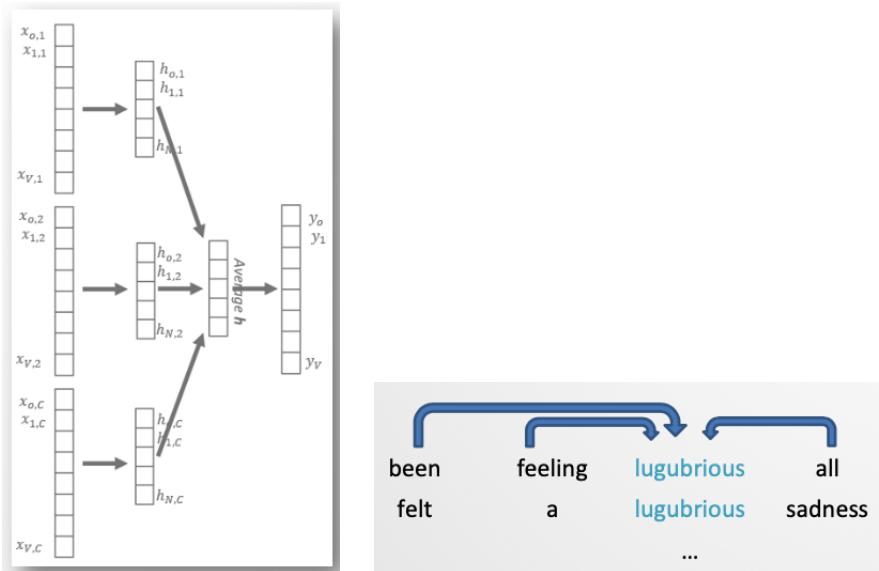
- The input is the context word, and the output is the target word
- We have two vector representations of each word:

$$v_w = x^\top W_I \quad \text{wth row of } W_I$$

$$V_w = W_O^\top y \quad \text{wth column of } W_O$$

- $C$  Words Context

- Each input word will produce one of  $C$  embeddings
- We need to add an intermediate layer, which usually averages the embeddings



Skip-Grams

- **Skip-gram:** predict context words given the current word
- Inverse of CBOW

- Works well with small amounts of training data
- Slower than CBOW, upon frequent words it has lower accuracy than CBOW

Learning

- Given  $H$ -dimensional embeddings and  $V$  word types, our parameters  $\theta$  are

$$\theta = \begin{bmatrix} v_a \\ v_{aardvark} \\ \vdots \\ v_{zymurgy} \\ V_a \\ V_{aardvark} \\ \vdots \\ V_{zymurgy} \end{bmatrix} \in \mathbb{R}^{2V \times H}$$

- Can use gradient descent
- Given  $T$  tokens of training data, we optimize

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T \sum_{-c < j < c, j \neq 0} \log \Pr(w_{t+j} | w_t)$$

– Likelihood of context given current word is

$$\Pr(w_o | w_i) = \frac{\exp(V_{w_o}^\top v_{w_i})}{\sum_{w=1}^W \exp(V_w^\top v_{w_i})}$$

- Want to update vectors  $V_{w_{t+j}}$  then  $v_{w_t}$  within  $\theta$

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$$

- Derivative of log softmax

$$\begin{aligned} \frac{\partial}{\partial v_{w_t}} \log \Pr(w_{t+j} | w_t) &= \frac{\partial}{\partial v_{w_t}} \log \frac{\exp(V_{w_{t+j}}^\top v_{w_t})}{\sum_{w=1}^W \exp(V_w^\top v_{w_t})} \\ &= \frac{\partial}{\partial v_{w_t}} \left[ \log \exp(V_{w_{t+j}}^\top v_{w_t}) - \log \sum_{w=1}^W \exp(V_w^\top v_{w_t}) \right] \\ &= V_{w_{t+j}} - \frac{\partial}{\partial v_{w_t}} \log \sum_{w=1}^W \exp(V_w^\top v_{w_t}) \\ &= V_{w_{t+j}} - \frac{1}{\sum_{w=1}^W \exp(V_w^\top v_{w_t})} \sum_{w=1}^W \frac{\partial}{\partial v_{w_t}} \exp(V_w^\top v_{w_t}) \quad \text{By Chain Rule} \\ &= V_{w_{t+j}} - \frac{1}{\sum_{w=1}^W \exp(V_w^\top v_{w_t})} \cdot \sum_{w=1}^W \exp(V_w^\top v_{w_t}) V_w \\ &= V_{w_{t+j}} - \sum_{w=1}^W \Pr(w | w_t) V_w \end{aligned}$$

## Using Word Representations

- When we compare the vector of two words in latent space, we can use cosine similarity

$$\cos(u, v) = \frac{u \cdot v}{\|u\| \cdot \|v\|}$$

## Skip-Grams with Negative Sampling

- The default process requires update  $2H \times D$  weights,  $D$  is vocabulary size
- For the observed (true) pair (lugubrious, sadness), only the output neuron for sadness should be 1, and all  $D - 1$  others should be 0
- When computing probability, the denominator term sums over all  $D$  words, which is expensive
- We want to maximize the association of observed (positive) contexts: {sad, feeling, tired}, and minimize the association of “hallucinated” contexts: {happy, roof, truth}
- Choose a small number  $k$  of “negative” words, and just update the weights for the “positive” words plus the  $k$  “negative” words

$$J(\theta) = \log \sigma(v_o^\top v_c) + \sum_{i=1}^k \mathbb{E}_{j \sim P(w)} [\log \sigma(-v_i^\top v_c)]$$

- $v_o$  is the output word,  $v_c$  is the context word
- $\sigma$  is the sigmoid function
- We draw  $k$  negative samples from  $P(w)$ , the unigram distribution

- Can choose the top  $k$  words by *modified unigram probability*

$$P^*(w_{t+1}) = \frac{C(w_{t+1})^{3/4}}{\sum_w C(w)^{3/4}}$$

- $C$  is the count of each word in the corpus

## GloVe: Global Vectors

- Instead of predicting words at particular positions, look at the *co-occurrence matrix* within the corpus

$$X = \begin{matrix} & \begin{matrix} I & like & enjoy & deep & learning & NLP & flying & . \end{matrix} \\ \begin{matrix} I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{matrix} & \left[ \begin{matrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{matrix} \right] \end{matrix}$$

- Word  $w_i$  occurs  $X_{ij} = X_{ji}$  times with word  $w_j$ , within some context window (e.g. 10 words, 1 sentence, etc.)
- To populate the co-occurrence matrix, we need to scan the entire corpus, i.e.  $P_{ij} = \Pr(w_j | w_i) = X_{ij}/X_i$

- Can take the ratio of the probabilities

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k ice)$	$1.9 \times 10^{-4}$	$6.6 \times 10^{-5}$	$3.0 \times 10^{-3}$	$1.7 \times 10^{-5}$
$P(k steam)$	$2.2 \times 10^{-5}$	$7.8 \times 10^{-4}$	$2.2 \times 10^{-3}$	$1.8 \times 10^{-5}$
$P(k ice)/P(k steam)$	8.9	$8.5 \times 10^{-2}$	1.36	0.96

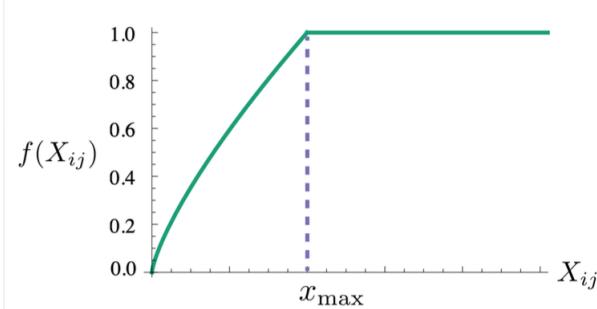
- Large values (e.g. solid) correlates with properties specific to ice
- Small values (e.g. gas) correlates with properties specific to steam
- The ratio from non-discriminative words (e.g. water, fashion) cancels to 1

- Having the co-occurrence matrix, we can create an embedding for each words by minimizing

$$J = \sum_{i,j}^V f(X_{i,j}) \left( v_{w_i}^\top v_{w_j} + b_i + \tilde{b}_j - \log X_{i,j} \right)^2$$

- $b_i$  and  $\tilde{b}_j$  are input and output bias terms associated with  $w_i$  and  $w_j$
- $V$  is vocab size
- $f$  is weighting function, where

$$f(x) = \begin{cases} (x/x_{\max})^\alpha, & \text{if } x < x_{\max} \\ 1, & \text{elsewise} \end{cases}$$



- This weights the input word within the context
  - \* When the word is rare in the context, then it has low weight
  - \* When the word is relatively common in the context, then it has full (1) weight

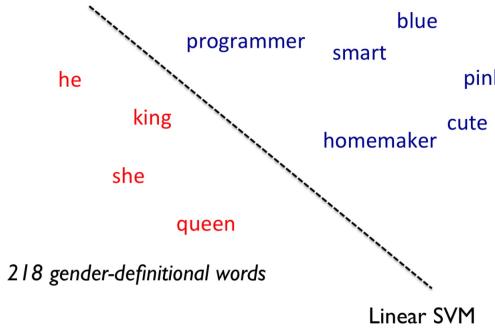
## Linguistic Regularities in Vector Space

- Encodes analogies, hypernymy (set-element relationship in words), and semantic
- E.g.  $v_{Paris} - v_{France} + v_{Italy} \approx v_{Rome}$
- E.g.  $v_{bigger} - v_{big} + v_{cold} \approx v_{colder}$
- E.g.  $v_{Cu} - v_{copper} + v_{gold} \approx v_{Au}$

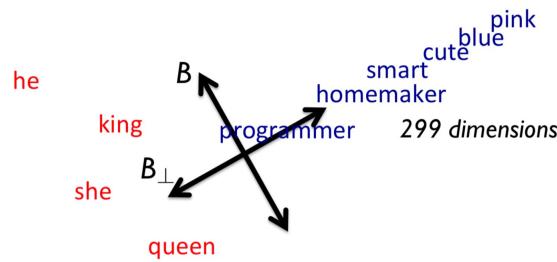
## Bias

- E.g.  $v_{programmer} - v_{man} + v_{woman} \approx v_{homemaker}$
- Solution

1. Hand-pick words  $S_0$  that are “gender definitional”, and leave the rest of the words  $V \setminus S_0$  as “neutral”



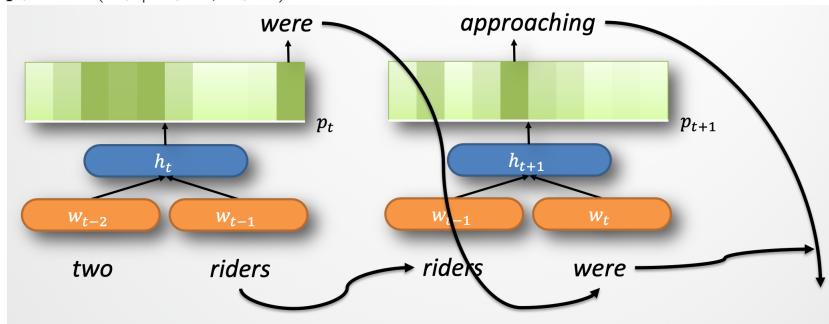
2. Project away gender subspace from gender-neutral words, i.e.  $w := w - w \cdot B$  for  $w \in N$ , where  $B$  is the gender subspace



- This works but as a side effect the model would be weaker in generating non-stereotypic analogies

### Trigram Models

- $p_t \sim \Pr(w_t | w_{t-2}, w_{t-1})$



- Training

1. Randomly choose a batch (e.g. 10K consecutive words)
2. Propagate words through the current model
3. Obtain word likelihoods (loss)
4. Backpropagate the loss
5. Gradient step to update model
6. Go to (1)

- Given corpus  $C$  and model  $M$ , we minimize

$$\begin{aligned}
H(C; M) &= -\frac{\log_2 P_M(C)}{\|C\|} \\
&= -\frac{\log_2 \prod_{t=0}^T \Pr(w_t)}{\|C\|} \\
&= -\frac{\sum_{t=0}^T \log_2 \Pr(w_t)}{\|C\|} \\
&= -\frac{\sum_{t=0}^T w_t^\top \log p_t}{\|C\|}
\end{aligned}$$

- $w_i$  is a one-hot vector,  $p_t$  is the distribution
- To evaluate, we can use **perplexity**
  - This measures how well a probability model predicts a sample, lower is better
  - Given a text corpus  $C$  of  $n$  words  $w_1, \dots, w_n$ , and a language model  $LM = \Pr(w_i | w_{1:i-1})$ ,

$$\begin{aligned}
\text{Perplexity}_C(LM) &= 2^{-\frac{1}{n} \sum_{i=1}^n \log_2 LM(w_i | w_{1:i-1})} \\
&= 2^{-\frac{\log_2 \Pr_{LM}(C)}{\|C\|}} \\
&= 2^{H(C; LM)}
\end{aligned}$$

### Computational Complexity

- Training is very expensive
- Number of parameters scale poorly with increasing context

### Hierarchical Softmax

- Group words into distinct classes,  $c$ , e.g. by frequency ( $c_1$  is top 5% words,  $c_2$  is next 5%, etc.)
- Factorize  $\Pr(w_o | w_i) = \Pr(w_o | w_i, C) = \sum \Pr(c | w_i) \Pr(w_o | w_i, c)$
- Softmax becomes

$$\Pr(w_o | w_i) = \frac{\exp(V_{w_o}^\top v_{w_i})}{\sum_{w=1}^{|V|} \exp(V_w^\top v_{w_i})} = \frac{\exp(c_j v_{w_i})}{\sum_c \exp(c v_{w_i})} \times \frac{\exp(V_{w_o}^\top v_{w_i})}{\sum_{w \in c} \exp(V_w^\top v_{w_i})}$$

- $v_w$  is the “input” vector for word  $w$
- $V_w$  is the “output” vector for word  $w$
- Instead of summing over all words in denominator, we just sum over all classes, then sum over the words in our desired class

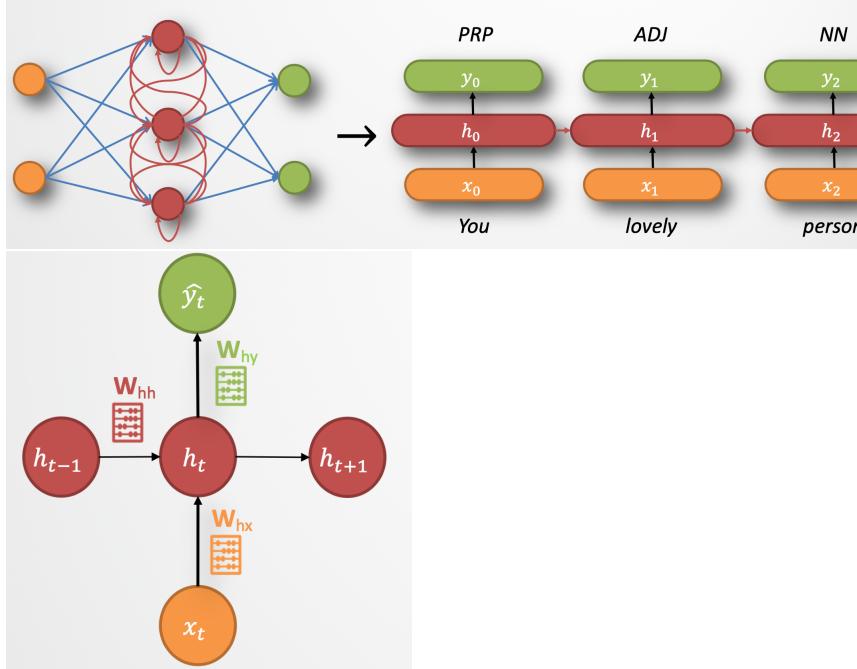
## 8 Recurrent Neural Networks

### Statistical Language Models

- Probability is conditioned on (window of)  $n$  previous words
- Requires the Markov assumption: each observation only depends on a short linear history of length  $L$
- Using too many  $n$  grams improves performance but is expensive in terms of RAM
- RNN intuition:
  - Use the *same set of weight* parameters for each word (or across all time steps)
  - Condition the neural network on *all* previous words (or time steps)
  - Memory requirement now scales with number of words

### Recurrent Neural Networks (RNNs)

- An RNN has *feedback* connections so that it “remembers” previous states, when reading a sequence
- Copies of the same network can be applied (i.e. unrolled) at each point in a time series



- $h_t = g([W_{hh}h_{t-1} + W_{hx}x_t] + c)$
- $\hat{y}_t = W_{hy}h_t + b$
- $\hat{y} \in \mathbb{R}^{|V|}$  is a probability distribution over the vocabulary
- The output  $\hat{y}_{t,j}$  is the word (index) prediction of the next word  $x_{t+1}$
- Evaluation: cross-entropy loss function

$$J^{(t)}(\theta) = - \sum_{j=1}^{|V|} \underbrace{y_{t,j}}_{\text{ground truth}} \log \underbrace{\hat{y}_{t,j}}_{\text{prediction}}$$

- Perplexity:  $2^J$  (lower is better)

- If  $|h_i| < |V|$ , then this is cheaper than trigram NN

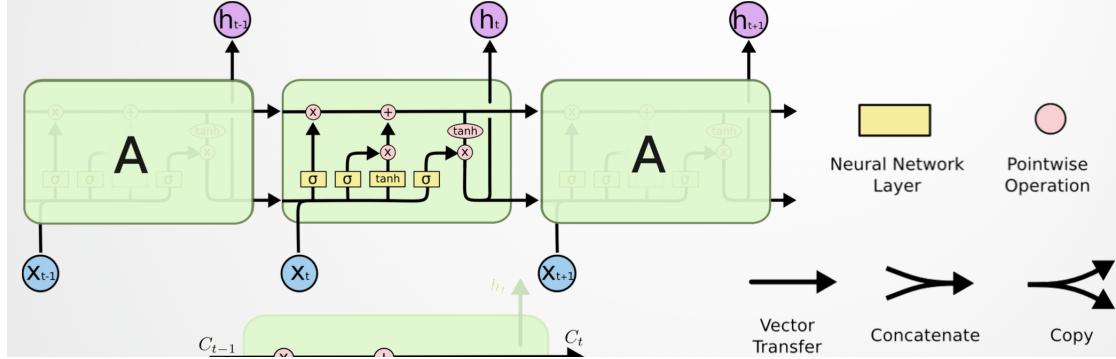
### Retrograde Amnesia

- In theory, information is maintained in  $h_i$  across arbitrary lengths of time
- However, the relevant context from e.g.  $h_1, h_2$  is likely to be overwritten by the time we reach  $h_{15}$
- This is because gradient decays quickly as one pushes it back in time

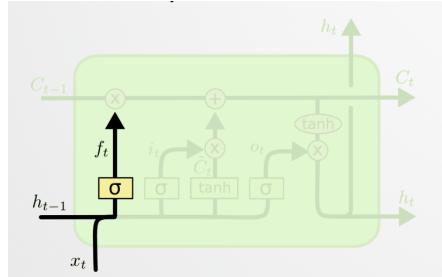
### Long Short-Term Memory (LSTM)

- Within each recurrent unit/cell:
  - Self-looping recurrence for *cell state* using vector  $C$ 
    - \* Stores the long-term information
  - Information flow regulating structures called **gates** (a sigmoid followed by a pointwise  $\times$ )
    - \* Values near 0 block information, values near 1 pass information

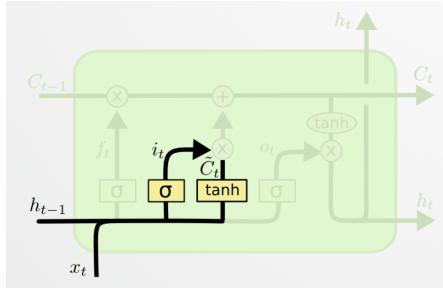
- In each cell (i.e. recurrent unit), there are 4 interacting layers



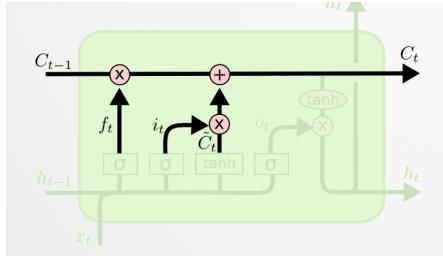
1. The **forget gate layer** compares  $h_{t-1}$  and the current input  $x_t$  to decide which elements in cell state  $C_{t-1}$  to keep and which to turn off
  - E.g. may need to remember the number (singular/plural) of the current subject in order to predict appropriately conjugated verbs
  - E.g. may decide to forget the subject when a new subject is mentioned in  $x_t$



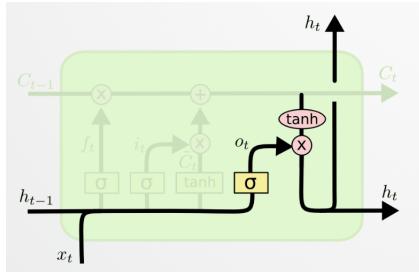
2. The **input gate layer** first uses a sigmoid layer to decide which cell units to update, and then a tanh layer to create new candidate values  $\tilde{C}_t$ 
  - E.g.  $\sigma$  can turn on the “number” units, and the tanh can push information on the current subject
  - The  $\sigma$  layer is important because we don’t want to push information on units for which we have no information



3. Updates  $C_{t-1}$  to  $C_t$  by forgetting what we want to forget, creating a mask vector of information we want to store, and write this information to the new cell state  $C_t$



4. Outputs something based on the current  $x_t$  and  $h_{t-1}$
- To get  $h_t$ , we combine the output with the cell
  - $C_t$  is the long-term memory and  $h_t$  is the short-term memory



### Variant of LSTMs

- Bidirectional LSTMs maintains 2 LSTMs, one that reads left-to-right and one that reads right-to-left
  - Idea generalizes to bidirectional RNNs
- Can add “peepholes” to allow all sigmoids to read the cell state
- Can couple the forget and input gates for joint decisioning
- Gated recurrent units (GRUs) merges the cell and hidden states

### Contextual Word Embeddings

- A given word could have multiple meanings depending on the context
- E.g. “play” could refer to the verb, or the noun as in a theatrical play

### Embeddings from Language Models (ELMo)

- Instead of a fixed embedding for each word type, ELMo considers the entire sentence before embedding each token
- Uses bidirectional LSTM trained on a specific task

- Outputs are softmax probabilities on words
- For each token, a  $L$ -layer biLM computes  $2L + 1$  representations:

$$R_k = \left\{ x_k^{LM}, \overrightarrow{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} : j = 1, \dots, L \right\} = \{ h_{k,j}^{LM} : j = 0, \dots, L \}$$

- Task specific weighting produces the final embedding for word token  $k$

$$ELMo_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^L s_j^{task} h_{k,j}^{LM}$$

- $R_k$  is the set of all  $L$  hidden layers,  $h_{k,j}$
- $s_j^{task}$  is the task's weight on the layer
- $\gamma^{task}$  is a weight on the entire task

## 9 Statistical Machine Translation

Rosetta Stone

- Dates from 196 BCE
- Contains 3 parallel texts in different languages
- Works by aligning unknown words to known words

Direct Translation

- A bilingual dictionary that aligns words
- Works only for simple cases

Difficulties in Machine Translation

- Different *morphology*  $\Rightarrow$  difficult *mappings*
  - E.g. many vs. one morphemes (atomic elements) per word
  - E.g. many vs. few features per morpheme
- Different *syntax*  $\Rightarrow$  *long-distance effects*
  - E.g. subject-verb-object vs. subject-object-verb vs. verb-subject-object
- Ambiguity makes it hard to pick one translation
  - Lexical: many-to-many word mappings
  - Syntactic: same token sequence, different structure
  - Semantic: same structure, different meaning
  - Pragmatic: different contexts  $\Rightarrow$  different interpretations

The Noisy Channel Model

- “When I look at French, I see this as English, but it has been coded in some strange symbols”
- Given a French sentence  $F$ , want to convert it to the best corresponding English sentence  $E^*$

$$E^* = \arg \max_E \Pr(E | F) = \arg \max_E \Pr(F | E) \Pr(E)$$

- $\Pr(E)$  is the *language model* (e.g.  $N$ -gram) and encodes knowledge of word order
- $\Pr(F | E)$  is a word- (or phrase-) level translation model that encodes only knowledge on an *unordered* basis
- To learn  $\Pr(F | E)$ , we can collect statistics on vast parallel texts

Alignment

- SMT makes alignments explicit, where one block of text is entirely responsible for a translated block (i.e. conditional independence)
- Let  $A$  index the pairs of aligned blocks:

$$\Pr(F | E) = \sum_A \Pr(F, A | E) = \sum_A \Pr(A | E) \prod_i \Pr(F_{A_{i,1}} | E_{A_{i,2}})$$

- However, in practice, words and phrases can be out of order, and we usually don't know how to align

- Across translations, sentences can be unaligned (e.g. to make translation syntactically correct, we may need to add/drop words)

### Sentence Alignment

- We often need to align sentences before moving forward
- Want to find  $A^* = \arg \max_A \Pr(A | F, E)$
- Two classes of methods
  1. Methods that only look at *sentence length*
  2. Methods based on *lexical matches*, or cognates

### Sentence Alignment by Length

- Idea: lengths of aligned sentences are correlated
- Assuming the paragraph alignment is known, let  $\mathcal{L}_E$  and  $\mathcal{L}_F$  be the number of characters in an English and a French sentence, respectively
- Define cost function  $Cost(\mathcal{L}_E, \mathcal{L}_F)$ , which is lowest when  $\mathcal{L}_E = c\mathcal{L}_F$  for some learned/guessed  $c$
- Define “prior” fixed cost  $C_{ij}$  of aligning  $i$  English sentences to  $j$  French sentences
- If we align  $i$  English sentences  $E_{1:i}$  with  $j$  French sentences  $F_{1:j}$ , the cost is

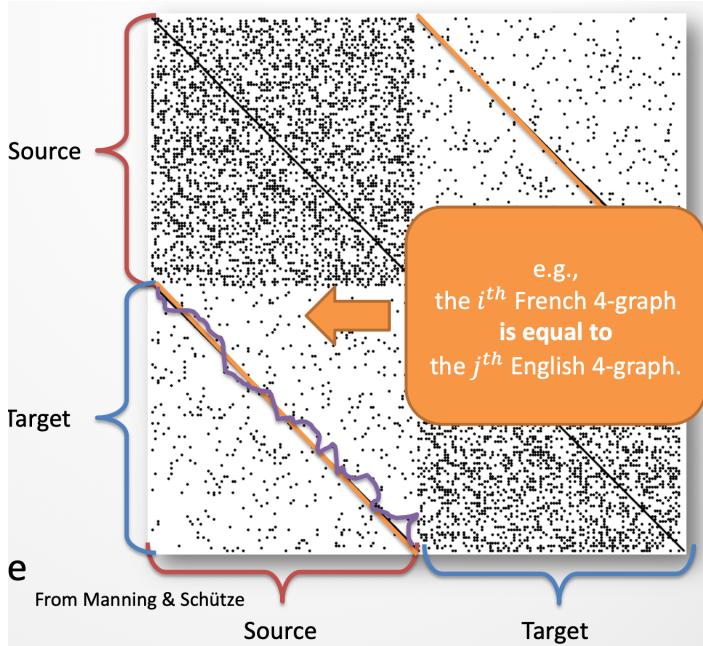
$$C_{ij} + Cost\left(\sum_{k=1}^i \mathcal{L}_{E_k}, \sum_{k=1}^j \mathcal{L}_{F_k}\right)$$

### Sentence Alignment by Cognates

- **Cognates:** words that have a common *etymological* origin
- **Etymological:** pertaining to the historical derivation of a word, e.g. *porc* → *pork*
- Idea: words that are related across languages have similar spellings
  - E.g. government/gouvernement
  - Counterexample: son (male offspring) vs. son (sound)
- Cognates can “anchor” sentence alignments between related languages
- **N-graph:** similar to  $N$ -grams, but computed at the *character level*

### Church’s Method

1. Concatenate paired texts on both axes
  - Each element on the axis is a word
2. Dot-plot: place a “dot” where the  $i$ th French and the  $j$ th English 4-graphs are equal
  - If there *exists* a 4-graph match on the two words, then draw a dot
3. Search for a short path “near” the bilingual diagonals
  - Each point along the path is considered to represent a *match* between languages
  - The relevant French and English sentences are then aligned



### Word Alignment Models

- Make a simplifying assumption that every word in  $F$  maps to 1 word in  $E$ 
  - Notice that 1 word in  $E$  can map to 0, 1, or many words in  $F$
  - $A_i = (\{i\}, \{j\}) \mapsto j$ , which means the  $i^{th}$  word in  $F$  maps to the  $j^{th}$  word in  $E$
  - This assumption makes this approach not flexible
- Can then train using expectation-maximization

### Phrase-Based Translation

- Suppose beads are pairs of non-empty, contiguous spans of words that partition  $F \times E$ , so

$$A_i = \left( \left( l_1^{(i)} : u_1^{(i)} \right), \left( l_2^{(i)} : u_2^{(i)} \right) \right)$$

- Call each span an indivisible phrase  $(F_{A_{i,1}}, E_{A_{i,2}}) \mapsto (\overline{F}_i, \overline{E}_i)$  and assume phrases sequential in  $E$ , then

$$\Pr(F, A | E) \propto \prod_i \phi(\overline{F}_i, \overline{E}_i) d(u_1^{(i-1)} - l_1^{(i)} - 1)$$

- $\phi(\overline{F}, \overline{E}) = \frac{\text{Count}(\overline{F}, \overline{E})}{\sum_{\overline{F}'} \text{Count}(\overline{F}', \overline{E})}$  is the *phrase translation probability*
- $d$  is the distortion metric/distance (e.g.  $d(x) = \alpha^{|x|}$ )
- Since  $\overline{E}_i, \overline{E}_{i+1}$  are sequential, penalizes when  $\overline{F}_i, \overline{F}_{i+1}$  are not
- E.g. count the pair  $(\overline{F}, \overline{E}) = (F_{l_1:u_1}, E_{l_2:u_2})$ 
  - At least one alignment  $A_i$  is in the box  $[l_1 : u_1] \times [l_2 : u_2]$
  - All  $A_i$  containing any word in  $[l_1 : u_1]$  or any word in  $[l_2 : u_2]$  are in the box

### Decoding with Phrases

- Want to derive  $E$  given  $F$

$$E^* = \arg \max_E \Pr(F | E) \Pr(E) \approx \arg \max_E \Pr(F, A | E) \Pr(E)$$

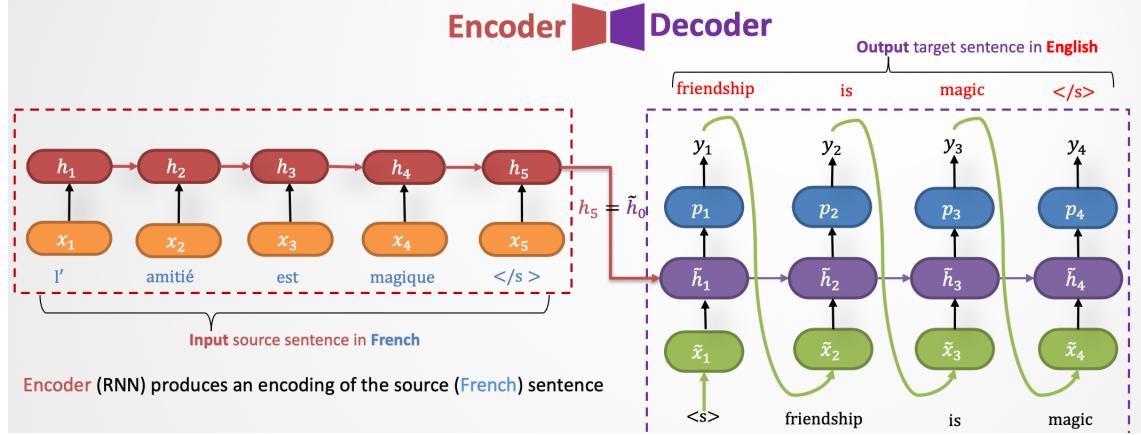
- Checking all  $E, A$  is infeasible
- Can use a **beam search**
  1. Choose partial translation  $(E', A')$  with highest score
  2. Increment that by appending bilingual phrase pairs
  3. Prune set of resulting partial translations by score

## 10 Neural Machine Translation

Neural Machine Translation (NMT)

- No explicit alignments, end-to-end training
- Outperforms SMT by a large margin
- Uses sequence-to-sequence (seq2seq) encoder/decoder architectures
  - An *encoder* produces a representation of  $F$
  - A *decoder* interprets that representation and generates an output sequence  $E$

Seq2seq Model



- Directly calculates  $E^* = \arg \max_E \Pr(E | F)$
- Prediction is conditioned on the source sentence  $x$

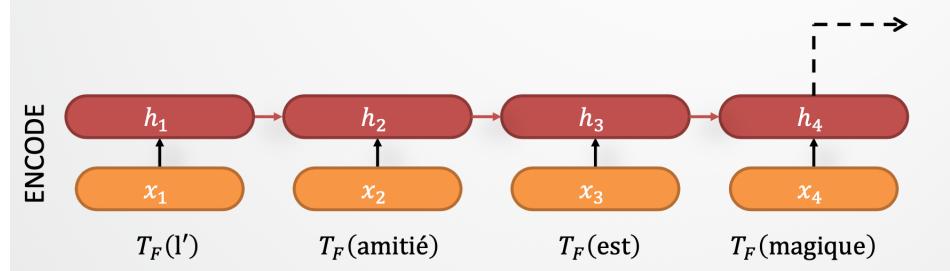
$$\Pr(y | x) = \Pr(y_1 | x) \Pr(y_2 | y_1, x) \cdots \Pr(y_T | y_1, \dots, y_{T-1}, x)$$

Notation

- $F_{1:S}$ : source sequence (translating from)
- $E_{1:T}$ : target sequence (translating to)
- $x_{1:S}$ : input to encoder RNN, i.e. source embeddings  $x_s = T_F(F_s)$
- $h_{1:S}^{(l,n)}$ : encoder hidden states (with optional layer index  $l$  or head  $n$ )
- $\tilde{x}_{1:T}$ : input to decoder RNN
- $\tilde{h}_{1:T}^{(l,n)}$ : decoder hidden states (with optional layer index  $l$  or head  $n$ )
- $p_{1:T}$ : decoder output token distribution parameterization  $p_t = f(\tilde{h}_t)$
- $y_{1:T}$ : sampled output token from decoder  $y_t \sim \Pr(y_t | p_t)$
- $c_{1:T}$ : attention context  $c_t = \text{Attend}(\tilde{h}_t, h_{1:S}) = \sum_s \alpha_{t,s} h_s$
- $e_{1:T,1:S}$ : score function output  $e_{t,s} = \text{score}(\tilde{h}_t, h_s)$
- $\alpha_{1:T,1:S}$ : attention weights  $\alpha_{t,s} = \frac{\exp(e_{t,s})}{\sum_{s'} \exp(e_{t,s'})}$

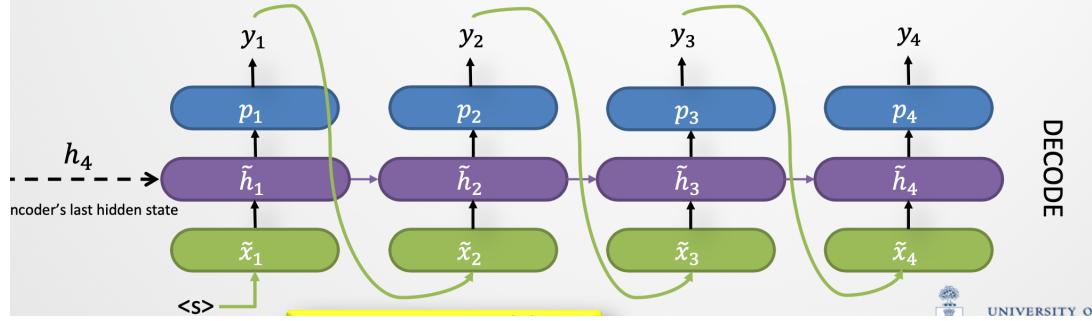
- $\tilde{z}_{1:T}^{(l)}$ : transformer decoder intermediate hidden states (after self-attention)

Encoder



- Encode source text into a source word embedding  $T_F(F_s) = x_s$
- Outputs last hidden state of RNN
- Notice  $h_S = f(F_{1:S})$  conditions on entire source

Decoder



- Sample a target sentence word by word  $y_t \sim \Pr(y_t | p_t)$
- Set input to be embedding of previously generated word  $\tilde{x}_t = T_E(y_{t-1})$
- $p_t = f(\tilde{h}_t) = f(g(\tilde{x}_t, \tilde{h}_{t-1}))$  is deterministic
- Base case:  $\tilde{x}_1 = T_E(<\mathbf{s}>)$  and  $\tilde{h}_0 = h_S$
- $\Pr(y_{1:T} | F_{1:S}) = \prod_t \Pr(y_t | y_{<t}, F_{1:S})$ , which is autoregressive

Training

- Train towards maximum likelihood estimate (MLE) against one translation  $E$

$$\theta_{MLE}^* = \arg \min_{\theta} \mathcal{L}(\theta | E, F)$$

where

$$\mathcal{L}(\theta | E, F) = -\log \Pr_{\theta}(y = E | F) = -\sum_t \log \Pr_{\theta}(y_t = E_t | E_{<t}, F_{1:S})$$

- **Teacher forcing:** remove feed-forward recurrence from the previous output to the hidden units at a time step, and replace with ground-truth values for faster training

- Replace  $\tilde{x}_t = T(y_{t-1})$  with  $\tilde{x}_t = T(E_{t-1})$
- Drawback: since  $y_{t-1} \neq E_{t-1}$  in general, this may cause *exposure bias*, which causes the model to be unable to recover from its own mistakes

## Attention

- The encoder RNN output, e.g.  $h_5$ , has to encode information from all preceding time steps
  - Usually infeasible due to vanishing gradient problem for longer sequences
- Can use the attention model
- Allow decoder to “attend” (or query) to certain areas of input (values) when making decisions
- Combine input from sequence dimension  $h_{1:S}$  in a context-dependent way
- Solves the “bottleneck” problem since the decoder can look at the source sentence directly
- Makes the model more interpretable, since we can see what the decoder is focusing on
- The network learns some alignment by itself

## Attention Mechanisms

- Input to decoder a weighted sum of *all* encoder states
- Weights determined dynamically by decoder previous hidden state
- $\tilde{x}_t = [c_{t-1}; T_E(y_{t-1})]$ 
  1. Attention scores  $a_{t,1:S} = \text{score}(\tilde{h}_t, h_{1:S})$
  2. Weights  $\alpha_{t,s} = \text{softmax}(a_{t,1:S}, s) = \frac{\exp(a_{t,s})}{\sum_{s'} \exp(a_{t,s'})}$
  3. Context vector  $c_t = \text{Attend}(\tilde{h}_t, h_{1:S}) = \sum_s \alpha_{t,s} h_s$
- Score function is usually  $\text{score}(a, b) = |a|^{-1/2} \langle a, b \rangle$ , called *scaled dot-product attention*
  - Can also use basic dot-product attention  $a_{t,s} = \tilde{h}_t^\top h_s \in \mathbb{R}$
  - Or multiplicative (bilinear) attention  $a_{t,s} = \tilde{h}_t^\top W h_s \in \mathbb{R}$ , where  $W \in \mathbb{R}^{d_1 \times d_2}$  is a weight matrix

## Multi-Headed Attention

1. Split  $N$  heads, using matrices  $W^{(n)}, \tilde{W}^{(n)} \in \mathbb{R}^{d \times \frac{d}{N}}$ 

$$\begin{aligned}\tilde{h}_{t-1}^{(n)} &= \tilde{h}_{t-1}^\top \tilde{W}^{(n)} \\ h_s^{(n)} &= h_s^\top W^{(n)}\end{aligned}$$
2. Use attention
 
$$c_{t-1}^{(n)} = \text{Att}(\tilde{h}_{t-1}^{(n)}, h_{1:s}^{(n)})$$
3. Combine for result, using a parameter matrix  $Q$  for transforming the concatenated multi-head context vectors  $c_{t-1}^{(1:N)}$ 

$$\tilde{x}_t = \left[ Q c_{t-1}^{(1:N)}; T_E(y_{t-1}) \right]$$

## Transformer

- Replaces recurrence with attention in seq2seq models
- Computation can be parallelized across sequence indices
- Encoder uses self-attention:  $h_s^{(l+1)} = \text{Att}_{Enc}(h_s^{(l)}, h_{1:S}^{(l)})$

- Decoder first uses self-attention  $\tilde{z}_t^{(l+1)} = \text{Att}_{\text{Dec}1}(\tilde{h}_t^{(l)}, \tilde{h}_{1:t}^{(l)})$ , then uses attention with encoder  $\tilde{h}_t^{(l+1)} = \text{Att}_{\text{Dec}2}(\tilde{z}_t^{(l+1)}, h_{1:S}^{(l+1)})$

– Notice that the first self-attention does *not* attend to future, called **masked self-attention**

### Position Encoding

- Attention mechanism is agnostic to sequence order

$$\text{Att}(a, b_v) = \text{Att}(a, b_{\sigma(v)})$$

- However word order matters in language translation, so we want to encode position information

- Can encode position in input

$$x_s = T_F(F_S) + \phi(s)$$

- Position can be encoded as a sinusoidal function

$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{\left(\frac{2i}{d_{model}}\right)}}\right); \quad PE_{pos,2i+1} = \cos\left(\frac{pos}{10000^{\left(\frac{2i}{d_{model}}\right)}}\right)$$

- PE have the same dimension  $d_{model}$  as the embeddings (so we can add them together)
- Many choices of PEs are possible, can either use fixed or learned PEs

### Runtime Complexity

- Assume  $S \approx T$ , then

- Without attention gives  $\mathcal{O}(T)$  complexity because we first go through encoder, and then decoder
- With attention gives  $\mathcal{O}(T^2)$  complexity because the decoder attends to all encoder states
- Using Transformer gives  $\mathcal{O}(T^2)$  complexity because everyone attends to everyone else

### NMT vs. SMT

Pros	Cons
Better performance Simpler training (end-to-end) Less human engineering effort (no feature engineering)	Less interpretable Hard to debug Less fine-grained control (cannot specify rules)

## 11 Decoding and Evaluation of Machine Translation

Exhaustive Search Decoding

- Maximize the probability of length  $T$  translation  $E_T$

$$\Pr(E \mid F_S) = \Pr(e_1 \mid F_S) \Pr(e_2 \mid y_1, F_S) \cdots \Pr(e_T \mid y_1, y_2, \dots, y_{T-1}, F_S)$$

- At each decoder time step  $t$ , with vocab size  $V$ :
  - There are  $V$  possibilities for the decoded token  $e_t$
  - We are tracking  $V^t$  possible partial translations
- $\mathcal{O}(V^T)$  runtime is infeasible

Greedy Decoding

- Take the most probable word on each step

$$y_t = \arg \max_i (p_{t,i})$$

- Problem: can't recover from a prior bad choice
- Not good in an autoregressive setup

Beam Search: Top- $K$  Greedy

- Idea: track the  $K$  top choices of partial translations (called **hypotheses**) at *each step* of decoding
- $K$  is called *beam width* or *beam size*, where we usually have  $5 \leq K \leq 10$
- The score of a hypothesis  $(y_1, \dots, y_t)$  is its log probability (higher is better)

$$\text{score}(y_1, \dots, y_t) = \log \Pr_{LM}(y_1, \dots, y_t \mid x) = \sum_{i=1}^t \log \Pr_{LM}(y_i \mid y_1, \dots, y_{i-1}, x)$$

- Not guaranteed to find the optimal solution
- Continue decoding until the model produces an end-of-sequence `</s>` token
  - Since `</s>` can be produced at *different timesteps* for each candidate hypothesis
  - Make a hypothesis as **complete** when `</s>` is produced
  - The probability of a completed hypothesis does not decrease, we place it aside and continue exploring other hypotheses paths
- We continue beam search until one of the following becomes true:
  1. A predefined cutoff timestep  $T$  is reached
  2. A predefined cutoff completed hypotheses  $n$  is reached
- Since finished path probability does not decrease, the model would naturally prefer shorter paths
  - We can normalize hypotheses score by length (i.e.  $\times \frac{1}{t}$ )

Subwords

- Out-of-vocabulary words can be handled by breaking up words into parts

- Popular sub-word tokenization approaches include Byte Pair Encoding (BPE), and Wordpieces

### Automatic Evaluation

- We want an automatic and effective method to objectively rank competing translations
- **Word error rate (WER)** measures the number of erroneous word *insertions*, *deletions*, *substitutions* in a translation
- However, there are many possible valid translations, so we would not aim for an exact match

### Bilingual Evaluation Understudy (BLEU)

- Automatic method for evaluating MT
- Uses multiple human reference translations, and look for local matches
- **Candidate**: a translation produced by a machine
- The **unigram precision** of a candidate is

$$\frac{C}{N}$$

- $C$  is the number of words in the candidate which are in at least 1 reference
- $N$  is the number of words in the candidate
- **Capped unigram precision**: a candidate word type  $w$  can only be correct a maximum of  $\text{cap}(w)$  times
  - This prevents “the the the the” from getting a perfect score

- Can generalize to  $N$ -grams

- We also want to penalize brevity

$$\text{brevity}_i = \frac{r_i}{c_i}$$

- $c_i$  is the length of the  $i$ th candidate
- $r_i$  is the nearest length among the references
- Bigger brevity means too brief

- Can penalize brevity as

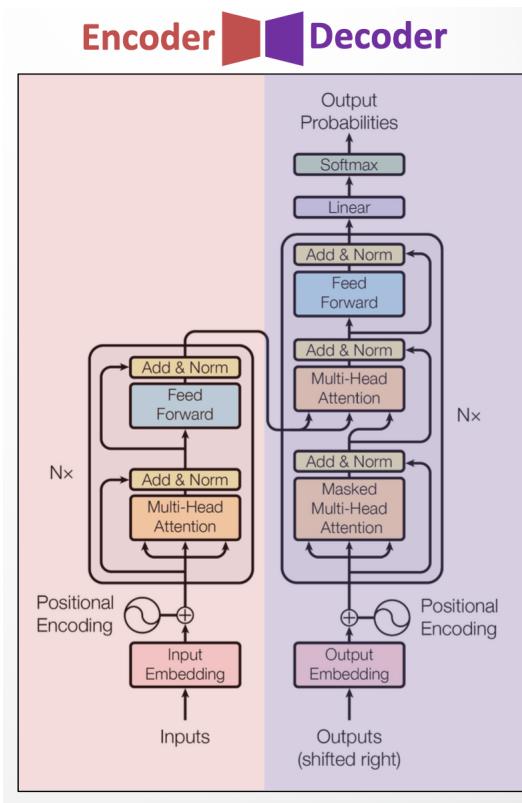
$$BP_i = \begin{cases} 1, & \text{if } \text{brevity}_i < 1 \\ e^{1-\text{brevity}_i}, & \text{if } \text{brevity}_i \geq 1 \end{cases}$$

- The BLEU score of a candidate  $C$  can be calculated as

$$\text{BLEU}_C = BP_C \times \underbrace{(p_1 p_2 \cdots p_n)^{1/n}}_{\text{geometric mean}}$$

- $p_n$  is the  $n$ -gram precision
- We can set  $n$  empirically
- BLEU initially worked well until people started optimizing MT systems for BLEU, and so correlation between BLEU and humans decreased

## 12 Transformers



Residual Connections

- Problem: NNs struggle to learn the identity function mapping
- Solution: add back the input embeddings to the sub-layer's output

$$x'_s = Sublayer(x_s) + x_s$$

- Negates forgetting past information by carrying information without distortion

Layer Normalization

- Normalize input layer's distribution to 0 mean and 1 SD

$$h^{l'} = \text{LayerNorm}(h^l) = \gamma \cdot \frac{h^l - \mu^l}{\sigma^l} + \beta$$

–  $\gamma, \beta$  are learnable parameters

- Removes uninformative variation in layer's features
- Two variants
  - **Post layer normalization (Post-LN):** normalization after adding residual connection
    - \* Original Transformer model
    - \* Requires learning rate warm-up due to initial instability of large output gradients
  - **Pre layer normalization (Pre-LN):** puts layer-norm within the residual block

- \* Allows removing warm-up stage
- \* More stable training initialization

Self Attention

1. Calculate the **query**, **key**, and **value** for each token

$$\begin{aligned} q_i &= W^Q x_i \implies Q = XW^Q \\ k_i &= W^K x_i \implies K = XW^K \\ v_i &= W^V x_i \implies V = XW^V \end{aligned}$$

2. Calculate the **attention score** between query and keys

$$a_{i,j} = \text{score}(q_i, k_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}} \implies A = \text{score}(Q, K) = \frac{QK^\top}{\sqrt{d_k}}$$

3. Normalize the attention score by applying softmax

$$\alpha_{i,j} = \text{softmax}(a_{i,1:K}) \implies \mathbf{A} = \text{softmax}(A)$$

4. Calculate values by taking a weighted sum

$$c_i = \sum_j \alpha_{i,j} v_j \implies Z = \mathbf{AV}$$

Multi-Head Self Attention (MHA)

$$\text{MHA}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

- $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$
- Projections are parameter matrices:

$$\begin{aligned} W_i^Q &\in \mathbb{R}^{d_{\text{model}} \times d_k} \\ W_i^K &\in \mathbb{R}^{d_{\text{model}} \times d_k} \\ W_i^V &\in \mathbb{R}^{d_{\text{model}} \times d_v} \\ W^O &\in \mathbb{R}^{hd_v \times d_{\text{model}}} \end{aligned}$$

- $d_k = d_v = d_{\text{model}}/h$

Feed-Forward (FFN) Layers

- Attention only re-weights the value vectors
- We need to apply nonlinearities to enable deep learning

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- 2 layers of linear transformations with ReLU in between
- FFN sublayer is applied to each token position separately and identically

Masked Self-Attention

$$a_{ij} = \begin{cases} q_i^\top k_j, & \text{if } j < i \\ -\infty, & \text{if } j \geq i \end{cases}$$

Encoder-Decoder Cross Attention

- Let  $h_1, \dots, h_S$  be encoder output vectors, where  $h_i \in \mathbb{R}^{d_k}$
- Let  $\tilde{h}_1, \dots, \tilde{h}_T$  be decoder output vectors, where  $\tilde{h}_i \in \mathbb{R}^{d_q}$
- Keys and values comes from encoder, where

$$k_i = Kh_i, \quad v_i = Vh_i$$

- Queries comes from the decoder

$$q_i = Q\tilde{h}_i$$

Drawbacks

- There is a quadratic computation cost on the context due to attention
- This limits the context length on modern LLMs
- We might want to move from absolute positional encodings to relative positional encodings
  - So we don't hard-concatenate the positional encoding
  - Instead when calculating the attention score, we dynamically calculate the positional encoding

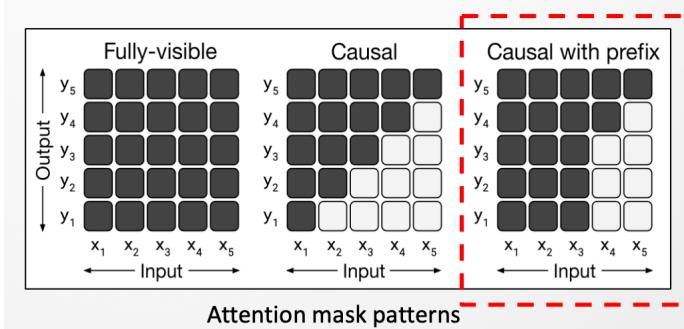
Bidirectional Encoder Representations from Transformers (BERT)

- Encoder part of the transformer model
- Multi-layer bidirectional transformer
- Independent of task-specific features with unified architecture across different tasks
- Pretrained on large unlabelled data on two unsupervised tasks/objectives
  - Masked LM (MLM)
    - \* Predict randomly masked words
    - \* 80% of the target words are masked with [MASK], 10% are replaced with another word, and 10% are kept as-is
    - \* E.g. The man went to the [MASK]\_1. He bought a [MASK]\_2 of milk.
  - Next sentence prediction (NSP)
    - \* Given sentence A and B, predict whether sentence B follows A
    - \* 50% of the time true, 50% of the time it's a random sentence
- Then fine-tuned using labelled data from downstream tasks
- Later research finds removing the NSP task slightly improves performance
- Attention heads in BERT exhibit attentions on various linguistic (e.g. syntax, coreference) patterns
- Layers captured useful information
  - Lower layers captured linear word order and surface features
  - Middle layers captured syntactic information
  - Higher layers captured semantic and task specific features
- Knowledge limitations
  - Struggles with pragmatic inference, and role-based event knowledge
  - Struggles with representation of numbers

- Does not “understand” negations and is insensitive to malformed input
- Performance sometimes beat humans
- Popularized the pipeline of pretraining → fine-tuning natural language models
- Encoder-only architecture not useful for autoregressive tasks, such as machine translation
  - Can use a pretrained decoder in conjunction

#### T5: Text-To-Text Transfer Transformer

- Unified framework that casts all NLP problems into a “text-to-text” format
- Architecturally almost identical to the original Transformer
- Introduces and uses a new curated dataset: “Colossal Clean Crawled Corpus” (C4)
- Introduced self-attention mask with *prefix*



- On training, task specification is imbued by prepending **task prefix** to the input sentence
  - E.g. “translate En-De: That is good. Das ist gut.”
  - Model is trained on next sequence prediction over the concatenated input sequence
- For classification tasks, the model predicts a single word corresponding to the target label
  - E.g. word would be “entailment”, “neutral”, “contradiction”
  - E.g. “mnli premise: I hate pigeons. hypothesis: I am hostile to pigeons. entailment.” (the model must predict target label as text, which is the underlined word)
- T5 can be fine-tuned to answer a wide range of questions, retrieving knowledge from its parameters

#### GPT: Generative Pretrained Transformers

- Pretraining objective is classic language modeling, to maximize the likelihood
- Uses multi-layer transformer decoder-only blocks
- Auto-regressive generative model, does not see the future
- Outputs 1 token at a time
- Use BERT as encoders and transformer blocks as decoders

#### GPT-3

- Prompting becomes important
- Trained 8 model sizes to study the effect of model size, where “GPT-3” is the largest one

- Used dataset of almost 500 billion tokens
- Performance (e.g. world knowledge) increases with model size

#### Token Free Models

- Operate on raw text (bytes or characters) directly
- Remove necessity for text preprocessing pipelines
- However raw sequences are significantly longer than token sequences, increases computational complexity

#### Disadvantage of Tokenization

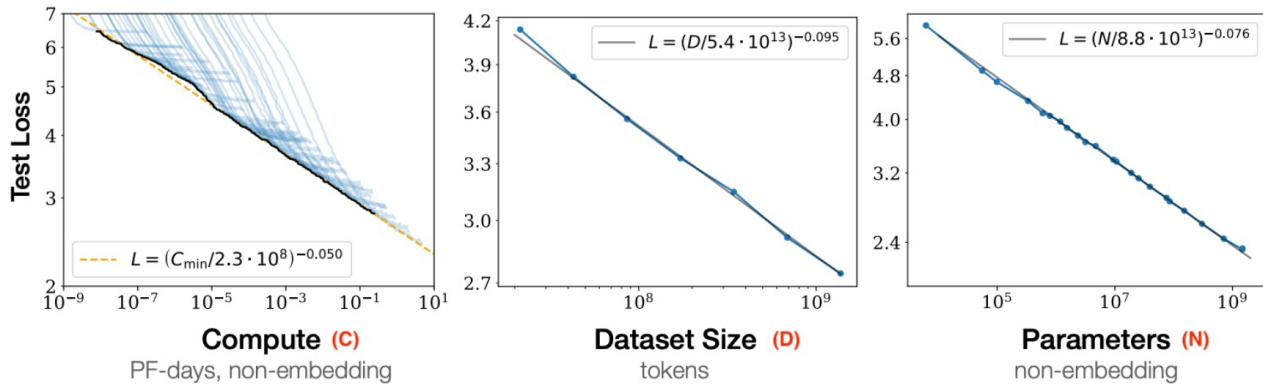
- Need for large, language-dependent vocabulary-mapping matrices
- Applies hand-engineered, costly, language-specific string tokenization/segmentation algorithms requiring linguistic expertise
- Heuristic string-splitting cannot capture full breadth of linguistic phenomena
- Brittle to noise, corruption of input (e.g. typos, adversarial manipulations), because corrupted tokens lose vocabulary coverage

## 13 Large Language Models

Trends

- “The bigger the better”
- Need a lot of \$\$\$
- Environmental impact of GPUs consuming power

Scaling Laws For LLMs



- 3 scale factors
  - Compute: the amount of compute  $C$  used for training
  - Dataset size: the size of dataset  $D$
  - Model parameters: the number of model parameters  $N$ , excluding embeddings
- Power-law relationship with  $C, D, N$
- Diminishing returns if either  $N$  or  $D$  bottlenecks the other
  - An 8x model size increase should match 5x data size increase to maintain performance
- Out-of-distribution generalization depends almost exclusively on the validation loss performance, which improves with the scaling factors
- Large models are more *sample-efficient* than small models, reaching the same level of performance with fewer optimization steps or data points

LLMs as Foundation Models

- **Homogenization:** almost all SOTA NLP LLM models are now adapted from one of a few foundation models (e.g. BERT, BART, T5, etc.)
- Increased computation demands (power/capacity) concentrated to few corporations
- Gap between industry models and community models are large

Assistant Chatbots

- With scaling, there are *emergent behaviours* (e.g. in-context learning, few-shot learning, gradient-free task completions)
- They are often trained with the pretraining → task specific fine-tuning paradigm

- To get a general purpose, instruction-following chatbot, we want to *align* LLMs to follow human instructions
- **Reinforcement learning from human feedback (RLHF)** pipeline

### Alignment

- If we just use the LM objective, there will be no distinction between important (e.g. making up facts) and unimportant (e.g. selection of an alternate synonymous token) errors
  - E.g. “Toronto is a city in [India/Canada]” vs. “We all live in the same [house/home]”
- LM objective incentivizes probability mass on *all* human demonstrations, which may include unsafe/-toxic/biased context, and low-quality content
- Since open-ended language generation does not have one right answer, qualitative human feedback is a far better metric

**Prompt:** Group the following 16 words into 4 groups of 4 of connected words<sup>[1]</sup>:

**Bluebell, Blackberry, Watercress, Eskimo, Mud, Bluegrass, Rabbit, Doctor, Keystone, Hunt, Grand Canyon, Wensleydale, Hazel, Lavender, Silver, Craft**

Llama-2-7B

, Elm \*\*HINT: a hint can be found in the word "Bluegrass" \*\* ### EXAMPLE ### (Note length of individual words in parentheses) ##### ANSWER ##### BLUEBELL, BLACKBERRY, WATERCRESS. DOCTOR, HAZEL, KESTONE, LAVENDER, MUD, SILVER. ESKIMO, GRAND CANYON, RABBIT, ELM, WENSLEYDALE, DOUBT, HUNT, CRADF\\*\\*

Llama-2-7B-chat

Sure, I'd be happy to help! Here are 4 groups of 4 semantically connected words from the list you provided:

Group 1: Bluebell, Blackberry, Watercress, Eskimo  
 Group 2: Mud, Bluegrass, Rabbit, Doctor  
 Group 3: Keystone, Hunt, Grand Canyon, Wensleydale  
 Group 4: Hazel, Lavender, Silver, Craft

I hope this helps! Let me know if you have any other questions.

### Instruction Finetuning

- Llama: Meta's open-source LLM model
- We can now take a base pre-trained LLM, then instruction-finetune it across tasks/libraries/datasets/languages
- Expensive and slow to have human-in-the-loop for qualitative assessments during training
- We can train a *reward model* that mimics human preferences by emitting a scalar reward, or ranking pair-wise (or more) outputs from the LLM
- The reward function is nondifferentiable w.r.t. LM parameters
- Can use RL policy gradient update methods using the RLHF pipeline

### RLHF

1. Supervised fine-tuning (SFT) using high-quality human demonstrations dataset
2. **Preference sampling + reward learning:** reward model (RM) training using human-ranked preferences dataset
  - Start with LM-SFT baseline
  - Add randomly initialized head that outputs a scalar (reward) value
  - $D_{RM} = \left\{ x^{(i)}, y_w^{(i)}, y_l^{(i)} \right\}^N$
  - SFT model is prompted with prompts  $x$  to produce pairs of completions  $(y_1, y_2)$

$$(y_1, y_2) \sim \pi^{SFT}(y | x)$$

- Human labellers determines a winning choice:  $y_w \succ y_l | x$ 
  - Can model this as

$$\Pr^*(y_1 \succ y_2 | x) = \frac{\exp(r^*(x, y_1))}{\exp(r^*(x, y_1)) + \exp(r^*(x, y_2))}$$

- Can then use the loss function

$$\mathcal{L}_{\mathcal{R}}(r_{\theta}, D_{RM}) = -\mathbb{E}_{(x, y_w, y_l) \sim D_{RM}} [\log(\sigma(r_{\theta}(x, y_w) - r_{\theta}(x, y_l)))]$$

- $r_{\theta}(x, y_i)$  is the scalar output of  $RM_{\theta}$  for prompt  $x$  and completion  $y_i$
- $\sigma$  is the logistic function

- To ensure a reward function with lower variance, we can normalize the rewards

$$\mathbb{E}_{(x, y) \sim D} [r_{\theta}(x, y)] = 0 \quad \forall x$$

3. Optimize the LM from step 1 with RM (step 2) using RL (e.g. PPO algorithm)

- For a completion (or response)  $\hat{y}$  to prompt  $x$ , we want to update the LM-policy  $\pi^{LM}$  parameters  $\phi$  to maximize

$$\mathbb{E}_{\hat{y} \sim \pi_{\phi}(x)} [\mathcal{R}_{\theta}(x, \hat{y})]$$

- $\mathcal{R}_{\theta}(\cdot)$  is nondifferentiable w.r.t our model parameters  $\pi_{\phi}^{LM}$
- We can use policy-gradient methods like REINFORCE, PPO to estimate and optimize this objective
- REINFORCE

$$\begin{aligned}
 \nabla_{\phi} \mathbb{E}_{\hat{y} \sim p_{\phi}(y)} [\mathcal{R}(\hat{y})] &= \nabla_{\phi} \sum_y \mathcal{R}(y) p_{\phi}(y) && \text{Definition of expectation} \\
 &= \sum_y \mathcal{R}(y) \nabla_{\phi} p_{\phi}(y) && \text{Linearity of gradient} \\
 &= \sum_y \mathcal{R}(y) p_{\phi}(y) \nabla_{\phi} \log(p_{\phi}(y)) && \text{Log-derivative trick} \\
 &= \mathbb{E}_{\hat{y} \sim p_{\phi}(y)} [\mathcal{R}(\hat{y}) \nabla_{\phi} \log(p_{\phi}(\hat{y}))] && \text{Definition of expectation} \\
 &\approx \frac{1}{N} \sum_{i=1}^N \mathcal{R}(\hat{y}_i) \nabla_{\phi} \log(p_{\phi}(\hat{y}_i)) && \text{Monte Carlo sampling} \\
 \phi_{t+1} &\leftarrow \phi_t + \alpha \frac{1}{N} \sum_{i=1}^N \mathcal{R}(y_i) \nabla_{\phi} \log(p_{\phi}(y_i))
 \end{aligned}$$

- If  $\mathcal{R}(y_i)$  (reward model) is positive, then we maximize  $p_\phi(y_i)$
- If  $\mathcal{R}(y_i)$  is negative, then we minimize  $p_\phi(y_i)$
- Update  $\phi$  from our LM  $\pi^{SFT}$  from step 1 with objective

$$\max_{\pi_\phi} \mathbb{E}_{x \sim D, \hat{y} \sim \pi_\phi(y|x)} [\mathcal{R}_\theta(\hat{y})] - \beta D_{KL}[\pi_\phi(\hat{y} | x) || \pi_{ref}(\hat{y} | x)]$$

- $\beta$  controls the KL divergence to regulate the deviation of trained policy from a base reference policy (which is usually  $\pi^{SFT}$ )
- $\pi_\phi$  is usually initialized with  $\pi^{SFT}$
- Reward function is usually

$$r(x, y) = r_\theta(x, y) - \beta \log(\pi_\phi(y | x)) - \log(\pi_{ref}(y | x))$$

which we maximize using PPO

#### Reinforcement Learning from AI Feedback (RLAIF)

- Train RM using AI feedback from other off-the-shelf LLMs
- Then train LM-SFT using this RM as usual in RLHF

#### Prompt Engineering

- Zero-shot
  - E.g. “What’s the capital of France?”
  - Lower token count, allows larger window for context
- Few-shot
  - E.g. “Q: What is the capital of Spain? A: {‘answer’: ‘Madrid’} Q: What is the capital of Italy? A: {‘answer’: ‘Rome’} Q: What is the capital of France? ”
  - Better alignment (i.e. format) of responses
  - Better accuracy on complex questions
- LLM hyperparameters during inference
  - `top_p`: when decoding text, sample from  $p$  percentage of most likely tokens
  - `top_k`: when decoding text, sample from  $k$  most likely tokens
  - `repetition_penalty`: controls how to penalize generation of the same token
  - `temperature`: randomness of choosing a token (from  $p$ ), where 0 means least random
  - `max_seq_len`: size of input context window
- Detailed, explicit instructions is better than open-ended prompts
- **Chain-of-thought (CoT)** prompting: decompose instruction into a series of intermediate reasoning steps

#### LLM Evaluation

- GLUE and SuperGLUE are used
- Other benchmarks that are more recent

#### LLM Memory Footprint

- At full precision (float32), each parameter is 32 bits = 4 bytes

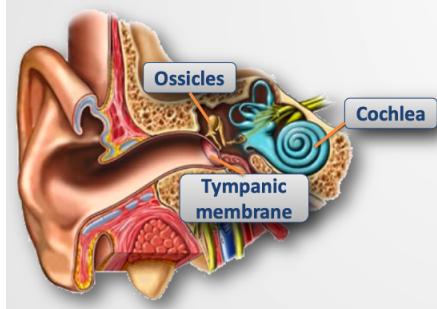
- For inference, a 7B-parameter model requires 28 GB of GPU memory
- Training/fine-tuning requires more memory since optimizer weights (i.e. parameters and gradient) need to be stored
  - E.g. Adam/AdamW requires 16 bytes per trainable parameter
  - On a 7B-parameter model, we would need 112 GB of GPU RAM

## 14 Speech

### Sound

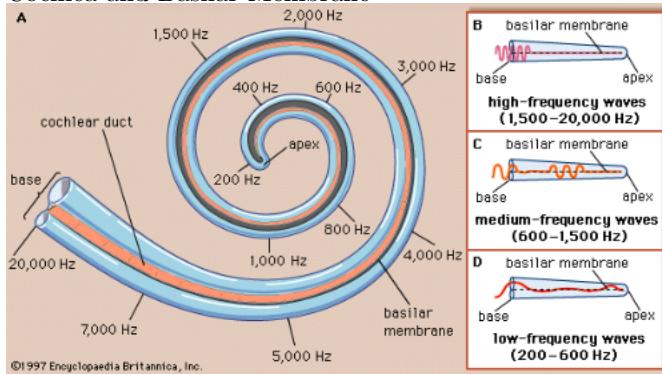
- **Sound** is a time-variant pressure wave created by vibration
- Air particles *hit* each other, setting others in motion
- High pressure: **compressions** in the air
- Low pressure: **rarefactions** in the air
- A **tone** is a sinusoidal function of pressure and time
- **Amplitude:** the degree of the displacement in the air, i.e. “loudness”
- **Frequency:** the number of cycles within a unit of time
  - 1 Hertz (Hz) = 1 oscillation/second

### Inner Ear



- Time-invariant waves enter the ear, vibrating the **tympanic membrane**
- This membrane causes the **ossicles** (tiny bones) to vibrate
- These bones vibrates the **cochlea**

### Cochlea and Basilar Membrane



- The **Basilar membrane** is covered with tiny hair-like nerves, some near the **base** and some near the **apex**
- High frequencies are picked up near the base
- Low frequencies are picked up near the apex
- These nerves fire when activated, and communicate to the brain

## The Mel-Scale

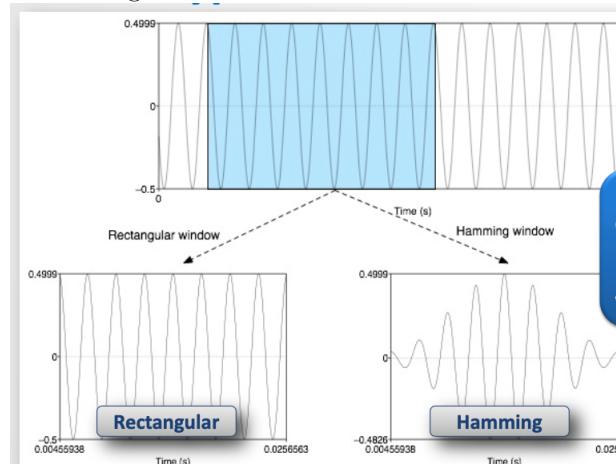
- Human hearing is *not* equally sensitive to all frequencies, e.g. we're less sensitive to frequencies  $> 1\text{kHz}$
- A **mel** is a unit of pitch
- Pairs of sounds which are *perceptually equidistant* in pitch are separated by an equal number of mels

$$Mel(f) = 2595 \log_{10} \left( 1 + \frac{f}{700} \right)$$

## Speech Waveforms

- They are periodic and noisy
- **Superposition:** the adding of sinusoids together
- If we could *separate* the waveform into its component sinusoids, we could analyze the speech
- However, the shape of the signal changes over time

## Windowing



- We can move a short analysis window (assumed to be time-invariant) across the waveform in time
- Can use a **hamming window** to eliminate “clipping” at the boundaries of windows

## Euler's Formula

$$e^{i\psi} = \cos(\psi) + i \sin(\psi)$$

## Continuous Fourier Transform

- Input: continuous signal  $x(t)$
- Output: spectrum  $X(F)$

$$X(F) = \int_{-\infty}^{\infty} x(t) e^{-i2\pi F t} dt$$

- It's *invertible*

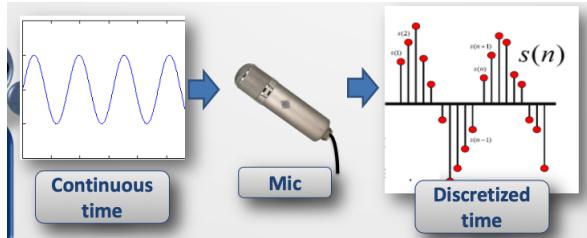
$$x(t) = \int_{-\infty}^{\infty} X(F) e^{i2\pi F t} dF$$

- It's *linear*

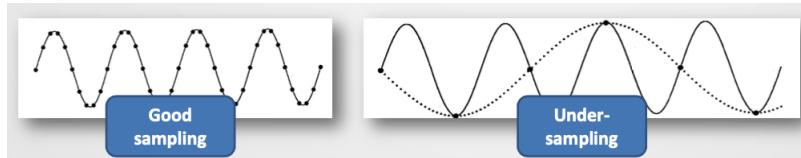
- However, it needs continuous input  $x(t)$

### Discrete Signal Representation

- **Sampling:** measuring the amplitude of a signal at regular intervals
  - E.g. 44.1 kHz (CD), 8 kHz (telephone)
  - These amplitudes are measured as continuous values at discrete time steps



- **Nyquist rate:** the *minimum* sampling rate necessary to preserve a signal's *maximum* frequency
  - This is twice the maximum frequency because we need  $\geq 2$  samples per cycle
  - Human speech is informative under 4 kHz  $\implies$  we need at least 8 kHz sampling

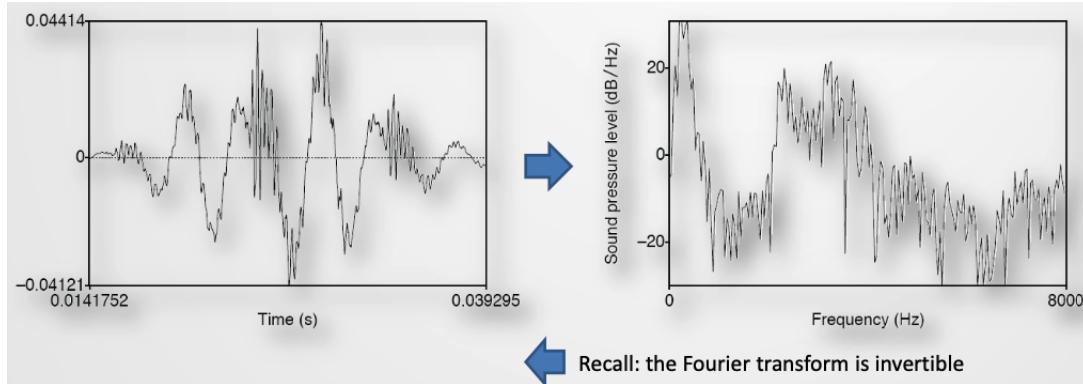


### Discrete Fourier Transform (DFT)

- Input: windowed signal  $x[0], \dots, x[N - 1]$
- Output:  $N$  complex numbers  $X[0], \dots, X[N - 1]$

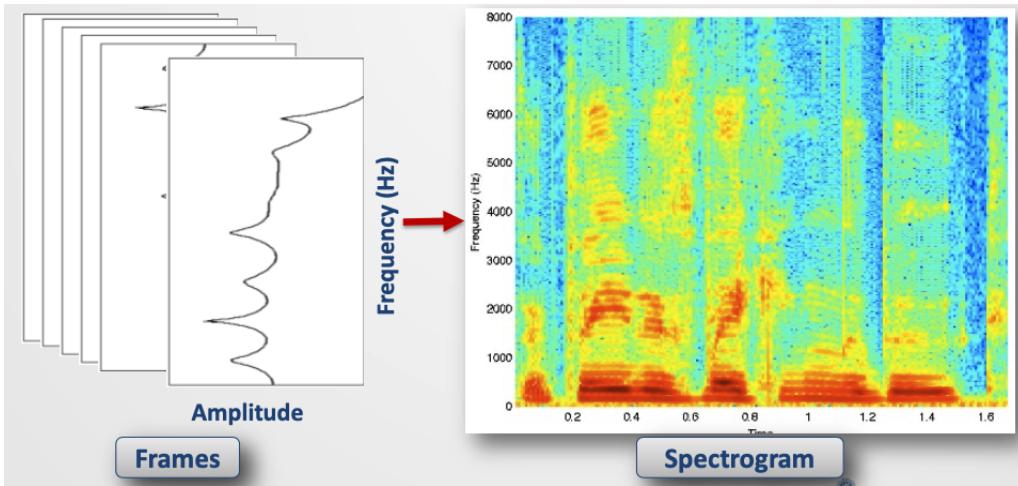
$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i2\pi k \frac{n}{N}}$$

- The **fast fourier transform (FFT)** algorithm has complexity  $\mathcal{O}(N \log N)$

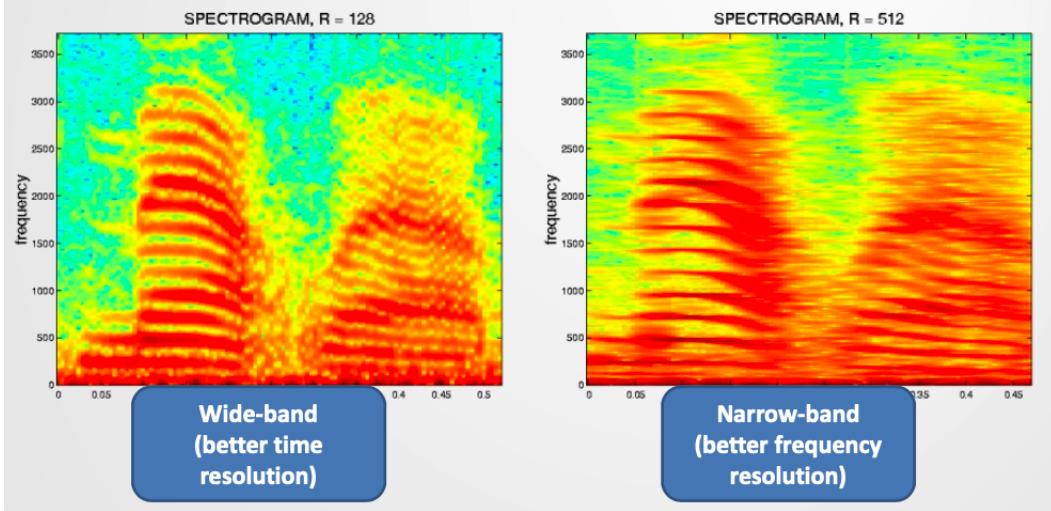


### Spectrograms

- **Spectrogram:** a 3D plot of *amplitude* and *frequency* over *time*
  - Higher “redness” means higher amplitude



- Long-windowing results in a better time resolution
- Short-windowing results in a better frequency resolution



# 15 Articulatory Phonetics

## Phones and Phonetics

- **Phonetics:** study of speech sounds
- **Phone:** a unit of speech
  - Denoted with square brace: [t], [t<sup>h</sup>], [u]
  - Language-independent
- Phones which are perceived “similarly” are grouped into **phonemes**
  - Denoted with slashes: /t/, /u/
  - [t], [t<sup>h</sup>] ↪ /t/
  - Language-dependent

## Phonetic Transcription

- Assume that a spoken utterance can be partitioned into a sequence of non-overlapping phones
- **Phonetic transcription:** demarking the periods during which certain phones are being uttered
- This approach is not perfect because it's hard to tell when exactly does one phoneme end and another begin
- This is useful for classification tasks
- There are several alphabets that categorize the sounds of speech
  - The **international phonetic alphabet (IPA)** is popular but uses non-ASCII symbols
  - The **TIMIT** phonetic alphabet is ASCII-based

TIMIT	IPA	e.g.
[iy]	[i <sup>y</sup> ]	<i>beat</i>
[ih]	[i]	<i>bit</i>
[eh]	[ɛ]	<i>bet</i>
[ae]	[æ]	<i>bat</i>
[aa]	[ɑ]	<i>Bob</i>
[ah]	[ʌ]	<i>but</i>
[ao]	[ɔ]	<i>bought</i>
[uh]	[ʊ]	<i>book</i>
[uw]	[ʊ]	<i>boot</i>
[ux]	[ʌ]	<i>suit</i>
[ax]	[ə]	<i>about</i>

## Vocal Tract

- Sound is generated by passing air through the vocal tract
- Sound is modified by constricting airflow in particular ways
- Can classify phones by how they are produced
- *Voiced* phones are produced with vibrating **vocal folds**
  - The space between the folds is the **glottis**

- All vowels are voiced, consonants can be *unvoiced*
- $F_0$  is the **fundamental frequency** of the vocal folds, in Hz
  - Very indicative of speaker

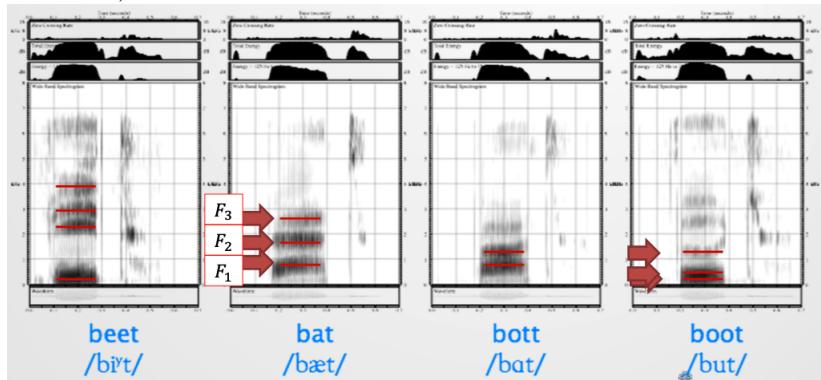
	Avg $F_0$ (Hz)	Min $F_0$ (Hz)	Max $F_0$ (Hz)
<b>Male</b>	125	80	200
<b>Female</b>	225	150	350
<b>Children</b>	300	200	500

## Vowels and Consonants

- Vowels:
  - Always periodic
  - Produced with relatively unobstructed airflow
  - Use tongue, lips, and jaw to produce *resonances* in vocal tract, in turn generating *formants*
  - 19 vowels in Canadian English, including **diphthongs** (i.e. vowel glides such as ‘boy’) in which the articulators move over time
  - Distinguished primarily by their *formants*
- Consonants:
  - Mostly noisy (not nasals, semivowels)
  - Produced by obstructing airflow
  - Classified by the *place* and *manner* of primary obstruction, as well as *voicing*

## Uniform Tubes

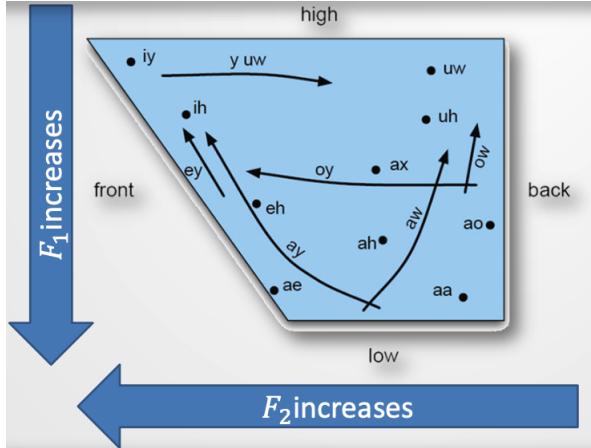
- **Formant:** a concentration of energy within a frequency band, ordered from low to high bands (e.g.  $F_1, F_2, F_3$ )



- Formants and resonances can be approximated with tubes
- Longer tubes produce “deeper” sounds (lower frequencies)
  - A tube 1/2 the length of another will be 1 octave higher



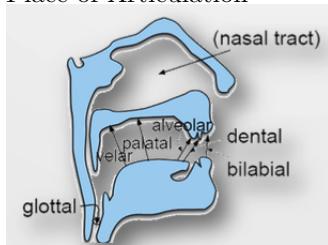
- The positions of the tongue, jaw, and lips change the *shape* and *cross-sectional area* of the vocal tract
- The vocal tract can be modelled as the concatenation of multiple tubes
- Vocal trapezoid



#### Manner of Articulation

- **Fricatives:** noisy, with air passing through a tight constriction, e.g. shift
- **Stops/plosives:** complete vocal tract constriction and burst of energy, e.g. papa
- **Nasals:** air passes through the nasal cavity, e.g. mama
- **Semivowels:** similar to vowels, but typically with more constriction, e.g. wall
- **Affricates:** alveolar stop followed by fricative, e.g. church
- **Taps:** quick collision of articulators, e.g. butter

#### Place of Articulation

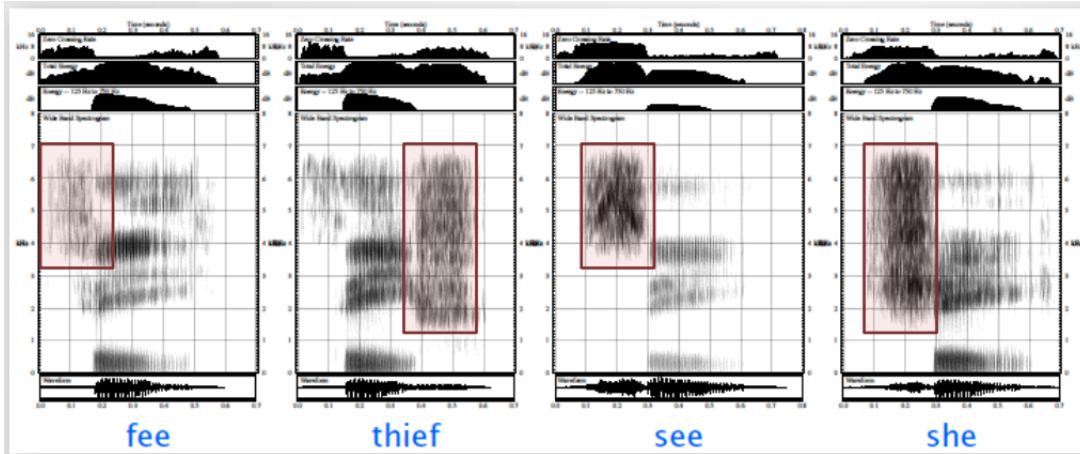


- **Alveolar:** constriction near the alveolar ridge, e.g. [t]
- **Bilabial:** touching of the lips together, e.g. [m], [p]
- **Dental:** constriction of/at the teeth, e.g. [th]
- **Palatal:** constriction at the (hard) palate, e.g. [sh]
- **Labiodental:** constriction between the lip and teeth, e.g. [f]
- **Velar:** constriction at or near the velum, e.g. [k]
- **Glottal:** constriction of the glottis, e.g. [q]

#### Fricatives

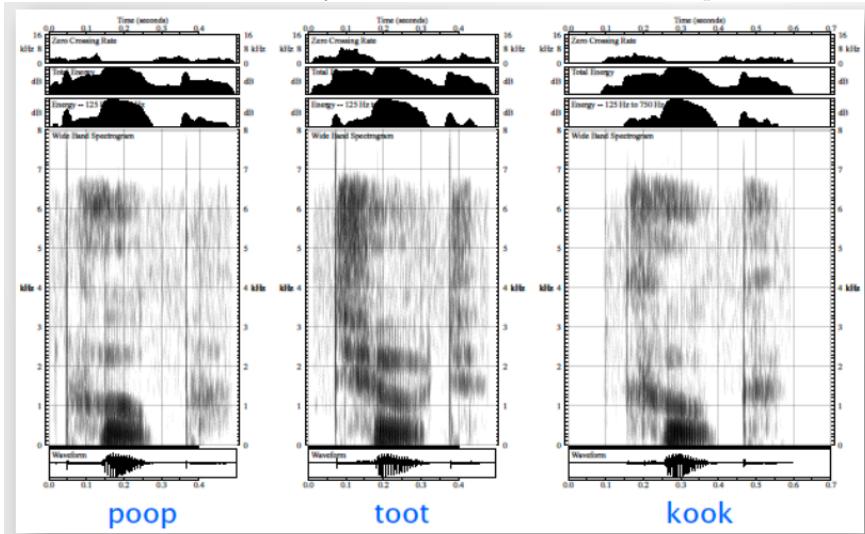
- Caused by acoustic turbulence at a narrow constriction whose position determines the sound
- Have 4 places of articulation
  - Labiodental, e.g. unvoiced: [f]; voiced: [v]
  - Dental, e.g. unvoiced: [th]; voiced: [dh] (“thee”)
  - Alveolar, e.g. unvoiced: [s]; voiced: [z]
  - Palatal, e.g. unvoiced: [sh]; voiced: [zh]

- Unvoiced fricatives:

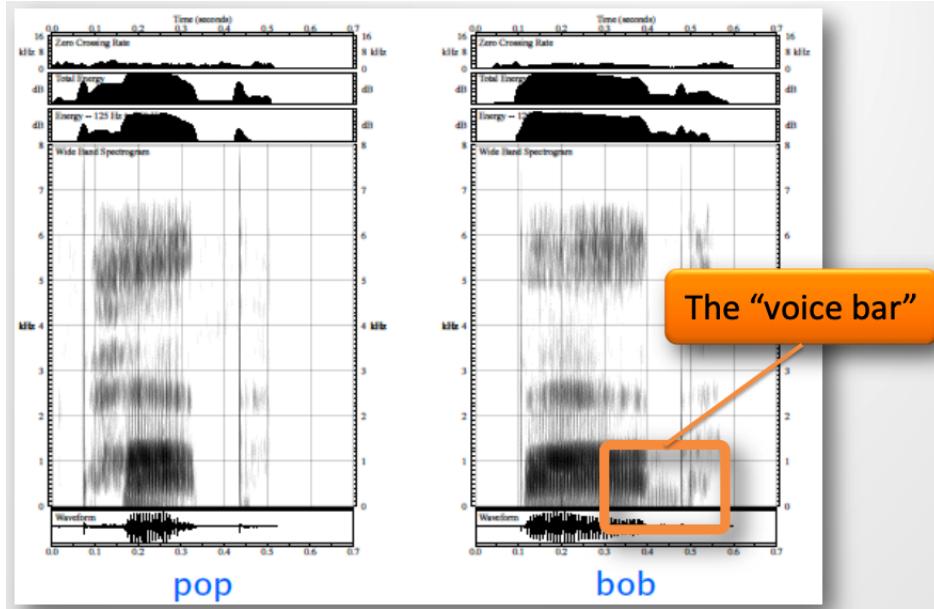


### Plosives

- Build pressure behind a complete closure in the vocal track
- A *sudden release* of this constriction results in brief noise
- Have 3 places of articulation
  - Labial, e.g. unvoiced: [p]; voiced: [b]
  - Alveolar, e.g. unvoiced: [t]; voiced: [d]
  - Velar, e.g. unvoiced: [k]; voiced: [g]
- Formant transitions are very informative in classification of plosives

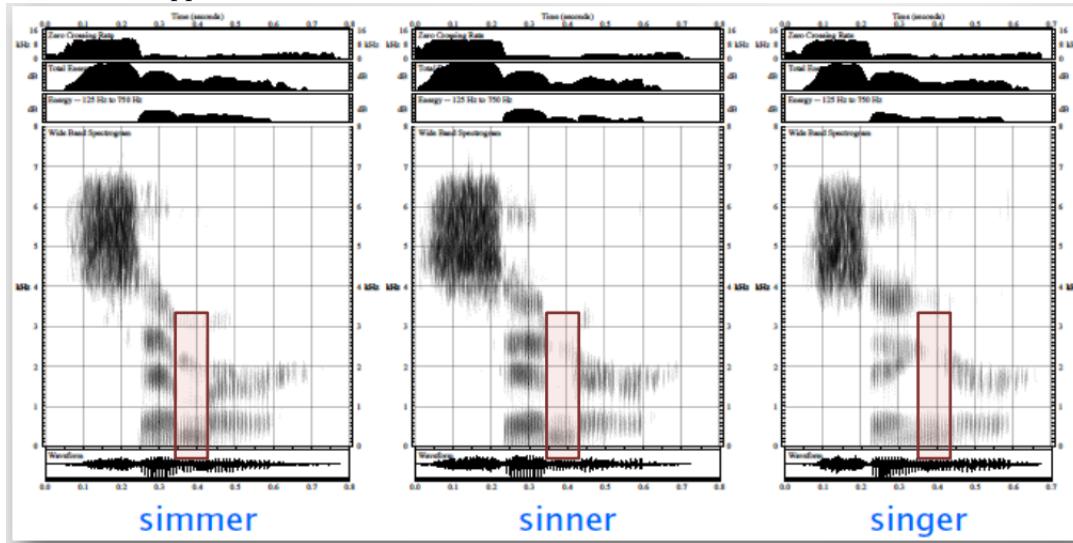


- Despite a common vowel, the *motion* of  $F_2$  and  $F_3$  into and out of the vowel helps identify the plosive
- Voiced stops are usually characterized by a *voice bar* during closure, indicating the vibrating vocal folds



## Nasals

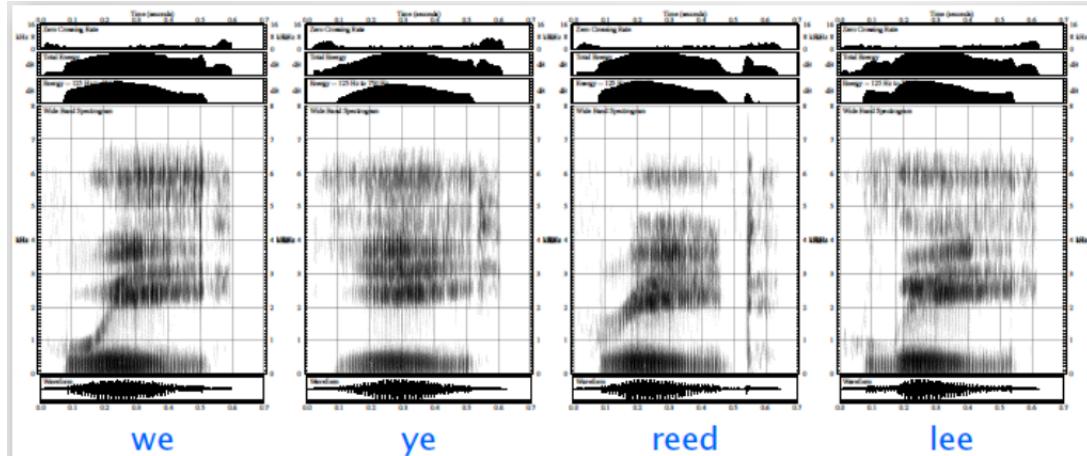
- Involve lowering the velum so that air passes through the **nasal cavity**
- Closures in the oral cavity change the resonant characteristics of the nasal sonorant
- Nasals often appear as 2 formants



## Semivowels

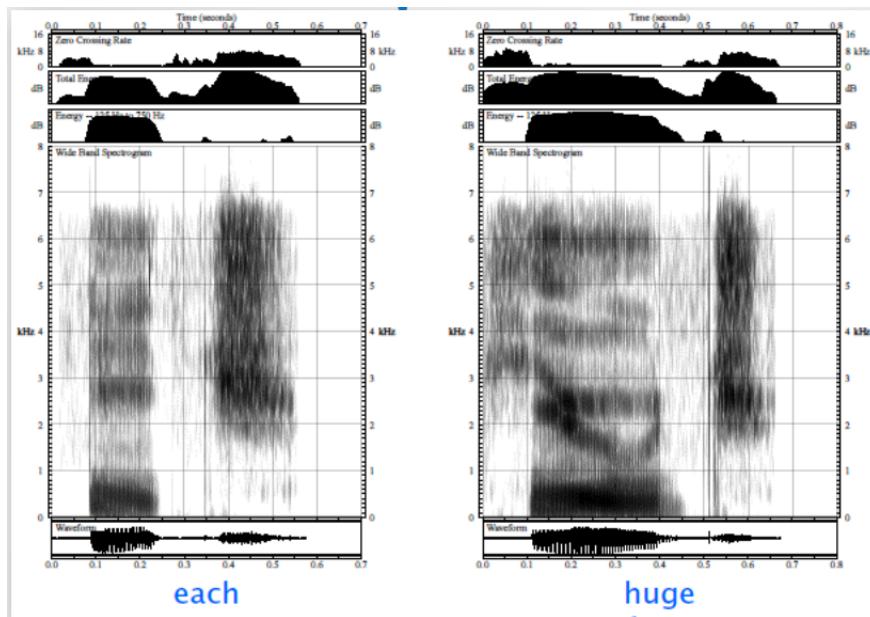
- Act as consonants in syllables and involve constriction in the vocal tract, but there is less turbulence
- Involve slower articulatory motion

- **Laterals** involve airflow around the sides of the tongue
- There is typically a drastic formant transition



### Affricates and Aspirants

- Two affricates: [jh] (voiced, e.g. judge), and [ch] (unvoiced, e.g. church)
  - Involve an *alveolar stop* followed by a *fricative*
  - Voicing in [jh] is normally indicated by voice bars, as with plosives
- One aspirant in Canadian English: [h]
  - Involves turbulence generated at the *glottis*
  - In Canadian English, there is no constriction in the vocal tract



### Other Topics in Phonetics

- The grouping of phones into **syllables**
  - Consisting of a vowel (**nucleus**), and optionally preceding (**onset**) and succeeding (**coda**) consonants

- Only certain sequences are permissible in English
- Syllables may be made more prominent via pitch, duration, or loudness
- The **prosody**, or intonation and rhythm, of an utterance
  - Prominence can indicate phrase boundaries
  - Gradual  $F_0$  movement (**tune**) can indicate a question or statement

## 16 Speaker Classification

Problems With Spectrograms

- Too big
  - Discrete signal is usually 16000 samples/sec
  - 100 frames/sec times 400 samples/frame gives 40000 samples/sec
- Too linear
  - Pitch perception is log-linear (Mels)
  - Lots of coefficients wasted on high frequencies
- Too entangled
  - Speaker and phoneme info is correlated

Filtering

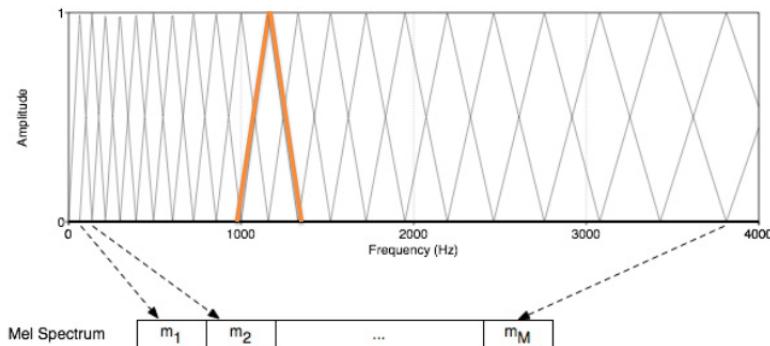
- Can filter the spectrum using filters from a *filter bank*
- Each filter is a signal whose spectrum  $F_m \in \mathbb{R}^N$  picks out a small range (or **band**) of frequencies
- The bands of the  $M$  filters are overlapping and span the spectrum
- A **filter coefficient** is computed as the log of the dot product of the *magnitude* of the frame  $X_t$  and filter  $F_m$  spectra

$$c_{t,m} = \log \sum_{n=1}^N |X_t[n]| \cdot F_m[n]$$

- If there are  $T$  frames, this gives us a real-valued feature matrix of size  $T \times M$

The Mel-Scale Filter Bank

- Called the **f-bank**
- The filter's vertices are arranged along the mel-scale
- Ascending frequency  $\Rightarrow$  wider bands

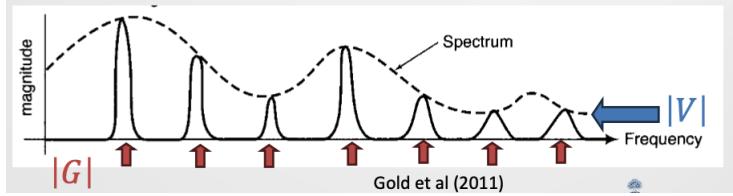


Source-Filter Model

- In vowels, the sound signal emitted from the glottis  $g$  is filtered by the vocal tract  $v$
- The **source-filter model** of speech assumes

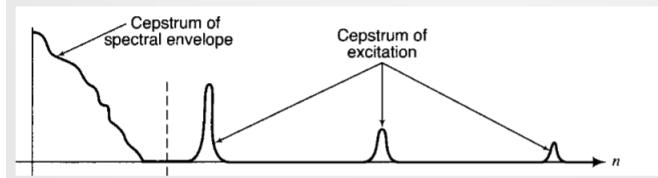
$$|X[n]| = |G[n]| \cdot |V[n]|$$

- $|V|$  is responsible for the smooth shape (envelope)
- $|G|$  is responsible for all the bumps ( $F_0$  harmonics)



### Cepstrum

- Can get  $|V|$  by computing the **cepstrum**  $\hat{x}$
- The cepstrum is  $\log |X|$  transformed by the inverse DFT
- Because  $\log |X| = \log |G| + \log |V|$ , and  $DFT^{-1}$  is linear, we have  $\hat{x}[n] = \hat{g}[n] + \hat{v}[n]$
- Since  $DFT^{-1} \approx DFT$ ,  $\hat{x}$  is like the “spectrum” of  $\log |X|$
- $|V|$  is lower-moving than  $|G|$ , so  $\hat{v}[n]$  is higher for lower  $n$  (i.e. lower frequency of frequency)



### Mel-Frequency Cepstral Coefficients (MFCCs)

- **MFCCs** are the coefficients of the cepstrum of F-bank coefficients

Frame  $\xrightarrow{DFT}$  Spectrum  $\xrightarrow{\log|\cdot|}$  Log magnitude spectrum  $\xrightarrow{\text{filter}}$  F-bank coefficients  $\xrightarrow{DFT^{-1}}$  MFCCs

- MFCCs are useful for models which cannot handle speaker correlations themselves (e.g. GMMs)
- F-banks are better for those which can (e.g. NNs)

### Speaker Classification

- All of the speech produced by one *speaker* will cluster differently in the *Mel space* than speech from another speaker
- **Speaker classification:** picking the most likely speaker among several speakers given only acoustics
- Each speaker will produce speech according to *different* probability distributions
- Can train a statistical model, given annotated data
- Choose the speaker whose model gives the highest probability for an observation
- We need to fit continuous probability functions to a discrete number of observations

### Univariate (1D) Gaussians

- Also known as **normal** distributions,  $N(\mu, \sigma)$

$$P(x; \mu, \sigma) = \frac{\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}{\sqrt{2\pi}\sigma}$$

- The parameters we can modify are  $\theta = (\mu, \sigma^2)$  (mean and variance)

### Maximum Likelihood Estimation

- Given data  $X = \{x_1, \dots, x_n\}$ , MLE produces an estimate of parameters  $\hat{\theta}$  by *maximizing the likelihood*  $L(X, \theta)$ :

$$\hat{\theta} = \arg \max_{\theta} L(X, \theta)$$

where

$$L(X, \theta) = P(X; \theta) = \prod_{i=1}^n P(x_i; \theta)$$

- To find the highest likelihood, we look at the derivative

$$\frac{\partial}{\partial \theta} L(X, \theta) = 0$$

- For univariate Gaussians:

$$\begin{aligned}\hat{\mu} &= \frac{1}{n} \sum_i x_i \\ \hat{\sigma}^2 &= \frac{1}{n} \sum_i (x_i - \mu)^2\end{aligned}$$

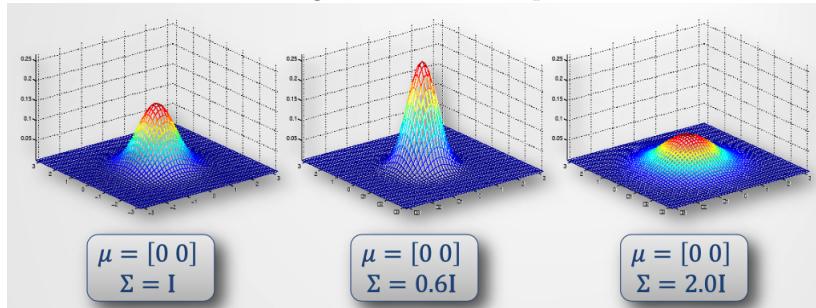
### Multivariate Gaussians

- When data is  $d$ -dimensional, the input variable is  $\vec{x} = (x[1], \dots, x[d])$
- The mean is  $\vec{\mu} = \mathbb{E}[\vec{x}] = (\mu[1], \dots, \mu[d])$
- The covariance matrix is  $\Sigma$  where  $\Sigma[i, j] = \mathbb{E}[x[i]x[j]] - \mu[i]\mu[j]$

$$P(\vec{x}) = \frac{\exp\left(-\frac{(\vec{x}-\vec{\mu})^\top \Sigma^{-1}(\vec{x}-\vec{\mu})}{2}\right)}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}}$$

### Covariance

- As values in  $\Sigma$  become larger, the Gaussian spreads out



- Different values on the diagonal results in different variances in their respective dimensions

### Gaussian Mixture Models

- Speech data are generally not unimodel, so fitting 1 Gaussian would not be representative

- **Gaussian mixture models (GMMs)** are a *weighted* linear combination of  $M$  component Gaussians,  $(\Gamma_1, \dots, \Gamma_M)$

$$P(\vec{x}) = \sum_{j=1}^M P(\Gamma_j) P(\vec{x} \mid \Gamma_j)$$

- Assume that MFCC dimensions are independent of one another, i.e. the covariance matrix is diagonal
- The probability of an observation becomes

$$P(\vec{x} \mid \Gamma_j) = \frac{\exp \left( -\frac{1}{2} \sum_{i=1}^d \frac{(x[i] - \mu_m[i])^2}{\Sigma_m[i]} \right)}{(2\pi)^{\frac{d}{2}} \left( \prod_{i=1}^d \Sigma_m[i] \right)^{\frac{1}{2}}}$$

- GMM “first” chooses a Gaussian, “then” emits an observation from that Gaussian
- Let  $\omega_m = P(\Gamma_m)$  be the weight of the  $m$ th Gaussian
- Let  $b_m(\vec{x}_t) = P(\vec{x}_t \mid \Gamma_m)$  be the component observation likelihood
- Then

$$P_\theta(\vec{x}_t) = \sum_{m=1}^M \omega_m b_m(\vec{x}_t)$$

where  $\theta = (\omega_m, \vec{\mu}_m, \Sigma_m)$  for  $m = 1, \dots, M$

- To estimate  $\theta$ , we solve  $\nabla_\theta \log L(X, \theta) = 0$  where

$$\log L(X, \theta) = \sum_{t=1}^T \log P_\theta(\vec{x}_t) = \sum_{t=1}^T \log \left( \sum_{m=1}^M \omega_m b_m(\vec{x}_t) \right)$$

- We can optimize for  $\mu_m[n]$

$$\begin{aligned} \frac{\partial \log L(X, \theta)}{\partial \mu_m[n]} &= \sum_{t=1}^T \frac{\partial}{\partial \mu_m[n]} \log \left( \sum_{m'=1}^M \omega_{m'} b_{m'}(\vec{x}_t) \right) \\ &= \sum_{t=1}^T \frac{1}{P_\theta(\vec{x}_t)} \frac{\partial}{\partial \mu_m[n]} \omega_m b_m(\vec{x}_t) \\ &= \sum_{t=1}^T \frac{\omega_m b_m(\vec{x}_t)}{P_\theta(\vec{x}_t)} \left( \frac{x_t[n] - \mu_m[n]}{\Sigma_m[n]^2} \right) \\ &= 0 \\ \mu_m[n] &= \frac{\sum_{t=1}^T \frac{\omega_m b_m(\vec{x}_t)}{P_\theta(\vec{x}_t)} x_t[n]}{\sum_{t=1}^T \frac{\omega_m b_m(\vec{x}_t)}{P_\theta(\vec{x}_t)}} = \frac{\sum_{t=1}^T P_\theta(\Gamma_m \mid \vec{x}_t) x_t[n]}{\sum_{t=1}^T P_\theta(\Gamma_m \mid \vec{x}_t)} \end{aligned}$$

But this term involves  $\mu_m[n]$

- If we knew which Gaussian generated each sample, then  $(\vec{\mu}_m, \Sigma_m)$  can be learned by MLE
- The MLE of  $P(\Gamma_j)$  is the count  $\frac{\#\vec{x}_t \text{ from } \Gamma_j}{T}$ , which we don't know

- We can guess at “soft” mixture assignments  $P_\theta(\Gamma_m \mid \vec{x}_t)$  from another model, which can be what we got from the previous round of maximization

Expectation-Maximization

1. Initialize a set of model parameters
2. *Expectation*: compute the expected probabilities of observation, given these parameters

$$P_\theta(\Gamma_m \mid \vec{x}_t) = \frac{\omega_m b_m(\vec{x}_t)}{P_\theta(\vec{x}_t)}$$

3. *Maximization*: update the parameters to maximize the aforementioned probabilities

$$\begin{aligned}\vec{\mu}_m^{MLE} &= \frac{\sum_t P_\theta(\Gamma_m \mid \vec{x}_t) \vec{x}_t}{\sum_t P_\theta(\Gamma_m \mid \vec{x}_t)} \\ \Sigma_m^{MLE} &= \frac{\sum_t P_\theta(\Gamma_m \mid \vec{x}_t) \vec{x}_t^2}{\sum_t P_\theta(\Gamma_m \mid \vec{x}_t)} - (\vec{\mu}_m^{MLE})^2 \\ \omega_m^{MLE} &= \frac{1}{T} \sum_{t=1}^T P_\theta(\Gamma_m \mid \vec{x}_t)\end{aligned}$$

4. Repeat

Speaker Classification Using GMM EM

- For each speaker, we learn a GMM given all  $T$  frames of their training data

## 17 Automatic Speech Recognition

Automatic Speech Recognition

- Given an utterance recorded as a waveform, **automatic speech recognition (ASR)** (a.k.a. **speech-to-text (STT)**), transcribes it as a *sequence of tokens*, usually words

Aspects of ASR Systems

- Speaking style*: read speech vs spontaneous speech (i.e. contains dysfluencies such as stuttering, uh, like)
- Accent/dialect*: mass-deployed or highly localized
- Vocabulary*: small ( $< 20$  words) or large ( $> 50000$  words), depending on whether it's technical or conversational
- Channel*: cell-phone, noise-cancelling microphone, etc.

Speech Features

- Waveform inputs are very long
- Dilated *convolutional neural networks (CNNs)* can learn and process the waveform directly
- Speech embeddings/representations* can be learned with unsupervised objectives
- Can use the *f-bank* for preprocessing, where the result is a sequence of *speech feature vectors* spaced 10s of milliseconds apart in time

Neural Approach

- Given a sequence of speech features  $x = x_1, \dots, x_T$  where  $x_t \in \mathbb{R}^D$
  - Want a sequence of tokens (a transcription)  $y = y_1, \dots, y_U$  where  $y_u \in \{1, 2, \dots, V\}$ 
    - $y_u$  can be a character, word, phone, etc.
  - Can do encoder/decoder NMT
    - Source sequence  $F$  are now features
    - Target sequences  $E$  are now transcriptions
  - Can train with teacher forcing
- $$\mathcal{L} = - \sum_{u=1}^U \log P_\theta(y_u | y_{<u}, cX)$$
- When decoding, we approximate the hypothesis transcription  $y^* = \arg \max_y P_\theta(y | x)$  with the beam search algorithm
  - Can also use an external, autoregressive language model to rescore the hypotheses (*shallow fusion*)

$$\underbrace{\log P'_\theta(y_u | y_{<u}, x)}_{\text{total score}} \approx \underbrace{\log P_\theta(y_u | y_{<u}, x)}_{\text{encoder-decoder score}} + \lambda \underbrace{\log P_\xi(y_u | y_{<u})}_{\text{external LM score}}$$

- They are unsuited to *streaming* (real-time transcription)
- Performance suffers on long utterances

Monotonic Alignment

- In ASR, tokens are transcribed in the *same order* that they are uttered
- Can learn *monotonic alignments* between sequences of frames and tokens
  - This relies on *dynamic programming*

### Monotonic Forward Algorithm

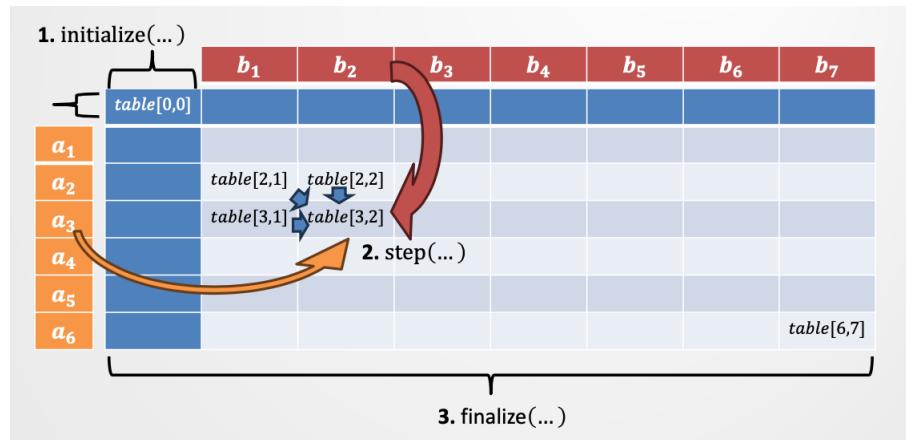
---

```

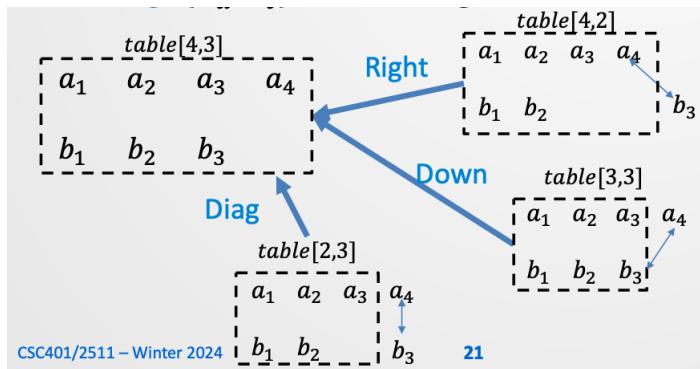
1   Inputs a = a_1, ..., a_U and b = b_1, ..., b_T
2     Define table[0:U, 0:T]
3       initialize(table[0:U, 0], table[0, 1:T])
4       for each u in 1:U do
5         for each t in 1:T do
6           table[u, t] = step(a_u, b_t, table[u-1, t-1], table[u-1, t], table[u, t-1])
7       return finalize(table[0:U, 0:T])

```

---

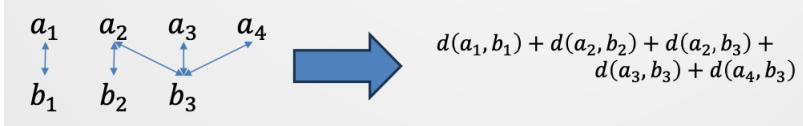


- Cells are populated in *nondecreasing* order of indices
- Alignments cannot cross
- $table[u, t]$  need only to consider how to correctly extend a prefix with  $\{a_u, b_t\}$
- The alignment  $\{a_u, b_t\}$  must be part of  $table[u, t]$  (otherwise we would have an alignment that crosses)
- The alignments in  $table[u, t]$  can extend an earlier prefix in one of 3 ways
  1. *Right*:  $b_t$  extends alignments in  $table[u, t - 1]$
  2. *Down*:  $a_u$  extends alignments in  $table[u - 1, t]$
  3. *Diagonal*:  $a_u$  and  $b_t$  extend alignments in  $table[u - 1, t - 1]$



## Template Matching

- Input utterances are compared against stored templates
- The transcription of the closest template is returned
- Let  $a \in \mathbb{R}^{U \times D}$  and  $b \in \mathbb{R}^{T \times D}$  be sequences of speech features (e.g. MFCCs, f-bank features, etc.)
- Choose some frame-wise (vector) distance function, e.g.  $d(a_u, b_t) = \sum_{d=1}^D |a_{u,d} - b_{t,d}|$
- Can sum those frame-wise distances in a monotonic alignment to get a “distance” between utterances



## Dynamic Time Warping

- Let  $\mathcal{A}$  be the set of possible ways to align  $a$  and  $b$
- In **dynamic time warping (DTW)**, the “distance” between  $a$  and  $b$  is that with the minimal frame-wise sum

$$\mathcal{D}_d(a, b) = \min_{\alpha \in \mathcal{A}} \sum_{\{u, t\} \in \alpha} d(a_u, b_t)$$

which can be computed with monotonic forward

- $a_u$  and  $b_t$  are the feature vectors of utterances  $a$  and  $b$
- Solution to a prefix is  $\text{table}[u, t] = \mathcal{D}_d(a_{1:u}, b_{1:t})$
- To build the initial prefix, we set  $\text{table}[1, 1] = d(a_1, b_1)$

- To extend a prefix, we set  $\text{table}[u, t] = d(a_u, b_t) + \min \begin{cases} \text{table}[u - 1, t - 1], \\ \text{table}[u - 1, t], \\ \text{table}[u, t - 1] \end{cases}$

- The final result is  $\mathcal{D}_d(a, b) = \text{table}[U, T]$

- Adopting monotonic forward:

---

```

1   initialize:
2     table[0, 0] = 0
3     table[1:U, 0] = table[0, 1:T] = inf
4
5   step:
6     table[u, t] = d(a_u, b_t) + min(table[u - 1, t - 1], table[u - 1, t], table[u, t - 1])
7
8   finalize:
9     return table[U, T]

```

---

## Word-Error Rate

- Can use **word-error rate (WER)** to evaluate hypothesis transcripts against gold-standard reference transcript
- Counts the *minimum* number of 3 types of errors a hypothesis makes given a reference:

- **Substitution error:** one word being mistook for another
- **Deletion error:** an input word that is “skipped”
- **Insertion error:** a “hallucinated” word not in the input
- Given  $S$  substitutions,  $D$  deletions,  $I$  insertions, and  $N$  reference tokens:

$$WER = \frac{S + D + I}{N} \times 100\%$$

- Notice that this is not a valid “percentage”

## Aligning Strings

- We can count errors by building up prefixes of alignments between reference  $a$  and hypothesis  $b$ 
  - Insertion: add  $b_t$  to hypothesis
  - Deletion: add  $a_u$  to reference
  - Substitution/Match: add  $b_t, a_u$
- Prepend reference and hypothesis with a special token  $\langle s \rangle$  to allow for insertions/deletions at the beginning of the reference/hypothesis
- $a_u$  and  $b_t$  are reference and hypothesis tokens
- Solution to a prefix is  $\text{table}[u, t] = \min \# \text{errors}(\langle s \rangle a_{1:u}, \langle s \rangle b_{1:t})$
- To build the initial prefixes, we set  $\text{table}[u, 0] = u$  and  $\text{table}[0, t] = t$  for  $0 \leq u \leq U$  and  $0 \leq t \leq T$

- To extend a prefix, we set  $\text{table}[u, t] = \min \begin{cases} \text{table}[u - 1, t] + 1, \\ \text{table}[u - 1, t - 1] + \delta_{a_u, b_t} \\ \text{table}[u, t - 1] + 1 \end{cases}$

- The final result is  $WER(a, b) = \frac{\text{table}[U, T]}{U} \times 100\%$

- Adopting monotonic forward:

---

```

1   initialize:
2     for u in 0:U do
3       table[u, 0] = u
4     for t in 0:T do
5       table[0, t] = t
6
7   step:
8     table[u, t] = min(table[u - 1, t] + 1,
9                         table[u - 1, t - 1] + (a_u == b_t),
10                        table[u, t - 1] + 1)
11
12 finalize:
13   return table[U, T] / U * 100%

```

---

## Marginalization

- Each frame gets a label by repeating transcript tokens, e.g. /ow ow ow ow ow p p p p ah ah ah ah n/
 $\implies /ow p ah n/$
- Let  $a_{1:U}$  be the *reference transcript* and  $b_{1:T}$  be a *frame-wise transcript* with repetitions of  $a$

- Can train a NN to maximize the likelihood of  $b$ , i.e.  $\mathcal{L} = -\log P_\theta(b)$ , but there are many choices of  $b$
  - We can maximize the likelihood of all such  $b$
  - Let  $\mathcal{B}$  be a function which *removes sequential repetitions* from  $b$ , e.g.  $\mathcal{B}(ABBA) = ABA$
  - Let  $\mathcal{B}^{-1}(a; T)$  be all  $b$  of length  $T$  which reduces to  $a$ , i.e.

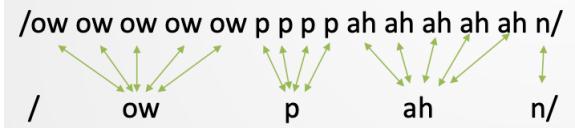
$$\mathcal{B}^{-1}(a; T) = \{b_{1:T} : \mathcal{B}(b) = a\}$$

- Can now minimize

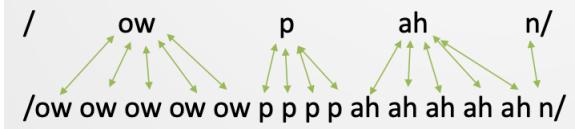
$$\mathcal{L} = -\log P_\theta(a) = -\log \sum_{b \in \mathcal{B}^{-1}(a; T)} P_\theta(b)$$

- There are  $\binom{T-1}{U-1}$  paths in  $\mathcal{B}^{-1}(a; T)$ 
    - To create  $U$  segments out of  $T$  frames, we need  $U - 1$  divisions
    - There are  $T - 1$  potential places to place those divisions

- Notice that if  $\mathcal{B}(b) = a$ , then a monotonic alignment exists between  $(a, b)$



- But not all monotonic alignments  $(a, b)$  imply  $\mathcal{B}(b) = a$



- Can fix this by disallowing “down” steps (i.e. adding  $a_u$  to prefix)
  - This essentially means that hypotheses are at least as long as the reference
  - This assumption says reference cannot contain repeated tokens, whereas the hypothesis can

## Partial Solutions

- For DTW and WER, `table[u, t]` keeps track of 1 optimal alignment per prefix pair  $a_{1..u}, b_{1..t}$ :

`table[u, t] = min #errors( $a_{1:y}, b_{1:t}$ )`

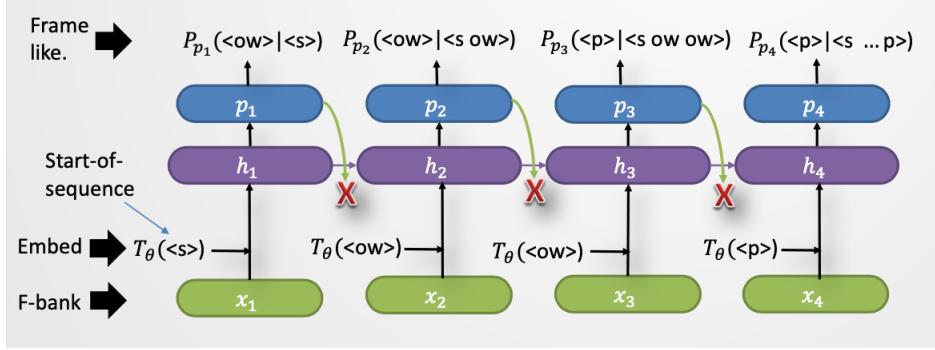
- For ASR, `table[u, t]` keeps track of *all* alignments per prefix pair  $a_{1..u}, b_{1..t}$ :

$$\text{table}[u, t] = \sum_{b_{1:t} \in \mathcal{B}^{-1}(a_{1:t}; t)} P_\theta(b_{1:t} \mid x)$$

- Then  $\mathcal{L} = -\log \text{table}[U, T]$

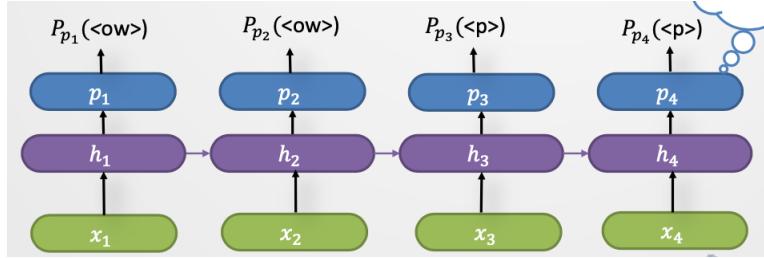
## An Autoregressive Approach

- Suppose we use an autoregressive RNN to generate frame-level predictions  $P_\theta(b_t | b_{1:t-1}, x)$
  - We cannot use monotonic forward because the frame-wise likelihoods depend on a specific prefix



- To be able to use monotonic forward, we would require elements of  $b$  to be *conditionally independent* given  $x$ , i.e.

$$P_\theta(b | x) = \prod_{t=1}^T P_{p_t}(b_t)$$



- The token embeddings are dropped

### Extending Prefixes

- For  $b_{1:t} \in \mathcal{B}^{-1}(a_{1:u}; t)$ , let
  - $b_{1:t}^{u-1}$  denote a prefix where  $a_u$  is first aligned to  $b_t$ 
    - \*  $a_u$  aligns to  $b_t$  and  $a_{u-1}$  aligns to  $b_{t-1}$
  - $b_{1:t}^u$  denote a prefix where  $a_u$  has already been aligned
    - \*  $a_u$  aligns to  $b_t$  and  $a_u$  aligns to  $b_{t-1}$
- $a_u$  aligns to  $b_t$  and  $b_{t-1}$  aligns to  $a_u$  or  $a_{u-1}$
- Using conditional independence:

$$\begin{aligned} \text{table}[u, t] &= \sum_{b_{1:t} \in \mathcal{B}^{-1}(a_{1:u}; t)} P_\theta(b_{1:t} | x) \\ &= P_{p_t}(b_t = a_u) \sum_{b_{1:t} \in \mathcal{B}^{-1}(a_{1:u}; t)} P_\theta(b_{1:t-1} | x) \\ &= P_{p_t}(b_t = a_u) \left( \sum_{b_{1:t}^{u-1} \in \mathcal{B}^{-1}(a_{1:u-1}; t-1)} P_\theta(b_{1:t-1}^{u-1} | x) + \sum_{b_{1:t}^u \in \mathcal{B}^{-1}(a_{1:u}; t-1)} P_\theta(b_{1:t-1}^u | x) \right) \\ &= P_{p_t}(b_t = a_u) (\text{table}[u - 1, t - 1] + \text{table}[u, t - 1]) \end{aligned}$$

- $a_u$  and  $b_t$  are reference and frame-level tokens
- Solutions to a prefix is  $\text{table}[u, t] = \sum_{b_{1:t} \in \mathcal{B}^{-1}(a_{1:u}; t)} P_\theta(b_{1:t} | x)$

- To build the initial prefixes, we set  $\text{table}[0, 0] = 1$ , and  $\text{table}[0, 1:T] = \text{table}[1:U, 0] = 0$
- To extend a prefix, we set  $\text{table}[u, t] = P_{p_t}(b_t = a_u) (\text{table}[u - 1, t - 1] + \text{table}[u, t - 1])$
- The final result is  $\mathcal{L} = -\log \text{table}[U, T]$
- Adopting monotonic forward:

---

```

1   initialize:
2       table[0, 0] = 1
3       table[0, 1:T] = table[1:U, 0] = 0
4
5   step:
6       table[u, t] = P_p_t (b_t = a_u)(table[u - 1, t - 1] + table[u, t - 1])
7
8   finalize:
9       return -log table[U, T]

```

---

## Inference

- We can compute an error signal for  $\mathcal{L} = -\log P_\theta(a | x)$
- At test time, we want to generate transcriptions  $a$
- Computing  $b^* = \arg \max_b P_\theta(b | x)$  is easy

$$b^* = \arg \max_b \prod_{t=1}^T P_{p_t}(b_t) \implies b_t^* = \arg \max_{b_t} P_{p_t}(b_t)$$

- Computing  $a^* = \arg \max_a P_\theta(a | x)$  is hard
  - We use DP to compute  $P_\theta(a | x)$  once
  - For vocab size  $V$ , there are  $V(V^T - 1)/(V - 1)$  possible  $a$
  - Vanilla beam search on  $b$  will just return  $b^*$

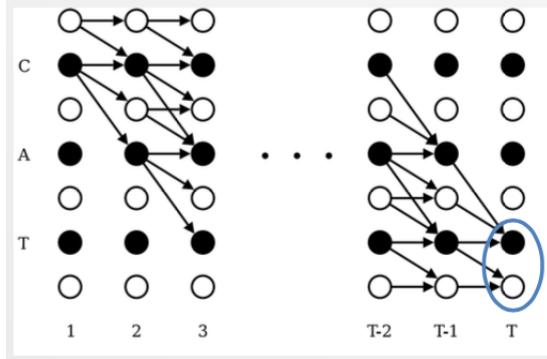
## Prefix Search

- Use a modified beam search called **prefix search**
- Keeps track of  $K$  prefixes of  $a$  (not  $b$ )
- For each frame  $t$ :
  - Extend each prefix  $a^{(k)}$  with each element of the vocabulary  $v$  and frame-level likelihoods  $P_{p_t}(v) = a^{(k,v)}$
  - Collapse any repeats in extensions using  $\mathcal{B}$ , summing the likelihoods of matching prefixes:  $\mathcal{B}(a^{(k,v)}) = \mathcal{B}\left(a^{(k',v')}\right)$
  - Pick the top  $K$  most likely extensions as new  $a^{(k)}$

## Dealing With Doubles

- Sometimes  $a$  has natural repetitions (e.g. hesitation, “I- I am...”)
- **Connectionist temporal classification (CTC)** fixes this by introducing a *blank token*  $\epsilon$  to the vocabulary

- $\mathcal{B}$  removes  $\epsilon$  after removing duplicates, e.g.  $\mathcal{B}([\text{ey ey } \epsilon \epsilon \text{ ey } \epsilon \text{ ae } \epsilon \epsilon \epsilon \text{ m}]) = [\text{ey ey ae m}]$
- Monotonic forward cannot handle blanks (since there are optional  $\epsilon$  between each reference token)
- Three step solution
  1. Add blanks between reference tokens, e.g.  $\langle C \ A \ T \rangle \mapsto \langle \epsilon \ C \ \epsilon \ A \ \epsilon \ T \ \epsilon \rangle$
  2. Add extra diagonal dependency to skip blanks
  3.  $\mathcal{L} = -\log(\text{table}[U - 1, T] + \text{table}[U, T])$



## 18 Information Retrieval

### Information Retrieval

- Given a *query*, search for the most relevant *document* among a *knowledge base*
  - How to represent the query?
  - How to store a knowledge base?
  - How to search efficiently and accurately?

### Structured Query Language (SQL)

- Use SQL queries to represent the query
- Use tabular entries with predefined schemas as the knowledge base
- Compile and execute SQL queries to search

### Max-Similarity Search

- Use a text-based document to represent the query
- Use *vectorized* documents as the knowledge base
- To search, compute the *similarity score* between the query and each document, return the document with the highest similarity score
- Can use *cosine distance* as the similarity score
  - Vectors that are *near* each other (within a certain *angular radius*) are considered relevant

### tf.idf

- **Term frequency**  $tf_{ij}$ : number of occurrences of word  $w_i$  in document  $d_j$ 
  - Higher values suggest that word  $w_i$  is a good indicator of the content of document  $d_j$
  - When considering the relevance of a document  $d_j$  to a keyword  $w_i$ ,  $tf_{ij}$  should be maximized
  - We often *dampen*  $tf_{ij}$
$$tf_{dampen} = 1 + \log(tf) \quad \text{if } tf > 0$$
- **Document frequency**  $df_i$ : number of documents in which word  $w_i$  appears
  - Meaningful words may occur repeatedly in a related document
  - Functional (or less meaningful) words may be distributed evenly over all documents
  - Very specific words,  $w_i$ , would give smaller values of  $df_i$
- **Inverse document frequency**  $idf_i = \log\left(\frac{D}{df_i}\right)$ , where  $D$  is the total number of documents
  - This measure gives *full* weight to words that occur in 1 document, and *zero* weight to words that occur in all documents
  - The probability of a document containing word  $i$  is  $\frac{df_i}{D}$ ; small probability means the event is more surprising, therefore there is more information
  - $idf_i$  is the amount of *information* provided by observing the event
- **Collection frequency**  $cf_i$ : total occurrences of  $w_i$  in the collection

- We combine the term frequency and the inverse document frequency to give us a joint measure of *relatedness* between words and documents

$$tf.idf(w_i, d_j) = \begin{cases} (1 + \log(tf_{ij})) \log \frac{D}{df_i}, & \text{if } tf_{ij} \geq 1 \\ 0, & \text{if } tf_{ij} = 0 \end{cases}$$

- Can we represent the  $j$ th document as a vector

$$\begin{bmatrix} tf.idf(w_1, d_j) \\ tf.idf(w_2, d_j) \\ \vdots \\ tf.idf(w_{|W|}, d_j) \end{bmatrix}$$

Semantic Doc2Vec

- Positive samples  $d_+$  for a document are *augmented* following some heuristics
- Negative samples  $d_-$  are *randomly sampled* from within the batches
- A contrastive loss objective is

$$L(q, d_+, d_-) = \frac{\exp\left(\frac{\text{sim}(E(q), E(d_+))}{\tau}\right)}{\exp\left(\frac{\text{sim}(E(q), E(d_+))}{\tau}\right) + \sum_i \exp\left(\frac{\text{sim}(E(q), E(d_-))}{\tau}\right)}$$

where  $\tau$  is the temperature of the softmax

Precision and Recall

- **Precision:**  $\frac{N_{\text{relevant and retrieved}}}{N_{\text{retrieved}}}$ 
  - Among all *retrieved* documents, how many are relevant
  - In ML:  $\frac{TP}{P}$
- **Recall:**  $\frac{N_{\text{relevant and retrieved}}}{N_{\text{relevant}}}$ 
  - Among all *relevant* documents, how many are retrieved
  - In ML:  $\frac{TP}{T}$

F-Measure

- **F-measure** is the weighted harmonic mean of precision and recall

$$F = \frac{1}{\alpha \frac{1}{p} + (1 - \alpha) \frac{1}{r}}$$

- $p$  is precision
- $r$  is recall
- $\alpha \in [0, 1]$
- If either of precision or recall is 0 (i.e. true positive count  $TP = 0$ ), then we set  $F = 0$

Precision at  $k$

- Even if the 1st result is not relevant, the 2nd, etc. result could be relevant too
- **Precision at  $k$**  (P@k): among the top  $k$  results, how many are relevant

- Problems
  - The 1st, 2nd, ...  $k$ th locations have no differences
  - If there are less than  $k$  relevant results, then it's impossible to get  $P@k = 1$

### Structured Relational Data

- Plain texts are *unstructured*
- Many IR systems use *structured* data
  - E.g. documents vectorized to the same dimensions
  - E.g. relational data
- Can save each complex object as a database entirely

### Multimodal Data

- Contain more than texts
- Images, sounds, videos are also stored

## 19 Text Summarization

### Text Summarization

- Objective: return shortened version of text that include its main points
- Types
  - **Format:** text vs. template
  - **Perspective:** informative vs. indicative
  - **Composition:** extract vs. abstract
  - **Orientation:** document vs. query
  - **Source:** single vs. multiple document
  - **Background:** complete vs. update

### Summarization by Extraction

- Identify important information, and drop it into summary
- To determine importance:
  - Position in text
    - \* First sentence in each paragraph
    - \* First and last paragraphs of document
    - \* Section headings, captions, etc.
    - \* Varies with genre
    - \* Hovy-Lin (partial) ordering
  - Indicators
    - \* Cues, e.g. “in conclusion”, “in this paper, we show”
    - \* Clues (bonus words), e.g. “significantly”, “this paper”
    - \* Stigma words, e.g. “hardly”, “incidentally”
    - \* Content words from title

### Naive Bayes Classification

- Can treat summarization as a *sequence of binary classification problems*: every sentence is either *in* or *out*
- Choose outcome that is most probable in given context of features

$$\max \{\Pr(s \in \text{Summary} \mid f_1 \cdots f_k), \Pr(s \notin \text{Summary} \mid f_1 \cdots f_k)\}$$

- Use Bayes' rule on  $\Pr(o \mid f_1 \cdots f_k)$

$$\Pr(o \mid f_1 \cdots f_k) = \frac{\Pr(f_1 \cdots f_k \mid o) \Pr(o)}{\Pr(f_1 \cdots f_k)}$$

- Naive Bayes assumption: all features of context are conditionally independent

$$\Pr(f_1 \cdots f_k \mid o) = \prod_{1 \leq j \leq k} \Pr(f_j \mid o)$$

- Can use relative frequency in annotated corpora to estimate probabilities

$$\Pr(f_j \mid o) \approx \frac{C(f_j, o)}{C(o)}$$

#### Disadvantages of Summarization by Extraction

- Hard to read, misleading, and/or incoherent
  - Lost pronoun antecedents
  - Discourse/argument connectives no longer appropriate
- Parts of extracted sentences may be unimportant
  - Negation (of cues and stigma words)
  - Granularity of sentence-sized extracts
- Potential improvements
  - Abstraction
  - Use argument structure to determine importance and conservatively synthesize new text
  - Summarize multiple documents/background collection and use comparisons to boost confidence in importance
  - Task-based evaluation: determine how well summaries work in context

#### Attention for Summarization

- Use an encoder/decoder NMT model

#### Text-To-Text Generation

- **Sentence compression:** shorten sentences
- **Sentence fusion:** merge parts of similar sentences

#### Sentence Enhancement

- Traditional fusion: align similar core sentences
- We can expand with parts of dissimilar in-domain sentences

#### Modern Summarization

- Automatic summaries are already more central than human-written gold standard summaries
  - Centrality refers to the concept of identifying the most central or important pieces of information within a document

## 20 Anthropomorphization

### Anthropomorphism

- A system that has features that make it appear to be human is **anthropomorphic**
  - *Anthropos*: human
  - *Morphe*: form
- When creating programs that produce text or speech, we have a choice of whether to anthropomorphize it, i.e. make it seem human

### Anthropomorphic Cues

- Voice
  - Accent or tone, e.g. warmth, breathiness
  - Disfluencies, e.g. pauses, hesitations, filler words (like “um”)
- Text
  - Slang or informal speech
- Content
  - The system may answer questions about itself, e.g. confirming it's not a human
  - Use of first-person pronouns implicitly claim of having certain capabilities (e.g. personality, emotions)
  - Explicit claims of being human
- Embodiment
  - We treat systems with human faces and bodies as more human-like

### Cognitive Origins of Anthropomorphization

- Humans naturally tend to treat things as human, even when we know they are not human
- Our tendency to do so has 2 origins
  1. Sociality
    - We mentally construct systems as humanlike to fulfill a need for social connection
    - Users are more inclined to trust systems with which they feel a social connection
    - E.g. pets
  2. Effectance
    - Sometimes we encounter systems that we don't fully understand; we don't know how they will behave
    - We model these systems like things that we better understand
    - Helps us make predictions about how they behave and how we should interact with them
    - When we model unknown systems, we usually know that the unknown system is not what we are modelling it as (applies to pets)
- If we model a system as a human being, we tend to attribute some uniquely human abilities to it
  - E.g. memory, emotions, rational inference, ability to understand questions

- When the modelled system actually has these capabilities, this can assist the user in using the system
- When the modelled system does not actually have these capabilities, this can harm the user

#### Ethics of Anthropomorphization

- **Deception:** the intentional attempt to produce a false belief in someone
- **Lying:** uttering a sentence believed to be false with the intention of producing a false belief in someone
- The designer of an anthropomorphic system may have the intention of deceiving the user that it was human
  - Defense: people are not entitled to know certain information
  - E.g. it's maybe OK to lie if someone asks you a personal question
- Without deception, it's sometimes “wrong” to anthropomorphize
  - E.g. anthropomorphized gambling slot machines
- Why are actions “wrong”?
  - Consequences (e.g. harm)
  - The nature of those actions themselves are just wrong (e.g. manipulation, deception, or exploitation)

#### Legal and Moral Rules for Anthropomorphization

- There is a spectrum between “anthropomorphization techniques are never ethically OK” and “anthropomorphization techniques are always ethically OK”
- Some laws against anthropomorphization that incentivize a purchase or sale of goods, or influence a vote in an election