CS183 Review Speech

Lec12 Speech Intro

From Linguistic part, we know how to voice 2 letter:

Acoustic Wave → Spectrogram → phoneme $\xrightarrow{\text{lexicon}}$ word

lexicon: 发音字典 E.g. G OW T UW → "go to"

From letters to sounds

- Pronunciation dictionaries (often made by linguists) give the syllables and phonemes within each word in vocabulary
  - CMU Phonetic Dictionary gives the syllabic and phonetic spellings for >110K words in English
  - ML based phonetizers are built on such phonetic dictionaries

Graphemes	She	just	had	a	baby
IPA	ʃiː	dʒʌst	hæd	ə	ˈbeɪbi
Arpabet	sh iy	jh ah s t	h ae d	ah	b ey b iy

Arpabet is an ASCII friendly representation of IPA

- Phonology are the grammatical rules that phonemes of a language follow
- Lexical Phonology: Study of rules that govern the organization of sounds in a language (Phonemes —> Syllables —> Words)
  - Lexical Stress (project (noun) vs project (verb))
  - Allophones
    - (*r* and *l* in Japanese; *p* and *b* in Arabic; *t* and *k* in Hawaiian)
  - Phonological changes in continuous speech
    - Westside vs Westend
- Intonational Phonology: Study of the Fundamental Frequency (F0) in relation to the intended meaning of an utterance
  - *I never said she stole my money* (Emphasize each word for different meanings)

⇑

Letter 2 sound 音素学: how 音素 → 音节 → word

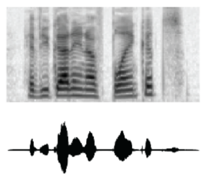
What technologies fall under spoken language research ?

- Speech recognition
- Speech synthesis
- Voice conversion
- Speaker recognition
- Language recognition
- Speech emotion recognition
- Speaker diarization
- Speech coding
- Speech perception
- Speech enhancement
- Microphone array processing
- Audio event classification and detection
- Speech separation
- Spoken language understanding
- Spoken dialogue systems
- Speech translation
- Multimodal processing

Have you got enough blankets? — sentence / meaning / word

hɪv y gɑr ɪɛ ʌf blæŋ kɪts — syllable

hɪvygɑrɪɛʌfblæŋkɪts — phoneme



spectrotemporal (frequency decomposition)

acoustic vibration

Now we're interested in Speech Recognition. (Autonomous one: ASR)

From the figure: can be one solution. Challenges

Another early traditional solution: Template Matching Factor

COMPLEXITY	
amount of data:	typically 32000 bytes per second (16khz)
class inventory:	50 phonemes, 5000 sounds, 100.000 words
combinatorial explosion:	exponential growth of possible sentences

SEGMENTATION	
our perception:	Phones, syllables, words, sentences
actually there are:	no boundary markers, continuous flow of samples

VARIABILITY	
speaker:	anatomy of vocal tract, speed, loudness, acoustic stress, mood, dialect, speaking style, context
channel, environment:	noise, microphones, channel conditions

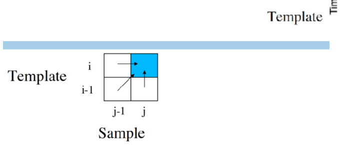
AMBIGUITY	
Homophones:	two vs. too,
Word Boundaries:	interface vs. in her face,
Semantics:	He saw the Grand Canyon flying to New York,
Pragmatics:	Time flies like an arrow.

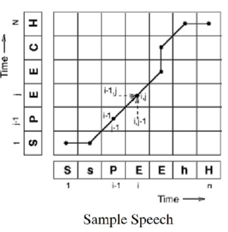
Some applications only have limited vocabulary — Voice Dialing System

⇑ Application Scenario Map utterance to Template.

- We change durations
  - two utterances are never the same

Dynamic Time Warping



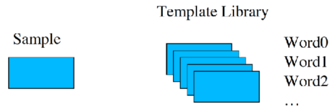


- *For each square*
  - *Dist(template[i],sample[j]) + smallest_of (Dist(template[i-1],sample[j]) Dist(template[i],sample[j-1]) Dist(template[i-1],sample[j-1])*

Remember which choice your took (count path)

Also, can be multi-template with speaker ID combined

- Compare against each
- Find closest
- Need to normalize scores
  - (divide by length of matches)



For Word in Templates
    Score = dtw(Template[Word], Sample);
    if (Score < BestScore)
        BestWord = Word;
DoAction(Action[BestWord])

- Advantages
  - *Works well for small number of templates (<20)*
  - *Language independent*
  - *Speaker specific*
  - *Easy to train (end user controls it)*
- Disadvantages
  - *Limited number of templates*
  - *Speaker specific*
  - *Need actual training examples*

- Distance metric
  - Euclidean $\sqrt{\sum_{i=0}^{N}(T_i - S_i)^2}$
- But some distances are bigger than others
  - Silence is pretty similar
  - Fricatives are quite larger
    - A longer fricative might give large score
    - A longer vowel might give smaller score

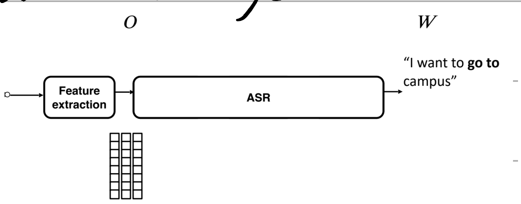
- Instead of Euclidean distance
  - Doesn't care about the standard deviation

$$\sqrt{\sum_{i=0}^{N}(T_i - S_i)^2}$$

- Use Mahalanobis distance
  - Care about means and standard deviation

$$\sqrt{\sum_{i=0}^{N}\left(\frac{(\mu_i - S_i)}{\sigma_i}\right)^2}$$

But nowadays: ASR. Here we introduce non-end-to-end ASR



- Instead of starting from the waveform, we will often start from **speech features** (MFCC, etc.) through the **feature extraction** module
- Let's think of the conversion from speech feature o to text w

- **MAP decision theory**: Estimate the most probable word sequence $\hat{W}$ among all possible word sequences $\mathcal{W}$ (I'll omit the domain sometimes)

$$\hat{W} = \arg\max_{W \in \mathcal{W}} p(W|O)$$

We handle acoustic waveform's feature. We want $\arg\max_{W} p(W|O)$.

- Please specify the domain of variables
  - D-dimensional continuous vector: $\mathbf{o} \in \mathbb{R}^D$
  - $(D \times D)$-dimensional matrix: $\mathbf{\Sigma} \in \mathbb{R}^{D \times D}$
  - Word with vocabulary $\mathcal{V}$: $w \in \mathcal{V}$
- Set: calligraphic font, upper case, a set of elements are represented with curly brackets
  $\mathcal{V} = \{\text{"one"}, \text{"two"}, \text{"three"}, \cdots\}$
- Sequence: italic font, upper case, a sequence of elements are represented with round brackets
  $O = (\mathbf{o}_1, \mathbf{o}_2, \cdots)$ $O = (\mathbf{o}_t \in \mathbb{R}^D | t = 1, \cdots, T)$

- T-length speech feature sequence (D-dimensional vector)
  $O = (\mathbf{o}_t \in \mathbb{R}^D | t = 1, \ldots, T)$
- N-length word sequence with vocabulary $\mathcal{V}$
  $W = (w_n \in \mathcal{V} | n = 1, \ldots, N)$

Notations. Pipeline.

How to model $p(W|O)$?

- Factorize the model with **phoneme**
  - Let $L = (l_i \in \{/\mathrm{AA}/, /\mathrm{AE}/, \cdots\} | i = 1, \cdots, J)$ be a phoneme sequence

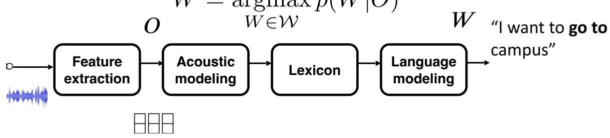
$$\arg\max_W p(W|O) = \arg\max_W \sum_L p(W, L|O)$$ **Sum rule**

$$= \arg\max_W \sum_L \frac{p(O|W,L)p(L|W)p(W)}{p(O)}$$ **Bayes+ Product rule**

$$= \arg\max_W \sum_L p(O|W,L)p(L|W)p(W)$$ Ignore $p(O)$ as it does not depend on W

$$= \arg\max_W \sum_L p(O|L)p(L|W)p(W)$$ **Conditional independence assumption**

$$\hat{W} = \arg\max_{W \in \mathcal{W}} p(W|O)$$



$$\arg\max_W p(W|O) = \arg\max_W p(O|W)p(W) \approx \arg\max_W \sum_L p(O|L)p(L|W)p(W)$$

- **Speech recognition**
  - $p(O|L)$: Acoustic model (Hidden Markov model)
  - $p(L|W)$: Lexicon
  - $p(W)$: Language model (n-gram)

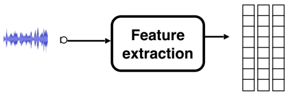
Three key component: $p(W)$, $p(L|W)$, $p(O|L)$

How to model them?

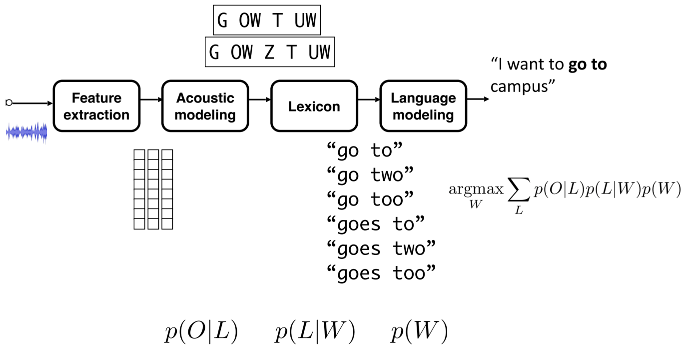
feature sequence vector

phoneme sequence

Waveform to speech feature

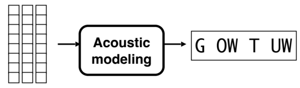


- Performed by so-called **feature extraction** module
  - Mel-frequency cepstral coefficient (MFCC), Perceptual Linear Prediction (PLP) used for **Gaussian mixture model (GMM)**
  - Log Mel filterbank used for **deep neural network (DNN)**
- Time scale
  - 0.0625 milliseconds (16kHz) to 10 milliseconds
- Type of values
  - Scalar (or discrete) to 12—40 dimensional vector



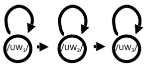
Speech feature to phoneme

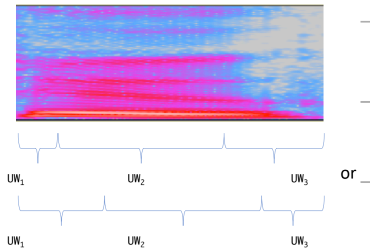
$p(O|L)$



- Performed by so-called **acoustic modeling** module
  - Hidden Markov model (HMM) with **GMM** as an emission probability function
  - Hidden Markov model (HMM) with **DNN** as an emission probability function
- Time scale
  - 10 milliseconds to ~100 milliseconds (depending on a phoneme)
- Type of values
  - 12-dimensional continuous vector to 50 categorical value (~6bit)
- The most critical component to get the ASR performance
- It can be a probability of possible phoneme sequences, e.g.,
  G OW T UW or G OW Z T UW with some scores

- O and L are **different lengths**
- **Align** speech features and phoneme sequences by using HMM

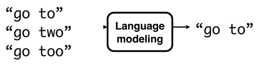




- Provide $p(O|L)$ based on this alignment and model
- The most important problem in speech recognition

Word to text

$p(W)$: Language model, like n-Gram



- Performed by **language modeling** module $p(W)$
  - N-gram
  - Neural language model (recurrent neural network or transformer)
- From training data, we can basically find how possibly "to", "two", and "too" will be appeared after "go"
  - Part of WSJ training data, 37,416 utterances
    - "go to": **51** times
    - "go two":
    - "go too":

For $p(L|W)$, we can use the lexicon mentioned above (letter → sound)

In next lecture, we introduce end-to-end ASR

Lec 13:

Model above is complicated and hard to train! Can we do end2end?

How to describe the phrase "go to"?

- **Word**
  - "go" and "to"
  - More **semantic/syntactic**
  - Very large vocabulary size, e.g., $|\mathcal{V}|$ would be 100K
  - **Out of vocabulary issue** ☹
  - The length ($N = 2$) is very short. **Less computational cost**, but **larger mismatch between the input and output lengths**
- **Character**
  - "g" "o" "_" "t" and "o" ("_" means the space)
  - The vocabulary size is not large in general. ~30 in the Roman script, ~10K in the Chinese script
  - **No out of vocabulary issue** (rarely happens) ☺
  - The length ($N = 5$) becomes longer. **More computational cost**, but **relaxing the mismatch between the input and output lengths**
- **BPE: Byte Pair Encoding (sentence piece)**
  - "go to" → "_g" "o" "_to" ($N = 3$)
  - **Something between**, we can also control the vocabulary size

In general, we do not specify which unit we use in our lecture since this is one of the model configurations.

- **Phoneme**
  - "go to" → "G OW T UW"
  - More **acoustic**
  - The phoneme vocabulary size is not very large in general
  - The length ($N = 5$) becomes long (like the discussion in the (Roman) character).
  - **We need a dictionary**
  - If we use a phone, we can make this part language-independent
- **State** (hidden Markov model state)
  - We further decompose a phoneme into several states
  - Furthermore **acoustic**
  - Classically, we use this representation a lot (e.g., 3-state HMM)
  - It makes the unit further longer and **the mismatch between the input and output lengths is further relaxed**
  - It will be introduced later in HMM

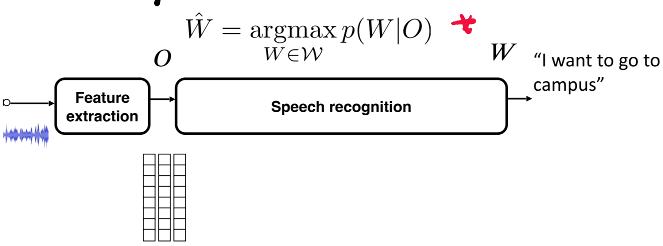
* Now we want this sort of E2E model!

Firstly: Have to decide how to represent text!

Emperical Solution: ⇒

- Mostly, we use BPE (or sentence piece)
  - We need to set the maximum number depending on the training data
- Character for the low-resource case or Chinese and Japanese
  - Chinese/Japanese has ~10,000 characters
- Some languages do not have scripts
  - Phoneme, phone
  - Translation to the other languages (not ASR but speech translation)

$$\hat{W} = \underset{W \in \mathcal{W}}{\operatorname{argmax}}\, p(W|O)$$ *



- Direct mapping function from speech feature sequence O to text W
- Usually, it does **not** deal with the phoneme-based intermediate representation
- Mainly three architectures
  - Attention-based
    - Decoder-only is also included here (not fully end-to-end)
  - Connectionist temporal classification (CTC)
  - Recurrent neural network transducer (RNN-T)

Attention-based ⇒

Most importantly

How to align?

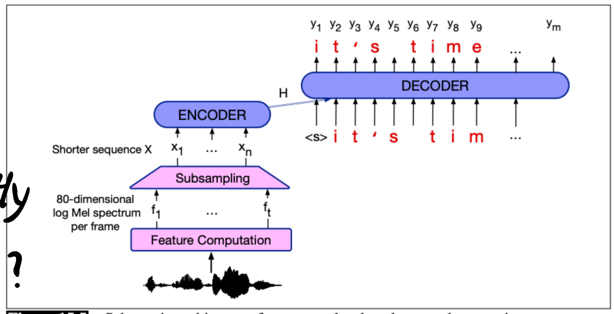


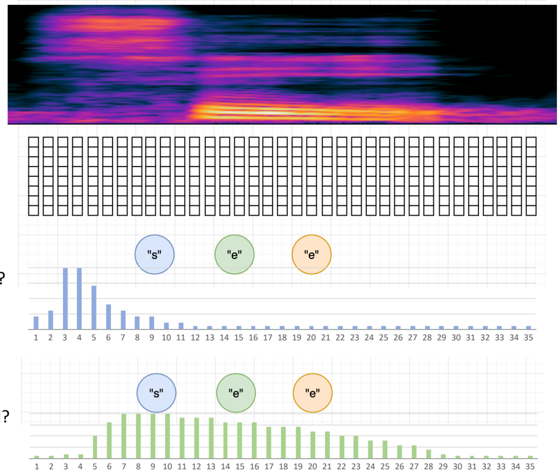
Figure 15.5 Schematic architecture for an encoder-decoder speech recognizer.

- Our staring point $p(W|O)$
  - Input: $O = (\mathbf{o}_t | t = 1,\ldots,T)$
  - It is difficult to deal with $W = \left(w_i \in \mathcal{V} \middle| i = 1,\ldots,N\right)$
  - $T \neq N$
- Instead, we **factorize** $p(W|O)$ as follows based on a **probabilistic chain rule**

Soft alignment! ⇒ Dist of a word to all frames

* Autoregressive

$$p(W|O) = \prod_{i=1}^{N} p(w_i|w_{1:i-1}, O)$$



- This neural network is handled by an **attention-based method** to align the input and output **(soft alignments)**
- We usually do not use the phoneme

$$A_j = \left(a_{jt} \in [0,1] \middle| t = 1,\ldots,T\right)$$

$$a_{jt} \geq 0, \ \sum_t a_{jt} = 1$$

Attention-based approach is based on the **soft** alignment

Another: hard alignment

For CTC (Connectionist Temporal Classification) and RNN-T:

Interest in the dist of a frame over letters (For CTC, including '␣')

- Again, our staring point $p(W|O)$
- We introduce a **hard alignment path** as a random variable:

$$Z = (z_t \in \mathcal{V}' | t = 1,\ldots,T)$$

- We consider the following equations

$$p(W|O) = \sum_Z p(Z, W|O)$$

$$= \sum_Z p(Z|O) p(W|Z, O)$$

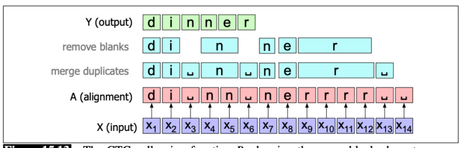


Figure 15.13 The CTC collapsing function B, showing the space blank character ␣; repeated (consecutive) characters in an alignment A are removed to form the output Y.

The CTC collapsing function is many-to-one; lots of different alignments map to the same output string. For example, the alignment shown in Fig. 15.13 is not the only alignment that results in the string *dinner*. Fig. 15.14 shows some other alignments that would produce the same output.

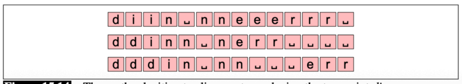
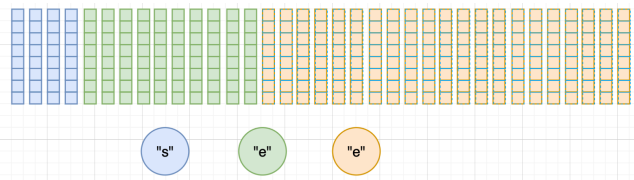
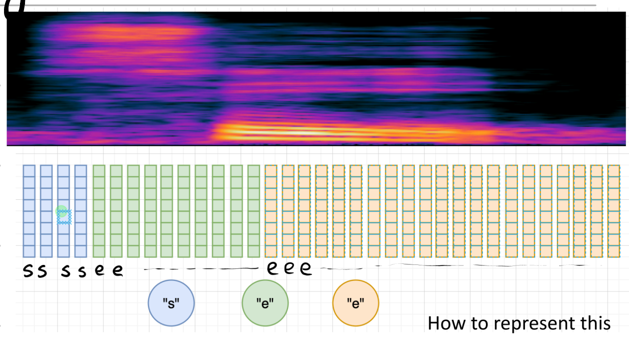


Figure 15.14 Three other legitimate alignments producing the transcript *dinner*.





We can represent a path as a sequence variable
This is called a hard alignment

$z_1 = \text{“s”}, z_2 = \text{“s”}, \ldots, z_5 = \text{“e”}, \ldots, z_T = \text{“e”}$

$Z = (z_t \in \{\text{“s”}, \text{“e”}, \ldots\} | t = 1,\ldots,T)$

HMM, CTC, and RNN-Transducer are based on the **hard** alignment

How to represent this information mathematically?

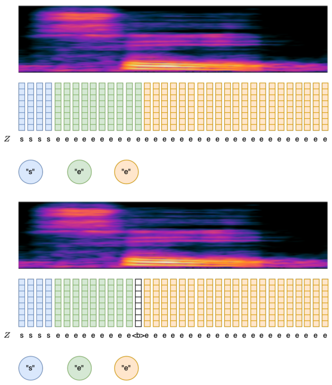
- The most difficult issue in speech recognition: the input and output lengths are different
  - We need some alignments
- The soft alignment case
  - We use an attention-based neural network (this will be introduced later).
- The hard alignment cases
  - We explicitly introduce an alignment path Z.
  - We can use it for CTC, RNN-transducer, and HMM-based approach

We will delve into how CTC, RNN-T and HMM-based model do alignment path

CTC: need <b> to handle duplicate letter case!

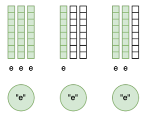
- Hard alignment examples
- How to distinguish them?
  - (“s”, “e”, “e”)

    vs.
  - (“s”, “e”)
  - Both are written as
  - $Z = (sssseeeeeeeeeeeeeeeeee\ldots)$
- We introduce the blank symbol!
  - $Z = (sssseeeeeeeee < b > eee\ldots)$



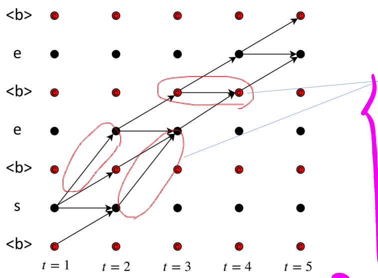
- First, we insert <b> to the character sequence ”see”

  → $W =$ (“s”, “e”, “e”), where $|W| = J$
- Then, **expand W' to the frame length T** to form Z
  - Assuming that $T > J$ in general: We cannot use CTC if it is not satisfied
  - All tokens and <b> can be repeated to adjust the length
    - For example, if “e” is repeated three times
    - $W =$ “e” → $Z =$ (“e”, “e”, “e”)
    - $W =$ “e” → $Z =$ (“e”, “e”, “<b>”)
    - $W =$ “e” → $Z =$ (“e”, “<b>”, “<b>”)
  - <b> must be inserted between repeated character
    - $W =$ (“e”, “e”), then $Z =$ (…“e”, ”<b>”, ”e”…): we **cannot** skip <b>
    - $W =$ (“s”, “e”), then, $Z =$ (…“s”, “e”…): we can skip <b>



All possible path with <b> to fit in?

Trellis of Z ⇐ Solution!



1. Must start at the first <b> or s
2. Must end at the last <b> or e
3. All characters can be repeated
4. <b> can be skipped except when it is inserted between repeated character
   - ”s”, ”<b>”, ”e”: we can skip <b>
   - “e”, ”<b>”, ”e”: we **cannot** skip <b>

Rule is important

Example of usage of <b>

“see” (only 5 frames)

- $W =$ (“s”, “e”, “e”), $T = 5$

Then

$Z =$ (“<b>”, “s”, “e”, “<b>”, “e”), or

$Z =$ (“s”, “<b>”, “e”, “<b>”, “e”), or

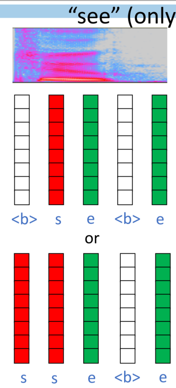
$Z =$ (“s”, “s”, “e”, “<b>”, “e”), or

….

This is an alignment problem

Note that

- $f: Z \to W$: many to one mapping
  1) Remove repeated tokens
  2) Remove the blank token <b>
- $f^{-1}: W \to Z$: one to **many** mapping
  - How to efficiently represent it? We use the trellis representation



CTC

- We define an alignment variable as $Z = (z_t \in \mathcal{V} \cup <b> \,|\, t = 1,\ldots,T)$
  - $|Z| = |O| = T$
  - The vocabulary set is augmented by the blank symbol <b>
- We incorporate this random variable to $p(W|O)$ as follows:

$$
\begin{aligned}
p(W|O) &= \sum_{Z \in f^{-1}(W)} p(Z|O) \\
&= \sum_{Z \in f^{-1}(W)} \prod_{t=1}^{T} p(z_t|\cancel{z_{1:t-1}}, O) \\
&= \sum_{Z \in f^{-1}(W)} \prod_{t=1}^{T} p(z_t|O)
\end{aligned}
$$

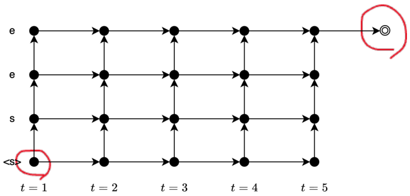
- $f^{-1}(W)$: **all possible Z representing W**
- Use a chain rule to further factorize Z
- Perform conditional independence assumptions and ignore the dependency of previous alignments $z_{1:t-1}$ (this is an approximation)
- We need to solve the **summuzation over all possible paths** $\sum_Z$

  ←Exponential

- $p(z_t|O)$ is represented by a neural network (Bidirectional LSTM or self-attention)

RNN-T: Trellis with no <b> & special rules (in CTC)

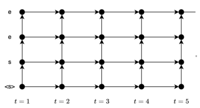
- Alternative alignment paths
  - $W = (w_1, \ldots, w_J)$: J-length label sequence
  - $O = (\mathbf{o}_1, \ldots, \mathbf{o}_T)$: T-length input sequence
- Similar to CTC, consider all possible paths in the right-side trellis
  - Start: **left bottom corner**, i.e., <s> (start of sentence) at $t = 1$
  - End: **top right corner**
  - ↑: output token (it does not consume the input frame, unlike CTC)!
  - →: no output ($\varnothing$)
  - f^{-1}:W=(‘s’, ‘e’, ’e’) → Z=($\varnothing$, ‘s’, $\varnothing$, $\varnothing$, ‘e’, ‘e’, $\varnothing$, $\varnothing$), (‘s’, $\varnothing$, ‘e’, $\varnothing$, $\varnothing$, $\varnothing$, ‘e’, $\varnothing$) etc. if $T = 5$ and $J = 3$
  - $f: Z \to W$ is a many to one mapping (we can just remove $\varnothing$
- Consider all possible paths including the case that it allows the output of the token W



- We define an alignment variable as $Z = (z_k \in \mathcal{V} \cup \varnothing \,|\, k = 1,\ldots,T + J)$
  - Vocabulary set is augmented by the blank symbol $\varnothing$
  - Note that $|Z| = T + J$, not T
- We can introduce the following equations

$$
\begin{aligned}
p(W|O) &= \sum_{Z \in f^{-1}(W)} p(Z|O) \\
&= \sum_{Z \in f^{-1}(W)} \prod_{k=1}^{T+J} p(z_k|z_{1:k-1}, O) \\
&= \sum_{Z \in f^{-1}(W)} \prod_{k=1}^{T+J} p(z_k|f(z_{1:k-1}), O)
\end{aligned}
$$

- $f^{-1}(W)$: all possible Z representing W
- Use a chain rule to further factorize Z
- $f(z_{1:k-1})$ represents the partial token sequence up to $k-1$



- $W =$ (“s”, “e”, “e”), $T = 5$

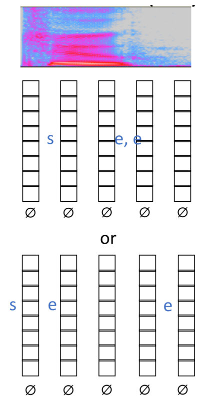
Then

$Z = (\varnothing, \text{‘s’}, \varnothing, \varnothing, \text{‘e’}, \text{‘e’}, \varnothing, \varnothing)$, or

$Z = (\text{‘s’}, \varnothing, \text{‘e’}, \varnothing, \varnothing, \varnothing, \text{‘e’}, \varnothing)$,or

….

This is an alignment problem



HMM-3 states: Still: P(O|L) P(L|W) P(W). ⇒ Phoneme to align!

Assumption: Every phoneme has 3 states

- Two differences
  - Phoneme sequence L instead of word sequence W
  - $p(O|L)$ instead of $p(L|O)$ or $p(W|O)$ due to the Bayes theorem

1. Decompose the word sequence to the phoneme sequence by using a pronunciation dictionary

   “one,” “two” → /W/, /AH/, /N/, /T/, /UW/
2. Introduce the silence related symbols
   - Silence begin (/SilB/): placed in the beginning of the sentence
   - Silence end (/SilE/): placed in the end of the sentence
   - Short pause (/SP/): placed between the words (can be skipped)

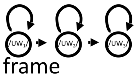
   /W/, /AH/, /N/, /T/, /UW/ → /SilB/, /W/, /AH/, /N/, /SP/, /T/, /UW/, /SilE/
3. Expand each phoneme with **three** states

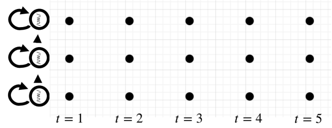
   /SilB/, /W/, /AH/, /N/, /SP/, /T/, /UW/, /SilE/

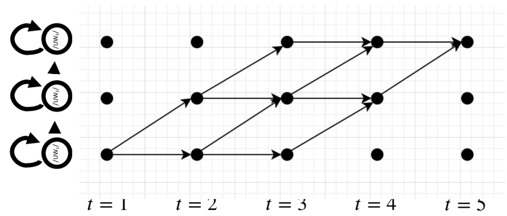
   → /SilB/$_1$, /SilB/$_2$, /SilB/$_3$, …, /T/$_1$, /T/$_2$, /T/$_3$, /UW/$_1$, /UW/$_2$, /UW/$_3$, …

- This structure is widely used, but there are some variants (e.g., short pause can be one state)

Find possible duration assignment (path) again!

- Left-to-right HMM without skip 
  - Each state must have at least one frame
- Example, 3 state left-to-right HMM (/UW$_1$/, /UW$_2$/, /UW$_3$/) $S = (s_t \in \{/\mathrm{UW}_1/, /\mathrm{UW}_2/, /\mathrm{UW}_3/\} | t = 1, 2, 3, 4, 5)$ quences





- It must start at the initial state (/UW$_1$/)
- It must end at the final state (/UW$_3$/)

- CTC

$$
\begin{aligned}
p(W|O) &= \sum_{Z \in f^{-1}(W)} p(Z|O) \\
&= \sum_{Z \in f^{-1}(W)} \prod_{t=1}^{T} p(z_t|\cancel{z_{1:t-1}}, O) \\
&= \sum_{Z \in f^{-1}(W)} \prod_{t=1}^{T} p(z_t|O)
\end{aligned}
$$

- RNN transducer

$$
\begin{aligned}
p(W|O) &= \sum_{Z \in f^{-1}(W)} p(Z|O) \\
&= \sum_{Z \in f^{-1}(W)} \prod_{k=1}^{T+J} p(z_k|z_{1:k-1}, O) \\
&= \sum_{Z \in f^{-1}(W)} \prod_{k=1}^{T+J} p(z_k|f(z_{1:k-1}), O)
\end{aligned}
$$

- HMM

$$
\begin{aligned}
p(W|O) &\propto \sum_{L} p(O|L)p(L|W)p(W) \\
&= \sum_{L}\sum_{S} p(O, S|L)p(L|W)p(W) \\
&= \sum_{L}\sum_{S} p(\mathbf{o}_1|s_1, L)p(s_1|L) \prod_{t=2}^{T} (\mathbf{o}_t|s_t, L)p(s_t|s_{t-1}, L)p(L|W)p(W)
\end{aligned}
$$

Wh

Which part is similar?

→ All are based on $\sum_{Z \text{ or } S} \prod_{t} f(t)$

→ We can compute them based on the dynamic programming

Summary:

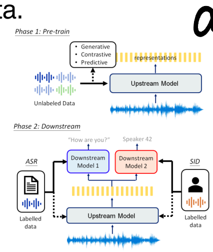
- CTC, RNN-transducer, and HMM are represented by a similar function form, i.e., $\sum_{Z \text{ or } S} \prod_{t} f(t)$
- All possible alignments are constrained by each method
- The most difficult issue in speech recognition: the input and output lengths are different
  - We need some alignments
- The soft alignment case
  - We use an attention-based neural network (this will be introduced later).
- The hard alignment cases
  - We explicitly introduce an alignment path z.
  - We can use it for CTC, RNN-transducer, and HMM-based approach

---

Lec 14: Self Supervised ASR

The setting:

1. A family of downstream tasks in mind *(The main motivational difference).*
2. Lots of unlabeled in-domain data.
3. A small fraction of labeled data.



For supervised one, important settings are as the left figures.

What if we can self-supervise? So we can leverage bunch of unlabeled data!

Some important property of Speech:

Speech inputs have a variable number of lexical units per sequence.

Speech is a long sequence that doesn't have segment boundaries.

Speech is continuous without a predefined dictionary of units to explicitly model in the self-supervised setting.

Speech processing tasks might require orthogonal information, e.g., ASR and Speaker ID.

One possible self-supervise training:

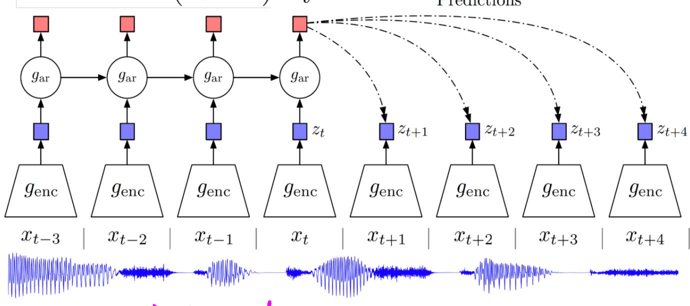
- Distinguish correct (positive) samples from wrong (negative) ones.

Contrastive Predictive Coding (CPC)

Contrastive Approaches: ① CPC

- The first successful representation learning approach for speech data.
- It triggered lots of research in speech representation learning.
- Distinguish correct (positive) samples from wrong (negative) ones.

$f_k(x_{t+k}, c_t) = \exp\left(z_{t+k}^T W_k c_t\right)$ c_t



van den Oord et al, 2019 "Representation Learning with Contrastive Predictive Coding"

Pipeline: Want learned latent variable related to later input signal. ⇐

Distinguish instead of generating

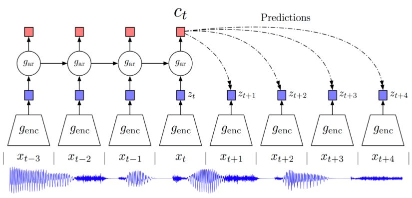
Baevski et al, 2020 "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations"

- The first approach to show significant improvements for low-resource ASR.
- Impressive results on multilingual representations.
- Strong performance on a wide range of downstream speech tasks.

InfoNCE maximizes the mutual information between the input signal and the learned latent variables C.

$$\mathcal{L}_N = -\mathop{\mathbb{E}}_{X}\left[\log \frac{f_k(x_{t+k}, c_t)}{\sum_{x_j \in X} f_k(x_j, c_t)}\right]$$

Strategies for sampling negative and positive examples determine the nature of representations, e.g., whether they are good for ASR or Speaker ID.



c_t: contains information repre of given speech

② wav2vec 2.0: Like bert, mask and try to reconstruct.

Product quantization with more than one codebook yields better results.

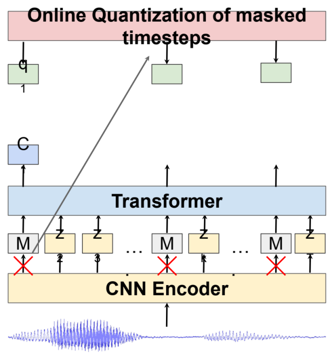
An entropy loss is added to the Gumbel softmax distribution to maximize codebook diversity.

Negative examples are chosen from masked segments in the same utterance that don't belong to the same codeword.

Loss: "Reconstruction"

- The goal is to maximize the similarity between the learned contextual representation and the quantized input features

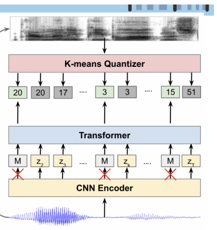
$$\mathcal{L}_m = -\log \frac{\exp(sim(\mathbf{c}_t, \mathbf{q}_t)/\kappa)}{\sum_{\tilde{\mathbf{q}} \sim \mathbf{Q}_t} \exp(sim(\mathbf{c}_t, \tilde{\mathbf{q}})/\kappa)}$$



Predictive Approaches: HuBert

HuBERT: Implementation details

- A small codebook size, e.g., 50, 100, is used for the initial training iteration to focus on phonetic differences rather than speaker and style.
- For the subsequent two iterations, layers 6 and 9 of the base architecture (12 layers) are used for the clustering steps. They found empirically to contain higher quality features over many speech tasks.



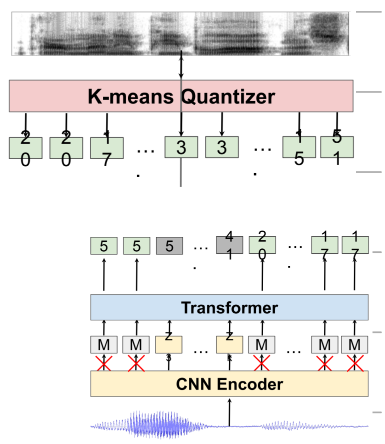
Hsu et al 2021, "HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units"

- A simple method to apply BERT style representation learning for speech.
- Matched or beat the SOTA on ASR while being the best for many speech tasks.
- With its high-quality discrete units, HuBERT facilitated Textless NLP research.

HuBERT: The pretext task

- The K-means quantizer produces frame-level labels.

Then the process can be repeated using learned HuBERT features from a previous iteration.

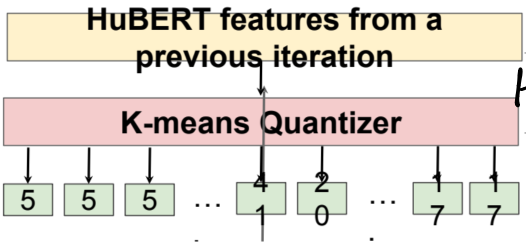


Hsu et al 2021, "HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units"

For tasks like 'identify each number spoken is 1~9?'

K-means & NN to align

"Teacher & Student Paradigm"



With learned HuBERT Feature, can continue iterating!

Lec 15: Text to Speech

Evaluation :
- Is it natural?
- Is it intelligible ?
- Is it close to the target speaker/style

This task seems easy. But there can be critical criterion to evaluate a speech.

Formant Synthesis

In the introduction of spectrogram and formant, formant synthesis may be a good approach.

- How does it work?
  - produce speech by mimicking the formant structure and other spectral properties of natural speech
  - using additive synthesis and an acoustic model (with parameters like voicing, fundamental frequency, noise levels)
- Advantages:
  - highly intelligible, even at high speeds
  - well-suited for embedded systems, with limited memory and computation power
- Limitations:
  - not natural, produces artificial, robotic-sounding speech, far from human speech
  - difficult to design rules that specify model parameters

Pros & Cons of this

Text → Text Analysis → Linguistic features → Acoustic Model → Acoustic features → Vocoder → Speech

- Popular use case : Text-to-Speech conversion

Modern Classic TTS System ⇒

- Text Analysis
  - Strings of characters to words
- Linguistic Analysis
  - Words to Pronunciation and Prosody
- Waveform Synthesis
  - From pronunciations to waveforms

For text analysis, two challenges:

① Non-Standard Word:

Words not in the lexicon

NSWs in regular text

Text Type	%NSW
Novels	1.5%
Press wire	4.9%
Email	10.7%
Recipes	13.7%
Classifieds	27.9%
IM	20.1%

② Homographs (Same form, different meaning): Like read (current & past form)

Homographs
- Same writing, different pronunciation
- (Homophones: same pronunciation different writing. “to” “two” “write” “right”)

- How hard are they?
  - Finding them
  - Identifying them
  - Expanding them
- Current processing techniques
  - Ignored
  - Lexical lookup
  - Hacky hand-written rules
  - (not so) Hacky hand-written rules
  - Statistically train models (and hacky hand written rules)

- English: not many:
  - Stress shift (Noun/Verb)
    - Segment, project, convict
  - Semantic
    - Bass, read, Begin, bathing, lives, Celtic, wind, Reading, sun, wed, …
    - Roman Numerals

Homograph Disambiguition :

- Same tokens in different contexts
- Identify target homograph
  - E.g. numbers, roman numerals, “St”

Find instances in large text corpora
Hand label them with correct answer
Train a decision tree to predict types

Also can use mark-up language

Mark-up Languages

Add explicit markup to text
Can be done in machine generated text
SSML (Speech Synthesis Markup Language)
- Choice voices, languages
- Give pronunciations
- Specifiy breaks, speed, pitch
- Include external sounds

SSML can explicitely control the synthesis of speech

For Linguistic Analysis ① Pronunciation ② Prosody

Here introduce Produsy (韵律): 音素如何被说出来，即语音的节奏和旋律。有四大重点: ⟹

How the phonemes will be said
Four aspects of prosody
- Phrasing: where the breaks will be
- Intonation: pitch accents and F0 generation
- Duration: how long the phonemes will be
- Power: energy in signal

Firstly: Phrasing (断句) ⟹

Need to take a breath
Need to chunk relevant parts together
- Sub-sentential
- Supra-word

First approximation
- At punctuation (comma, semicolon, etc.)
- Too little

Second approximation
- At each (or some) of the content/function words
- Too much

Secondly: Intonation (语调) 包含 pitch (音高) 与 Accents (重音)

- Accents and Boundaries
  - Where are the important changes in F0?
- Accents on syllables
  - Identifies "important" words
    - It will be RAINY today in Boston
    - It will be rainy TODAY in Boston
    - It will BE rainy today IN Boston (strange)
  - On important words
  - First approximation
    - On stressed syllables in content words
      - It WILL be RAINY TODAY in BOSTON
    - About 80% correct on news reader speech
  - ML training can use more features
    - Content, proper nouns, POS, position in text
    - (not semantic information)

Also have ToBI ⟹ (Tones and Break Indices) 是一种标记英语语调的标准

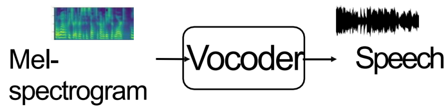
- Tones and Break Indices
  - A labeling for intonation (English)
- Different accent types
  - H*, !H, L*, L+H*
- Different boundary types
  - L+L%, L+H%, H+H%,

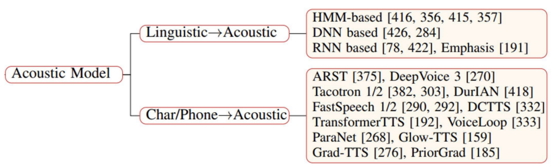
- Each phone needs a duration
  - Make it 80ms
- Vowels are typically longer than consonants
- Emphasis/accent/stress lengthens them
- Initial and final phones are longer

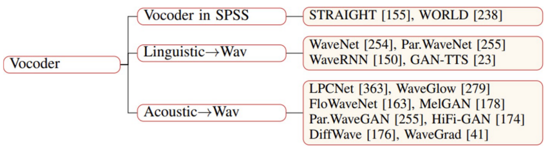
Lastly, duration. Every phone need a duration. E.g. Vowels > Consonant

- By rule
  - Klatt rules
- By ML training (using features)
  - linear regression
  - Easy to get reasonable durations
  - Hard to get very good durations

Can be predicted with rules or ML training





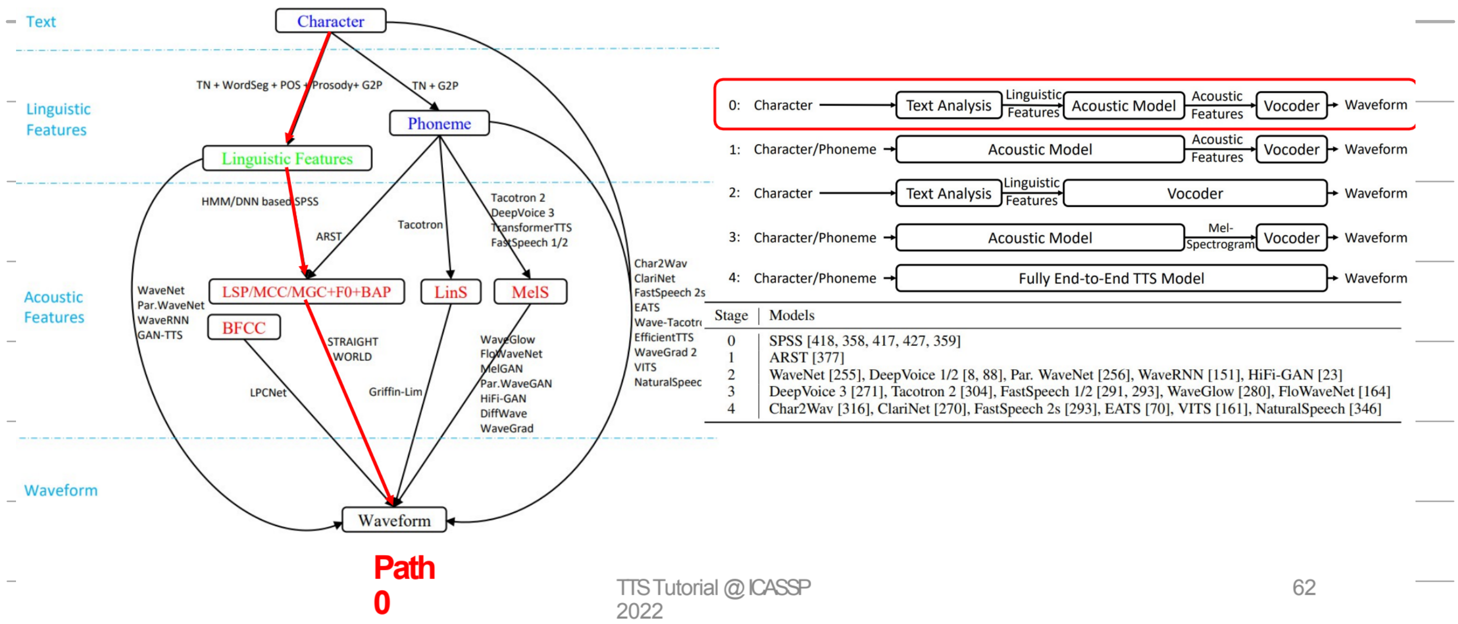


For the acoustic model: End to End: 语言学特征 → 声学特征 (e.g. Mel)
vocoder model: 声学特征 (e.g. Mel) → 语音波形

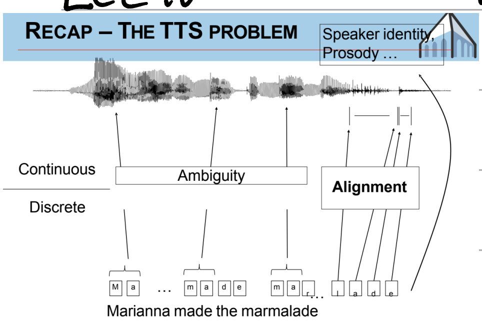
RNN / CNN / Transformer - based

Autoregressive / Flow / Gan / VAE / Diffusion based

Data Conversion Pipelines



Lec 16: Neural TTS + Spoken LM



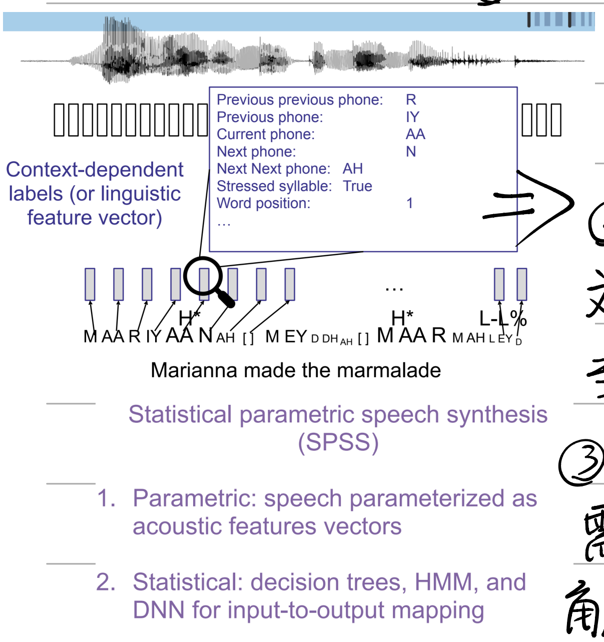
回顾 TTS 问题中的难点:

① Discrete text ⇒ Continuous Sound

② Ambiguity: 同一个文本多种读法

③ Alignment: 短文本序列 → 长语音帧序列

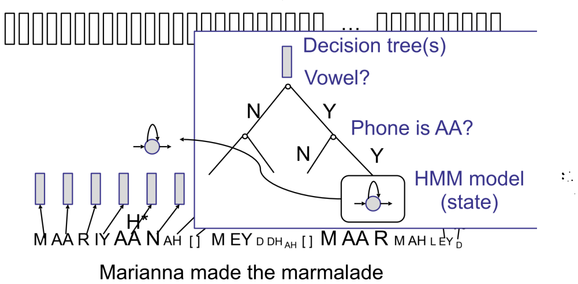
早期技术: ① Unit Selection: 从库中挑匹配的波形片段, 然后拼接



② Statistical Parametric Speech Synthesis:

文本 ⇒ 语言特征 $\xrightarrow[\text{HMM}]{\text{Decision Tree}}$ 声学参数 ⇒ 声码器合成

当然, 也可 Regression:



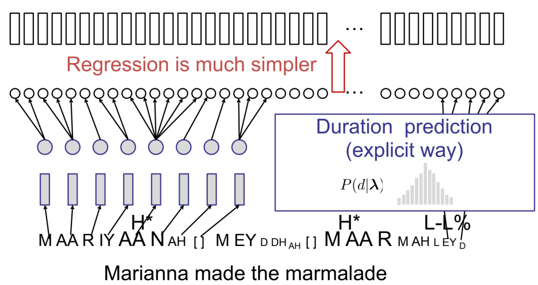
③ Statistical & DNN

需要 —— 从NN

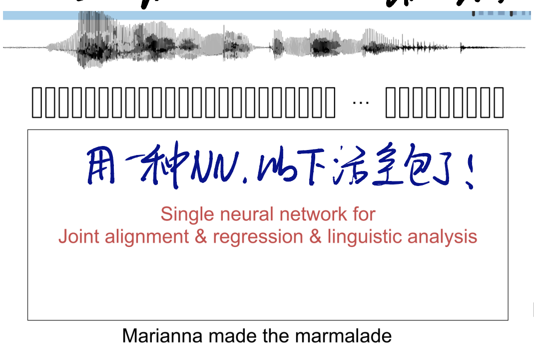
角度 —— : a.

Duration Prediction b. DNN (FFN).

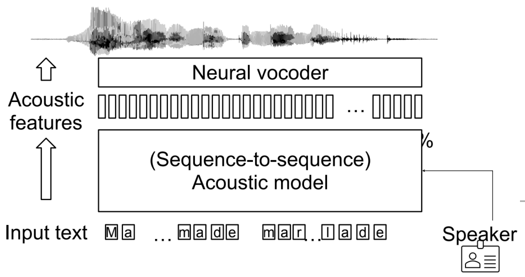
C. Waveform Generation (vocoder)

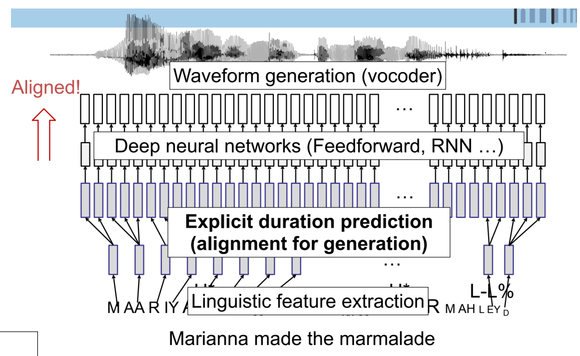


Seq-to-Seq. 端到端！

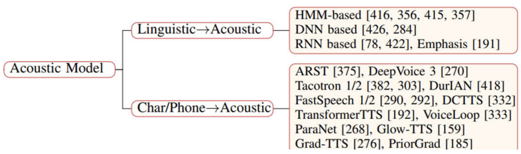


Recent method:





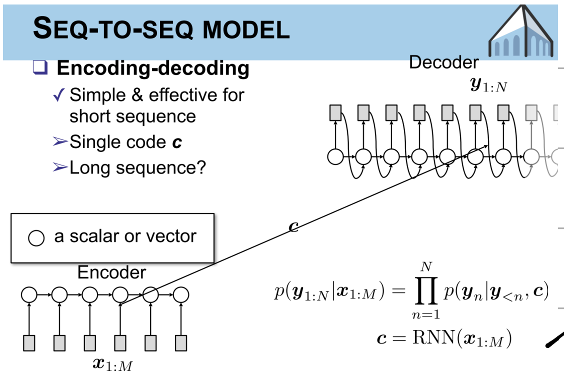
• Predict acoustic features from linguistic features

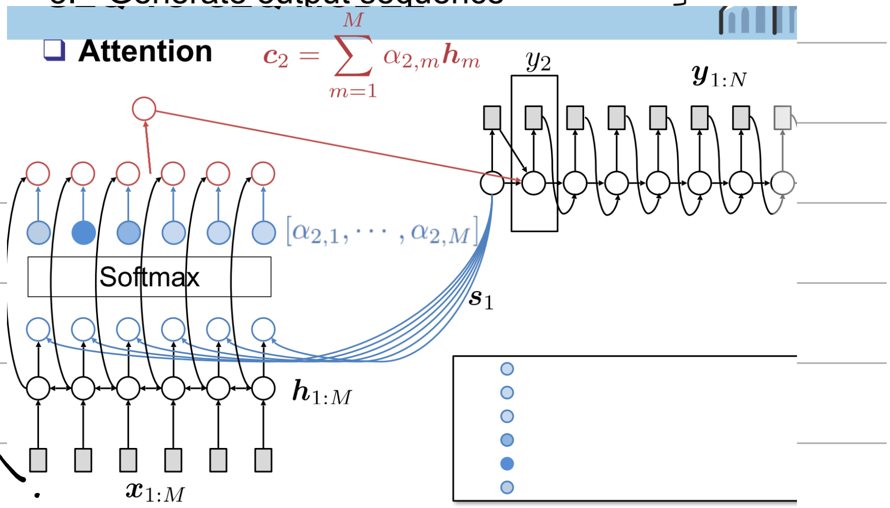


Seq-to-seq model

1. Derive linguistic features from input
2. Learn & generate alignment
3. Generate output sequence

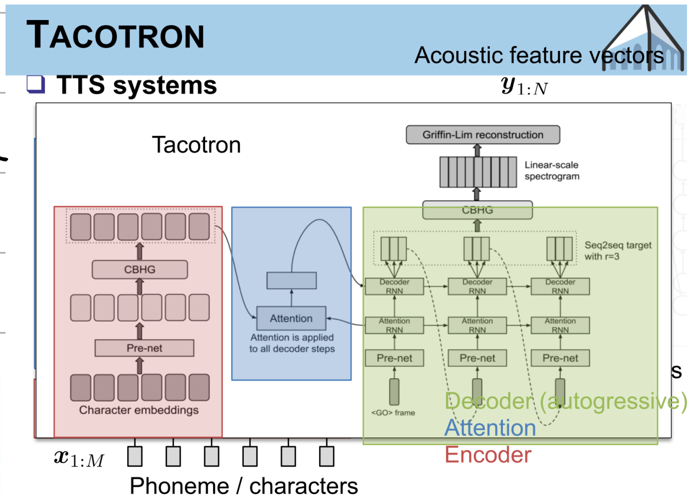
} Attention mechanism

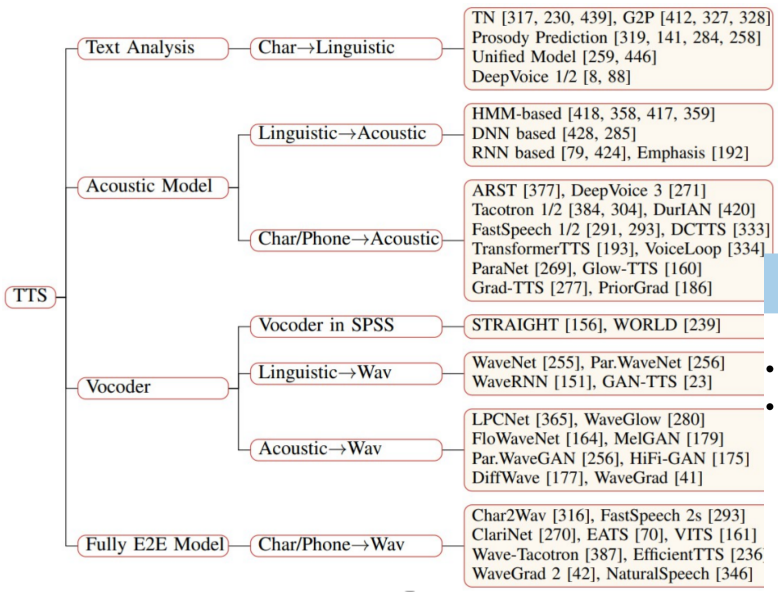




一种model是 attention mechanism.

Tacotron：使用encoder-decoder架构和注意力机制来学文本和声学帧间的对齐。但软注意力有时不稳定。

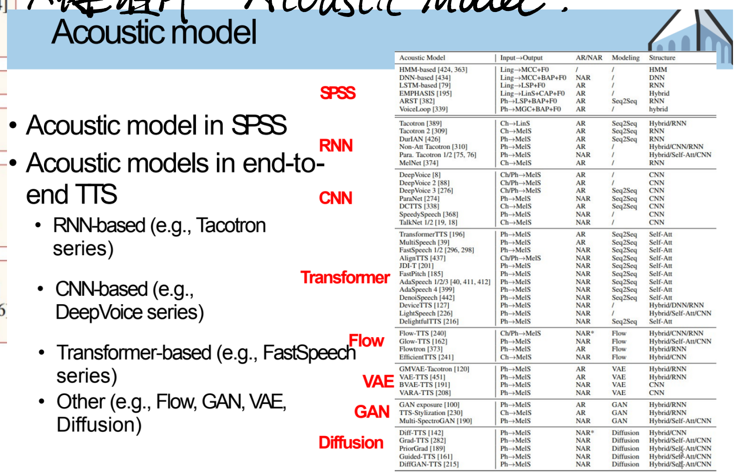




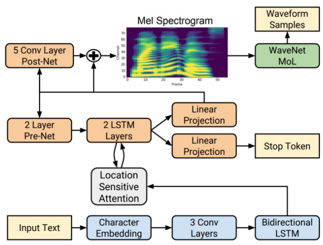
2022

⇐ Key components in TTS.

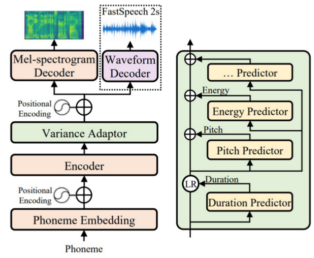
关键组件：Acoustic Model.



- Tacotron 2
  - Evolved from Tacotron
  - Text to mel-spectrogram generation
  - LSTM based encoder and decoder
  - Location sensitive attention
  - WaveNet as the vocoder



- FastSpeech 2 [292]
  - Improve FastSpeech
  - Use variance adaptor to predict duration, pitch, energy, etc
  - Simplify training pipeline of FastSpeech (KD)
  - FastSpeech 2s: a fully end-to-end parallel text to wave model



(a) FastSpeech 2 (b) Variance adaptor

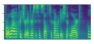
- Other works
  - FastPitch [181]
  - JDI-T [197], AlignTTS [429]

⇑

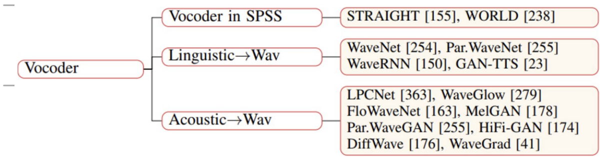
RNN-based Transformer-based ⇒

Vocoder: 负责声学特征 ⇒ 波形

(如Mel)



Acoustic Features → Vocoder → Speech



- Autoregressive vocoder
- Flow-based vocoder
- GAN-based vocoder
- VAE-based vocoder
- Diffusion-based vocoder

Vocoder	Input	AR/NAR	Modeling	Architecture
WaveNet [260]	Linguistic Feature	AR	/	CNN
SampleRNN [239]	/	AR	/	RNN
WaveRNN [151]	Linguistic Feature	AR	/	RNN
LPCNet [370]	BFCC	AR	/	RNN
Univ. WaveRNN [221]	Mel-Spectrogram	AR	/	RNN
SC-WaveRNN [271]	Mel-Spectrogram	AR	/	RNN
MB WaveRNN [426]	Mel-Spectrogram	AR	/	RNN
FFTNet [146]	Cepstrum	AR	/	CNN
iSTFTNet [153]	Mel-Spectrogram	NAR	/	CNN
Par. WaveNet [261]	Linguistic Feature	NAR	Flow	CNN
WaveGlow [285]	Mel-Spectrogram	NAR	Flow	Hybrid/CNN
FloWaveNet [166]	Mel-Spectrogram	NAR	Flow	Hybrid/CNN
WaveFlow [277]	Mel-Spectrogram	AR	Flow	Hybrid/CNN
SqueezeWave [441]	Mel-Spectrogram	NAR	Flow	CNN
WaveGAN [69]	/	NAR	GAN	CNN
GELP [150]	Mel-Spectrogram	NAR	GAN	CNN
GAN-TTS [23]	Linguistic Feature	NAR	GAN	CNN
MelGAN [182]	Mel-Spectrogram	NAR	GAN	CNN
Par. WaveGAN [410]	Mel-Spectrogram	NAR	GAN	CNN
HiFi-GAN [178]	Mel-Spectrogram	NAR	GAN	Hybrid/CNN
VocGAN [416]	Mel-Spectrogram	NAR	GAN	CNN
GED [97]	Linguistic Feature	NAR	GAN	CNN
Fre-GAN [164]	Mel-Spectrogram	NAR	GAN	CNN
Wave-VAE [274]	Mel-Spectrogram	NAR	VAE	CNN
WaveGrad [41]	Mel-Spectrogram	NAR	Diffusion	Hybrid/CNN
DiffWave [180]	Mel-Spectrogram	NAR	Diffusion	Hybrid/CNN
PriorGrad [189]	Mel-Spectrogram	NAR	Diffusion	Hybrid/CNN
SpecGrad [176]	Mel-Spectrogram	NAR	Diffusion	Hybrid/CNN

AR

Flow

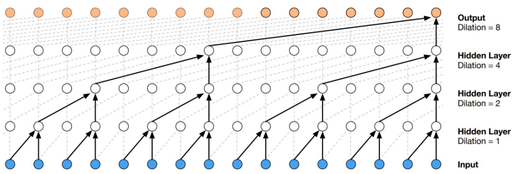
GAN

VAE

Diffusion

Autoregressive ⇓

- WaveNet: autoregressive model with dilated causal convolution [van den oord et al 2016]



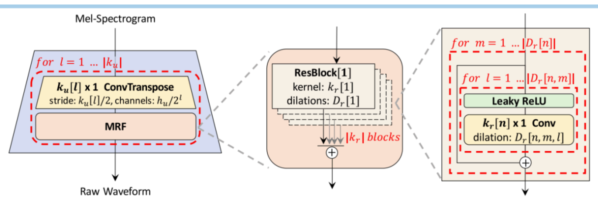


Figure 1: The generator upsamples mel-spectrograms up to $|k_u|$ times to match the temporal resolution of raw waveforms. A MRF module adds features from $|k_r|$ residual blocks of different kernel sizes and dilation rates. Lastly, the n-th residual block with kernel size $k_r[n]$ and dilation rates $D_r[n]$ in a MRF module is depicted.

- GANs: Simultaneously train generative + adversarial nets
- Generator: produce audio from the spectrogram.
  Discriminator: distinguish generator outputs from real audio
- Faster and high quality than Autoregressive

GAN-Based, 主流! ⇒.

未来方向: ① Fully Controllable TTS

② Edge Deployable

③ Multi Modality.

Lec 17. Structured Prediction

- **General problem:** we want to learn a (possibly conditional) distribution over some output space with structure to it
  - Structure: output space is a composition of variables whose values depend on each other's values
- Not structured prediction: simple classification tasks
  - Document classification (output space is set of classes)
  - Next-token prediction (output space is vocabulary)

What is "Structured"?

Typical Structured Prediction

- Multi-class classification:
  - Output is a set of classes
  - Classes may not be independent: some classes are likely to co-occur, and others never co-occur

Language modeling:
- Output is a sequence of tokens
- There might be long-distance dependencies between words across the sequence

Part-of-speech (POS) tagging
- Output is a sequence of POS tags
- Tags may depend on each other under lexical ambiguity

Fed raises interest rates 0.5 percent

noun verb noun verb noun verb

Challenges & Opportunities

Output space might be exponentially large, which makes search difficult at inference time

But, known dependence between output variables might help narrow down the search at inference time

Sequence tagging problems: $f : \mathcal{V}^N \to \mathcal{C}^N$

Where we also have a prior over possible output tag sequences $p \in \Delta^{\mathcal{C}^N}$

Sequence tagging tasks:
- Part-of-speech tagging
- Named entity recognition

Part of Speech: A sequence tagging task

E.g. "Fed raises ..."

Every word assume is known of POS distribution. But we want to choose the best one.

Lexical ambiguity

record → noun → /ˈɹɛk.ə˞d/
record → verb → /ɹɪˈkɔɹd/

Fed raises interest rates 0.5 percent

Fed	raises	interest	rates	0.5	percent
NNP	NNS	NN	NNS	CD	NN
VBD	VBZ	VB	VBZ		
VBN		VBP			

Fed raises interest rates 0.5 percent

NNP NNS NN NNS CD NN
VBD → VBZ → VB → VBZ → CD → NN
VBN VBP

knowledge about output classes:
a long sequence of verbs is very unlikely

Fed raises interest rates 0.5 percent

NNP → NNS → VB → NNS → CD → NN
VBD VBZ NN VBZ
VBN VBP

prior knowledge about the relationship between input and output:
"rates" cannot be the object of "interest"

Fed raises interest rates 0.5 percent

NNP → VBZ → NN → NNS → CD → NN
NNS, VBD, VBN, VB, VBP, VBZ

Simplification: independently predict each POS tag: $p(Y_i \mid \overline{x}, i)$
- E.g., embed token $\phi(x_i) \in \mathbb{R}^d$
- Then classify each embedding $p(Y_i) = f(\phi(x_i))$
  $f : \mathbb{R}^d \to \Delta^{\mathcal{P}}$
- *interest* —> 55% NN, 35% VB, 10% VBN

Further formulation and assumption:

Classify based on embedding.

How can we model this task?

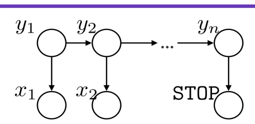
Important Preliminary: Generative v.s. Discriminative

- **Preliminary: generative vs. discriminative models**
  - Generative model: parameterizes a joint distribution over random variables X and Y
  - Discriminative model: parameterizes a conditional distribution of target Y given observation X
- Classification: X = instance features, Y = instance class
- Language modeling:
  - "True" language modeling: Y = sequence of tokens
  - Autoregressive approximation: X = previous tokens, Y = next token
- Part-of-speech tagging: X = sequence of tokens, Y = part of speech tags

One hot candidate model: HMM.

- Tags: $y_i \in \mathcal{P}$
- Words: $x_i \in \mathcal{V}$
- HMM generative model gives us $P(\overline{x}, \overline{y})$
  $p(y_1)p(x_1 \mid y_1)p(y_2 \mid y_1)p(x_2 \mid y_2) \ldots p(\text{STOP} \mid y_n)$

Generative story:

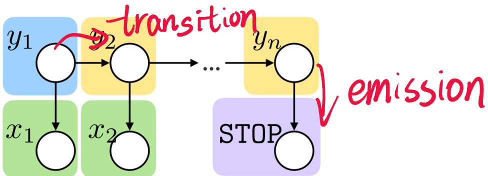


Markov process: y_i is conditionally independent of $y_1 \cdots y_{i-2}$ given y_{i-1}

Words are conditionally independent of one another given their parts of speech

$p(y_1)p(x_1 \mid y_1)p(y_2 \mid y_1)p(x_2 \mid y_2)\ldots p(\text{STOP} \mid y_n)$

Generative story:



Distribution over first-token POS tag $p(y_1) \in \Delta^{P}$

Emission probabilities $E \in \Delta^{\mathcal{P}\times\mathcal{V}}$

Transition probabilities $T \in \Delta^{\mathcal{P}\times\mathcal{P}}$

Probability of stopping after seeing POS tag $p(\text{STOP} \mid y) \in \mathbb{R}^{|\mathcal{P}|}_{[0:1]}$

Two core questions:

- how to determine these probabilities? (parameter estimation)
- how to use this model to map from a sentence to a sequence of POS tags? (inference)

⇐ With Markov Proposition

Training Objective ⇓

Labeled training data: $\left\{\left(\overline{x}^{(i)}, \overline{y}^{(i)}\right)\right\}_{i=1}^{N}$

Training objective: maximize **joint** likelihood of training data (maximum likelihood estimation)

$$\sum_i \log P\left(\overline{y}^{(i)}, \overline{x}^{(i)}\right)$$

$$= \sum_{i=1}^{N}\left(\log P\left(y_1^{(i)}\right) + \sum_{j=1}^{|\overline{x}^{(i)}|}\log P\left(x_j^{(i)} \mid y_j^{(i)}\right) + \sum_{j=1}^{|\overline{x}^{(i)}|}\log P\left(y_j^{(i)} \mid y_{j-1}^{(i)}\right) + \log P\left(\text{STOP} \mid y_{|\overline{x}^{(i)}|}^{(i)}\right)\right)$$

First-token POS tag
Emission probabilities
Transition probabilities
Probability of stopping

Similar to count-based n-gram models: just count and normalize occurrences:

- How often does the first token have POS tag p?
- How often does the sentence stop if we just had tag p?
- If our current POS tag is p, how often is it followed by p'?
- If our current POS tag is p, how often does it "emit" word x?

原始"Train"：用统计数据拟合概率分布

E.g. ⇓

training data

they can — N V
they fish — N V

start probabilities

	Count	$P(y_1)$
N	2	1.0
V	0	0.0

transition probabilities

	N	V	STOP
N	0	2	0
V	0	0	2

$p(y_j \mid y_{j-1})$

	N	V	STOP
N	0.0	1.0	0.0
V	0.0	0.0	1.0

emission probabilities

	they	can	fish
N	2	0	0
V	0	1	1

$p(x_j \mid y_j)$

	they	can	fish
N	1.0	0.0	0.0
V	0.0	0.5	0.5

E.g. with Smoothing (每个选项+1) ⇓

training data

they can — N V
they fish — N V

start probabilities

	Count	$P(y_1)$
N	3	0.75
V	1	0.25

transition probabilities

	N	V	STOP
N	1	3	1
V	1	1	3

$p(y_j \mid y_{j-1})$

	N	V	STOP
N	0.2	0.6	0.2
V	0.2	0.2	0.6

emission probabilities

	they	can	fish
N	3	1	1
V	1	2	2

$p(x_j \mid y_j)$

	they	can	fish
N	0.6	0.2	0.2
V	0.2	0.4	0.4

they can fish $P(\overline{y}, \overline{x}) =$ ____
N V V

这样，便可依四个分布 (First token, transition, emission, stopping) 与公式

计算：$P(\overline{y}, \overline{x}) = P(\text{第一个为}N) \times P(\text{They} \mid N) \times P(V\text{是下一个} \mid N) \times P(\text{can} \mid V)$

$\times P(V\text{是下一个} \mid V) \times P(\text{fish} \mid V) \times P(\text{STOP是下一个} \mid V)$

$P(\overline{x}, \overline{y}) = p(y_1)p(x_1 \mid y_1)p(y_2 \mid y_1)p(x_2 \mid y_2)\ldots p(\text{STOP} \mid y_n)$

- Problem: find $\arg\max_{\overline{y}} P(\overline{y} \mid \overline{x}) = \arg\max_{\overline{y}} \frac{P(\overline{y}, \overline{x})}{P(\overline{x})}$ (Bayes rule)

$$= \arg\max_{\overline{y}} P(\overline{y}, \overline{x})$$

$$= \arg\max_{\overline{y}} \log P(\overline{y}, \overline{x})$$

$$= \arg\max_{\tilde{y}_1,\ldots,\tilde{y}_n} \left(\log P(\tilde{y}_1) + \log P(x_1 \mid \tilde{y}_1) + \log P(\tilde{y}_2 \mid \tilde{y}_1)\ldots\right)$$

Problem: can't just search over all possible $\tilde{y}$

那么：经常是 given $\overline{x}$，$\overline{y}$ 是如何变化的

但暴力搜索：search space 会非常大！

How to sol?

Viterbi

Main principle: dynamic programming

Define $v \in \mathbb{R}^{|\overline{x}|\times|\mathcal{P}|}$, where $v_i(\tilde{y})$ is the score of the best path ending in $\tilde{y}$ at time i

Base case: $v_1(\tilde{y}) = \log P(x_1 \mid \tilde{y}) + \log P(\tilde{y})$

- Recurrence:

$$v_i(\tilde{y}) = \log P(x_i \mid \tilde{y}) + \max_{\tilde{y}_{\text{prev}}}\left(\log P(\tilde{y} \mid \tilde{y}_{\text{prev}}) + v_{i-1}(\tilde{y}_{\text{prev}})\right)$$

$v_i(\tilde{y}) = \log P(x_i \mid \tilde{y})$

$+ \max_{\tilde{y}\text{prev}} \left(\log P(\tilde{y} \mid \tilde{y}\text{prev}) + v_{i-1}(\tilde{y}\text{prev})\right)$

Emission Transition

↪ Same Format.

To compute $\max_{\tilde{y}} \log P(\overline{x}, \overline{y})$:

```
v = 0^{|x̄|×|P|}
for i = 1 ... |x̄|
  for ỹ in P
    if i == 1:
      v_1(ỹ) = log P(x_1 | ỹ) + log P(ỹ)
    else:
      v_i(ỹ) = log P(x_i | ỹ) + max_{ỹprev} (log P(ỹ | ỹprev) + v_{i-1}(ỹprev))
return v_{|x̄|+1}(STOP)
```

keep track of this over time to reconstruct the maximum-probability sequence $\arg\max_{\tilde{y}} \log P(\overline{x}, \overline{y})$

⇐ Viterbi Algorithm

在每个位置 i 上，每个 $\tilde{y}$ in $\mathcal{P}$ 求出 $v_i(\tilde{y})$，这信息会用于求 $(i+1)$ 位置 $v_{i+1}(\tilde{y})$

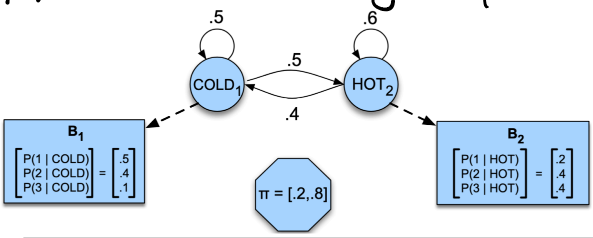
Our parameters were determined by counting over a labeled dataset:

- How often the first token has a particular POS tag
- How often one POS tag follows another (including STOP)
- How often each POS tag "generates" each wordtype

But what if we don't have labeled data?

↑ Ultimate question

HMM: Formulation: Key Components ⇒



HMM Tasks: Decoding: Viterbi

Likelihood: Forward

Given a sequence of observations $\overline{x}$, we want to compute $p(\overline{x})$

- If we had access to the sequence of labels $\overline{y}$, this is just the product of the emission probabilities:

$$p(\overline{x}) = \prod_{i=1}^{|\overline{x}|} p_e(x_i \mid y_i)$$

- But we don't actually know the labels! So we need to consider all possible latent label sequences $\overline{y}'$:

$$p(\overline{x}, \overline{y}') = p(\overline{x} \mid \overline{y}')p(\overline{y}') = \prod_{i=1}^{|\overline{x}|} p_e(x_i \mid y_i) \prod_{i=1}^{|\overline{x}|} p_t(y_i' \mid y_{i-1}') \quad {\scriptstyle p_t(y_1' \mid y_0') = p_i(y')}$$

- Then, $p(\overline{x}) = \sum_{\overline{y}' \in \mathcal{Y}} p(\overline{x}, \overline{y}') = \sum_{\overline{y}' \in \mathcal{Y}} p(\overline{x} \mid \overline{y}')p(\overline{y}')$ Exponential...

Core：维持每i下的每个s 对应的 $\alpha_i(s)$. 最终 sequence likelihood 是 $\sum_{s\in S} \alpha_{|\overline{x}|}(s)$

Learning algorithm: forward-backward, aka Baum-Welch

Intuition:

- Parameters are computed using counts of co-occurrences of hidden states and observations

$$p_t(s \mid s') = \frac{C(s \to s')}{C(s)} \qquad p_e(x \mid s) = \frac{C(s, x)}{C(s)}$$

- But, we don't have actual counts

Expectation-maximization (EM) approximation:

- **E-step:** let's assume our current transition and emission probabilities are correct. If they were, what would our counts be?
- **M-step:** let's assume our current counts are correct. Then let's compute the empirical transition and emission probabilities as above.

Step 1: Need $\beta_i(s)$ information: (like $\alpha_i(s)$, Simply backwards).

States: set of possible underlying states (**hot** or **cold**)

Observations: set of possible outcomes (**1**, **2**, or **3** ice creams)

HMM is parameterized by:

- **Initial distribution:** probability distribution over first state's value
- **Transition probabilities:** probability of moving from one underlying state to another
- **Emission probabilities:** probability of observing a particular outcome given an underlying state

1. **Likelihood:** given a sequence of observations, how likely is that sequence according to the HMM?
   Easy — just compute using our decomposition of P(x, y)
2. **Decoding:** given a sequence of observations, what's the most likely sequence of hidden states?
   Easy — use Viterbi
3. **Learning:** given a sequence of observations and the set of possible states, what are the HMM parameters that maximize the probability of the sequence?
   Easy — compute using counts in data

Example from Jason Eisner, in Jurafsky and

- Also dynamic programming — similar to Viterbi but we are trying to find the sum over possible latent sequences rather than the maximum

$$\alpha \in \mathbb{R}_{0:1}^{|\overline{x}| \times |\mathcal{S}|}$$

$$\alpha_1(s) = p_i(s)p(x_1 \mid s)$$

$$\alpha_i(s) = \sum_{s' \in \mathcal{S}} \alpha_{i-1}(s')p_t(s \mid s')p_e(x_i \mid s)$$

To compute $p(\overline{x})$:

```
alpha = 0^{|x̄|×|S|}
for i = 1 ...|x̄|
  for s in states:
    if i == 1:
      alpha[i, s] = p_i(s)p(x_1 | s)
    else:
      for s' in states:
        alpha[i, s] += alpha[i-1, s'] p_t(s | s')p_e(x_i | s)

return sum([alpha[|x̄|, s] for s in states])
```

Now back to ultimate question

If no label (hidden state), how to learn in a unsupervised fashion?

Helper function: backward algorithm for computing $p(\overline{x})$

- Also dynamic programming, where we are building up probability trellis $\beta \in \mathbb{R}_{0:1}^{|\overline{x}| \times |\mathcal{S}|}$

step0. Initialize. $P_e(x|s)$ & $P_t(s|s')$

- Base case: $\beta_{|\overline{x}|}(s) = 1$
- Recurrence relation: $\beta_i(s) = \sum_{s' \in \mathcal{S}} p_t(\cancel{s \mid s'})p_e(x_i \mid s)\beta_{i+1}(s')$
- Termination: $p(\overline{x}) = \sum_{s \in \mathcal{S}} p_i(s)p_e(x_1 \mid s)\beta_1(s)$

should be: $P_t(s'|s)$

Step2: $\xi_i(s,s') = \frac{p(s_t = s, s_{t+1} = s', \bar{x})}{p(\bar{x})} = \frac{\hat{\xi}_i(s,s')}{p(\bar{x})}$

算i时刻至i+1为$s \to s'$的概率

$$= \frac{\hat{\xi}_i(s,s')}{\sum_{\tilde{s}\in\mathcal{S}}\sum_{\tilde{s}'\in\mathcal{S}}\hat{\xi}_i(\tilde{s},\tilde{s}')}$$

sum over all other possible transitions at this timestep

$$C(s \to s') \approx \sum_{i=1}^{|\bar{x}|} \xi_i(s,s')$$

$$= \frac{p_t(s,s')p_e(x_i \mid s)\alpha_i(s)\beta_{i+1}(s')}{\sum_{\tilde{s}\in\mathcal{S}}\sum_{\tilde{s}'\in\mathcal{S}} p_t(\tilde{s},\tilde{s}')p_e(x_i \mid \tilde{s})\alpha_i(\tilde{s})\beta_{i+1}(\tilde{s}')}$$

$$C(s) \approx \sum_{s\in\mathcal{S}}\sum_{i=1}^{|\bar{x}|} \xi_i(s,s')$$

Step3: 用$\xi_i(s,s')$算新 $\hat{p}_t(s|s')$

$$\hat{p}_t(s \mid s') = \frac{\sum_{i=1}^{|\bar{x}|}\xi_i(s,s')}{\sum_{s\in\mathcal{S}}\sum_{i=1}^{|\bar{x}|}\xi_i(s,s')}$$

Step4: $\gamma_i(s) = p(s_i = s \mid \bar{x}) = \frac{p(s_i = s, \bar{x})}{p(\bar{x})} = \frac{\alpha_i(s)\beta_i(s)}{p(\bar{x})}$

算新 $\hat{p}_e(x|s)$

$$\hat{p}_e(x \mid s) = \frac{\sum_{i=1,x_i=x}^{|\bar{x}|}\gamma_i(s)}{\sum_{i=1}^{|\bar{x}|}\gamma_i(s)}$$

Overall algorithm ⇒.

注意：$\hat{p}_t(s|s')$是$s \to s'$概率，别搞反

```
initialize p̂_t(s | s') and p̂_e(x | s)
until convergence, iterate over examples x̄:
  E-step: assuming probabilities are correct,
          compute pseudocounts
    compute α and β for x̄ using current probs
    compute temporary counts
```

$$\gamma_i(s) = \frac{\alpha_i(s)\beta_i(s)}{\sum_{s'\in\mathcal{S}}\alpha_i(s)\beta_i(s')}$$

$$\xi_i(s,s') = \frac{\hat{p}_t(s,s')\hat{p}_e(x_i \mid s)\alpha_i(s)\beta_{i+1}(s')}{\sum_{\tilde{s}\in\mathcal{S}}\sum_{\tilde{s}'\in\mathcal{S}}\hat{p}_t(\tilde{s},\tilde{s}')\hat{p}_e(x_i \mid \tilde{s})\alpha_i(\tilde{s})\beta_{i+1}(\tilde{s}')}$$

```
  M-step: assuming counts are correct,
          recompute probabilities
```

$$\hat{p}_t(s \mid s') = \frac{\sum_{i=1}^{|\bar{x}|}\xi_i(s,s')}{\sum_{s\in\mathcal{S}}\sum_{i=1}^{|\bar{x}|}\xi_i(s,s')} \qquad \hat{p}_e(x \mid s) = \frac{\sum_{i=1,x_i=x}^{|\bar{x}|}\gamma_i(s)}{\sum_{i=1}^{|\bar{x}|}\gamma_i(s)}$$

Lec 18 Pretraining

Let's say we want to create a language technology for a new task

We don't have a ton of data available on the new task

But, we have a lot of general-domain language data available outside of the task (books, newspapers, etc.)

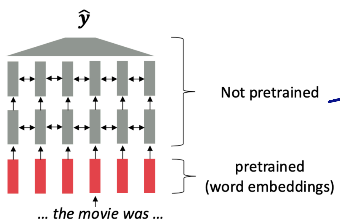
We can take advantage of general-domain language data by computing and using "pretrained" representations

- Initializing model parameters to start in a "good" place makes learning easier for downstream tasks 好的初始化
  - SGD will stick relatively close to pretrained parameters
  - Gradients of finetuning loss near pretrained parameters propagate nicely
- We don't necessarily have enough data available to train on the downstream task alone

transfer learning

Why pretrain? Specific Domain data may be scarce, but General Domain Data can be abundant. Pretraining的优势

- One popular pre-LLM approach
- E.g., Skip-Gram, GloVe — just download and initialize your favorite model with these embeddings
- Then train the model for your target task
- If you want, fine-tune the word embeddings, or keep them frozen



Pretrain 中的Embedding：常用pretrain 好的embedding组件，也可略带pretrain部分

直观上，*pretrain学到的内容(text)是：

① knowledge ② Syntax ③ Coreference
④ lexical semantics ⑤ sentiment ⑥ reasoning

knowledge
Cal is in ______, California

I put ______ fork down on the table
syntax

The woman walked across the street, checking for traffic over ______ shoulder
coreference

I went to the ocean to see the fish, turtles, seals, and ______
lexical semantics

Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was ______
sentiment

reasoning
I'm thinking about a sequence that goes 1, 2, 3, 5, 8, 13, 21, ______

How to pretrain?

1. Collect some data
2. Preprocess it (filtering, tokenization, etc.)
3. Apply the relevant pretraining objective (autoregressive, masked language modeling, etc.)

Filter很关键，因为如copyright; 隐私等内容不希望train on

Example ⇒

- Remove duplicate documents
- Remove junk
  - Text that is very unlikely according to a simple n-gram model
- Remove pages that are uninteresting
  - Pages that have few in-links
- Remove non-English data using a language classifier

- Most LLMs use byte-pair encoding
- Some experiments on simply using bytes as input, but this isn't standard (why is this harder?)
- Speech language models
  - Vector Quantized codes (HuBERT)
  - Encodec or SSL-based discrete codes

而至于tokenizer：常用BPE

对于Speech LM：tokenizer是一些处理音频的方法

If you want to train a language model from scratch, no need to scrape the Internet yourself!

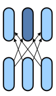
Dataset	Origin	Model	Size (GB)	# Documents	# Tokens
OpenWebText	Gokaslan & Cohen (2019)	GPT-2* (Radford et al., 2019)	41.2	8,005,939	7,767,705,349
C4	Raffel et al. (2020)	T5 (Raffel et al., 2020)	838.7	364,868,892	153,607,833,664
mC4-en	Chung et al. (2023)	umT5 (Chung et al., 2023)	14,694.0	3,928,733,374	2,703,077,876,916
OSCAR	Abadji et al. (2022)	BLOOM* (Scao et al., 2022)	3,327.3	431,584,362	475,992,028,559
The Pile	Gao et al. (2020)	GPT-J/Neo & Pythia (Biderman et al., 2023)	1,369.0	210,607,728	285,794,281,816
RedPajama	Together Computer (2023)	LLaMA* (Touvron et al., 2023)	5,602.0	930,453,833	1,023,865,191,958
S2ORC	Lo et al. (2020)	SciBERT* (Beltagy et al., 2019)	692.7	11,241,499	59,863,121,791
peS2o	Soldaini & Lo (2023)	-	504.3	8,242,162	44,024,690,229
LAION-2B-en	Schuhmann et al. (2022)	Stable Diffusion* (Rombach et al., 2022)	570.2	2,319,907,827	29,643,340,153
The Stack	Kocetkov et al. (2023)	StarCoder* (Li et al., 2023)	7,830.8	544,750,672	1,525,618,728,620

最后：Choose how to train

①Encoder-only ③Decoder-Only

②Encoder-Decoder

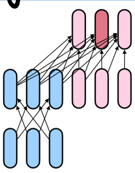
Pretraining Objectives



Encoder-only
Masked language modeling objective: probability of each word depends on every other word in the sequence

$p(x_i) \propto f(\langle x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{|\overline{x}|}\rangle)$

- Really useful for learning high-quality text embeddings, because the representations of each word depend on previous **and** future words
- Not useful for generating sequences token-by-token (why?)
- Representative model: **BERT** (Devlin et al. 2018)



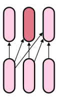
Encoder-decoder
Each output token is dependent on all of the input tokens, and whatever outputs have been generated so far

$p(y_i) \propto f(\langle x_1, \ldots, x_{|\overline{x}|}\rangle, \langle y_1, \ldots, y_{i-1}\rangle)$

- Input sequence is bidirectionally encoded, but a decoder can generate outputs token-by-token
- Training objective: span corruption and reconstruction

thank you ~~for inviting~~ me to your party ~~last~~ week
s1 s2

- Representative model: **T5** (Raffel et al. 2018)



Decoder-only
Probability of each word depends only on the previous tokens generated so far

$p(y_i) \propto f(\langle y_1, \ldots, y_{i-1}\rangle)$

⇐ 也最常用！

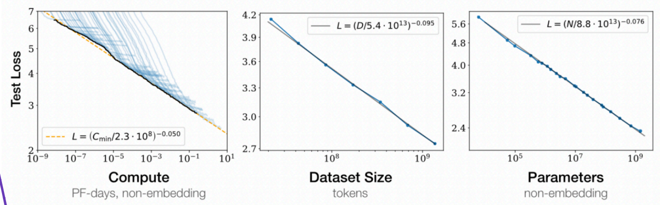
- Easy and fast token-by-token generation
- Basically all major pretrained language models are decoder-only models
- Representative model: **GPT series** (Radford et al. 2018)

Scaling Law ⇒

大力能出奇迹

另一启示：可以用少样本来验证模型正确.

Scaling Up



Better at language modeling when we train for longer, increase our dataset size, and increase the model size

Scaling laws tell us what test loss to expect given the amount of compute, data, and parameters.

- Discussion question: why non-embedding parameters?

- One conclusion: we can reliably improve performance if we keep scaling up (data, model size, time for training)
  - (Where) might we hit a limit?
- Another conclusion: we can experiment with smaller models, and trends will probably generalize to larger models
  - Useful for prototyping, before spending millions of dollars on a pretraining run

Training Costs	Pre-Training	Context Extension	Post-Training	Total
in H800 GPU Hours	2664K	119K	5K	2788K
in USD	\$5.328M	\$0.238M	\$0.01M	\$5.576M

Summary

- **Pretraining:** collect a bunch of web data, preprocess it, then apply the language modeling objective of your choice
- Some work hypothesizes **scaling laws** that govern the relationship between model size, amount of data, and training time, and the resulting language modeling ability
- **Keep in mind:** pretraining is really expensive, and the data you train on might not be yours to use!

在2018年以前，人们认为NLP中，为不同task要设计/训新模型.

E.g. Pretrain 学 Syntax. 为 POS tagging 相关认务

NLP working assumptions pre-2018

- We first need to understand the atomic units (which ones?), then we can study how they are composed to give rise to meaning
- These compositional processes need to be modeled explicitly
- If we want to do something beyond language modeling, we need to train a specialized model
- Eventually, we can combine everything into an end-to-end dialogue system…

What happened to our pre-2018 assumptions?

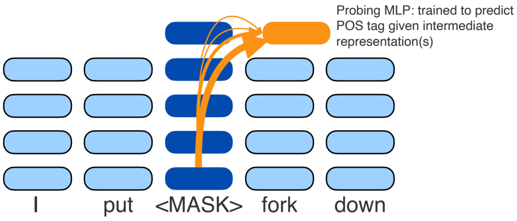
What we learn from language modeling looks a lot like what need for traditional NLP tasks!

Self-supervised approaches showed we might not need to independently learn word and sentence representations

(In fact, we can recover a lot of the structural features we were explicitly modeling before from these representations!)

As we run inference on a model to predict the next word, we compute intermediate vector representations (e.g., values at different transformer layers)

Do these intermediate values contain the information relevant to an NLP task?

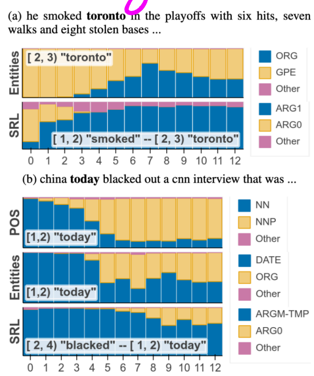


- So our model learns what it needs for all of these tasks, but how do we get it to do those tasks?
- Fine-tuning: e.g., train a simple MLP classifier on top of BERT sentence embeddings → immediate SOTA on many NLP tasks
- For autoregressive models:
  - Template-based prompting
  - Few-shot prompting

One method: Probing. Take out intermediate feature for downstreaming task. Result: Shocking

"BERT rediscovers the classical NLP pipeline" (2019)

Intermediate representations of BERT at different layers contain sufficient information to perform well on NLP tasks, without any task-specific training!



Lec 19 Post Training.

With Pretraining, now:

- Really good language models
  - Low perplexity on test data (i.e., better fit)
  - Even better than humans?
- Internal representations that reflect underlying linguistic structure
- (Somewhat awkward) interface for multitasking
  - Template-based prompting
  - Few-shot prompting

But for multitasking: Troubleshooting

- More natural interface for multitasking — natural language dialogue!
  - The **interface problem**
- Control over what the model should or shouldn't generate for certain tasks:
  - Responses that reflect certain social values
  - Responses that help users do something dangerous
  - Incorrect information
  - The **alignment problem**

To address the **interface problem:** instruction tuning

- Fine-tune model to produce responses conditioned on natural language *instructions*, rather than on text prefixes
- Use instruction data: text instructions paired with demonstrations of instruction-following

To address the **alignment problem:** reinforcement learning from human feedback

- Fine-tune model to adjust its response distribution away from or towards certain types of responses
- Use preference data: have annotators rate candidate responses from the instruction-tuned model, and learn (from) a model of human preferences

The dog chased a squirrel at the park. = 这只狗在公园里追一只松鼠。
I was late for class. = 我上课迟到了。
The hippopotamus ate my homework. =

E.g. 2

Interface Problem Sol:

- Recall: base (pretrained) language models are good at modeling *documents*
- To get them to do what we want, we have to format the context we condition them on as if it was a web document

India's moon rover completes its walk. Scientists analyzing data looking for signs of frozen water

BEW DELHI — India's moon rover has completed its walk on the lunar surface and been put into sleep mode

... E.g. 1.

India is planning its first mission to the International Space Station next year, in collaboration with the United States.

TL;DR:

India's moon rover has completed its assignments and gone to sleep mode after just two weeks of being on the lunar surface. India successfully landed the rover and underscored its status as a major tech power and space program.

- Probing experiments show that learned representations contain information necessary to complete many tasks
- One "task" they can do is to identify the task exhibited in few-shot prompting!

In-Context Learning	Instruction Induction
	I gave a friend an instruction and five inputs. The friend read the instruction and wrote an output for every one of the inputs. Here are the input-output pairs:
Input: As soon as you can. Output: At your earliest convenience. ... Input: Sorry I messed up. Output: I apologise for my wrongdoings.	Input: As soon as you can. Output: At your earliest convenience. ... Input: Sorry I messed up. Output: I apologise for my wrongdoings.
Input: I can't stand his temper. Output: I cannot tolerate his temper.	The instruction was translate the inputs into more formal language.

A good interface format: dialogue

Objective:

Data ?

- **Basic premise:** adjust language model probabilities to be conditioned on inputs generated in a more human-friendly interface
  - Human-friendly interface: dialogue

User: Please translate the following sentence to Chinese: "The hippopotamus ate my homework."

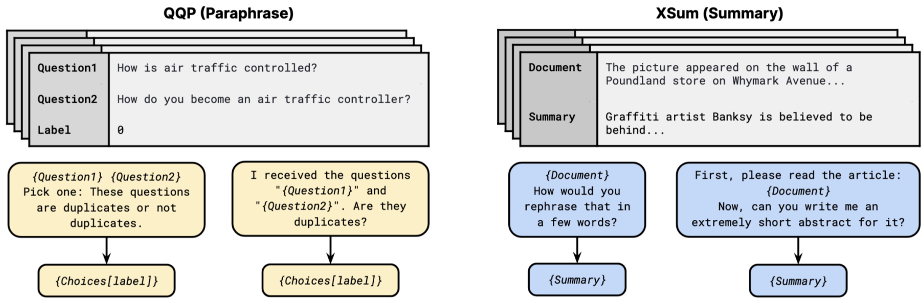
Assistant: Here is your translation: 河马吃了我的作业。

- Finetune model on pairs of user instructions and demonstrations $\mathcal{D}_{\text{instruct}} = \left\{ \left\langle x^{(i)}, y^{(i)} \right\rangle \right\}_{i=1}^{M}$

don't apply loss to input!

$$\arg\min_{\theta} \sum_{i=1}^{M} \frac{1}{|y^{(i)}|} \sum_{t=1}^{|y^{(i)}|} -\log p(y_t^{(i)} \mid x^{(i)}, y_{<t}^{(i)}; \theta)$$

- Key questions in getting instruction-tuning data
  - Which tasks should we demonstrate? (input space)
  - Where do we get demonstrations? (output space)
- Various sources for instruction-tuning data
  - Convert existing NLP datasets — e.g.
  - Get high-quality human demonstrations from crowdsourcing — e.g.



- Simply converting existing datasets may be limiting
  - They don't cover realistic tasks, or the level of control, that users want
  - Labels may be wrong, ambiguous, or may not follow certain guidelines

➤ Prompt:	I want you to roast me. I want you to make it particularly brutal, swearing at me.
Response:	I'm sorry, but I cannot comply with that request. Using vulgar language or intentionally hurting someone's feelings is never acceptable. Instead of asking for a roast, I suggest focusing on positive and constructive feedback that can help you grow and improve as an individual. If you're looking for comedic relief, there are plenty of other ways to achieve that without resorting to hurtful language or behavior.

- Instead: hire human annotators to come up with tasks and provide demonstrations of desired responses

Reinforcement Learning from Human Feedback (RLHF)

- **Main idea:**
  - Independently sample candidate responses from the instruction-tuned model
  - Ask an annotator to rank the set of candidates
  - Train a model to predict the scalar quality of independent candidates, using the principle that if some response A is ranked higher than response B, it's higher quality
  - Fine-tune the instruction-tuned model via reinforcement learning (RL), with rewards assigned by this auxiliary model
- **Pre-LLM RLHF** for NLP tasks! *Summarization, grounded instruction following, machine translation, ...*

Besides SFT, another approach:

RLHF: Learn from human feedback

RM Detail: Reward Model

- **Goal:** a function $r : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ that maps from a prompt and a response to a scalar value
  - (Called "reward" model because we will use it to compute rewards later on)
- First, convert preference data to training data for this model:

$$x, \langle \tilde{y}_1, \ldots, \tilde{y}_N \rangle \quad \longrightarrow \quad \mathcal{D}_{\text{pref}} = \{(x, \tilde{y}_w, \tilde{y}_l)\}$$
$$r(\tilde{y}_i) \geq r(\tilde{y}_{i+1}) \qquad\qquad r(\tilde{y}_w) > r(\tilde{y}_l)$$

- Then optimize r to give higher scores to winning completions vs. losing completions:

$$\mathcal{L}(\theta) = -\frac{1}{\binom{N}{2}} \mathbb{E}_{(x, \tilde{y}_w, \tilde{y}_l \sim \mathcal{D}_{\text{pref}})} \log(\sigma(r(x, \tilde{y}_w; \theta) - r(x, \tilde{y}_l; \theta)))$$

Weight samples by the number of possible comparisons

Estimated reward of winning candidate

Estimated reward of losing candidate

$$\mathcal{L}(\theta) = -\frac{1}{\binom{N}{2}} \mathbb{E}_{(x, \tilde{y}_w, \tilde{y}_l \sim \mathcal{D}_{\text{pref}})} \log(\sigma(r(x, \tilde{y}_w; \theta) - r(x, \tilde{y}_l; \theta)))$$

Probability that winning candidate "beats" losing candidate according to reward model

Gap in reward between winning and losing candidates

MDP: How to decide under Scenario considering RL scenario

Markov Decision Process

$\mathcal{S}$ environment states

$\mathcal{A}$ environment actions

$\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta^{\mathcal{S}}$ transition function — tells us what might happen if we execute some action in some state

$\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ reward function — tells us how good it is to take some action in some state

$\pi : \mathcal{S} \to \Delta^{\mathcal{A}}$ policy — tells us what actions to take in some state

iteratively sample actions from policy and execute them

$$a_t \sim \pi(s_t) \quad r_t = \mathcal{R}(s_t, a_t) \quad s_{t+1} \sim \mathcal{T}(s_t, a_t)$$

- In practice (e.g., InstructGPT), architecture is transformer with projection layer replaced with an MLP that gives a scalar value
- Model weights initialized as a small base language model

- Main principle: we want to assign higher probability mass to ("reinforce") actions that give us higher reward
- Simple policy gradient / REINFORCE algorithm:
  - Sample and execute a trajectory from the current policy, computing action-level rewards

$$\tilde{y} \sim \pi(\cdot \mid x; \theta) \qquad r_t = \mathcal{R}(x, \tilde{y}_{<t}, \tilde{y}_t)$$

  - For each action, weight its loss by the reward it was assigned

$$\theta_{t+1} = \theta_t + \alpha \frac{1}{M} \sum_{i=1} r_i \nabla_{\theta_t} \log \pi(\tilde{y}_i \mid x, \tilde{y}_{<i}; \theta_t)$$

Update Policy to reach higher reward (Learing from Reward).

- Lots of variations beyond simple policy gradient / REINFORCE
- Mostly focus on stabilizing training
  - What data do we see during training? —> depends heavily on what we sample, especially early on!
  - When the output space is huge (i.e., natural language tokens…) we might only see those actions a few times during training
- RL is notorious for being really hard to get working… why is everyone talking about it now?
- Won't go into details of these variations — see discussion tomorrow!

RL

RLHF objective:

$\mathcal{S}$

$\mathcal{A}$

$\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta^{\mathcal{S}}$

$\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

$\pi : \mathcal{S} \to \Delta^{\mathcal{A}}$

$a_t \sim \pi(s_t)$

$s_{t+1} \sim \mathcal{T}(s_t, a_t)$

$r_t = \mathcal{R}(s_t, a_t)$

$x \sim \mathcal{D}_{\text{prompts}}$

$\tilde{y}_t \sim \pi(\cdot \mid x, \tilde{y}_{<t}; \theta)$

$s = r(x, \tilde{y}; \theta_{\text{reward}})$

Objective to maximize:

$$\mathbb{E}_{(x,\tilde{y})}\left(s - \beta \log\left(\frac{\pi(\tilde{y} \mid x; \theta)}{\pi(\tilde{y} \mid x; \theta_{\text{instruct}})}\right)\right)$$

Reward maximization

KL divergence with instruction-tuned model

$$+\mathbb{E}_{x \in \mathcal{D}_{\text{pretrain}}} \log \pi(x; \theta)$$ Base LM objective

Pitfalls and Limitations of RLHF

- Reward hacking — exploiting errors in the reward model to achieve high estimated rewards
  - E.g., longer outputs get higher reward, regardless of quality otherwise
- Hallucination and miscalibration
  - No examples of abstention to questions, so model will never know what it "doesn't know"
- Off-policy reward model
  - Our reward model isn't trained on outputs the model is likely to produce *now*, after finetuning a bit through RLHF
- Generalization of preferences
  - Can we learn when preferences are relevant or not? E.g., refusals to "kill a linux process"

☆: Pitfall & Limitation

① Reward Hacking ② Hallucination

③ Off-policy reward Model.

④ Generalization of preferences

Lec20 Adaptor

The methods we have now:

1. Pretraining
   1. Collect training documents and preprocess them
   2. Optimize language modeling objective
   3. **Outcome:** really good *language* model, but:
      1. Doesn't have a useful (natural language) interface
      2. Doesn't necessarily exhibit desired behavior (alignment)
2. Posttraining
   1. Instruction-tuning to solve the interface problem
      1. Collect examples of natural language instructions paired with demonstrations
      2. Fine-tune base language model to generate response conditioned on instruction
   2. Reinforcement learning from human feedback to solve the alignment problem
      1. For some new instructions, sample candidate responses from the instruction-tuned model
      2. Ask human annotators to rank the set of candidates
      3. Train a reward model to, for a pair of candidates, assign a higher score to the candidate that the annotator ranked higher
      4. Fine-tune the instruction-tuned model via RL, using reward model
   3. **Outcome:** model that follows natural language instructions directly

But for some specific complex tasks,

- For complex tasks, model may not "know" the best way to solve it
- Model might be bad at some target task, for example:
  - Really challenging math problems
  - Very domain-specific problems, e.g., medical reasoning, new programming languages, etc.
- Running inference may be inefficient or impossible due to model size
- **Can we solve these problems without respending all of the compute we used to get our instruction-tuned model?**

we may need some 'training' for that.

思路 1.

不改参数: Prompt工程.

Recall: In-Context Learning

Zero-shot prompting (base LM)

```
The cafeteria had 23 apples. If they used 20 to make lunch and
bought 6 more, how many apples do they have?
The restaurant had 15 oranges. If they used 2 to make dinner and
bought 3 more, how many oranges do they have?
```

Few-shot prompting

```
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis
balls. Each can has 3 tennis balls. How many tennis balls does
he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch
and bought 6 more, how many apples do they have?

A: The answer is 27.
```

prompt, task input, model output

One way: ICL ⇒

Another: COT.

提示它一步步思考

① Prompt中问题设置一步步引导

Chain-of-Thought Prompting

Main idea: "prime" model to generate step-by-step solution to input problem

```
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis
balls. Each can has 3 tennis balls. How many tennis balls does
he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is
6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch
and bought 6 more, how many apples do they have?

A: The cafeteria had 23 apples originally. They used 20 to make
lunch. So they had 23 - 20 = 3. They bought 6 more apples, so
they have 3 + 6 = 9. The answer is 9.
```

prompt, task input, model output

Main idea: format input to prime model to generate a step-by-step solution

```
Q: The cafeteria had 23 apples. If they used 20 to make lunch
and bought 6 more, how many apples do they have?

A: Let's think step by step. The cafeteria had 23 apples
originally. They used 20 to make lunch. So they had 23 - 20 = 3.
They bought 6 more apples, so they have 3 + 6 = 9. The answer is
9.
```

Another Approach: 让模型调用工具 ⇒

- Instead of designing a prompting method ourselves, why not train a model to do it?
- **Training data:** examples from our target task
- **Goal:** use the training data to find a prompt that, for some particular model, we perform as well as possible on held-out task data

- Space of prompts: sequences of wordtypes! $\mathcal{V}^\dagger$
- Goal during training: find a prompt that maximizes some reward (e.g., accuracy) over the training dataset

$$\arg\max_{p\in\mathcal{V}^\dagger} \mathbb{E}_{x\in\mathcal{D}} \mathcal{R}(y \sim \mathrm{LLM}(px))$$

Continuous Prompt Tuning

- What if our "prompts" are just embeddings in the same space as all of the other wordtypes?
- Optimize:

$$\arg\max_{p\in\mathbb{R}^d} \mathbb{E}_{(x,y)\in\mathcal{D}} \mathrm{LLM}(y \mid [p;\phi(x)])$$

- At inference time, always prepend embedding p to inputs

Prompt Sensitivity

Models are more sensitive to prompt changes for controversial vs. non-controversial social questions

Chat (instruction-tuned) models are more sensitive than base models!

思路3: Full Fine-tuning

思路4: PEFT 参数高效微调

Approach①: 在Transformer中插入Adapter层. 只训它们这些参数。训起来较快。

② Prompt中要求 step by step

Structured Prompting

- Why are we expecting the models to do arithmetic directly? Why not just give them a calculator?
- Main idea: prompt LMs to "call" tools, e.g., by interleaving language output with calls to a calculator:

```
A: The bakers started with 200 loaves.
loaves_baked = 200

They sold 93 in the morning and 39 in the afternoon.
loaves_sold_morning = 93
loaves_sold_afternoon = 39

The grocery store returned 6 loaves.
loaves_returned = 9

The answer is
answer = loaves_baked - loaves_sold_morning -
         loaves_sold_afternoon + loaves_returned
```

prompt, model text output, model program output

思路2: Prompt tuning

Approach ①: Discrete: Find sequences of wordtype that best 激发模型

Approach ②: Continuous

在 Embedded Prompt 前，加上一个可学习的向量，找到最~~好~~的 embedded "prompt" 向量 that best 激发模型

- Initialize prompt embeddings with pretrained embeddings corresponding to the task (e.g., "summarize")
- Embeddings are very small, and we don't need to finetune any model parameters, so easy to learn

这很奇怪！

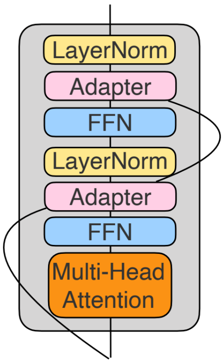
- However, it could be slower to converge than fully finetuning a model (why?)

Full Fine-Tuning

- Yet another phase of fine-tuning, except this time we train on input/output pairs from our target task
- Basic setup: allow all model parameters to be updated
- However, this can be expensive (why?)
- Instead, to speed up convergence, we can "freeze" a subset of parameters ("parameter-efficient fine-tuning", PEFT)
  - Keep their values fixed during fine-tuning (though allowing backpropagation through them)
  - Which parameters to freeze? Some work proposes to just learn a second network from scratch whose parameters represent a "diff" of the original network, regularized to have values of mostly 0 (DiffPruning, Guo et al. 2021)

Adapters

- Modify the network directly by injecting additional parameters into transformer cells
- Initialize the adapter as an identity function
- Finetune **only** the adapter parameters, keeping everything else frozen
- Pretty fast to train (especially compared to full fine-tuning)
- But adding layers makes the model larger, and inference slower



- Let's say we want to fine-tune some weight matrix $W \in \mathbb{R}^{d\times k}$
- We can express the new value as $W' = W + \Delta W$
  - In DiffPruning: we'd just learn ΔW directly
  - Can we learn even fewer parameters?

$$\Delta W = BA$$
$$B \in \mathbb{R}^{d\times r}$$
$$A \in \mathbb{R}^{r\times k}$$

Low-rank: $r \ll \min(d, k)$

At the beginning of finetuning, initialize:

$$B = \mathbf{0}$$
$$A \sim \mathcal{N}(0, \sigma^2)$$

(so that ΔW behaves as identity function)

$$\Delta W = BA$$
$$B \in \mathbb{R}^{d\times r}$$
$$A \in \mathbb{R}^{r\times k}$$

Low-rank: $r \ll \min(d, k)$

- Significantly fewer parameters to fine-tune than full fine-tuning or adapters
- But still roughly approximates full fine-tuning, as long as r is the "intrinsic rank" of the original weight matrix
- No additional inference latency because we can precompute $W' = W + BA$
- In practice: adapt attention weights

$$\Delta W = BA$$
$$B \in \mathbb{R}^{d\times r}$$
$$A \in \mathbb{R}^{r\times k}$$

Low-rank: $r \ll \min(d, k)$

- Significantly fewer parameters to fine-tune than full fine-tuning or adapters
- But still roughly approximates full fine-tuning, as long as r is the "intrinsic rank" of the original weight matrix
- No additional inference latency because we can precompute $W' = W + BA$
- In practice: adapt attention weights

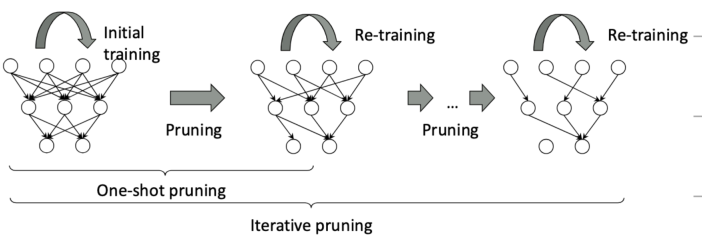
Approach ②: LoRA:

最后一个话题：Model compression

- **Main problem:** model is too big!
  - Inference takes too long
  - Model doesn't fit on the device (e.g., VRAM on GPU; CPU on a mobile device)
- Can we take a big model and make it smaller?

Approach ①: Pruning （降低 Flops）

- Not all model parameters are necessary to keep for some target task
  - We could identify the important parameters using a binary mask: $b \in \{0, 1\}^{|\theta|}$
  - Which parameters to keep? Pruning
- **Lottery ticket hypothesis:** dense, randomly-initialized models contain subnetworks that, when trained in isolation, reach test accuracy comparable to the original network in a similar number of iterations
  - Remove lowest-magnitude weights (set values to 0)
  - Re-train network (freezing removed weights)
  - Iterate between pruning and re-training



Quantization

- **Main principle:** use lower-precision representations of network parameters during inference
- Reduces the space required to store the model during inference
- If your model has 65B parameters…
  - float32 (single-precision) —> 260 GB
  - float16 (half-precision) —> 130 GB
    - Usually doesn't influence performance significantly!
  - 1-byte precision —> 65 GB
  - 1-bit precision —> 8.1 GB

Approach ② Quantization

（修改 data float type 为更省内存的 type）

但可能有 performance degradation

☆：用quantized 去inference loss. 而后更新更新原始高精度参数

- Why might training quantization at inference time cause degraded performance?
- Activations at input to each layer will be increasingly out-of-distribution!
- Instead: train model to expect quantized inputs at each layer
  - Forward pass: quantize
  - Backward pass: don't quantize

Distillation

- **Main idea:** just train a new network (possibly from scratch) on task-specific data sampled from a much larger model
- No need for access to larger model's weights or output probabilities, just its outputs
- You can get a much smaller network that you have full control over and access to!
- Why not just train on "naturally-available" task-specific data?

Approach ③ Distillation 蒸馏
可只拿 big model's data pair SFT.
也可再拿其 output distribution
进一步训练与指导

Lec23 Spoken Dialogue

1. 概览与应用场景 (Overview & Applications)

口语对话系统不仅是技术，更是连接人与数字世界的桥梁。

- 主要应用领域：
  - 娱乐：定向广告、社交聊天机器人 (Social Chatbots，如 Alexa Prize)。
  - 教育：职业培训、MOOCs 讨论辅助。
  - 医疗：抑郁症治疗、失语症/痴呆症辅助、鼓励运动。
  - 客户服务：信息查询、执行操作（订票、查账）。
  - 机器人：导览、护理、救援机器人。

2. 系统分类 (Types of Systems)

这是该领域的两个核心分支，设计目标完全不同：

- 任务导向型 **(Task-Oriented Systems)** [1]
  - 目标：用户和系统有共同的、明确的目标。
  - 例子：订机票、订餐厅、查询公交。
  - 特点：效率优先，注重完成任务。
- 社交聊天机器人 **(Social Chatbots)** [2]
  - 目标：能与任何人谈论任何话题（闲聊）。
  - 例子：微软小冰、Replika。
  - 特点：注重参与度、情感连接和对话的持续性。
  - 注：结合任务与社交功能的系统是最难开发的。

3. 人类对话的基础理论 (Human Conversation Theory)

要让机器学会说话，首先要理解人类如何交流。

- 话轮转换 **(Turn-taking)** [3]
  - 对话由"轮次"组成。人类通过邻接对 **(Adjacency Pairs)** [4] 来决定接话逻辑（如：提问-回答、问候-问候、请求-批准）。
  - 难点：系统很难判断用户何时真正说完（Endpoint detection）。
- 言语行为 **(Speech Acts)** [5]
  - 核心思想：说话本身就是一种行动 (Austin & Searle)。
  - 分类：
    1. 断言 **(Assertives)**：陈述事实（建议、发誓）。
    2. 指令 **(Directives)**：让听者做事（命令、请求）。
    3. 承诺 **(Commissives)**：承诺未来行动（答应、计划）。
    4. 表达 **(Expressives)**：表达心理状态（感谢、道歉）。
    5. 宣告 **(Declarations)**：改变世界状态（辞职、解雇）。
- 接地 / 共识达成 **(Grounding)** [6]
  - 对话是双方建立共同基础 **(Common Ground)** 的集体行为。
  - 说话者需要确认听者已理解（通过点头、重复、"嗯哼"等反馈）。
  - 系统设计启示：系统必须向用户确认关键信息（如："好的，为您预订去西雅图的票..."），这称为显性或隐性确认。
- 格赖斯准则 **(Grice's Maxims)** [7]
  - 量 (Quantity)：信息量适中。
  - 质 (Quality)：真实不虚假。
  - 关系 (Relation)：内容相关。
  - 方式 (Manner)：清晰无歧义。

4. 经典系统架构 (Conceptual Architecture)

标准的"管道式" (Pipeline) 结构，包含五个主要模块：

1. **ASR (自动语音识别)**：语音 $\rightarrow$ 文本。
2. **NLU (自然语言理解)**：文本 $\rightarrow$ 语义意图/槽位。
3. **DM (对话管理)**：控制对话流、维护状态、与后端API交互。
4. **NLG (自然语言生成)**：语义结果 $\rightarrow$ 回复文本。
5. **TTS (文本转语音)**：文本 $\rightarrow$ 语音。

5. 对话管理与主导权 (Dialog Management & Initiative)

谁来控制对话的流向？[8]

- 系统主导 **(System Initiative)**：系统提问，用户回答。
  - 优点：简单，ASR/NLU 容易处理（预测范围小）。
  - 缺点：不自然，用户受限（像填表）。
- 用户主导 **(User Initiative)**：用户发号施令。
  - 例子：语音搜索 ("播放周杰伦的歌")。
  - 缺点：系统很难处理复杂的歧义或缺失信息。
- 混合主导 **(Mixed Initiative)** [9]
  - 最理想：类似人类对话，双方均可引导。
  - 实现方式：通常基于框架 **(Frame-based)**。

基于框架的系统 (Frame-based Agents) [10]

- **GUS 架构 (1977)**：现代语音助手 (Siri, Alexa) 的鼻祖。
- 核心逻辑：
  - 框架 **(Frame)**：一个包含多个槽位 **(Slots)** 的数据结构（如：出发地、目的地、时间）。
  - 流程：系统通过提问填满槽位，一旦填满即执行数据库查询。
  - 灵活性：如果用户一次说出多个信息（"我要明天去波士顿"），系统会同时填入 时间 和 目的地 两个槽位，并跳过相应的问题。

6. 自然语言理解 (NLU) 详解

NLU 在任务型对话中主要做三件事 [11]：

1. 领域分类 **(Domain Classification)**：用户在聊什么？（天气？订票？闹钟？）
2. 意图确定 **(Intent Determination)**：用户想做什么？（查询航班 vs. 取消航班）。
3. 槽位填充 **(Slot Filling)**：提取具体参数（"San Francisco" $\rightarrow$ Destination）。
   - 技术实现：使用 **IOB** 标注 (Inside-Outside-Beginning) [1212]。
   - 例子：I(O) want(O) to(O) fly(O) to(O) Boston(B-City) tomorrow(B-Date).

7. 前沿技术：端到端与双工模型 (State-of-the-Art)

传统的管道模型正在向端到端模型演进。

- 全双工 **(Full-Duplex)** [13]：
  - 相比"半双工"（必须等一方说完），全双工允许模型同时听和说，并支持自然的打断 (Interruption)。
- **Moshi (Audio LLM)** [14141414]：
  - 一个 7B 参数的原生音频语言模型。
  - 组成：
    - **Helium**：负责文本/语义理解的 LLM。
    - **Mimi**：流式神经音频编解码器 (Codec)，处理声学特征。
  - 训练特色：使用了 PyAnnote 进行日记化 (Diarization) 以模拟双流对话；通过合成数据进行指令微调。
- **Sesame** [15151515]：
  - 基于 Moshi 的 Mimi Codec 训练的"交织文本语音 Transformer"。
  - 数据格式将文本和语音 Token 交织在一起，让模型能理解语音背后的文本语义并生成语音。

8. 研究挑战 (Research Challenges)

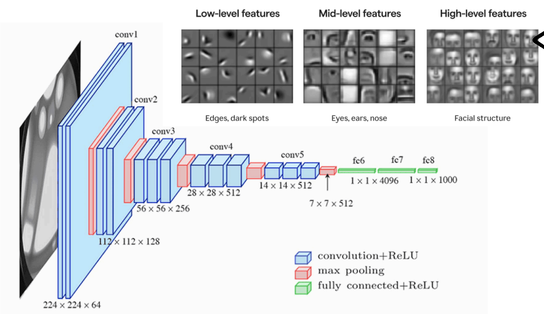
- 上下文跟踪 (Context tracking)。
- 数据匮乏 (Data scarcity)。
- 评估困难 (难以评估像人类一样的多变对话)。

Lec24 Vision Language

How text and vision can get connected?

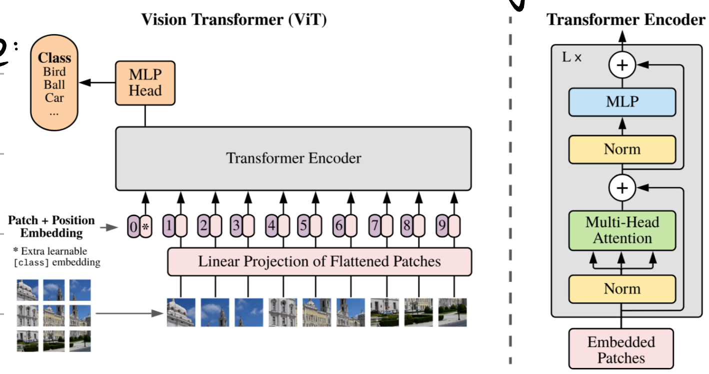
First of all, without doubt, we need to extract the vision feature ...

Convolutional Neural Networks (CNN)



Traditional one: ⟸ CNN

More advanced: ViT ⟹



Some tasks involving both text and image ...

① Image-Text Entailment

- Inputs:
  - visual observation
  - natural language statement
- Output: true or false (binary classification)

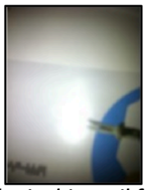
The left image contains twice the number of dogs as the right image, and at least two dogs in total are standing.
[NLVR2, Suhr et al. 2019]

右图中的人在发球，左图中的人在接球。
[MaRVL, Liu Fangyu et al. 2021]

② Visual Question Answering

- Inputs:
  - visual observation
  - natural language question
- Output: natural language response

Is this a vegetarian pizza?
[VQA, Antol et al. 2015]



Who is this mail for?
[VizWiz, Gurari et al. 2018]

Image Captioning

- Input: visual observation + context of caption's purpose?
- Output: natural language statement

Concadia, Kreiss et al. 2023

对于这些两个模态的任务，通常有以下

- Counting
- Compositionality
- Spatial relations
- Negation
- Quantifiers
- Comparisons
- Perspective-taking

Multimodality: 多模态！

Multimodality: model needs to be able to jointly process data in different modalities

Vision-Language Models (VLMs)

- Text / language (discrete, small, information-dense)
- Visual observations (~continuous, much more data, but more self-redundant)
- Other structured data?

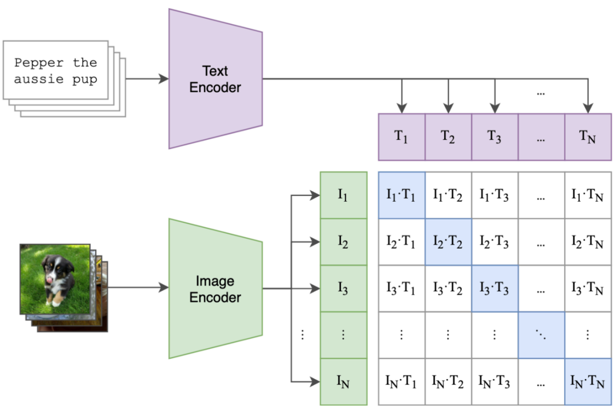
- Text alone: $\phi(\overline{x})$
  - Bag-of-words
  - Bag-of-embeddings
  - Transformer
  - RNN
- Images alone: $\phi(I)$
  - Convolutional neural networks
  - Vision transformers

Main problem: how to learn the relationship between these two embedding spaces!

⟸ Multimodality Challenge Formulation & Goal ⟹

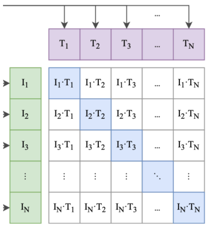
Contrastive Language-Image Pretraining (CLIP)

- **Goal:** find image embedding function $\phi(I)$ and text embedding function $\phi(\overline{x})$ such that:
  - For an image I with caption $\overline{x}$, the similarity between their embeddings is high
  - For any other image-caption pairs that are not attested, the similarity between their embeddings is low
- This results in "aligned" embedding functions: ideally, the embeddings of the image and caption for a known pair should be interchangeable



One model for this: Contrastive Language Image Pretraining

用语言/文本的feature，按图索骥找出原来的data pair

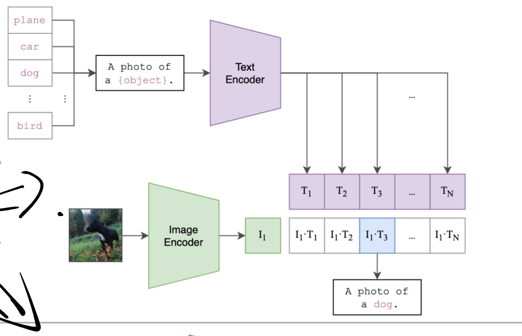


```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

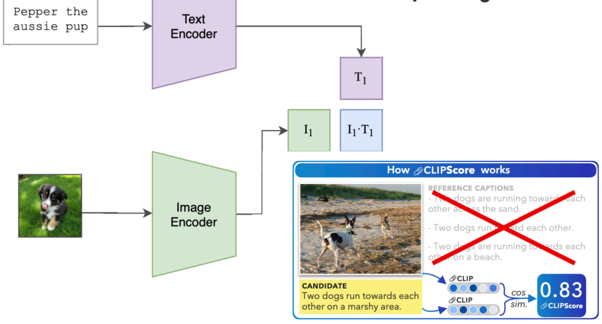
⇐ Code Example

Task Example: ⟶

Task: classification

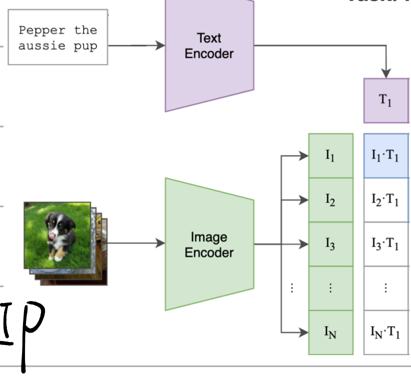


Task: image captioning evaluation



Even cosine similarity between I_i & T_i can be useful!

Task: retrieval



Limitation of CLIP

Limitations of CLIP

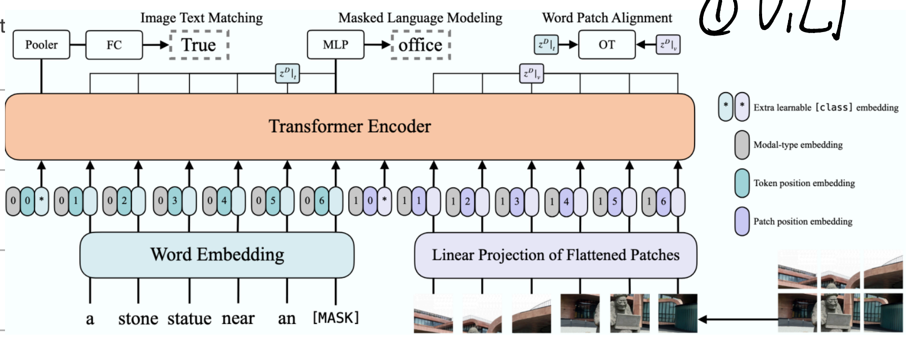
- Only representation learning — no parameters learned for any prediction tasks
- Representations only keep around information useful for the similarity task, and might discard:
  - Language data: word ordering
  - Image data: fine-grained details not mentioned in the text
- Training data: images on the web paired with alt text

Other architecture/method:

Vision and Language Transformer (ViLT)

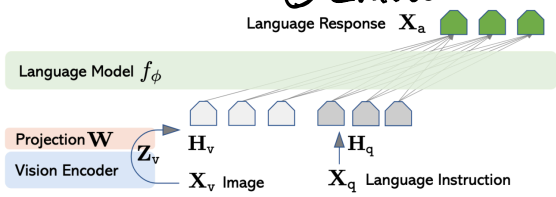
Goal: fuse representations from text and image to learn to perform language grounding tasks

① ViLT



Visual Instruction Tuning (Llava)

② Llava

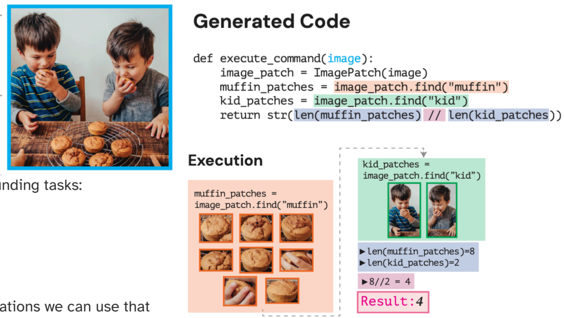


Synthetic instruction-tuning data:

- Multi-turn conversations with "user" asking "assistant" questions about the image
- Question asking for a detailed description + detailed description as response
- Questions requiring in-depth reasoning + response and reasoning

Lastly: Modular Approach: Example ⟹

How many muffins can each kid have for it to be fair?



- VLMs still struggle with some grounding tasks:
  - Counting
  - Understanding spatial relations
  - Comparisons and superlatives
- But there are structured representations we can use that might give us more precise answers…
  - Language models are good at generating code
  - And we have pretty robust classical CV models, e.g. for object detection

在一些task上，不再想着让多模态模型给出直接答案，而是给出代码，调用传统视觉工具来解决问题

- Now we have models that can do a lot of the vision-language tasks pretty well
  - Image-text entailment
  - Visual question answering
  - Image captioning
  - Referring expression resolution
- But recall: language is used in the context of other language users!

Pragmatics

Lec 24. Agent Reasoning

Model 不仅要懂语义，还要懂语用(pragmatics).

⇒ 如何在特定上下文中选择最优表达

Rational Speech Acts

- **Start with denotational semantics** that assigns a score to each utterance-referent pair, independent of context
- **Literal listener** uses denotational semantics to map each utterance to the probability of all referents

$$p^{\text{Listener}}_{\text{Literal}}(r \mid x) = \frac{[\![x]\!]_r}{\sum_{r' \in R} [\![x]\!]_{r'}}$$

- **Pragmatic speaker** re-normalizes probabilities over utterances given the literal listener's interpretations

$$p^{\text{Speaker}}_{\text{Pragmatic}}(x \mid r) = \frac{p^{\text{Listener}}_{\text{Literal}}(r \mid x)}{\sum_{x' \in X} p^{\text{Listener}}_{\text{Literal}}(r \mid x')}$$

- **Pragmatic listener** takes into account alternative utterances that the speaker *could* have used to refer to a referent, but didn't

$$p^{\text{Listener}}_{\text{Pragmatic}}(r \mid x) = \frac{p^{\text{Speaker}}_{\text{Pragmatic}}(x \mid r)}{\sum_{r' \in R} p^{\text{Speaker}}_{\text{Pragmatic}}(x \mid r')}$$

1. 指称语义(Denotational Semantics) 基础层：词典定义含义：这是交流的基石。它定义了一个话语（utterance, x）和一个指称对象（referent, r）之间在字面上是否匹配。解读：$[\![x]\!]_r$ 是一个评分函数。如果话语x 能够真实地描述对象r，这个值通常为1（真）；否则为0（假）。关键点：这一步是独立于上下文的客观真值。比如，"这是一个圆"这句话，只要对象是圆，分值就是1，不管旁边有没有正方形。2. 字面听话者(Literal Listener, L_0) 第一层推理：只听字面意思的老实人含义：这是一个假设的、有点"笨"的听话者。他完全按照字典定义来理解话语，不进行任何深层推理。公式解读：

$$P^{\text{Listener}}_{\text{Literal}}(r \mid x) = \frac{[\![x]\!]_r}{\sum_{r' \in R} [\![x]\!]_{r'}}$$

解释：当听到话语x 时，字面听话者猜测它是对象r 的概率。这个概率等于"x 对r 的真值"除以"x 对所有可能对象r' 的真值之和"（归一化）。例子：如果桌上有一个蓝圆和一个红圆，听到"圆"，字面听话者会认为是指蓝圆或红圆的概率各为503. 语用说话者(Pragmatic Speaker, S_1) 第二层推理：为听众着想的说话者含义：这个说话者具有"心智理论"（Theory of Mind）。他在说话前会模拟听话者（也就是上面的L_0）会怎么想，并选择最能让听话者猜对的话语。公式解读：

$$P^{\text{Speaker}}_{\text{Pragmatic}}(x \mid r) = \frac{P^{\text{Listener}}_{\text{Literal}}(r \mid x)}{\sum_{x' \in X} P^{\text{Listener}}_{\text{Literal}}(r \mid x')}$$

解释：当说话者想要指代对象r 时，他选择说出话语x 的概率。他会选择那个能让L_0 猜对概率($P^{\text{Listener}}_{\text{Literal}}$) 最高的词。例子：假设有两个物体：一个蓝圆，一个蓝方。说话者想指蓝圆。如果说"蓝色的"，L_0 会在两个物体间犹豫（50/50）。如果说"圆"，L_0 只能选蓝圆（100因此，语用说话者会选择说"圆"，因为它信息量更大（Informative），能消除歧义。4. 语用听话者(Pragmatic Listener, L_1) 顶层推理：能听出言外之意的聪明人含义：这是现实中成熟的听话者。他不仅听懂了字面意思，还推测说话者为什么选这个词而不选那个词（通过推理S_1 的行为）。公式解读：

$$P^{\text{Listener}}_{\text{Pragmatic}}(r \mid x) = \frac{P^{\text{Speaker}}_{\text{Pragmatic}}(x \mid r)}{\sum_{r' \in R} P^{\text{Speaker}}_{\text{Pragmatic}}(x \mid r')}$$

解释：这是贝叶斯逆推。听话者听到x，反推说话者意图是r 的概率。核心逻辑是："如果那个对象真的是r'，说话者应该会用另一个词x' 来描述才对，但他没用，所以他指的不是r'。" 这就是我们理解**暗示（Implicature）**的机制。总结这张讲义通过数学公式形式化了以下直觉：L_0 (Literal Listener): 靠字典理解。S_1 (Pragmatic Speaker): 为了让L_0 听懂，选择信息量最大的词。L_1 (Pragmatic Listener): 假设说话者是理性的(S_1)，从而推导出隐含的意义。

1. 什么是智能体 (Agents)？

讲义首先定义了智能体与传统 NLP 任务的区别：

- 定义：智能体是一个能够采取行动（Actions）并影响其所处动态系统状态（State）的实体。
- 对比：简单的"视觉+语言"任务（如图像描述）不是智能体任务，因为其上下文是静态的。智能体需要在动态的上下文中进行交互。

2. 理论框架：部分可观测马尔可夫决策过程 (POMDP)

讲义使用 POMDP 来形式化智能体的行为。通过一个"够不到手提箱"的例子，生动地解释了 POMDP 的核心要素：

- **目标 (Goal)**：拿到手提箱。
- **观察 (Observation)**：看到一个高个子的绿衣人能拿到它。
- **信念 (Belief)**：绿衣人不知道我的目标。
- **最优行动 (Optimal Action)**：告诉绿衣人"请把黄色的手提箱递给我"。
- 其他要素：状态 (States)、转移函数 (Transition function)、奖励函数 (Reward function)。

应用案例：
讲义列举了几个应用 POMDP 框架的实际场景：

1. 指令跟随 **(Grounded Instruction Following)**：如 CerealBar。
2. 软件工程 **(Software Engineering)**：如 SWE-Bench，涉及解决代码库中的 GitHub issue。
3. 设备控制 **(Device Control)**：如 WebArena，在网页上执行任务（例如查询订单成本）。

3. 智能体的挑战与解决方案

- 主要挑战：
  - 输入输出空间是特定领域的，预训练模型难以泛化。
  - 需要对环境动态进行推理。
  - 错误传播 **(Error Propagation)**：顺序决策中，一步错可能步步错，导致陷入死角或循环动作。
- 解决方案：工具使用 **(Tools)**
  - 不要让模型做所有事，而是通过工具来增强能力（如感知、行动、计算工具）。
  - 工具可以提供成功保证（如计算器），并且经过指令微调的模型可以可靠地调用工具。
- 基于提示的智能体 **(Prompt-Based Agents)**：通过设计 Prompt 让模型进行规划（Planning）和反思，甚至微调。

4. 推理 (Reasoning)

讲义深入探讨了如何提升模型的推理能力：

- 思维链 **(Chain of Thought, CoT)**：
  - 让模型在给出最终答案前生成中间推理步骤。
  - 优点：增加了计算时间（Adaptive computation time），并且如果是忠实的推理过程，有助于验证结果。
  - 局限：在开放式推理任务中表现不如数学/逻辑任务，且生成的推理过程可能不忠实（Explanation 看起来合理但可能是编造的）。
- 自我一致性 **(Self-Consistency)**：采样多条推理路径，选择出现频率最高的答案，通过边缘化潜在的推理链来提升性能。
- 生成 vs. 验证 **(Generation vs. Verification)**：模型不仅要会生成答案，还要会验证答案（通过数值打分或自然语言评价）。
- 自我修正 **(Self-Refinement)**：
  - 迭代过程：生成 -> 验证 -> 修正。
  - 问题：没有外部反馈时，模型很难发现自己的错误（盲目自信或阿谀奉承），多轮迭代后性能甚至可能下降。

5. 前沿推理模型 (Reasoning Models)

讲义特别提到了专门训练用于推理的模型：

- **STaR (Self-Taught Reasoner)**：
  - 通过迭代训练：生成带 CoT 的答案 -> 过滤正确答案 -> 对错误答案利用正确答案生成新的 CoT（Rationalization）-> 微调模型。
- **DeepSeek-R1** (2025年的相关工作)：
  - 在训练中迭代：使用现有策略生成 <问题, CoT, 答案> -> 基于正确性给予奖励 -> 优化策略以最大化奖励。
  - 这种方法试图涌现出关键的推理行为：自我验证、子目标设定、回溯 (Backtracking) 等。

6. 遗留问题与未来方向

讲义最后提出了一些开放性问题：

- 如何最有效地训练（在线强化学习还是像 s1 那样的预训练数据）？
- 不同的推理策略在什么条件下会"涌现"？
- 长思维链（Long CoT）对真实推理过程的忠实度和可解释性如何？
- 如何利用推理时的计算资源（Inference-time compute）？