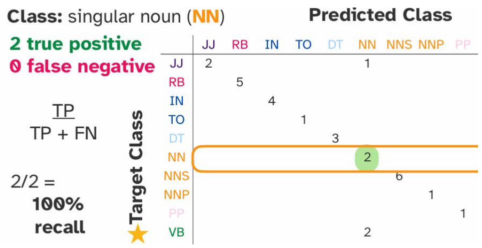


F1: $\frac{2 \times P \times R}{P + R}$ WER: $\frac{S+D+I}{N}$

WER can be greater than 100% (eg. Too many insertions)

d [i, j] = min (d [i-1, j] + 1 ,
d [i , j -1] + 1 ,
d [i -1 , j -1] + local_substitution (i , j))

Optimal path: tells us what the errors were

$G_n(x)$ is the set of n-grams in sequence x
$G_1(y) = \{$the, most, natural, form, of, language, use, is, dialogue, .$\}$
$G_2(y) = \{$the most, most natural, natural form, form of, ...$\}$

$C(s, x)$ is the number of occurrences of n-gram s in x
C(the, y) = 1 C(most, y) = 1 C(natural form, y) = 1

$$P_n(\hat{y}, y) = \frac{\sum_{s \in G_n(\hat{y})} \min(C(s,\hat{y}), C(s,y))}{\sum_{s \in G_n(\hat{y})} C(s,\hat{y})}$$

$$\mathrm{BP}(\hat{y}, y) = \begin{cases} 1 & |\hat{y}| \geq |y| \\ e^{(1-(|y|/|\hat{y}|))} & |\hat{y}| < |y| \end{cases}$$

$$\textsc{Bleu} = \mathrm{BP}(\hat{y}, y) \exp\left(\sum_{n=1}^{N} \frac{1}{N} \ln P_n(\hat{y}, y)\right)$$

In well-defined tasks, we can define slots that need to be filled before providing information to the user. However, this reduces the expressivity of dialogue

Use another model (or a set of rules) to “simulate” a human user

Allows scaling up experiments, including to more complex domains

Allows stability across system evaluations

But doesn’t reflect the real-world complexity of actual use case, e.g., user adaptation to systems over time

Why do we need benchmark?

Estimating how well our models will work on real-world data

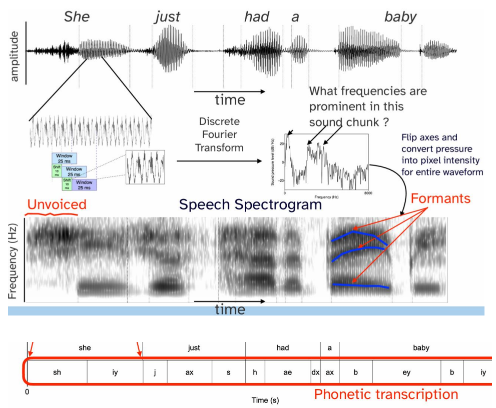
Shared understanding of model performance with standardized evaluations

Building trust within a community in proving how well a new model does

Driving progress towards specific tasks and capabilities

What are the properties of the outputs models produce in general? • Consistency: when evaluating models on the same task (or variations of it) multiple times, how consistent is its behavior? • Diversity: do the outputs models generate have the same distributional properties as human language?

Pitfall: Dataset Contamination, Spurious Correlations, Defining “Human Performance”, The Long-Tail Paradox



- Spectrogram reveals some segmental structure with distinct properties
- These are phonemes — perceptually distinct speech sounds

IPA: Phoneticians compiled a common set of sounds used to codify different speech sounds (across languages)

Initial vocabulary is the set of all bytes (characters) across the texts

Until the target vocabulary size is reached, repeat the following:

- Tokenize all of the texts using the current vocabulary
- Find the most common bigram in the tokenized texts, then add it to the vocabulary as a new wordtype

Denotational semantics: the symbol *refers to* something in the context in which the language is used

“pet” is a hypernym of “dog”; “draught animal” is a hyponym of “domesticated animals”

Core principle: **distributional hypothesis**

- Words that are used in similar contexts have similar meanings
- Context: typically, other words in a text, but really anything can be context!

$$\arg\max_{\theta} \prod_{(w,c)\in\mathcal{D}} p((w,c)\in\mathcal{D}) \prod_{(w,c)\in\mathcal{D}'} p((w,c)\notin\mathcal{D})$$

$$p((w,c)\in\mathcal{D}) = \frac{1}{1+\exp(-\mathrm{score}(c,w))}$$

$$p((w,c)\notin\mathcal{D}) = 1 - p((w,c)\in\mathcal{D})$$

Negative samples: sample and train on $l * |D|$ pairs (w', c') where

$w' \sim p(W)$ unigram prior over words $c' \sim p(C)$ unigram prior over contexts

$$p((w,c)\in\mathcal{D}) = \frac{1}{1+\exp(-\cos(\phi(c), \phi(w))}$$

$$\text{a : b :: c : ?} \equiv d = \arg\max_{i \in \mathcal{W}} \frac{(\phi(b) - \phi(a) + \phi(c)) \cdot \phi(i)}{||(\phi(b) - \phi(a) + \phi(c)) \cdot \phi(i)||}$$

- $\mathcal{D} = \{w_i^A, w_i^B\}_{i=1}^M$: dataset pairing *M* word translations
- Find a matrix *W* such that $\min_W \sum_{i=1}^{M} ||W\phi(w_i^A) - \phi(w_i^B)||^2$

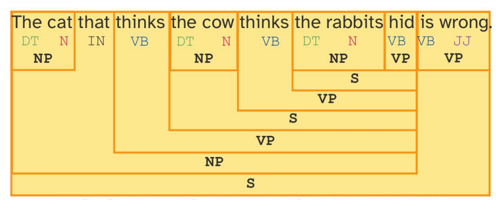
CFGs offer arbitrary expressivity through recursive structure

- Set of nonterminal symbols
- Set of terminal symbols (wordtypes)
- Set of production rules defining how nonterminal symbols could be expressed via the composition of other nonterminal and terminal symbols

Nonterminal symbols
DT N VB JJ IN
S NP VP SBAR

Terminal symbols (vocabulary)
are, cat, cow, hid, is, rabbits, that, the, thinks, wrong, ...

Production rules
DT → {*the, a, an, ...*}
N → {*cat, cow, rabbits, dogs, ...*}
VB → {*hid, is, thinks, was, are, ...*}
JJ → {*wrong, right, blue, red, ...*}
IN → {*that, in, of, because, ...*}
NP → {DT N, NP IN VP}
VP → {VB, VB JJ, VB S}
S → {NP VP, VP}



NP is singular, so its corresponding VP should be too

Nonterminal symbols
DT N VB JJ IN
S NP VP

Production rules
NP → {DT N, NP IN VP}
VP → {VB, VB JJ, VB S}
S → {NP VP, VP}

CFG is often called phrase structure or constituency grammar

Each production rule describes a **constituent**

Constituent constructions are independent of one another (this is why the grammar is context-free)

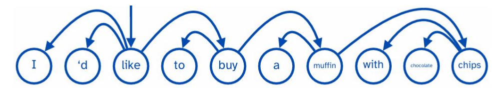
- Augmenting a CFG with agreement (e.g., distinguishing plural vs. singular NPs and plural vs. singular VPs plural) means it is no longer context-free
- Also, some languages aren’t even context-free, not even considering agreement:

CCG: Combinatory Categorial Grammar

Elements of a CCG:

- Lexical items (wordtypes)
- Each paired with a syntactic type (≈ nonterminal or composition thereof)

the : NP/N	dog : N	John : NP	bit : $(S\backslash NP)/NP$
If a Noun appears to the right, then it creates a Noun Phrase	Noun	Noun Phrase	If a Noun Phrase appears to the right, then it creates an element with the type S \ NP
NP → DT N DT → *the*	N → *dog*	NP → PRO PRO → *John*	If an NP appears to the left of that element, it creates a Sentence



Truth-Conditional Semantics:

We’d like the outcome of our semantic parsing to be a some code that can evaluate to true or false (i.e., $\mathcal{W} \rightarrow [0,1]$)

```
everyone                     likes                         Pepper
S / (S \ NP)                 (S \ NP) / NP                 NP
< e → t > → t                e → < e → t >                 e
λ f . ∀ x                    λ x, y . likes (y, x)         Pepper
(person(x) → f(x))
                                          S \ NP
                                          e → t
                                   λ y . likes (y, Pepper)
                          S
                          t
λ f . ∀ x (person(x) → f(x)) (λ y . likes (y, Pepper))
∀ x (person(x) → (λ y . likes (y, Pepper)) (x)
∀ x (person(x) → likes (x, Pepper))
```

Logical operators, like ∨, ∧, and ¬
*Pepper is clever **and** curious*

Quantifiers like ∀ and ∃
***Some** cats like water*

Relationships between functions ⇒ and ⇔
Squares are rectangles (∀x (square(x) → rectangle(x)))

Verbs can have tenses, and can be modified with adverbs

We can talk about beliefs others have

Some combinations of meanings are nonsensical (unevaluable)
green ideas

Sentences aren’t just statements — sometimes they are commands, questions, etc.

Sentences exist in the context of previous sentences and their meanings

Semantics: mapping from surface form (sequence of tokens) to formal executable representation

Pragmatics: executing the logical form against some context to acquire its denotation

Speech Act; Presupposition; Implication

By interpreting speech as action, we can ascribe intent to utterances that isn’t obvious from their formal representation

Propositions that must be true about a world in order to compute the denotation of a particular sentence

Propositions suggested by an utterance, but not explicitly expressed

Gricean Maxims; Common Ground

General principles we believe we mutually hold about how what kinds of utterances we should add to conversation given what's been said so far:

During interaction, we maintain some representation of what we believe is <u>mutually known</u> by conversation participants

Elements of Scenario Design

Interaction dynamics also depend heavily on the properties of the context itself:

- Incentive structure
- Environment design — perception and action, novelty
- Participants — how many, any existing structures among them, roles, a priori asymmetries
- Communication channel

Work in computational linguistics, psycholinguistics, and cognitive science aims to characterize the relationship between scenario design and linguistic behavior

Multilingual MLP and its challenges

For any task we expect out of language technologies, they should work for any language

Data Modality, Data Scarcity, Dialectical Variation, Speech System, Morphology, Lexical Semantics, Syntax, Semantics, Idioms, Difference in Language Use, Change

Autoregressive language modeling:

- The probability of a sequence is a product of local token probabilities
- The probability of a token depends on the ones that came before it

Masked language modeling:

- The probability of a sequence is a product of local token probabilities
- The probability of a token depends on the ones that came before *and after* it

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1} \ldots x_n)$$

Let's make a Markov assumption:
The probability of word at index i only depends on the n - 1 words that came before it

$$p(X_i = x) = p(x \mid x_1, \ldots, x_{i-1})$$

$$\approx p(x \mid \underbrace{x_{i-n+1}, \ldots, x_{i-1}}_{\text{Preceding (n-1)-gram}})$$

$$\approx \frac{C(x_{i-n+1}, \ldots, x_{i-1}, x) \text{ count of n-gram}}{C(x_{i-n+1}, \ldots, x_{i-1}) \text{ count of prev (n-1)-gram}}$$

For an n-gram language model, we need to store counts for:

- All sequences of length n ($\mathcal{V}^n$)
- All sequences of length n - 1 ($\mathcal{V}^{n-1}$)

What if the count of the target n-gram is 0?

- Solution: add a small number to the count for every n-gram (aka "smoothing")

What if our n-1-gram prefix has a count of 0?

- Solution: condition on a shorter n-gram prefix (e.g., the previous n-2, or n-3, etc.) instead (aka "backoff")

Can't learn anything from the counts of n-grams containing similar words

Without a big n, cannot handle long-distance dependencies

Measure of Fit (拟合优度度量)

Likelihood: probability of the data under our model

$$\prod_{i=1}^{M} p_\theta(\overline{x}_i)$$

Negative log likelihood (fixes float underflow)

$$-\sum_{i=1}^{M} \log p_\theta(\overline{x}_i) = -\sum_{i=1}^{M}\sum_{j=1}^{N} \log p_\theta(x_j^i \mid x_1^i, \ldots, x_{j-1}^i)$$

Perplexity: inverse probability of data, normalized by number of tokens in the dataset

$$= \exp\left(-\frac{1}{\sum_{i=1}^{M}|\overline{x}^i|}\sum_{i=1}^{M}\sum_{j=1}^{N} \log p_\theta(x_j^i \mid x_1^i, \ldots, x_{j-1}^i)\right)$$

Adjust the Temperature when generating

Operation: modify the logits *before* computing probabilities

$$s(w) = f(w \mid x_1, \ldots, x_{i-1}; \theta) \quad \Leftarrow \text{logits}$$

$$p(X_i = w \mid x_1, \ldots, x_{i-1}) = \frac{\exp(s(w)/\tau)}{\sum_{w' \in \mathcal{V}} \exp(s(w')/\tau)}$$

A better approximation for global argmax: **beam search**

- During generation, we maintain a "beam" of n sequences instead of just one
- At each generation step i,
  - We select the n most likely next tokens $\mathcal{X}_i$ for each prefix, and create n more sequences
  - Then we look at all the n^2 sequences so far, and discard all but the n most likely sequences
- At the end, we select the sequence that has the highest probability among the set

Masking Out Wordtypes: (&constrained)

ϵ sampling top-k sampling top-p (nucleus) sampling

Identify the set of n tokens $\mathcal{E}$ such that $\forall\, x \in \mathcal{E}$, $p(x \mid x_1, \ldots, x_{i-1}) \geq \epsilon$

Identify the set of k tokens $\mathcal{K}$ that have the highest probabilities under $p(X_i \mid x_1, \ldots, x_{i-1})$

Identify the set of n tokens $\mathcal{P}$ that have the highest probabilities under $p(X_i \mid x_1, \ldots, x_{i-1})$ and their cumulative probability is p

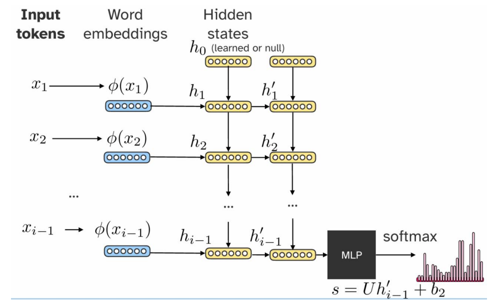
Similar to before: given a set of possible continuations $\mathcal{C} \subseteq \mathcal{V}$ we will set the probabilities of all other tokens to 0, then renormalize using $\sum p(X_i \mid x_1, \ldots, x_{i-1})$

$p(x_i \mid x_{i-n+1}, \ldots, x_i; \theta)$ neural n-gram

$$= \frac{\exp(s(x_i \mid x_{i-n+1}, \ldots, x_{i-1}; \theta))}{\sum_{x' \in \mathcal{V}} \exp(s(x' \mid x_{i-n+1}, \ldots, x_{i-1}; \theta))}$$

Softmax over wordtype scores

$$s(x \mid x_{i-n+1}, \ldots, x_{i-1}; \theta) = f_\theta(\phi(x_{i-n+1}), \ldots, \phi(x_{i-1}))[x]$$

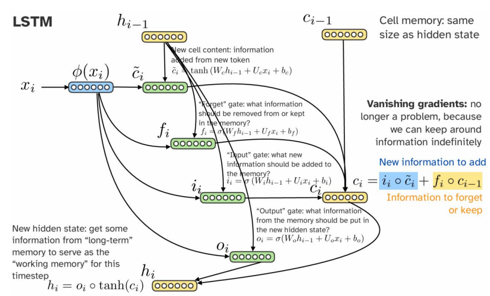


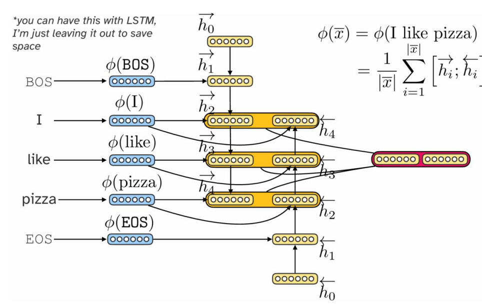
$$\theta^* = \arg\min_{\theta \in \mathbb{R}^{|\mathcal{V}| \times d}} \left(-\sum_{\overline{x} \in \mathcal{D}}\sum_{i=1}^{|\overline{x}|} \log p(x_i \mid x_1, \ldots, x_{i-1}; \theta)\right)$$

Vanishing gradients: gradient signal from many timesteps in the future is negligible compared to gradient signal from nearby tokens

Exploding gradients: if the activations are too big, the gradients become too big, and the values of the weights become too big, eventually getting values closer to ∞

Can solve using gradient clipping (if the norm of the gradient is greater than a threshold, clamp its value or scale it down)

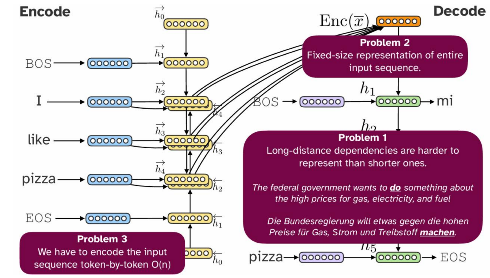




$$\text{Enc}(\overline{x}) = \text{Pool}\left(h_1, \ldots, h_{|\overline{x}|}\right) \in \mathbb{R}^d$$

"Pooling" functions map from a sequence of items of type t to a single item of type t

Seq2seq (With challenges); Attention



We encoded our input sequence into hidden states: $h_1, \ldots, h_{|\overline{x}|} = \mathbf{h} \in \mathbb{R}^{d \times |\overline{x}|}$

Now we want to predict the next word y_{i+1}

We have access to the previous decoder hidden state g_i

First, compute attention scores for each input hidden state using similarity between query (g_i) and keys ($\mathbf{h}$): $s_i = a(g_i, \mathbf{h}) \in \mathbb{R}^{|\overline{x}|}$

Then, take softmax of attention scores to get a distribution over keys: $\alpha_i = \text{softmax}(s_i) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$

Finally, compute a weighted sum of values ($\mathbf{h}$) using this distribution $c_i = \sum_{j=1}^{|\overline{x}|} \alpha_{i,j} h_j \in \mathbb{R}^d$

Use the weighted sum to predict the next word: $p(Y_{i+1} \mid \overline{x}, y_1, \ldots, y_i) = \text{softmax}(f(c_i, g_i))$

Use the weighted sum to update the decoder hidden state: $g_i = g(g_{i-1}, c_{i-1}, y_i)$

$$h_1, \ldots, h_{|\overline{x}|} = \mathbf{h} \in \mathbb{R}^{d \times |\overline{x}|}$$

O(n) $s_i = a(g_i, \mathbf{h}) \in \mathbb{R}^{|\overline{x}|}$

O(n m) $\alpha_i = \text{softmax}(s_i) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$

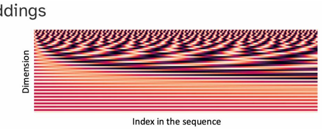
$$c_i = \sum_{j=1}^{|\overline{x}|} \alpha_{i,j} h_j \in \mathbb{R}^d$$

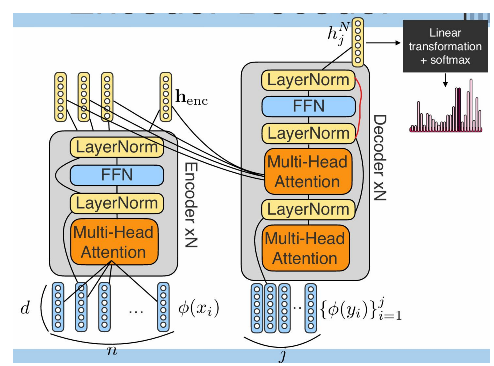
$p(Y_{i+1} \mid \overline{x}, y_1, \ldots, y_i) = \text{softmax}(f(c_i, g_i))$ O(m)

$$g_i = g(g_{i-1}, c_{i-1}, y_i)$$

- Sinusoidal embeddings

$$p_i = \begin{bmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{bmatrix}$$





But we don't want the model to learn to rely on "future" words in the sequence!

Solution: during training, set $s_{ij} = -\infty$ if i (query index) < j (key/value index)

BERT: **Instead: randomly sample some tokens to replace with a MASK placeholder token in the input, then train the model to predict those words**