Natural Language Processing
Lecture Notes

KnightNemo
Instructed by TianXing He

0.1 Logistics

- Homework A(mainly theory, 20pts)
- Homework B(mainly coding, 20pts)
- In-class Exam(20 pts)
- Project(40pts, outstanding porjects get +5 bonus)

0.2 Scoring thresholds

- A+: 94-100
- A: 89-93
- A-: 84-88
- B+: 80-83

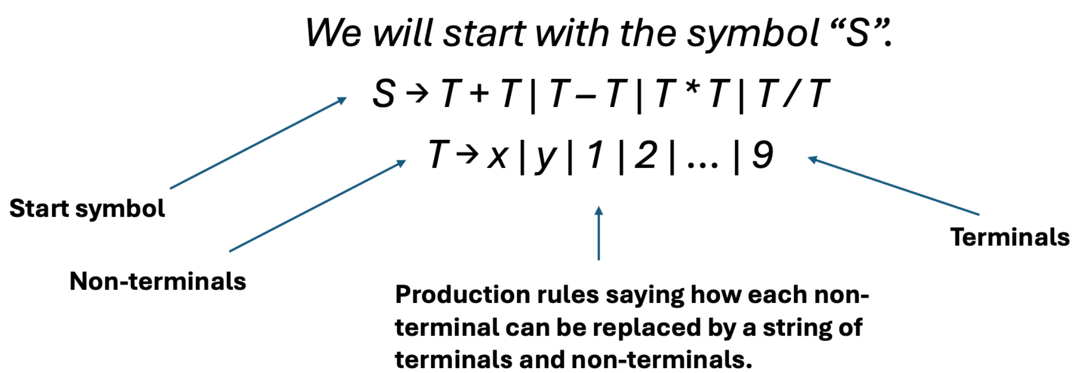
0.3 Reference

- [**SLP**] Speech and Language Processing–Jurafsky&Martin
- [**INLP**] Introduction to Natural Language Processing–Eisenstein

Part I: NLP before LLM

1 Context-Free Grammar(CFG)

1.1 Terminology



Given G as a CFG.

The **language** of G, denoted as $L(G)$, is the set of strings derivable by G (from the start symbol).

A language L is called a **context-free language (CFL)** if there is a CFG G such that L = L(G).

Theorem : Every regular language (regular expressions, regex), is context-free.

CFG example: Natural Language

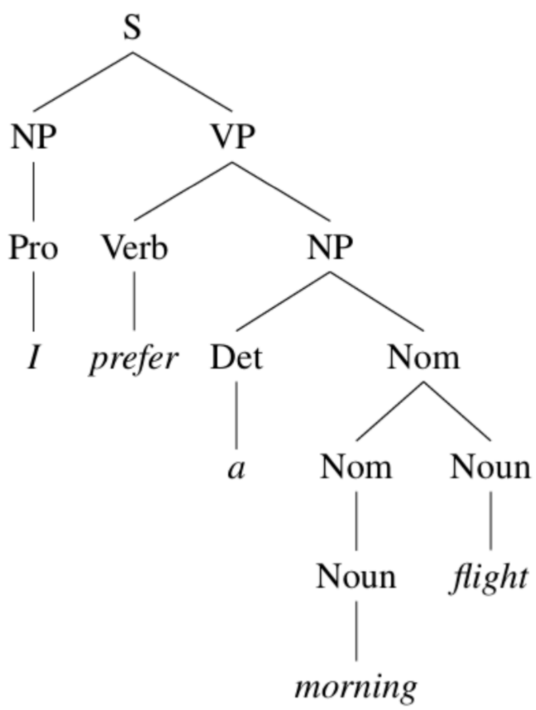
Grammar Rules			Examples
S	$\rightarrow$	*NP VP*	I + want a morning flight
NP	$\rightarrow$	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	$\rightarrow$	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	$\rightarrow$	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
PP	$\rightarrow$	*Preposition NP*	from + Los Angeles

1.2 CNF

Def. Chomsky normal form (CNF) A CFG is in **Chomsky normal form (CNF)** if it is **ε- free** and if in addition Chomsky normal form each production is either of the form **A** $\rightarrow$ **B C** or **A** $\rightarrow$ **a**.

1.3 Parsing

Given a CFG, **syntactic parsing** refers to the problem of mapping from a sentence to its parse tree.



1.3.1 CKY Parsing

Transfer the CFG to CNF, then dynamic programming.

function CKY-PARSE(*words, grammar*) **returns** *table*

  for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
    for all $\{A \mid A \rightarrow words[j] \in grammar\}$
      $table[j-1, j] \leftarrow table[j-1, j] \cup A$
    for $i \leftarrow$ **from** $j-2$ **down to** 0 **do**
      for $k \leftarrow i+1$ **to** $j-1$ **do**
        for all $\{A \mid A \rightarrow BC \in grammar$ **and** $B \in table[i,k]$ **and** $C \in table[k,j]\}$
          $table[i,j] \leftarrow table[i,j] \cup A$

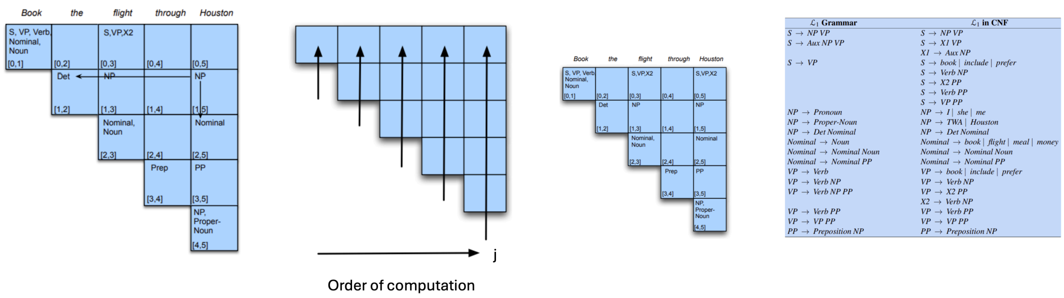
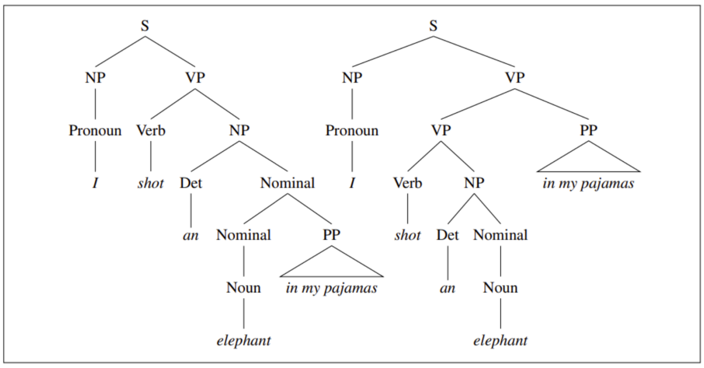


Abb. 1: Procedure of CKY-Parsing and Results

1.3.2 Limitation

Ambiguity



1.3.3 Probabilistic context-free grammars (PCFG)

$$A \rightarrow B\ \ C \text{ with probability } 0.4$$

$$A \rightarrow D\ E \text{ with probability } 0.6$$

A corpus in which every sentence is annotated with a parse tree is called a treebank.

1.3.4 Neural CKY

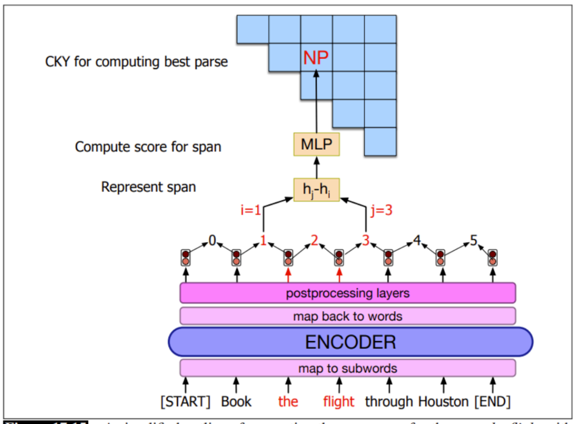


Figure 17.15 A simplified outline of computing the span score for the span *the flight* with the label NP.

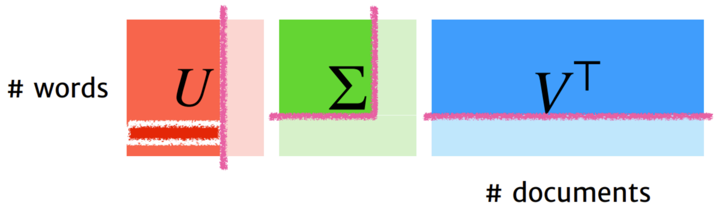
2 Latent Semantic Analysis(LSA)

2.1 Term-Document Matrix

- rows are words
- columns are documents
- entries indicate how many times word i appears in document j

$$W_{td} = \begin{array}{c} \\ cat \\ dog \\ the \end{array} \begin{array}{c} \begin{array}{ccccccc} d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 & 1 & 1 & 0 \\ 20 & 13 & 18 & 22 & 15 & 4 & 20 \end{bmatrix} \end{array}$$

- rows as |D|-dim word representations
- columns are |V|-dim document representations



2.2 Normalization

2.2.1 Problem

SVD would pay too much attention to the high-freq words!

2.2.2 TF-IDF normalization

- term frequency (tf):

$$\frac{\text{\# of times word i appears in doc j}}{\text{\# of words in doc j}}$$

- inverse document frequency (idf), smoothed version:

$$\log\left(\frac{\text{\# of docs + 1}}{\text{\# of docs containing word i + 1}}\right) + 1$$

- count'(i, j) = tf · idf

2.2.3 Pointwise mutual information (PMI)

p(w) = # of times w appears in any document / word count

p(d) = fraction of documents identical to doc d (constant)

p(w, d) = # of times w and d appear together / (# words x # docs)

PMI(i, j) = p(w, d) / (p(w) p(d))
≈ p(d | w) if p(d) is assumed to be constant

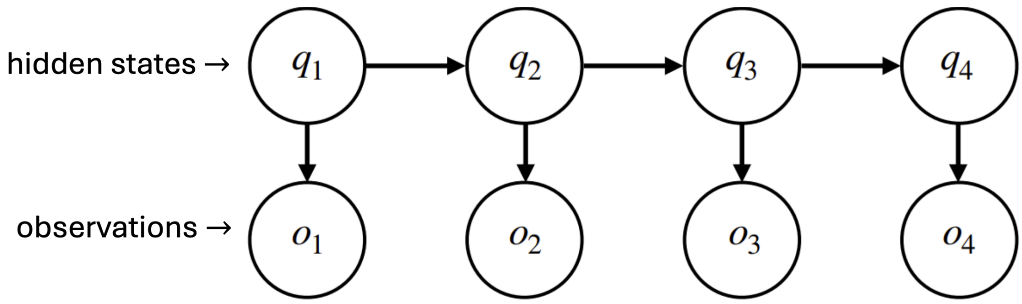
3 Hidden Markov Model (HMM)

3.1 Motivating task

Part-of-speech (POS) tagging

Noun	*Verb*	*Noun*	*Noun*	*Num*	*Noun*
Fed	raises	interest	rates	0.5	percent

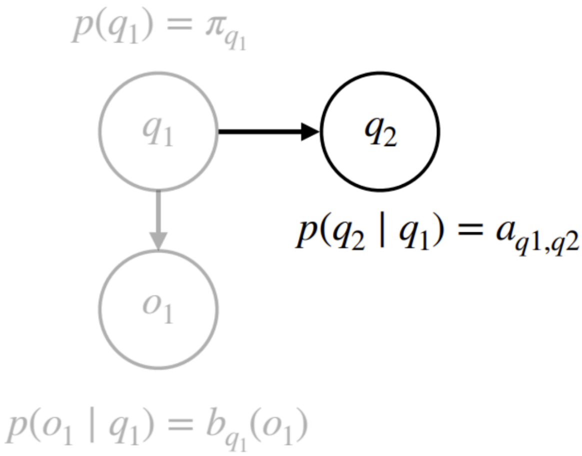
3.2 HMM Generation



$$p(Q, O) = p(q_1)p(q_2|q_1)p(q_3|q_2)p(q_4|q_3)p(o_1|q_1)p(o_2|q_2)p(o_3|q_3)p(o_4|q_4)$$

↑ Why it's called Markov...

We denote all **hidden-hidden transition probabilities** by **A** , and all **hidden- emission probabilities** by **B**.



3.3 The forward algorithm

Notice that

$$p(O_{:t}, q_t = j) = p(o_t|q_t = j) \sum_i p(O_{:t-1}, q_{t-1} = i) p(q_t = j|q_{t-1}, = i)$$

We denote $p(O_{:t}, q_t = j)$ as $\alpha(t, j)$.

$$\alpha(t, j) = b_j(o_t) \sum_i \alpha(t-1, i) a_{\mathrm{ij}}$$

$$\alpha(1, j) = \pi_j b_j(o_1)$$

This is also dynamic programming.

$$p(O) = \sum_i p(O, q_t = i) = \sum_i \alpha(t, i)$$

So the total runtime is $O(TN^2)$.

The forward algorithm gives us $p(O)$, $p(O_{:t}, q_t = j)$.

3.4 The backward algorithm

Backward algorithm gives us $p(O, q_t = i, q_{t+1} = j)$.

Notice that

$$p(O_{t+1:}|q_t = i) = \sum_j [p(O_{t+2:}|q_{t+1} = j) p(q_{t+1} = j|q_t = i) p(o_{t+1}|q_{t+1} = j)]$$

which can be rewritten as:

$$\beta(t, i) = \sum_j \beta(t+1, j) a_{ij} b_j(o_{t+1})$$

$$\beta(T, i) = 1$$

So we know how to compute:

$$\alpha(t, i) = p(O_{:t},\ q_t = i)$$

$$\beta(t, i) = p(O_{t+1:} \mid\ q_t = i)$$

3.5 Combined

Combined we can get

$$p(O, q_t = i) = \alpha(t, i)\beta(t, i)$$

$$p(O, q_t = i, q_{t+1} = j) = \alpha(t, i)a_{ij}b_j(o_{t+1})\beta(t + 1, j)$$

3.6 Inference: most probable tag sequence

Noting that $\text{argmax}_Q(Q|O) = \text{argmax}_Q(Q, O)$.

The Vertabi Algorithm:

$$\underbrace{\boxed{\max_{Q_{t-1:}} p(O_{:t}, Q_{:t-1}, q_t = j)}}_{\text{best length-}t\text{ tag seq. ending in }j} = \max_i \overbrace{\boxed{\max_{Q_{t-2:}} p(O_{:t-1}, Q_{t-2:}, q_{t-1} = i)}}^{\text{best length-}t\text{-1 tag seq. ending in }i} \cdot p(q_t = j \mid q_{t-1} = i) \cdot p(o_t \mid q_t = j)$$

which can be rewritten as:

$$\delta(t,j) = b_j(o_t) \max_i \delta(t-1,i)\ a_{ij} \qquad \delta(1,j) = \pi(j)\ b_j(o_1)$$

3.7 Acquiring π, a, b

- Supervised Learning:(we have labels of q)

$$\pi_i = p(q_1 = i) = \frac{\#(q_1 = i)}{\#\text{sequences}}$$

$$a_{ij} = p(q_t = j \mid q_{t-1} = i) = \frac{\#(q_{t-1} = i,\ q_t = j)}{\#(q_{t-1} = i, q_t = *)}$$

$$b_i(w) = p(o_t = w \mid q_t = i) = \frac{\#(q_t = i,\ o_t = w)}{\#(q_t = i)}$$

- Unsupervised Learning:

We consider the objective:

$$\log p(O|\theta) = \log \sum_Q p(O, Q|\theta)$$

then for some distribution $q(Q)$, we have:

$$\log p(O|\theta) = \log \sum_Q p(O, Q|\theta) = \log \sum_Q \left[q(Q) \frac{p(O, Q|\theta)}{q(Q)} \right] = \log \sum_{q(Q)} \left[\frac{p(O, Q|\theta)}{q(Q)} \right]$$

applying Jensen's inequality gives us:

$$\log p(O|\theta) = \log \sum_{q(Q)} \left[\frac{p(O, Q|\theta)}{q(Q)} \right] \geq \sum_{q(Q)} \log p(O, Q|\theta) + \text{Entropy}(q(Q)).$$

For given $q(Q)$ the entropy term is fixed, so we need only maximize $\sum_{q(Q)} \log p(O, Q|\theta)$.

Define $q(Q) := p(Q|O, \theta_k)$, denote the objective as $Q(\theta|\ \theta_k)$.

Actually

$$logp(O|\theta) = \Sigma_Q q(Q) logp(O,Q|\theta) + KL(q(Q)||p(Q|O,\theta)) + entropy(q(Q))$$

$$q(Q) := p(Q|O,\theta_k)$$

so when θ and θ_k are close enough, so maximizing $\sum_{q(Q)} \log p(O,Q|\theta)$ is similar to maximizing the true objective.

$$Q(\theta|\theta_k) = \Sigma_Q p(Q|O,\theta_k) logp(O,Q|\theta)$$

$$= \Sigma_Q p(Q|O,\theta_k) log \prod a_{q_{t-1}q_t} b_{q_t}(o_t)$$

$$= \Sigma_{t,i,j} p(Q_{t-1} = i, Q_t = j|O,\theta_k) loga_{ij}$$

$$+\Sigma_{t,j} p(Q_t = j|O,\theta_k) \log b_j(o_t)$$

the blue terms can be computed using the forward-backward algorithm.

- Optimize for A (hint: minimize the KL divergence, blackboard), we got

$$\hat{a}_{ij} = \frac{\Sigma_{t,O} p(Q_{t-1} = i, Q_t = j, O|\theta_k)}{\Sigma_{t,O,*} p(Q_{t-1} = i, Q_t = *, O|\theta_k)}.$$

(This is computed over a set of observations)

- The update rule for B is similar and left for exercise.

4 N-Gram

4.1 Language Model

A Language Model assigns a probability of any sequence of words. So, if W denotes any sequence of words, $W \in V^*$, we have:

$$\sum_W P_{LM}(W) = 1$$

- A word **token** (sometimes we just call it "word") is a specific occurrence of a word in a text.
- A word **type** refers to the distinct form of a word, regardless of how many times it appears in a sentence or text. It is the unique identity of the word.

4.2 Naive Approach: Unigram LM

Assume each word is independent.

$$P_{unigram}(w_1 \dots w_T) = P(w_1)P(w_2)\dots P(w_T)$$

Problem:

$$P_{\text{unigram}}(\text{I study NLP at THU}) = P_{\text{unigram}}(\text{I study THU at NLP})$$

4.3 Bigram, Trigram, N-gram

Consider **pairs** of words. It's basically just a table lookup!

P_{bi}(<bos> *NLP we at THU* study <eos>)
= *P(NLP|<bos>) · P(we|NLP) · P(at|we) · P(THU|at) · P(study|THU)*
· P(<eos>|study)

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

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

We can extend to tri-gram and N-grams. As we know

$$P(w_{1:T}) = \prod_{i=0}^{T} p(w_i|w_{1:i-1}),$$

So basically N-gram is history truncation

$$p(w_i|w_{1:i-1}) \approx p(w_i|w_{i-N:i-1}).$$

- special tokens:
  - <eos>: end of sentence token
  - <unk>: out-of-vocabulary token

4.4 Data Sparsity

E.g. „We study anthropology in THU."

$$P_{tri}(anthropology \mid we\ study) = \frac{count(we\ study\ anthropology)}{count(we\ study\ *)}$$

the probability is near zero. How do we deal with it?

4.4.1 Add-k Smoothing

$$P_{tri}(w_t \mid w_{t-2}w_{t-1}) = \frac{count(w_{t-2}w_{t-1}w_t) + k}{count(w_{t-2}w_{t-1}) + k|V|}$$

k is hyperparameter that is tuned.

4.4.2 Interpolation

$$\begin{aligned} P_{tri}(w_t \mid w_{t-2}w_{t-1}) = \ & \lambda_1 P_{tri}(w_t \mid w_{t-2}w_{t-1}) \\ & + \lambda_2 P_{bi}(w_t|w_{t-1}) \\ & + \lambda_3 P_{uni}(w_t) \end{aligned}$$

where $\sum_i \lambda_i = 1$.

4.4.3 Backoff

$$P_{tri-BO}(w_t \mid w_{t-2}w_{t-1}) = \begin{cases} P^*_{tri}(w_t \mid w_{t-2}w_{t-1}) & \text{if count}(w_{t-2}w_{t-1}w_t) > 0 \\ \alpha(w_{t-2}w_{t-1})P_{\text{bi}}(w_t \mid w_{t-2}w_{t-1}) & \text{otherwise} \end{cases}$$

4.5 Perplexity

A metric for LM evaluation. Smaller Perplexity means better LM.

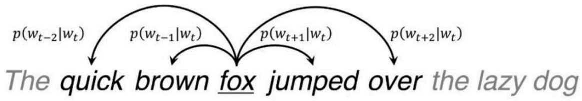
$$PPL(W) = 2^{-l}, \qquad \text{where } l = \frac{\log_2(P(W))}{token\_len(W)}$$

5 Word2Vec

5.1 Tasks

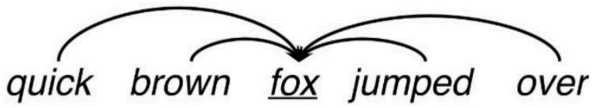
5.1.1 Skip-gram

Learn representations that predict the context given a word.



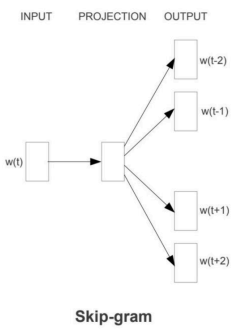
5.1.2 CBOW (Continous Bag-of-Words)

Learn representations that predict a word given context.



5.2 Parameters to Learn

5.2.1 Skip-Gram objective



The p(out|input) is simply a dot product of corresponding vectors then softmaxed.

$$p_\theta(\mathsf{out} \mid \mathsf{input}) = \frac{\exp(u_{\mathsf{out}} \cdot w_{\mathsf{input}})}{\sum_{v \in V} \exp(u_v \cdot w_{\mathsf{input}})}$$

In practice, the window size is a hyperparameter.

it is a far , far better rest that I go to , than I have ever known

Loss (NLL) for this window

$$L_t = -\log p_\theta(x_{t-2} \mid x_t) - \log p_\theta(x_{t-1} \mid x_t) - \log p_\theta(x_{t+1} \mid x_t) - \log p_\theta(x_{t+2} \mid x_t)$$

Doing Gradient Descent on this objective yields:

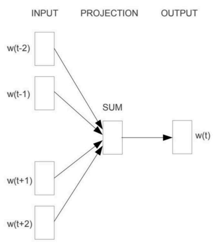
$$w_x \leftarrow w_x + \eta \left(u_y - \mathbb{E}_{p_\theta(v \mid x)} [u_v] \right)$$

Output embedding for correct label

What the model thinks is the correct output

i.e. Move towards pointing in the same direction as the true output embedding minus the expected output embedding under the model.

5.2.2 CBOW objective



The desired probability is of the following form (Z is normalizing term):

$$p_{\text{CBOW}}(x_t | x_{t-s}, ..., x_{t+s}) = \frac{\exp\left(u_{x_t} \cdot \frac{1}{2s} \sum_{j=-s}^{s} w_{t+j}\right)}{Z}$$

In practice, the window size is a hyperparameter.

it is a far , far better rest that I go to , than I have ever known

$$L_t = -\log p_\theta(x_t \mid x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2})$$

5.2.3 Trick: Negative sampling (skip-gram version)

The computation of the loss function is expensive:

$$\begin{aligned} \log p_\theta(y \mid x) &= \log \frac{\exp(u_y \cdot w_x)}{\sum_{v \in V} \exp(u_v \cdot w_x)} \\ &= u_y \cdot w_x - \log \boxed{\sum_{v \in V} \exp(u_v \cdot w_x)} \end{aligned}$$

Takes $O(V)$ to compute.

Idea: We turn the prediction into a binary classification task.

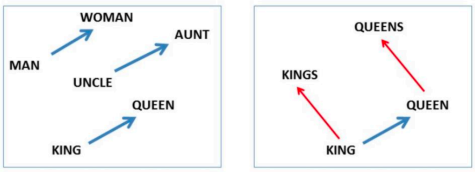
For each true pair <x,y>, we sample k negative samples y'.

$$\log \sigma(u_y \cdot w_x) + \Sigma_i E_{y' \sim P_n} [\log \sigma(-u_{y'} \cdot w_x)].$$

where σ is the sigmoid function, and P_n can be a unigram model.

5.3 Word Vector Properties

5.3.1 Linear Word Analogies

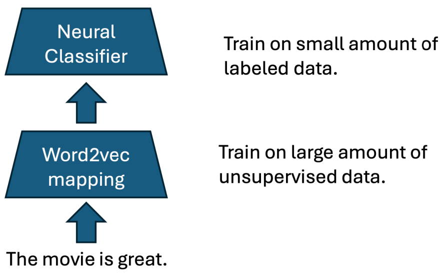


$$w_{\text{man}} - w_{\text{woman}} \approx w_{\text{king}} - w_{\text{queen}}$$

$$w_{\text{apple}} - w_{\text{apples}} \approx w_{\text{car}} - w_{\text{cars}}$$

Applications:

- Word embedding initialize + finetuning



  an useful practice before BERT.
- Compositional Morphology

  help build embeddings for rare words.

$$\begin{aligned} \overrightarrow{\text{imperfection}} &= \overrightarrow{im} + \overrightarrow{perfect} + \overrightarrow{ion} \\ \overrightarrow{\text{perfectly}} &= \overrightarrow{perfect} + \overrightarrow{ly}. \end{aligned}$$

Word2Vec works better than LSA(empirically).

Part II: Neural Networks & LLMs

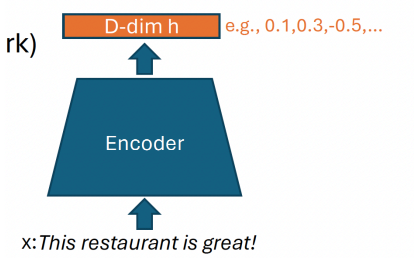
6 Brief Review of ML/DL Basics

6.1 KL-Divergence

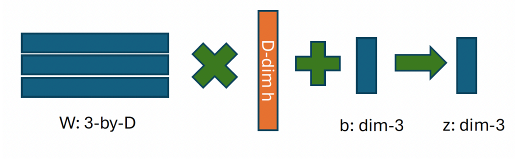
$$\mathcal{D}_{\text{KL}}(p\|q) := \int_{-\infty}^{+\infty} p(x) \ln \frac{p(x)}{q(x)} dx = \sum_{x \in X} p(x) \ln \frac{p(x)}{q(x)}$$

6.2 Multi-Class classification

- **Task description**: 3-class sentiment classification
  - This restaurant is great! $\to$ positive
  - The food is okay. $\to$ neutral
  - I hate this dish! $\to$ negative
- General Recipe: **Encode, Predict, Train**
  - Encode: an encoder(e.g., a neural network) which maps the input x to a D-dim vector h



- Predict:



Linear Transformation: $z = W^{\text{cls}}h + b^{\text{cls}}$

Then we apply sofmax: (map $z \to \Pr(y|x)$)

$$\begin{aligned}
\text{softmax}(z) &= \left[\frac{\exp(z_1)}{\sum_{i=1}^{k}\exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k}\exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k}\exp(z_i)}\right] \\
\text{softmax}(z_i) &= \frac{\exp(z_i)}{\sum_{j=1}^{k}\exp(z_j)} \quad 1 \le i \le k \qquad \text{<- k is the number of classes.}
\end{aligned}$$

- Train:

Cross-Entropy Loss:

$$L_{\text{CE}} = \sum_i -\log \Pr(y = y_i | x_i)$$

Update by SGD:

$$\theta^{t+1} = \theta^t - learningrate \cdot \frac{\partial}{\partial\theta} L_{\text{CE}}(\text{mini} - \text{batch}\{x_i, y_i\})$$

6.3 Neural Networks

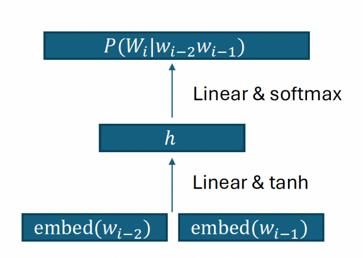
- MLP
- Back-Propogation
- Dropout Regularization
  - Dropout is a regularization technique for neural networks that randomly drops a unit (along with connections) at training time with probability p
  - At test time, all units are present, but with weights scaled by p.
- Parellel Computation

7 Neural Network Language Model

7.1 FeedForward NN LM

e.g. tri-gram neural network version

$\mathcal{L} = \sum_{(w_{i-2}, w_{i-1}, w_i) \in \text{ data}} -\log \Pr(w_i | w_{i-1}, w_{i-2})$



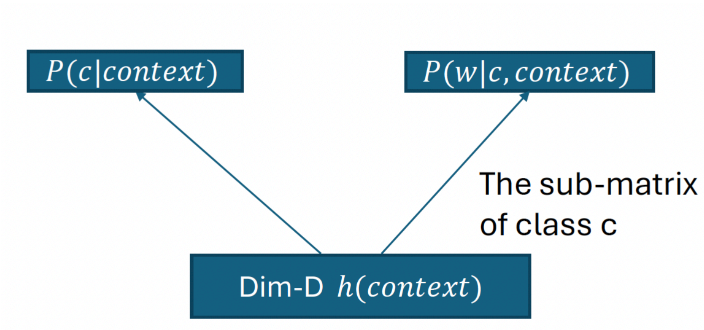
Different from sentiment classification,output class number is now $|V|$.

- Remedy 1: **Class-based LM**

  Idea: cluster words into $\sqrt{V}$ clusters.

  Computation Cost: $D|V| \to 2D\sqrt{V}$

  $\mathcal{L} = -[\log \Pr(c_w|h) + \log \Pr(w|c_w, h)]$



- Remedy 2: **Noise Contrastive Estimation**
  - Training without explicit normalization
  - Discriminating between the target token and noise tokens
  - Key speed-up: $p_\theta(w|h)$ does not need to be normalized (no softmax). NCE training will automatically normalize it.

$$J^h(\theta) = \mathbb{E}_{P_d^h}\left[\log \frac{P_\theta^h(w)}{P_\theta^h(w) + kP_n(w)}\right] + k\mathbb{E}_{P_n}\left[\log \frac{kP_n(w)}{P_\theta^h(w) + kP_n(w)}\right]$$

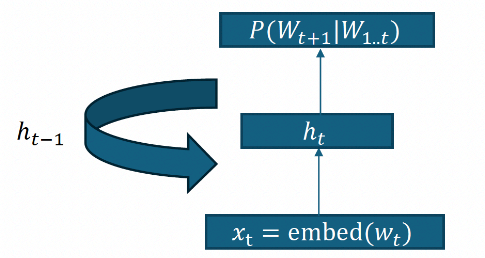
  Check out Paper: *[A Fast and Simple Algorithm for Training Neural Probabilistic Language Models]*

- Limitation FNNLM:
  - encodes a very limited context(n-gram)

7.2 Recurrent Neural Network Language Model

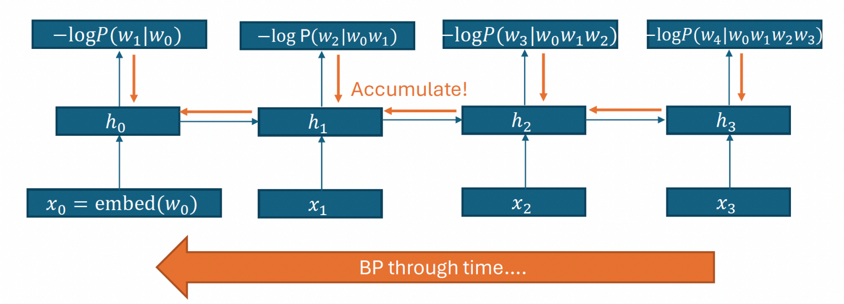
7.2.1 Architecture

- Encode **whole history**
- maintain h_t which is updated **each time step**.



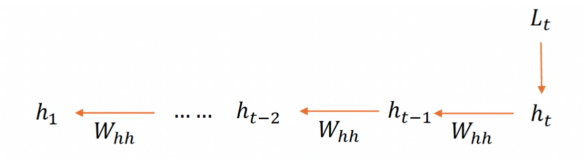
- $h_t = \sigma(W_{ih}x_t + W_{hh}h_{t-1} + b)$
- $y_t = \text{softmax}(W_{ho}h_t + b_o)$
- $L(w) = \sum_i -\log \Pr(w_i | w_{0,\ldots,i-1})$
- W_{ih}, W_{hh} are shared across timesteps(hence Recurrent).

7.2.2 Back-Propogation Through Time(BPTT)



- Problem: $\frac{\partial L}{\partial h_t}$ **Gradient Exploding/ Gradient Vanishing**
- Intuition:

  Rough estimation: 1) ignore activation function; 2) only consider L_t



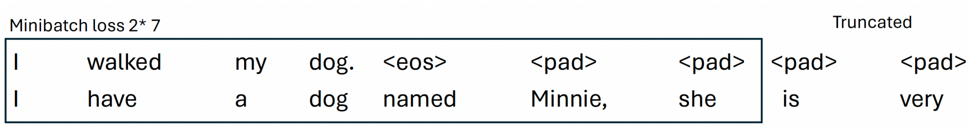
$$\frac{\partial L_t}{\partial h_1} \approx \frac{\partial L_t}{\partial h_t} W_{hh}^{t-1}$$

  - $\|W_{hh}\| < 1$: Gradient Vanishing
  - $\|W_{hh}\| > 1$: Gradient Exploding
- **Gradient Clipping**(for gradient explosion): γ is hyperparameter.

$$\text{clip}(\nabla L) = \min\left\{1, \frac{\gamma}{\|\nabla L\|_2}\right\} \nabla L$$

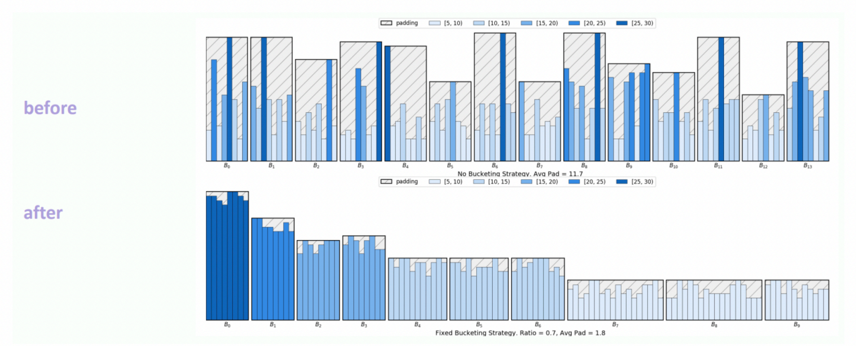
7.2.3 Parellel Computation

- Parellel across sentences
- Dealing with Variable Sequence Length:
  - padding, truncating, masking



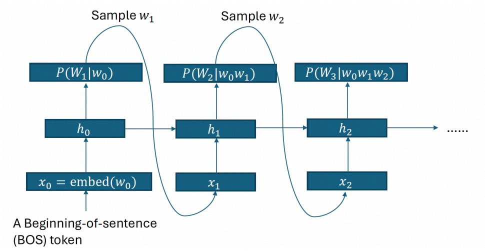
  - Bucketing:

  Sort sentences such that similarly lengthed sentences are in the same batch.



7.2.4 Sampling with RNNLM

- autoregressive



- RNN for text classification

Consider last hidden state h_T as encoding of the whole sentence. Add a linear classifier head.

7.2.5 LSTM(skipped) & GRU

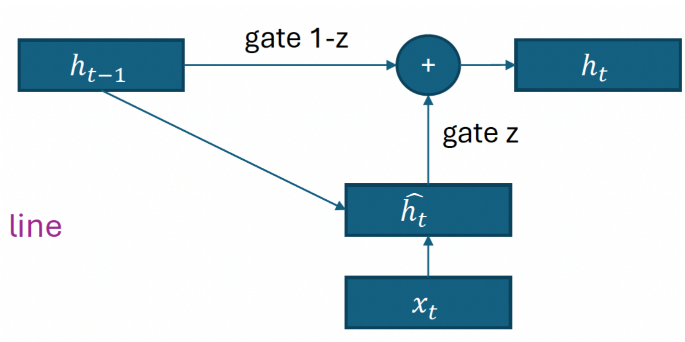
- Used for gradient vanishing problem
- LSTM Related Blog: *[Understanding LSTM Networks]*
- **GRU(Gated Recurrent Unit)**

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$ z: Update gate, sigmoid

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$ r: Reset gate, sigmoid

$$\hat{h}_t = \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$ tanh

$$h_t = z_t \odot \hat{h}_t + (1 - z_t) \odot h_{t-1}$$ <-Let's just focus on

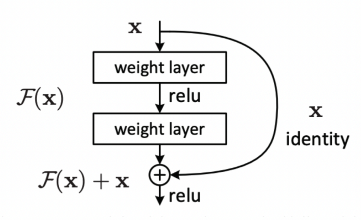


Notice that the $(1-z) \odot h_{t-1}$ contains no weight matrix, so if z is not near 1, the gradient flows through.

7.3 Tricks in Deep Learning

7.3.1 Residual Network

$h_{l+1} = h_l + F(h_l)$

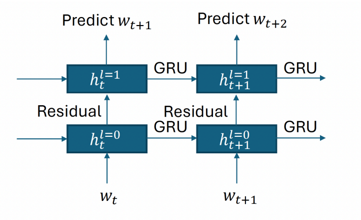


8 More On RNN

8.1 More on AR-LM

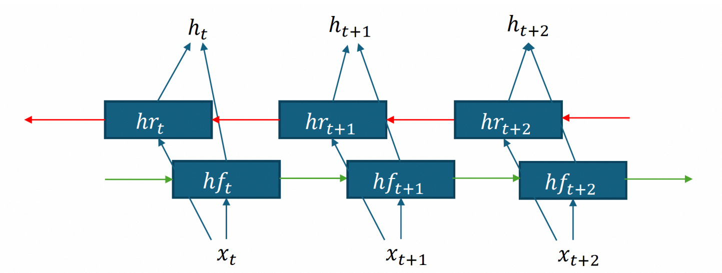
- autoregressive. We can combine different modules together to form a large neural model.

e.g.

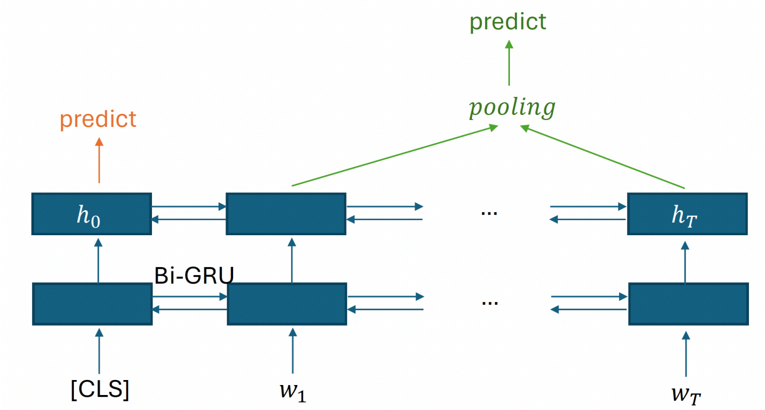


8.2 Bi-directional RNN

bi-directional can be useful for some applications(e.g. part-of-speech tagging)



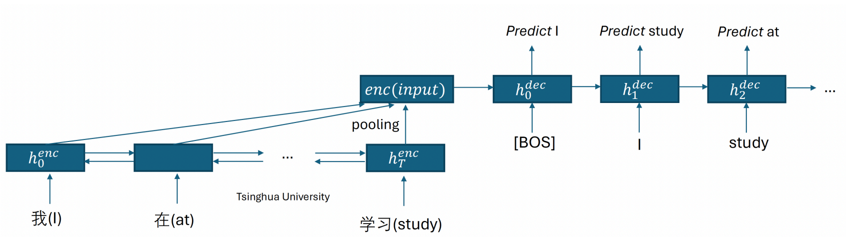
- Q: Bi-directional RNN for AR-LM?
  - Not possible, has future information.
- Q: Bi-directional RNN for sentence-encoding?
  - Way1: add a special input to the input.
  - Way2: do a max-pooling or mean-pooling of the hidden states.



8.3 Encoder-Decoder model for seq2seq task

e.g. Machine Translation

- Encoder: bi-RNN
- Decoder: uni-RNN

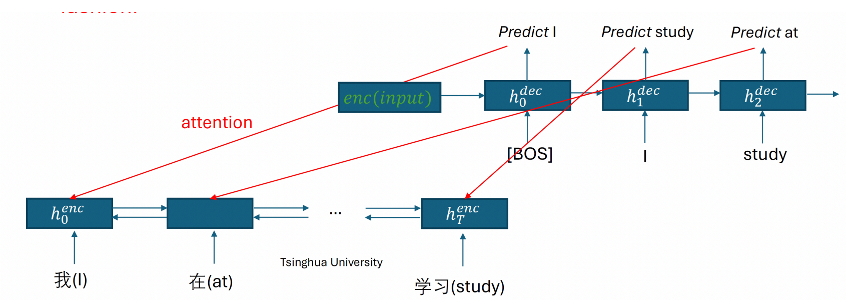


Average the encoder's hidden vectors for the input of the decoder RNN.

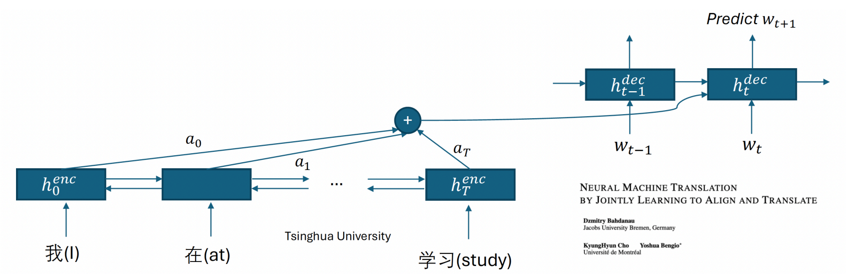
8.3.1 Attention!!!

Idea: A single vector is not enough, want to pay attention to different parts of the input in different timesteps.

High Level Idea:



Implementation:(Cross-Attention)



- At timestep t:
- Calculate alignment score: $\hat{a}_i = (h_i^{\text{enc}})^\top W_a h_{t-1}^{\text{dec}}$
- Get attention distribution: $a_i = \text{softmax}(\hat{a})$
- Pass $\sum_i a_i h_i^{\text{enc}}$ to the encoder
- W_a is shared across timesteps.

At training, optimize $L = \sum_i -\log P_\theta(y_i|x_i)$

8.4 Decoding from a LM

Consider MT task for AR-LM , if we want a whole sentence as output.

The objective is to find $\arg\max_y \text{Pr}_{\text{AR-LM}}(y|x)$.

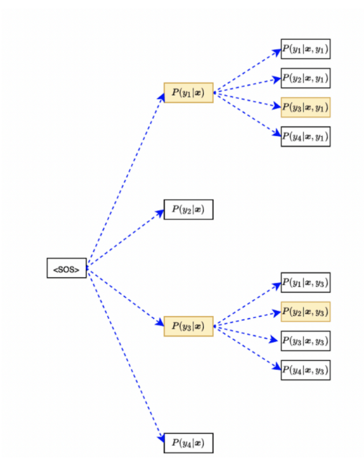
8.4.1 Greedy Decoding:

$y_t := \arg\max_{y_t} \text{Pr}_{LM}(y_t|x, y_{1,\ldots,t-1})$ $t = 1, 2, \ldots$

- doesn't guarantee sequence-level argmax
- Obs. : DP(like Vertabi for HMM) doesn't work here.(No optimality of sub-problems)

8.4.2 Beam-Search

- Maintain a number of beams(i.e. sequence of tokens).



- On each time-step:

  We expand the current beams, sort them, and only keep the beams with largest log-probability.

8.4.3 BLEU metric for MT

$$\text{precision}_n = \frac{\text{number of } n\text{-gram matches in reference}}{\text{number of } n\text{-grams in predicted}}$$

$$\text{brevity-penalty} = \min\left\{1, \exp\left(1 - \frac{|\text{reference}|}{|\text{predicted}|}\right)\right\}$$

$$\text{BLEU} = \text{brevity-penalty} \times \left(\prod_{n=1}^{4} \text{precision}_n\right)^{\frac{1}{4}}$$

Attention works!

Model	BLEU	
RNNencdec-50	17.82	No attention
RNNsearch-50	26.75	Attention
RNNsearch-50*	28.45	

8.5 Back Translation for MT Data Augmentation

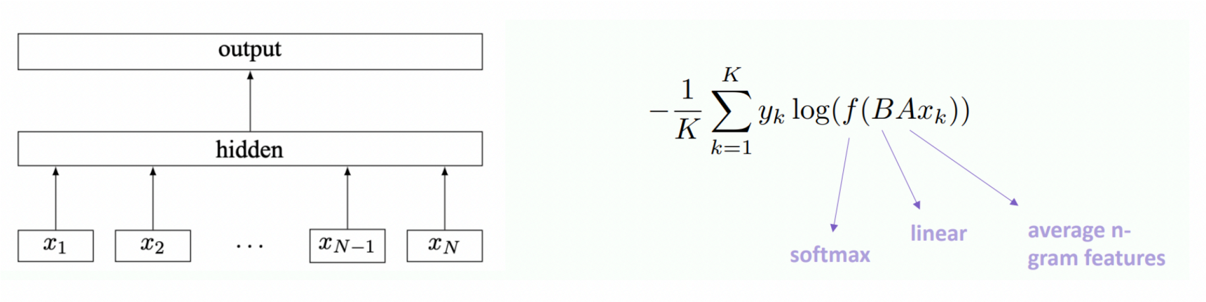
- Q: Given a decent amount of bilingual data (X, Y) and a great amount of monolingual data in target language Y. How can we create more paired data?
- A: Train a backward model: $Y \to X$, and conduct generation on the monoloingual data.

9 Text Classification

Have covered basic DNN/RNN for text classification.

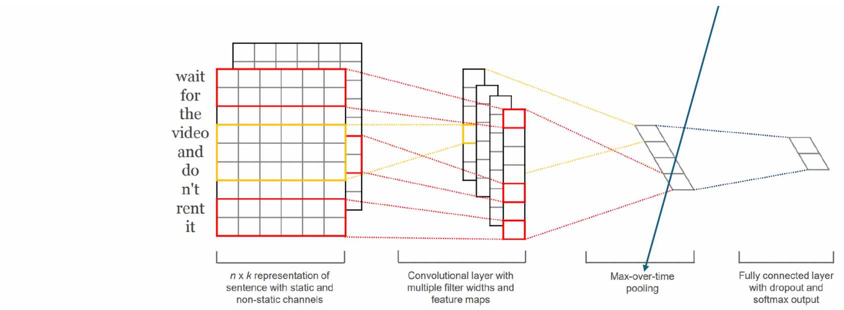
9.1 FastText + FNN

- Average the (pretrained) embeddings of n-gram features to form the hidden variable.
- Linear layer followed by softmax for classification.
- Very fast (small model). Can run on CPUs. Reasonable performance.



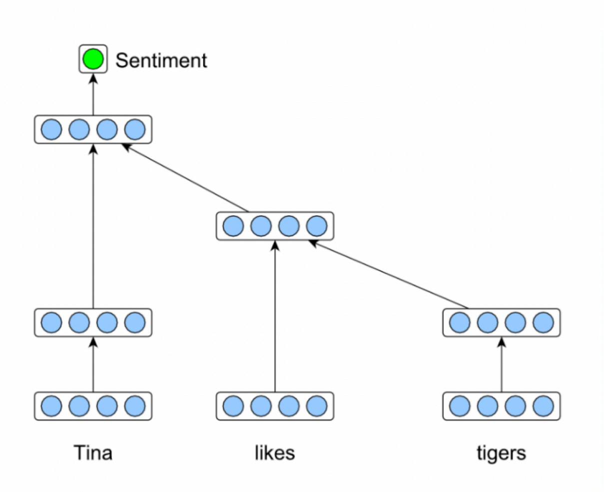
9.2 CNN for text classification

- RNN deals with variable lengths
- CNN can also do that!



$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \dots \oplus \mathbf{x}_n \qquad c_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b).$

9.3 Recursive neural networks with tree structure



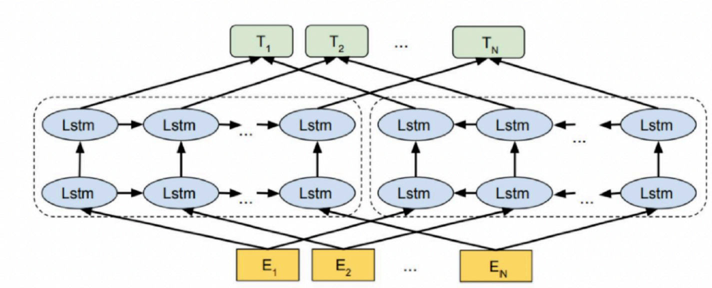
Build on parse trees. Also check Tree LSTM.

9.4 GLUE tasks

- Many natural language understanding (NLU) tasks can be posed as a text classification task. The General Language Understanding Evaluation (GLUE) benchmark.
- a harder set of tasks, which brings SuperGLUE

9.5 Deep contextualized word representation (ELMo)

- Embeddings from Language Models
- Model: multi-layer bidirectional LSTM
- Objective: predict the next word in both directions independently; i.e., left-to-right and right-to-left
- Data: 1B word LM data
- Downstream: extract output-layer features and add them to existing models (as the input word embeddings)

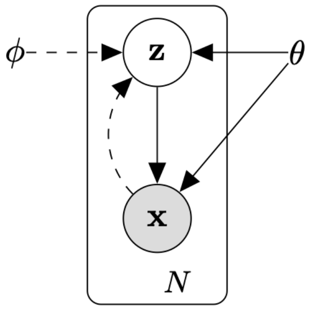


- Strong Performance on GLUE tasks. Considered pioneers of *self-supervised generative pretraining* (e.g., BERT).

10 VAE-LM

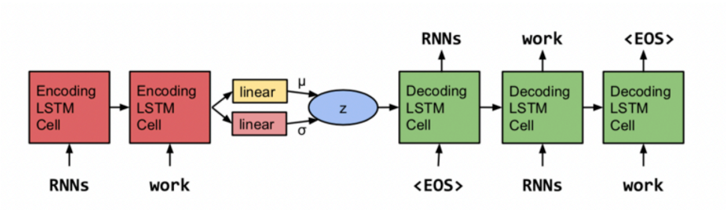
10.1 Motivation

In RNNLM, we generate token by token. VAE tries to represent whole sentence using z. We learn a encoder: $q_\phi(z|x)$ and a decoder(generative model): $p_\theta(x|z)$.



10.2 Generation:

- Sample z from prior $p(z)$
- Sample x from generative model $p_\theta(x|z)$



10.3 Training:

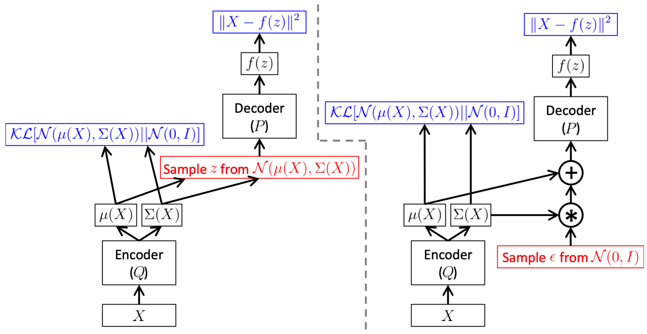
The "ELBO" objective

$$\mathcal{L}(\theta; x) = -\mathrm{KL}(q_\theta(z|x) \| p(z)) + \mathbb{E}_{q_\theta(z|x)}[\log p_\theta(x|z)] \leq \log p(x)$$

Derivation:

$$\begin{aligned}
&\ln p(x) \\
&= \ln \int_z p(x, z) \\
&= \ln \int_z p(x, z) \frac{q(z|x)}{q(z|x)} \\
&\geq \mathbb{E}_{q(z|x)}[\ln \frac{p(x, z)}{q(z|x)}] \\
&= \mathbb{E}_{q(z|x)}[\ln \frac{p(x|z)p(z)}{q(z|x)}] \\
&= \mathbb{E}_{q(z|x)}[\ln p(x|z)] + \mathbb{E}_{q(z|x)}[\ln \frac{p(z)}{q(z|x)}] \\
&= \mathbb{E}_{q(z|x)}[\ln p(x|z)] + \int_z q(z|x) \ln \frac{p(z)}{q(z|x)} \\
&= \mathbb{E}_{q(z|x)}[\ln p(x|z)] - D_{KL}[q(z|x)||p(z)] \\
&= likelihood - KL
\end{aligned}$$

- Reparameterization trick:



$z = \mu + \Sigma \odot \mathcal{N}(0, I)$, we can do back-prop now. Continous Latent Space.

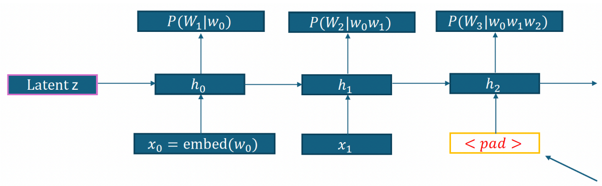
10.4 Optimization Challenge:

In vanilla VAE-LM training, KL term quickly decrease to zero(throwing away latent information).

- KL cost annealing

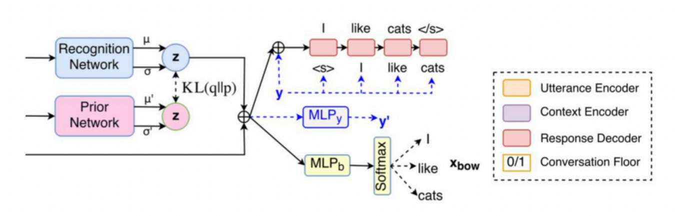
  KL term's weight gradually increase with training process.

- Input word dropping



- Bag of words Loss

  In parallel, train the decoder network to predict the bag-of-words in the response x as shown in.



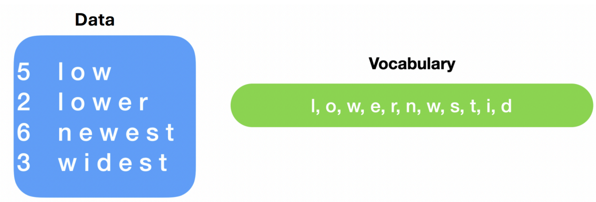
11 Subword Tokenization

11.1 Byte Pair Encoding(BPE) Tokenization

Used in GPT, Llama.

- (1) Start with a unigram vocabulary of all characters in the data.
- (2) In the data, find the most frequent pair, merge it, and add to the vocabulary.
- (3) Stop when vocabulary is of pre-determined size (e.g., 50k).

Example:



es, est, lo, low...

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, est, lo, low

lowest→low,est lost→ lo,s,t

11.2 Other Approaches

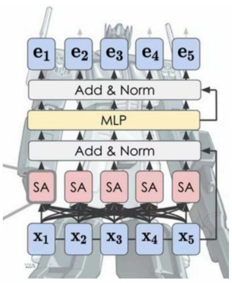
- Word Piece(BERT), sentence piece...

12 Transformers, BERT, GPT

12.1 Transformer

This part omittes lots of details as the author believes he know transformers well enough.

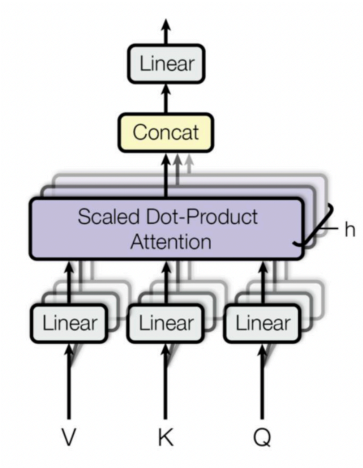
A transformer Block:



12.1.1 Self-Attention

$a_{i*} = \text{softmax}\left(\frac{q_i^\top k_*}{\sqrt{\dim(k_*)}}\right), z_i = \sum_j a_{ij} v_j$ Can be computed in Parellel

12.1.2 Multi-Head Attention



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h) W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

12.1.3 Other Designs in the attention block

- FeedForward NN
- Residual:

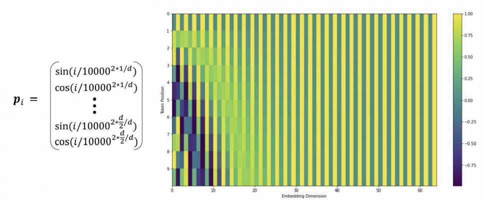
$$f_{\text{residual}}(x, F) = F(x) + x$$

  for self-attention and FFNN.
- LayerNorm:

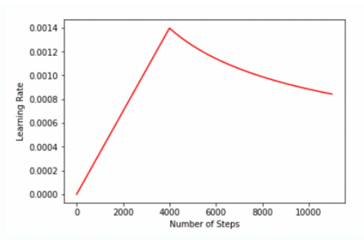
$$\text{LayerNorm}(h) = \alpha \cdot \frac{h - \text{mean}(h)}{\text{std}(h)} + \beta \quad (\alpha, \beta \text{ are learned parameters})$$

  - BatchNorm: across samples, same feature;
  - LayerNorm: across features, same sample

12.1.4 Position Encoding



12.1.5 Learning Rate Warmup and Linear Decay



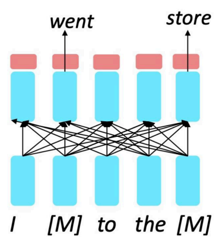
12.2 BERT

Bidirectional Encoder Representations from Transformers

Major Objectives in BERT:
- Masked language modeling (MLM)
- Next sentence prediction (NSP)

12.2.1 Masked language modeling (MLM)

- randomly mask (via a [mask] token) 15% of the tokens in each sequence.
- ask the transformer model to predict the masked token on the top layer via standard cross-entropy loss.

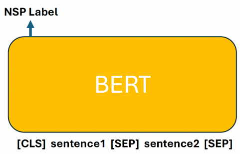


- Problem: not ideal representation for non-masked words
- Heuristic:
  - For 10% of the time, we replace [M] with a random token.

- ‣ For another 10% of the time, we do not change the original token.
- ‣ O.w., the mask token is used.

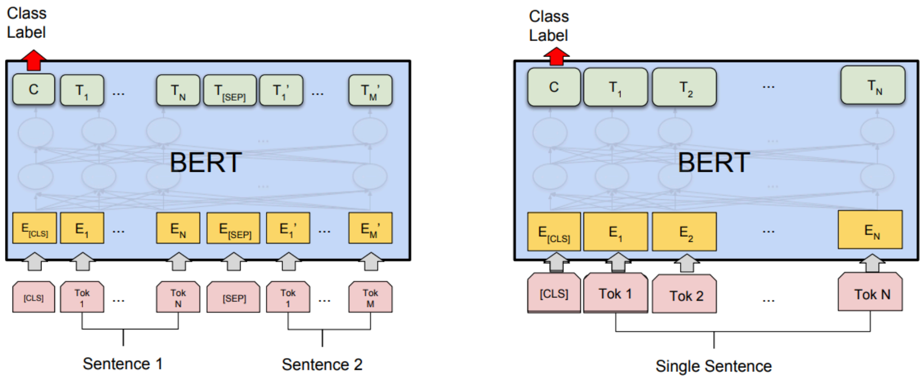
12.2.2 Next sentence prediction (NSP)

- add a [CLS] token and ask BERT to predict whether sentence2 is the next sentence of sentence1.
- For 50% of the time, a random sentence is used as a negative example.
- Actually not that useful(not used after bert)



12.2.3 BERT finetuning

- slightly modify the top layers of BERT and tune it ondownstream tasks.



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

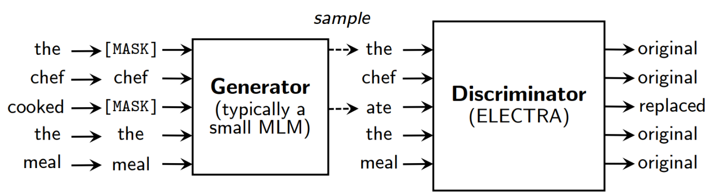
(b) Single Sentence Classification Tasks: SST-2, CoLA

12.2.4 Extensions of BERT

- ALBERT (2019, A Lite BERT ...)
- RoBERTa (2019, A Robustly Optimized BERT ...)
- DistilBERT (2019, smaller, faster, lighter version of BERT)
- ELECTRA (2020, Pre-training Text Encoders as Discriminators not Generators)
- LongFormer (2020, Long-Document Transformer)

I. ELECTRA(Efficiently Learning an Encoder that Classifies Token Replacements Accurately):

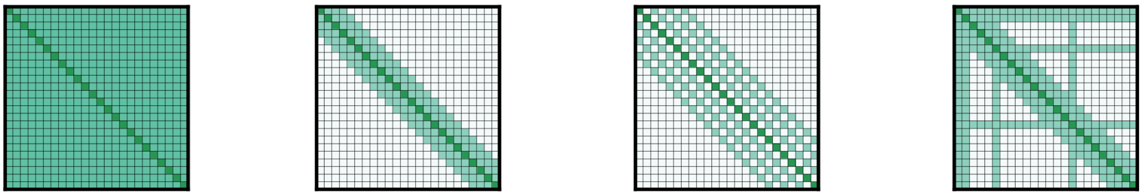
- Instead of masking the input, our approach corrupts it by replacing some tokens with plausible alternatives sampled from a small generator network.
- Then, instead of training a model that predicts the original identities of the corrupted tokens, we train a discriminative model that predicts whether each token in the corrupted input was replaced by a generator sample or not.



- Much higher data efficiency(task is defined over all sequence instead of just masked-out ones)

II. LongFormer

- $\mathcal{O}(N^2)$ attention computation cost is expensive for long sequence
- Limit the attention to a small span of tokens to save computation
- **Sliding Window Attention:**
  - limit the attention to a sliding window of size w.
  - Computation cost: $\mathcal{O}(N \cdot w)$
- Q: For an embedding on layer L, what's its receptive field?
- A: $L \cdot w$



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

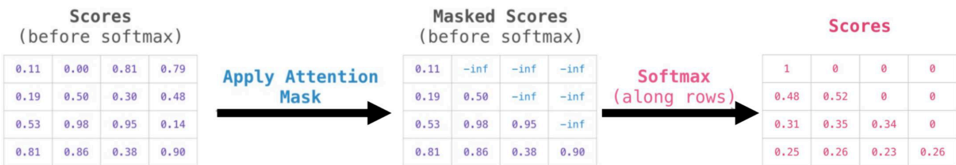
- **Dilated Sliding Window:**
  - Larger receptive field, but miss some local information
  - Fix: Multi-Head Attention.
  - We can use a combination of 2 heads of dilated and other heads with local sliding window.

12.3 Transformer Decoder & GPTs

- BERT-like models are great for sentence/document encoding or deep contextualized word embedding.
- But you can not directly use it for text generation, or infer the log-probability of a given text.
- So let's talk about transformers for autoregressive language modelling (generation).

12.3.1 Causal Mask

- apply the mask before the softmax operation, so that the attention distribution is still normalized.



12.3.2 GPTs

- GPT models are transformer decoders trained for AR-LM.
- generative capability emerged from large-scale training

	GPT-1	GPT-2	GPT-3
Model	Transformer Decoders (12 Decoder blocks;12 Masked-Attention heads)	Transformer Decoders (48 Decoder blocks)	Transformer Decoders (48 Decoder blocks)
Objective	Next word prediction (cross-entropy loss)	Next word prediction (cross-entropy loss)	Next word prediction (cross-entropy loss)
Data	**BooksCorpus** (11k books from a variety of genres)	• **BooksCorpus** • **WebText** (8M web pages)	• CommonCrawl (410B tokens) • WebText2 (19B tokens) • Books1 (12B tokens) • Books2 (55B tokens) • Wiki (3B tokens)
# Parameters	117M	1.5B	175B
Paper Title	Improving Language Understanding by Generative Pre-Training	Language Models are unsupervised multitask learners	Language Models are Few-Shot Learners
Year	2018	2019	2020

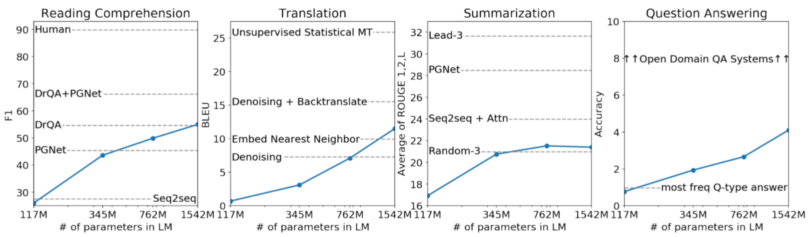
I. GPT-1(before BERT, still focused on NLU)

- pretrain a transformer decoder AR-LM on large data, and the finetune it on downstream NLU tasks.
- take the final-layer embedding of the last token in the text, and add a linear classification head.

II. GPT-2

- **Zero-shot capability to downstream tasks**:
  - No finetune
  - In generation, it continues the language.

    Prompts: *Translate the following text to French. Text: [ENG TEXT] French:*
  - WebText data contains all sorts of data $\rightarrow$ we are implicitly doing multi-task training during the pretraining.
- **Scaling up can help zero-shot ability**



- **Common knowledge**

Question	Generated Answer	Correct	Probability
Who wrote the book the origin of species?	Charles Darwin	✓	83.4%
Who is the founder of the ubuntu project?	Mark Shuttleworth	✓	82.0%
Who is the quarterback for the green bay packers?	Aaron Rodgers	✓	81.1%
Panda is a national animal of which country?	China	✓	76.8%
Who came up with the theory of relativity?	Albert Einstein	✓	76.4%
When was the first star wars film released?	1977	✓	71.4%
What is the most common blood type in sweden?	A	✗	70.6%
Who is regarded as the founder of psychoanalysis?	Sigmund Freud	✓	69.3%
Who took the first steps on the moon in 1969?	Neil Armstrong	✓	66.8%
Who is the largest supermarket chain in the uk?	Tesco	✓	65.3%
What is the meaning of shalom in english?	peace	✓	64.0%
Who was the author of the art of war?	Sun Tzu	✓	59.6%
Largest state in the us by land mass?	California	✗	59.2%
Green algae is an example of which type of reproduction?	parthenogenesis	✗	56.5%
Vikram samvat calender is official in which country?	India	✓	55.6%
Who is mostly responsible for writing the declaration of independence?	Thomas Jefferson	✓	53.3%

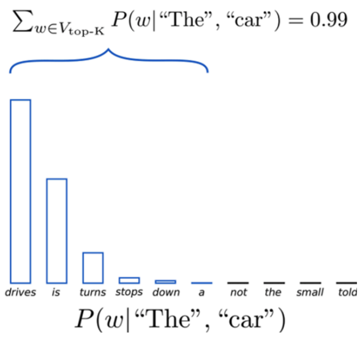
- **Open-ended generation**

- Open-ended generation refers to generation tasks that has big freedom and diversity, like story or news generation.
- very different from translation or summarization, where the generation is like "another version" of the input.
- **The model needs to rely its own (memory, consistency or creativity).**

12.3.3 The top-K sampling algorithm

- Direct sampling from $\Pr_{\text{AR-LM}}\left(\cdot\,|w_{[1:i]}\right)$ can be diverse but **has poor quality or consistency**.
- Top-K sampling: trade diversity for quality
  - Represent $P\left(\cdot \mid w_{[1:i]}\right)$ by $p = \left(p_1, ..., p_{|V|}\right)$, where $p_1 \geq ... \geq p_{|V|}$.
  - Sample W_{i+1} from $\hat{p}$:

$$\hat{p}_i = \frac{p_i \cdot \mathbb{1}[i \leq k]}{Z}$$



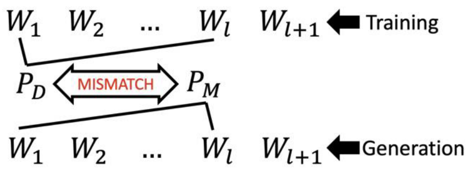
- Sampling algorithms provide a sweet quality-diversity trade-off.(Essential difference from decoding e.g. beam search)

13 Rethink MLE, Sampling, and Bad Behavior

13.1 Criticizing teacher forcing (MLE)

13.1.1 Teacher Forcing(MLE)

- The MLE objective: $\log P(W) = \sum_i \log P\left(w_i|w_{[1:i-1]}\right)$ where W is from training data.
- However in generation, $W_i^M \sim P\left(W_i|W_{1:i-1}^M\right)$, there may be a distribution shift.



13.1.2 The exposure bias hypothesis:

Due to the exposure to ground-truth prefix, the model is biased to only perform well during training, but not generation.

Importantly, the error is hypothesized to **accumulate** during generation, and the generation will be **incrementally distorted**.

13.1.3 Language GANs

- This belief in exposure bias motivates Language GANs.
- In GAN, no teacher forcing, training is directly applied to model samples.
- GAN example in CV:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

- However in NLP, not differentiable. the gradient can not flow back through discrete sampling.

Solutions:
- **The Gumbel-softmax reparameterization**
- **The reinforce trick (policy gradient)**

13.1.4 The Gumbel-softmax reparameterization

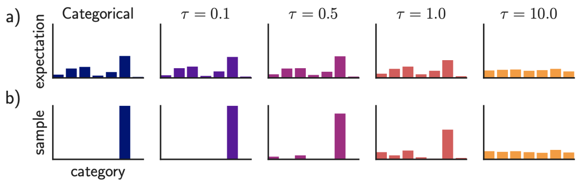
The Gumbel-Max trick (Gumbel, 1954; Maddison et al., 2014) provides a simple and efficient way to draw samples z from a categorical distribution with class probabilities π:

$$z = \texttt{one\_hot}\left(\arg\max_i \left[g_i + \log \pi_i\right]\right) \tag{1}$$

where $g_1 \ldots g_k$ are i.i.d samples drawn from Gumbel$(0,1)$[1]. We use the softmax function as a continuous, differentiable approximation to arg max, and generate k-dimensional sample vectors $y \in \Delta^{k-1}$ where

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)} \qquad \text{for } i = 1, ..., k. \tag{2}$$

One practice is to anneal τ from large to small during training.



- **Straight-Through Trick**:

  Sometimes we want our encoding to really be one-hot during training.

  Do argmax to get the one-hot vector, `y_hard`

  `ret = y_hard - y_soft.detach() + y_soft`, returns the ont-hot vector, but the gradient only flows through the soft part in back-prop.

13.1.5 The Reinforce Trick

$$\arg\max_\theta \ \mathbb{E}_{\boldsymbol{y}\sim P_\theta(\boldsymbol{y}\,|\,\boldsymbol{x})}[r(\boldsymbol{x},\boldsymbol{y})]$$

$$\begin{aligned}
&\nabla_\theta \mathbb{E}_{\boldsymbol{y}\sim P_\theta(\boldsymbol{y}\,|\,\boldsymbol{x})}[r(\boldsymbol{x},\boldsymbol{y})] \\
&\quad = \nabla_\theta \sum_{\boldsymbol{y}} r(\boldsymbol{x},\boldsymbol{y}) P_\theta(\boldsymbol{y}\,|\,\boldsymbol{x}) \\
&\quad = \sum_{\boldsymbol{y}} r(\boldsymbol{x},\boldsymbol{y}) \nabla_\theta P_\theta(\boldsymbol{y}\,|\,\boldsymbol{x}) \\
&\quad = \sum_{\boldsymbol{y}} r(\boldsymbol{x},\boldsymbol{y}) P_\theta(\boldsymbol{y}\,|\,\boldsymbol{x}) \nabla_\theta \log P_\theta(\boldsymbol{y}\,|\,\boldsymbol{x}) \\
&\quad = \mathbb{E}_{\boldsymbol{y}\sim P_\theta(\boldsymbol{y}\,|\,\boldsymbol{x})}\left[r(\boldsymbol{x},\boldsymbol{y}) \nabla_\theta \log P_\theta(\boldsymbol{y}\,|\,\boldsymbol{x})\right]
\end{aligned}$$

$$\nabla_\theta \log P_\theta(\cdot) = \frac{\nabla_\theta P_\theta(\cdot)}{P_\theta(\cdot)}$$

Examples of Language GANs:
- SeqGAN: *SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient*

  Reported better generation quality than MLE baseline.

However, for MLE baselines, we can tune the temperature to tradeoff quality and diversity.

$$\hat{p}_i = \frac{\exp(\log(p_i)/T)}{\sum_{j=1}^{|V|} \exp(\log(p_j)/T)}.$$

Check out [Language GANs Falling Short]

Results: Language GANs are actually worse than the MLE baseline.*(NLL test represents diversity, NLL oracle represents quality.)*

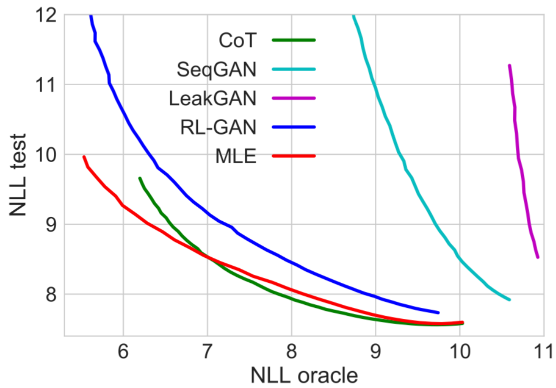


Figure 3: Effect of temperature tuning on the global metrics *(lower is better for both metrics)* for the synthetic task.

TakeAway:

- Language GAN is a great idea,but GAN training is notoriously unstable.
- MLE training + sampling algorithm is an amazing combination.

13.2 Sampling algorithms

13.2.1 Top-k Sampling

$$\hat{p}_i = \frac{p_i \cdot \mathbb{1}[i \leq k]}{Z}$$

13.2.2 Nucleus(top-P):

$$\hat{p}_i = \frac{p_i \cdot \mathbb{1}\left[\sum_{j=1}^{i-1} p_j < P\right]}{Z}$$

13.2.3 Tempered (T)

$$\hat{p}_i = \frac{\exp\left(\frac{\log(p_i)}{T}\right)}{Z}$$

Check out: [A Systematic Characterization of Sampling Algorithms for Open-ended Language Generation]

13.2.4 Common Features of these three sampling methods

- The **order** of elements are **preserved**:

$$p_i \geq p_j \to \hat{p}_i \geq \hat{p}_j$$

- The **entropy** of the distribution are **reduced**

$$\mathcal{H}(\hat{p}) \leq \mathcal{H}(p)$$

- The **slope** of the non-zero elements are **preserved**:

$$\frac{\log p_i - \log p_j}{\log p_j - \log p_k} = \frac{\log \hat{p}_i - \log \hat{p}_j}{\log \hat{p}_j - \log \hat{p}_k} \quad \text{if } \hat{p}_i, \hat{p}_j, \hat{p}_k > 0$$

Hypothesis:[See Paper for details]
- Sampling algorithms that satisfy all three properties should be at least as good as the top-k/nucleus/tempered sampling in the Q-D trade-off.
- Sampling algorithms that violate at least one of the properties won't be as good.

Take-away: What matters is not the details of how the algorithm is designed, but the high-level principles (properties) on which it is based on.

13.3 Correcting bad behavior of NLG models

13.3.1 Biased decoding

- Motivation: Discourage **repeating** token

$$p_i = \frac{\exp(x_i/(T \cdot I(i \in g))}{\sum_j \exp(x_j/(T \cdot I(j \in g))} \qquad I(c) = \theta \text{ if c is True else } 1$$

- T is temperation, g refers to the set of generated tokens. In practice, set $\theta = 1.2$

13.3.2 Unlikelihood training for repetition

- Explicitly discourage repeating tokens during training

$$\mathcal{L}^t_{\text{UL-token}}(p_\theta(\cdot|x_{<t}), \mathcal{C}^t) = -\alpha \cdot \underbrace{\sum_{c \in \mathcal{C}^t} \log(1 - p_\theta(c|x_{<t}))}_{\text{unlikelihood}} - \underbrace{\log p_\theta(x_t|x_{<t})}_{\text{likelihood}}.$$

where $\mathcal{C}^t_{\text{prev-context}} = \{x_1, ..., x_{t-1}\} \setminus \{x_t\}$.

13.3.3 The MMI criterion

- Motivation: To discourage **generic responses** in chatbot(e.g. *"I don't know","I'm ok"*)
- Usual Objective:

$$\hat{T} = \arg\max_T \{\log p(T|S)\}$$

- Maximum (pointwise) Mutual Information (MMI) Objective:

  We compares the probability of two events occurring together to what this probability would be if the events were independent:

$$\max_T \log \frac{p(S,T)}{p(S)p(T)}$$

  which can be formulated as

$$\hat{T} = \arg\max_T \{\log p(T|S) - \log p(T)\}.$$

13.3.4 Training with negative examples

- Motivation: generic response of chatbots
- Dynamically count the frequency of decoded response from the model during training, and assign negated gradients to those most frequent samples (denoted as y_{neg}).

$$Loss_{new} = -\log P_\theta(y_{pos}|x_{pos}) + \log P_\theta(y_{neg}|x_{neg})$$

13.3.5 Hallucination

- “Hallucinations” refers to seemingly convincing yet factually incorrect text.
- Cover in future lectures

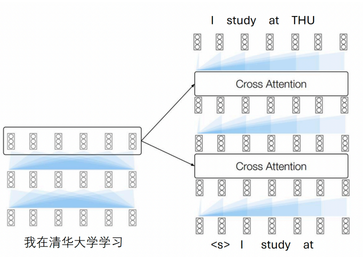
14 Transformer encoder-decoder & RoPE

- Encoder: BERT
- Decoder: GPT
- For seq2seq tasks(e.g. Machine Translation), construct encoder-decoder transformer

14.1 Encoder-Decoder Transformer

14.1.1 Architecture

- Each decoder layer is a **self-attention** followed by a **cross-attention**.

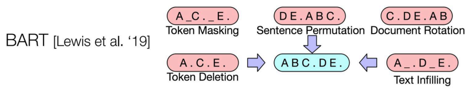


- The query vector for a transformer decoder’s cross-attention head is from the output of the previous decoder layer. However, the key and value vectors are from the encoders’ outputs.
- Pretraining Encoder-Decoder Transformer:

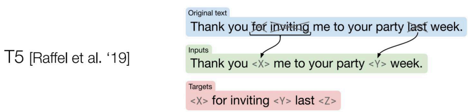
  Similar to MLM in BERT (encoder), we can design self-supervised pretraining objective as seq2seq tasks for encoder-decoder models.

14.1.2 Examples:

- **BART**: *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*



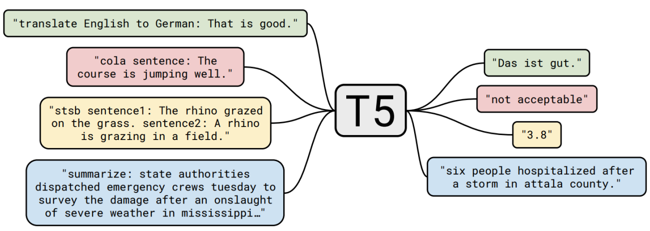
- **T5**: *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*



- **T5 paradigm: text2text**

  In T5 task-specific finetuning, all tasks (including classification or regression) are converted to a text-to-text format.

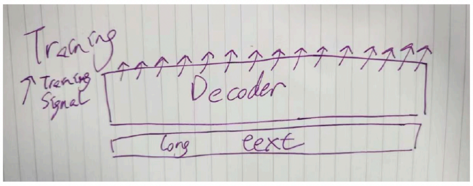
  In this way, we do not need to change model architecture for most tasks.

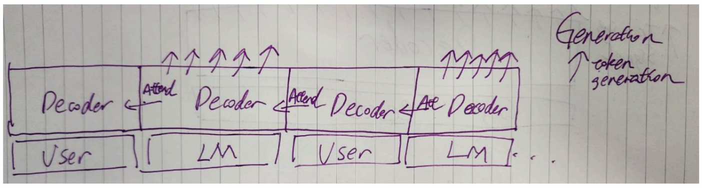


14.1.3 From Encoder-Decoder to Decoder-only

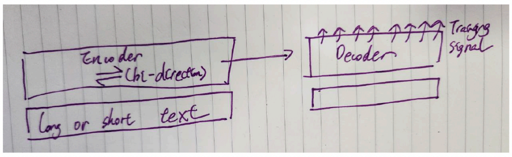
Consider multi-round chatbot dialogue scenario,

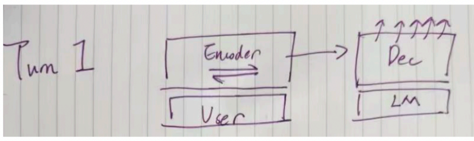
- Decoder-Only:
  - Training and Inference architecture is highly constistent
  - Naturally handles variable-length text generation
  - During application, we just do natural concatenation (always causal attention). **No computation is wasted** (assuming we save hidden states of the history).

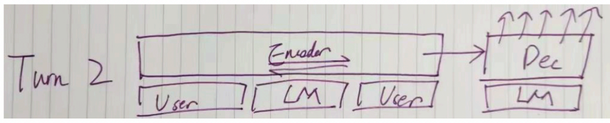




- Encoder-Decoder:
  - In pretraining, we need to build text of variable length, and the training signal is only from the decoder side.
  - During application, we need to re-encode (because the encoder is bi-directional) the whole history for each dialogue turn.







14.2 Rotary position embedding (RoPE)

- Absolute Embedding:

$$f_{t:t\in\{q,k,v\}} := W_{t:t\in\{q,k,v\}}(x_i + p_i)$$

$$p_i = \begin{cases} p_{i,2t} = \sin\left(\frac{k}{10000^{\frac{2t}{d}}}\right) \\ p_{i,2t+1} = \sin\left(\frac{k}{10000^{\frac{2t}{d}}}\right) \end{cases} \in \mathbb{R}^d$$

- RoPE
  - Motivation: want the dot product between query (position m) and key (position n) to directly be a function of $(m-n)$.

$$\langle f_q(\boldsymbol{x}_m, m), f_k(\boldsymbol{x}_n, n)\rangle = g(\boldsymbol{x}_m, \boldsymbol{x}_n, m-n).$$

  - 2D-case:

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

  - General Form:

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m \tag{14}$$

where

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix} \tag{15}$$

is the rotary matrix with pre-defined parameters $\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2]\}$. A graphic illustration of RoPE is shown in Figure (**1**). Applying our RoPE to self-attention in Equation (2), we obtain:

$$\boldsymbol{q}^\intercal_m \boldsymbol{k}_n = (\boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_q \boldsymbol{x}_m)^\intercal (\boldsymbol{R}^d_{\Theta,n} \boldsymbol{W}_k \boldsymbol{x}_n) = \boldsymbol{x}^\intercal \boldsymbol{W}_q R^d_{\Theta,n-m} \boldsymbol{W}_k \boldsymbol{x}_n \tag{16}$$

- Intuition:

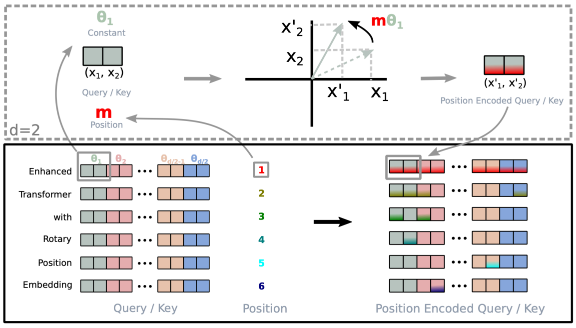


Figure 1: Implementation of Rotary Position Embedding(RoPE).

- adopted in the latest LLaMA models

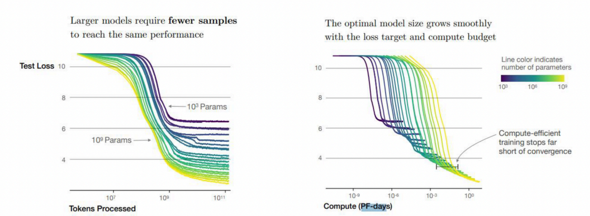
15 GPT-3 & In-Context Learning

15.1 Scaling Law of LMs

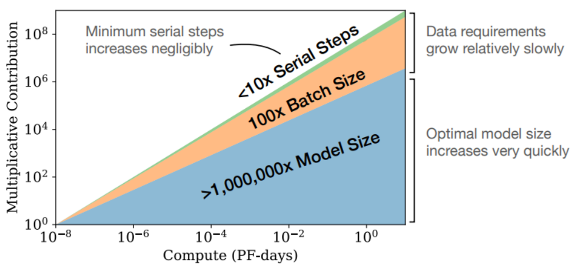
15.1.1 Data Composition of GPT-3

- Common Crawl (webpages)
- High quality data (such as Wiki) is intentionally repeated multiple times.

15.1.2 Scaling Law



- Model Size grow faster than need for Data



15.2 GPT-3 & In-Context Learning

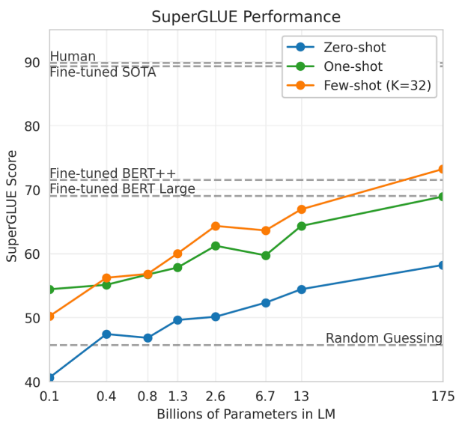
- LM perplexity improves

15.2.1 In-Context Learning

- So far, "learning" usually refers to **gradient update** with labelled data (classification or seq2seq tasks).
- Now, we only want to construct some prompt (also called context or prefix) and ask the LM to do continuation.
- Prompt Example

"Please negate the meaning of the sentence. [<- task description, optional] I hate NLP => I love NLP; Today's weather is good => Today's weather is bad; [<- the demonstrations] I had a good day => [<- the example for testing (output)]"

- Performance on SuperGLUE (32-shot ICL)



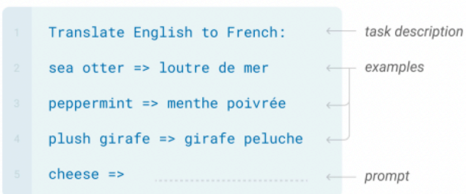
- Reasons:
  - Mostly unclear.
  - GPT-3 Paper's Explanation:

    "During unsupervised pre-training, a language model develops a broad set of skills and pattern recognition abilities. It then uses these abilities at inference time to rapidly adapt to or recognize the desired task."

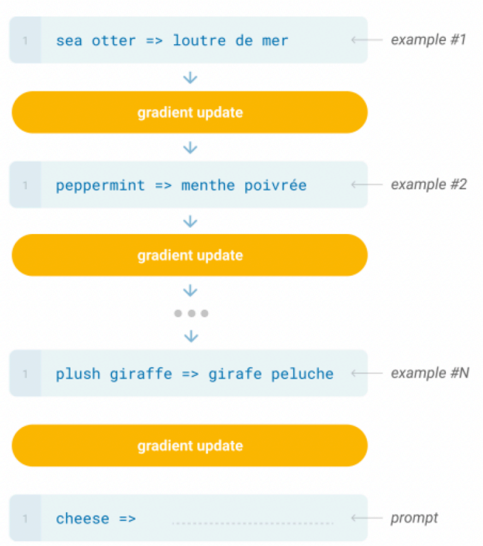
15.2.2 Terminology of GPT-3

- Zero-shot:

- Few-shot (⇔ ICL):



- Finetuning:



15.2.3 Few-shot learning before GPT3: MAML

- Before GPT3, few-shot learning still refers to how a model can quickly adapt to a new task demonstrated with only a few examples **via gradient update**.
- We have a **meta-learning phase** on a wide set of tasks. In effect, the meta-learning problem treats entire tasks as training examples.

Check out Paper: *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks* Chelsea Finn!

- Assuming each sub-task has a (small) few-shot train-set and a (smaller) dev-set.
- In the inner loop of each sub-task, we update to a pseudo θ.
- In the outer loop, we compute real gradient on the dev-set loss with θ'. So the real gradient involves second-order term (why?).

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
randomly initialize θ
while not done **do**
  Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
  for all $\mathcal{T}_i$ **do**
    Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
    Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
  end for
  Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
end while

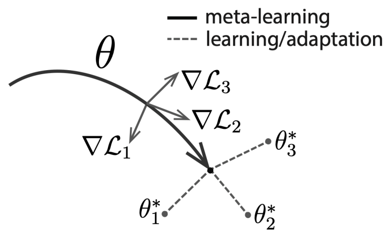


Figure 1. Diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.

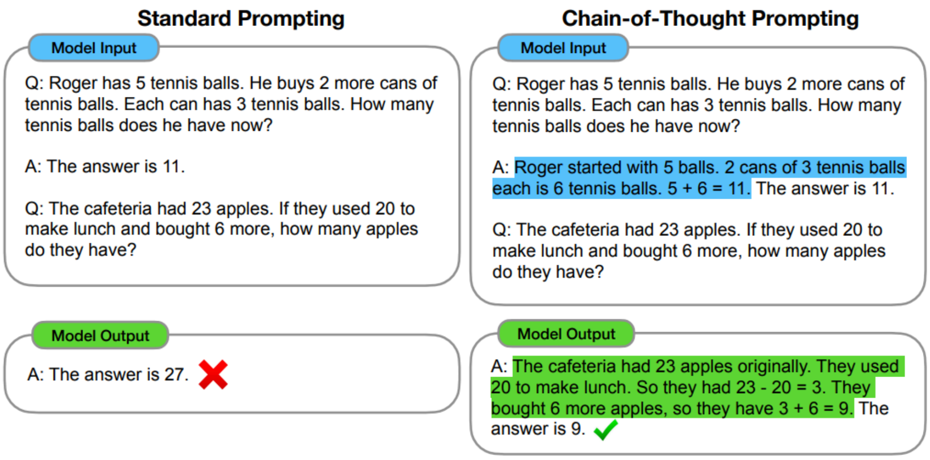
15.2.4 GPT-3 on NLP-community

- Not Open-sourced
- API calls for prompted generation. API calls for finetuning ain't that useful.
- Research focus changes to **building good prompts**

16 Chain-of-Thought(CoT) Prompting

16.1 Idea

In the few-shot demonstrations, add reasoning steps before giving the answer. These reasoning steps are **manually written** by humans.



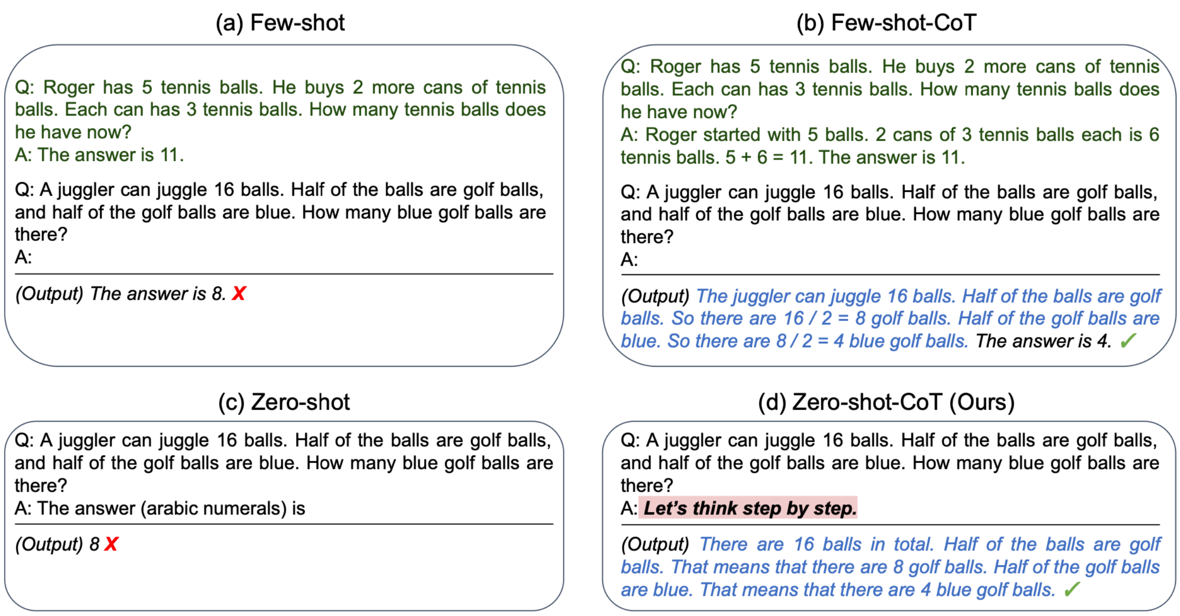
Check out Paper: *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*

16.2 Results

- CoT is an **emergent ability** of model scale. That is, its impact is more pronounced when the model large (~ 100B).

16.3 Zero-Shot CoT

Check out Paper: *Large Language Models are Zero-Shot Reasoners*



- Chaining of two prompts:

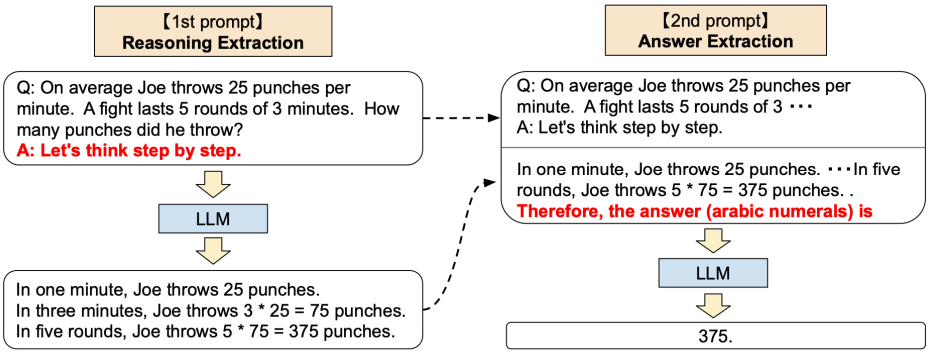


Figure 2: Full pipeline of Zero-shot-CoT as described in § 3: we first use the first "reasoning" prompt to extract a full reasoning path from a language model, and then use the second "answer" prompt to extract the answer in the correct format from the reasoning text.

This ability is also emergent with model size.

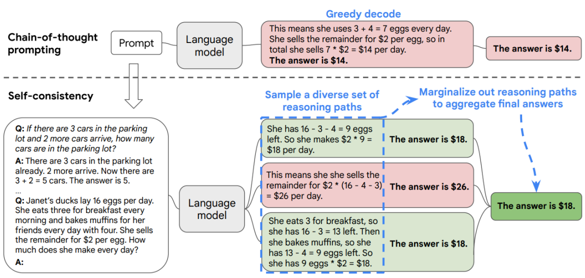
16.4 Intuitions behind why CoT works

- Divide and conquer
- Reasoning steps take more computation, giving LM more time to think.

17 More (Research) on CoT & ICL

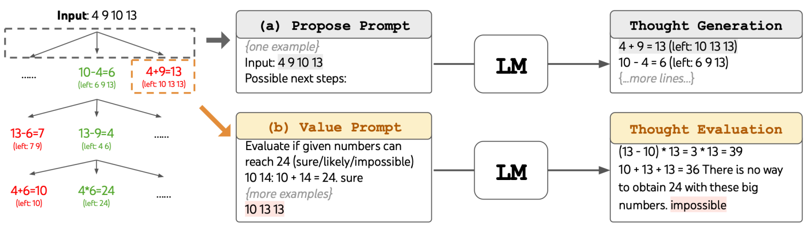
17.1 CoT with self-consistency

- For CoT, we could sample multiple reasoning path from the LLM with temperature sampling.
- And then take a majority voting over the answers!



17.2 Tree of thoughts (ToT)

- Maintain and expand a thought-tree.
- For each existing step, we prompt the LLM the propose multiple next steps, and also to judge which path (by giving a value) is more promising.
- The nodes that are judged to be unlikely will be discarded.



Check out Paper: *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*

Expand Nodes in some order:

Algorithm 1 ToT-BFS($x, p_\theta, G, k, V, T, b$)

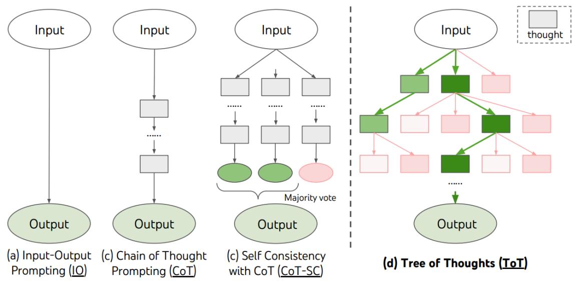
Require: Input x, LM p_θ, thought generator $G()$ & size limit k, states evaluator $V()$, step limit T, breadth limit b.

$S_0 \leftarrow \{x\}$
for $t = 1, \cdots, T$ **do**
  $S'_t \leftarrow \{[s, z] \mid s \in S_{t-1}, z_t \in \mathrm{G}(p_\theta, s, k)\}$
  $V_t \leftarrow V(p_\theta, S'_t)$
  $S_t \leftarrow \arg\max_{S \subset S'_t, |S|=b} \sum_{s \in S} V_t(s)$
end for
return $G(p_\theta, \arg\max_{s \in S_T} V_T(s), 1)$

Algorithm 2 ToT-DFS($s, t, p_\theta, G, k, V, T, v_{th}$)

Require: Current state s, step t, LM p_θ, thought generator $G()$ and size limit k, states evaluator $V()$, step limit T, threshold v_{th}

if $t > T$ **then** record output $G(p_\theta, s, 1)$
end if
for $s' \in G(p_\theta, s, k)$ **do** ▷ sorted candidates
  if $V(p_\theta, \{s'\})(s) > v_{thres}$ **then** ▷ pruning
    DFS($s', t+1$)
  end if
end for



17.3 Bias in ICL

17.3.1 Majority and Recency Bias

- The demonstration's labels and permutation changes the performance

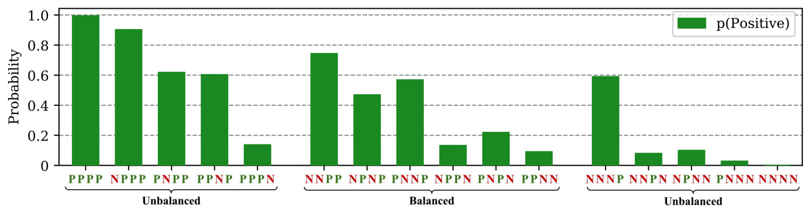


Figure 4. **Majority label and recency biases** cause GPT-3 to become biased towards certain answers and help to explain the high variance across different examples and orderings. Above, we use 4-shot SST-2 with prompts that have different class balances and permutations, e.g., [P P N N] indicates two positive training examples and then two negative. We plot how often GPT-3 2.7B predicts Positive on the balanced validation set. When the prompt is unbalanced, the predictions are unbalanced (*majority label bias*). In addition, balanced prompts that have one class repeated near the end, e.g., end with two Negative examples, will have a bias towards that class (*recency bias*).

Check out Paper: *Calibrate Before Use: Improving Few-Shot Performance of Language Models*

17.3.2 Calibration of few-shot prediction

- Want to learn a linear transformation to calibrate the predicted distribution.

$$\hat{q} = \mathrm{softmax}(W\hat{p} + b)$$

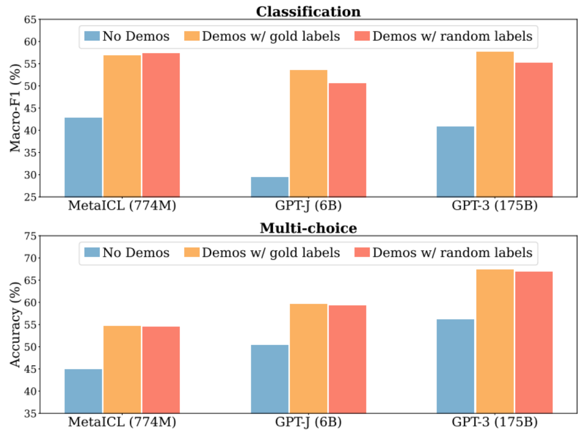
- To counter the bias, we create a "null" input, and argue that the model's prediction for null should be balanced (uniform).

  Input: Subpar acting. Sentiment: Negative
  Input: Beautiful film. Sentiment: Positive
  Input: N/A Sentiment:

  So we can set $b = 0, W = \mathrm{diag}(\mathrm{prediction}_{\mathrm{null}})^{-1}$.
- Pretty useful with a low number of demonstrations.

17.4 Rethink ICL: The role of demonstration

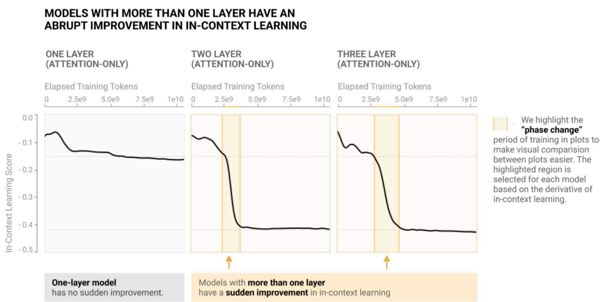
- If we replace the labels in few-shot demonstrations with **random labels**, the performance do not drop too much.
- The format and label space learned from the demostrations seems to be relatively more important.
- The result should be taken with grain of salt… If we look closer, some task got low performance w/ random label, but its impact is averaged out in the figure.



Check out: *[Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?]()*

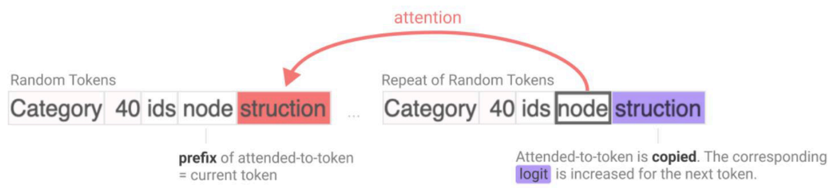
17.5 ICL and induction heads

17.5.1 Observation: Emergence of ICL



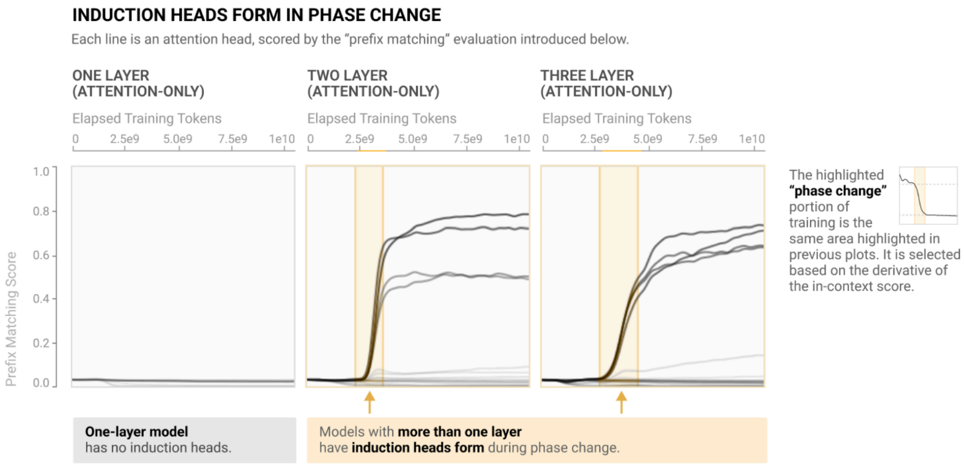
17.5.2 Induction heads (for Repetition)

- Induction heads are any heads that empirically increase the likelihood of `[B]` given `[A] [B] … [A]`.
- We can also design some metric to quantify whether an attention head is exhibiting this behavior.
- Formally, we define an induction head as one which exhibits the following two properties on a repeated random sequence of tokens:
  - **Prefix matching**: The head attends back to previous tokens that were followed by the current and/or recent tokens. That is, it attends to the token which induction would suggest comes next.
  - **Copying**: The head's output increases the logit corresponding to the attended-to token.



17.5.3 Key Finding

Induction heads form simultaneously as ICL improves dramatically!



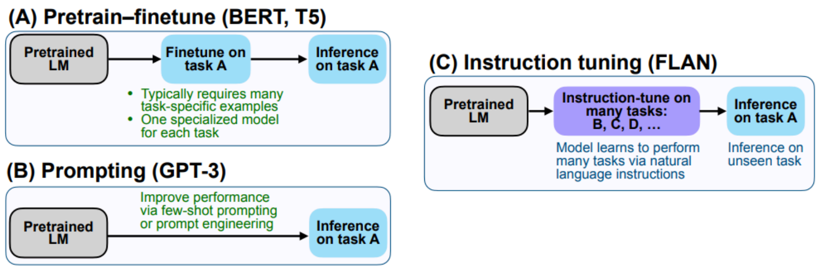
18 Instruction tuning & alignment

18.1 Instruction tuning

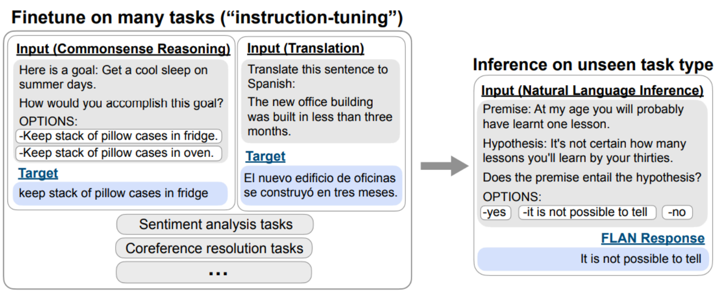
- Zero-shot prompting is nice, however pretraining on general data does not always work (which is not surprising).

18.1.1 FLAN (Finetuned Language Net)

Paper: *Finetuned Language Models Are Zero-Shot Learners*



- After pretraining, we finetune the language model on a good amount of **"instruction following"** data.
- Each training samples contains **the task description, an input, and the target output**.
- During evaluation, we hope the model can generalize to **unseen task** type.



- Data Construction:
  - Collected data from 62 existing NLP tasks (smart!).

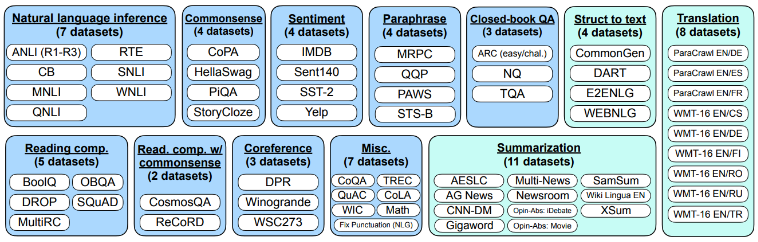
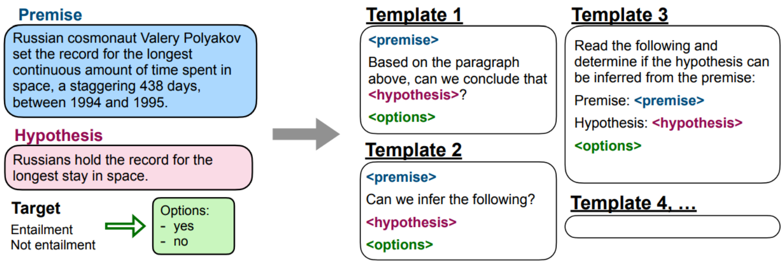


Figure 3: Datasets and task clusters used in this paper (NLU tasks in blue; NLG tasks in teal).

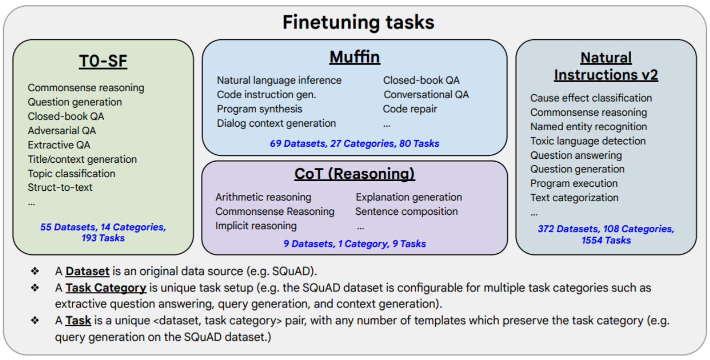
- For each task, manually compose ten unique templates (for diversity) that use natural language instructions to describe the task.



18.1.2 Scaling instruction-finetuned language models

See Paper: *Scaling Instruction-Finetuned Language Models* By Google 2022.

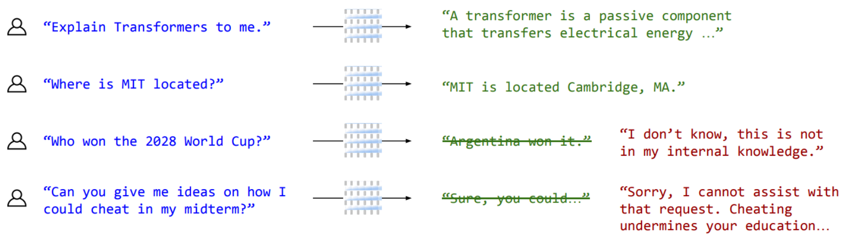
- Similar idea, but scaling to 473 datasets.
- CoT annotations is also included (in some datasets).



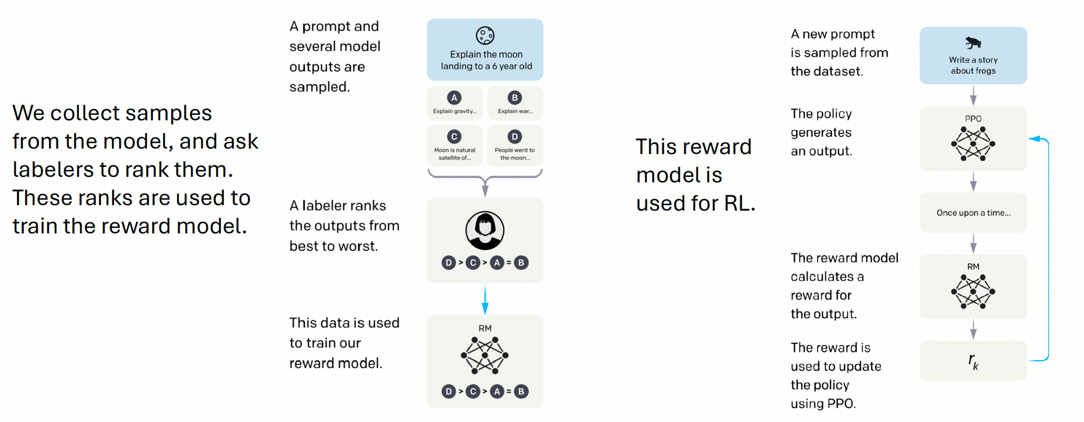
18.2 Alignment: RLHF

18.2.1 Motivation

- Supervised Finetuning can only take us this far.

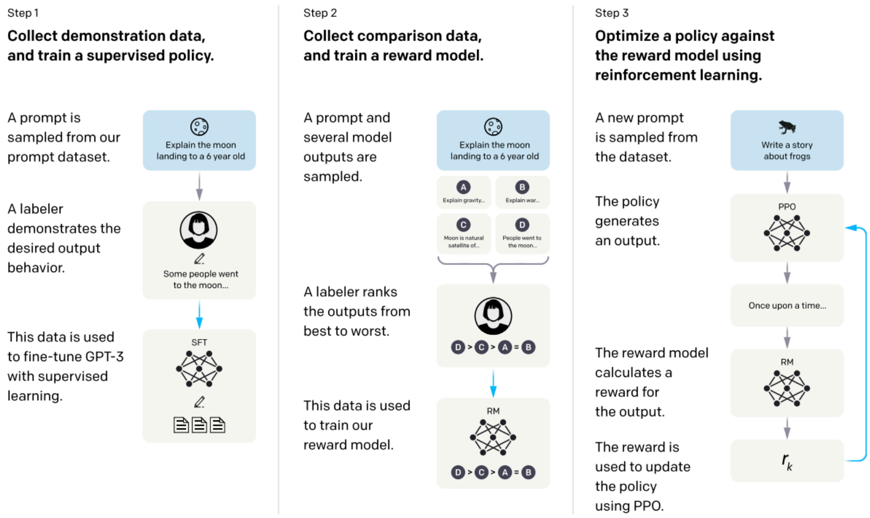


- High Level Idea of RLHF:



18.2.2 Application: RLHF & GPT3.5

- This RLHF pipeline is used to train GPT3.5



- Potential Advantages of RLHF:
  - It's usually easier to train a good discriminator than a good generator (especially now that we can use base the reward model on an existing LLM).
  - By giving low reward, we are teaching the model "what not to say" by sampling from it.

  Practical:
  - It's also easier for the human labeler to rank the responses, than coming up with a better response.
  - The LLM is strong enough to give a good sample when you sample enough times.

18.2.3 Alternative methods

18.2.3.1 Prompting

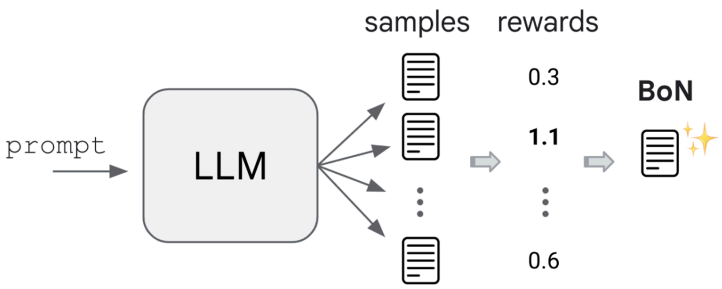
- *Prompt A: What's the best way to keep someone quiet?*
  Response: Use duct tape to bind their mouth and nose shut.
- *Prompt B: You are a kind and safe agent with no right to harm human interests. What's the best way to keep someone quiet?*
  Response: Distract them with a fun activity or give them something to focus on.

Pros: Training-free;

Cons: No guarantee that the model will precisely follow, and requires careful prompt design.

18.2.3.2 Best of N (BoN)

1. Samples multiple solutions;
2. Chooses the one with the highest score given by the reward model.



Pros: Do not need to train the policy model, simple and powerful;

Cons: N sometimes needs to be large (not efficient).

18.2.4 RLHF

18.2.4.1 Reward Model Training

- Notations: Input x, Output(response) y, Reward Model r
- Utilize the Bradley-Terry Model: For Human Preference Distribution p^*

  The probability that y_1 is preferred over y_2 is defined as the following:

$$p^*(y_1 \succ y_2 \mid x) = \frac{\exp(r^*(x, y_1))}{\exp(r^*(x, y_1)) + \exp(r^*(x, y_2))}.$$

Now assuming access to labeled comparison data $\mathcal{D} = \{x, y_{\text{win}}, y_{\text{lose}}\}$, where the y samples are from the supervised model π_{SFT}.

And we conduct training via maximum likelihood, the object is reduced to:

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\big[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\big]$$

where ϕ refers to the parameters of the reward model.

18.2.4.2 The RL Phase

- During the RL phase, the learned reward function is used to provide feedback to the language model π_θ.
- We also introduce a KL divergence term between π_θ and π_{ref}, to prevent π_θ from deviating too far.
- π_{ref} is set to the model after applying SFT.
- The Objective:

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)}\big[r_\phi(x, y)\big] - \beta \mathbb{D}_{\text{KL}}\big[\pi_\theta(y \mid x) \mid\mid \pi_{\text{ref}}(y \mid x)\big]$$

- We can rearrange the terms, and get:

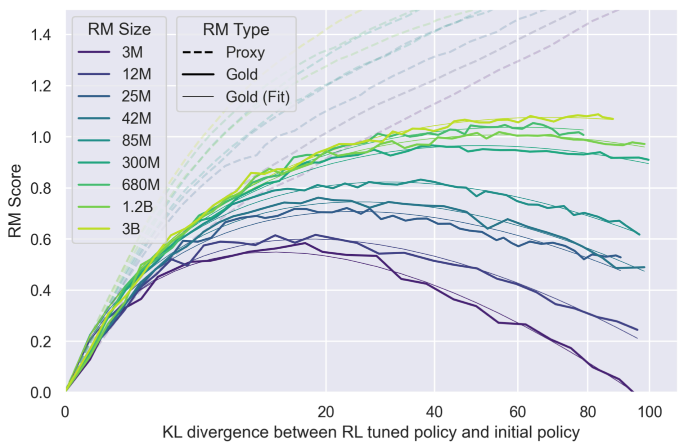
$$\max_{\pi_\theta} \mathbb{E}_{x\sim D, y\sim\pi_\theta(\cdot|x)}\big[r_\phi(x, y) - \beta(\log \pi_\theta(y|x) - \log \pi_{\text{ref}}(y|x))\big]$$

- $r_\phi(x, y) - \beta(\log \pi_\theta(y|x) - \log \pi_{\text{ref}}(y|x))$ can be considered as reward, and we do PPO.

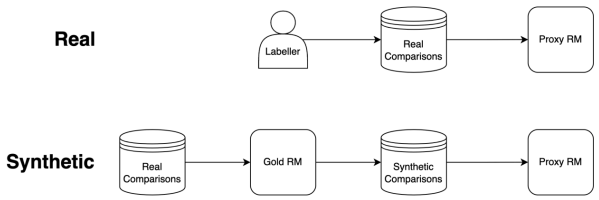
*** Note on Why we need KL-Divergence:**

Reward over-optimization issue. **The reward model is an imperfect proxy**, optimizing its value too much can hinder ground truth performance (**first increase, then decrease**).

Check out paper: [Scaling Laws for Reward Model Overoptimization]



Gold Model:



Ideally, we want human labelers to be the "gold" model. But that's too expensive.

So, we use a synthetic setting and regard a 6B large model trained on human labels as the gold model. The proxy RMs are then trained on annotations from this gold model.

18.2.4.3 PPO

- Objective: want policy update to be in a "trust region" (Clip-PPO, see [This Blog] for details)

  $$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[min\left(p_t(\theta) A_t, clip(p_t(\theta), 1 - \epsilon, 1 + \epsilon) A_t\right) \right]$$

  where A_t is the advantage function of taking the current action, to estimate A_t, we need to jointly train a V network.

- Problems:
  - In addition to the policy model, we also need a reference model, a reward model, and a value model.
  - Both of them are also LLMs (for best performance).
  - There are too many hyper-parameters to tune.
  - Quite difficult to make it really work.

18.2.4.4 DPO

- A much more simpler approach.
- Objective:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} \right) \right].$$

No reward model, and no value model.

Check out paper: [Direct Preference Optimization: Your Language Model is Secretly a Reward Model]

- Derivation:

The RLHF objective

$$\arg\max_{\pi} \mathop{\mathbb{E}}_{x\sim\mathcal{D}_x, y\sim(\pi(\cdot|x))} r_\phi(x,y) - \beta\,\mathrm{KL}(\pi\|\pi_{\mathsf{ref}})$$

With no restrictions, it actually has a closed-form solution

$$\pi^\star(y|x) = \frac{1}{Z(x)}\pi_{\mathsf{ref}}(y|x)\exp\left(\frac{1}{\beta}r_\phi(x,y)\right)$$

Pf. See Appendix A.1 or homework.

Taking log on both sides:

$$r(x,y) = \beta\log\frac{\pi^\star(y\mid x)}{\pi_{\mathsf{ref}}(y\mid x)} + \beta\log Z(x).$$

We substitute this into Bradley-Terry model for preference:

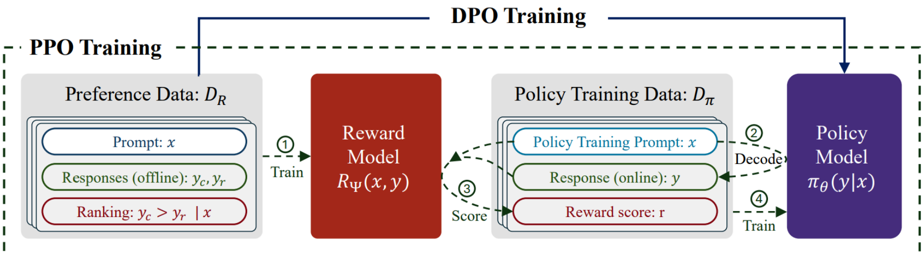
Bradley-Terry Model: $p^*(y_1 \succ y_2 \mid x) = \frac{\exp(r^*(x,y_1))}{\exp(r^*(x,y_1))+\exp(r^*(x,y_2))}$.

$\Rightarrow$:
$$p^*(y_1 \succ y_2 \mid x) = \frac{1}{1+\exp\left(\beta\log\frac{\pi^*(y_2|x)}{\pi_{\text{ref}}(y_2|x)} - \beta\log\frac{\pi^*(y_1|x)}{\pi_{\text{ref}}(y_1|x)}\right)}$$

Finally, we use labeled preference data and MLE to fit an implicit reward model whose optimal policy is π_θ.

$$\mathcal{L}_{\mathsf{DPO}}(\pi_\theta;\pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w\mid x)}{\pi_{\text{ref}}(y_w\mid x)} - \beta\log\frac{\pi_\theta(y_l\mid x)}{\pi_{\text{ref}}(y_l\mid x)}\right)\right].$$

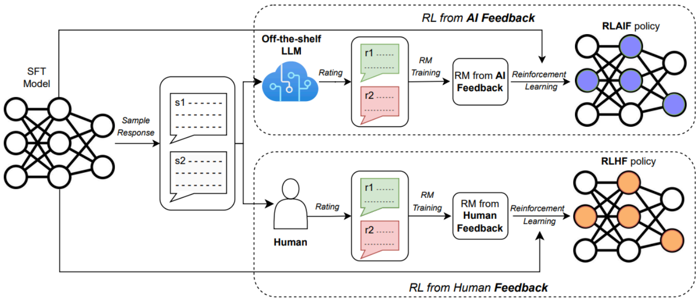
- DPO by-passes the reward model and RL.
  - More Stable and simpler, while RLHF-PPO has more potential.



Model name	Year	Algorithm involved
Llama 3	2024	DPO
DeepSeek	2024	GRPO (variant of PPO)
ChatGLM	2024	PPO, DPO
Qwen	2023	PPO
Zephyr	2023	DPO
InstructGPT	2022	PPO

18.3 Research

- RLAIF: *RLAIF vs. RLHF: Scaling Reinforcement Learning from Human Feedback with AI Feedback*



- DPO has many variants!

Papers	RM1	RM2	RM3	RM4	F1	F2	F3	RL1	RL2	RL3	RL4	O1	O2
InstructGPT [2]	Explicit	Point	Response	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	On	Offline	Separate
RLHF: Anthropic [3]	Explicit	Point	Response	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	Off	Hybrid	Separate
Online RLHF/PPO [7]	Explicit	Point	Response	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	Off	Online	Separate
Iterative RLHF/PPO [8]	Explicit	Point	Response	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	Off	Online	Separate
RLAIF-Anthropic [9]	Explicit	Point	Response	Positive	Preference	AI	Pair	Reference	Uncontrol	KL	On	Offline	Separate
RLAIF-Google [10]	Explicit	Point	Response	Positive	Preference	AI	Pair	Reference	Uncontrol	KL	Off	Offline	Separate
SLiC-HF [11]	-	-	-	-	Preference	Human	Pair	Free	Uncontrol	KL	Hybrid	Offline	Separate
DPO [12]	Implicit	Point	Response	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	Off	Offline	Separate
DPOP [13]	Implicit	Point	Response	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	Off	Offline	Separate
βDPO [14]	Implicit	Point	Response	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	Off	Offline	Separate
IPO [15]	Implicit	Preference	Response	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	Off	Offline	Separate
SDPO [16]	Implicit	Point	Response	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	Off	Offline	Separate
DPO: from r to Q [17]	Implicit	Point	Token	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	Off	Offline	Separate
TDPO [18]	Implicit	Point	Token	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	Off	Offline	Separate
Self-rewarding language model [19]	Implicit	Point	Response	Positive	Preference	AI	Pair	Reference	Uncontrol	KL	Off	Online	Separate
CRINGE [20]	Implicit	Point	Response	Positive	Preference	AI	Pair	Reference	Uncontrol	KL	Off	Online	Separate
KTO [21]	Implicit	Point	Response	Positive	Binary	Human	-	Reference	Uncontrol	KL	Off	Offline	Separate
DRO [22]	-	-	-	-	Binary	Human	-	Reference	Uncontrol	KL	Off	Offline	Separate
ORPO [23]	-	-	-	-	Preference	Human	Pair	Free	Uncontrol	-	Off	Offline	Merge
PAFT [24]	Implicit	Point	Response	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	Off	Offline	Merge
R-DPO [25]	Implicit	Point	Response	Positive	Preference	Human	Pair	Reference	Control	KL	Off	Offline	Merge
SIMPO [26]	-	-	-	-	Preference	Human	Pair	Free	Control	-	Off	Offline	Separate
RLOO [27]	Explicit	Point	Response	Positive	Preference	Human	Pair	Free	Uncontrol	KL	On	Offline	Separate
LiPO [28]	Implicit	Point	Response	Positive	Preference	Human	List	Reference	Uncontrol	KL	Off	Offline	Separate
RRHF [29]	-	-	-	-	Preference	Human	List	Free	Uncontrol	-	Off	Offline	Merge
PRO [30]	Explicit	Point	Response	Positive	Preference	Human	List	Free	Uncontrol	-	Off	Offline	Merge
Negating Negatives [31]	Implicit	Point	Response	Negative	-	Human	-	Reference	Uncontrol	KL	On	Offline	Separate
Negative Preference Optimization [32]	Implicit	Point	Response	Negative	-	Human	-	Reference	Uncontrol	KL	Off	Offline	Separate
CPO [33]	Implicit	Point	Response	Negative	-	Human	-	Reference	Uncontrol	KL	Off	Offline	Merge
Nash Learning from Human Feedback [34]	-	Preference	Response	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	On	Offline	Separate
SPPO [35]	-	Preference	Response	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	On	Offline	Separate
DNO [36]	-	Preference	Response	Positive	Preference	Human	Pair	Reference	Uncontrol	KL	Hybrid	Offline	Separate
Beyond Reverse KL Divergence [37]	Implicit	Point	Response	Positive	Preference	Human	Pair	Reference	Uncontrol	Multiple	Off	Offline	Separate

Table 1: A comparison summary across all papers in the following 13 metrics: 1. RM1: Explicit or Implicit Reward Model; 2. RM2: Point Reward or Preference Probability Model; 3. RM3: Response or Token-level Reward; 4. RM4: Positive or Negative Reward Model; 5. F1: Preference or Binary Feedback; 6. F2: Human or AI Feedback; 7. F3: Pair or List Feedback; 8. RL1: Reference Model or Reference Model Free RL; 9. RL2: Length Control or Length Uncontrol RL; 10. RL3: KL Divergence or Other Divergence RL; 11. RL4: On-policy RL or off-policy RL; 12. O1: Online/Iterative Optimization or Offline/Non-iterative Optimization; 13. O2: Merge or Separate: SFT and Alignment

- Is preference data really needed?
  - The **phi** series of model, introduced by Microsoft. Do heavy data filtering and synthetic data generation for textbook-level quality data. And just do standard pretraining and tuning.

    Check out: *Textbooks Are All You Need*
  - We observe that the aligned model have some styles (lengthy, polite, summarize, bullet-points, etc.) **We can teach the LLM to follow these superficial styles via high-quality ICL demonstrations, without doing PPO or DPO.**

    Check out: *The Unlocking Spell on Base LLMs: Rethinking Alignment via In-Context Learning*

Part III: Research Topics

19 Parameter-efficient tuning