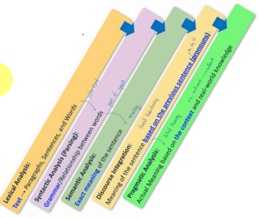
Lect 01- Intro



NLP
understand langauge
Communication
Human - Machine: QA, dialog
Human - Human: MT, spellcheck

NLP Task

Input X	Output Y	Task
Text	Continuing Text	Language Modeling
Text	Text in Other Language	Translation
Text	Label	Text Classification
Text	Linguistic Structure	Language Analysis
Image	Text	Image Captioning

Method of creating NLP

1. Rules: Bag of words
2. Prompting: Prompt w/o train
3. Finetune: learned from <X, Y>

Tokenization

1. Word - High OOV
2. Subword - moderate
3. Char - no meaning

Best Integrity (Thai)
split word if meaning not change
แม่น้ำ != แม่ + น้ำ
หมอฟัน ~= หมอ + ฟัน

Tokenizer Algor

1. Longest match - match current word long as possible in Vocab
   - ป้ายกลับรถ — Start scanning with "ป" as the starting point
   - ป้ายกลับรถ — Keep scanning …
   - ป้าย/กลับรถ — No more words start with "ป้าย", move to the next point
   - …
   - ป้าย/กลับ/รถ

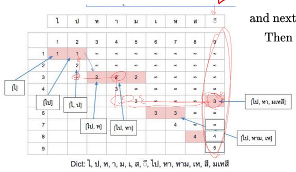
2. Maximal match (DP O(n^2))

dp[i][j] = fewest words from [1, j] (all in vocab), and current word start from [i, j]

$$dp_{i,j} = \begin{cases} \min_{k \le i-1} dp_{k,i-1} + 1, & \text{if } sent_{i:j} \in V \\ \infty, & \text{otherwise} \end{cases}$$

basecase: dp[i][i]=1, i>0

Backtracking starts from (i, n) where $i = \arg\min_k dp[k][n]$
and next step start from column: $j = i - 1$
Then find i again $i = \arg\min_k dp[k][j]$
repeat until column is 0



ML => better to handle OOV

One hot: sparse = curse of dim, no meaning

neural embedding: learnable mapping table
Vocab size — emb_dim

```
CLASS torch.nn.Embedding(num_embeddings, embedding_dim, padding_idx=None, max_norm=None,
norm_type=2.0, scale_grad_by_freq=False, sparse=False, _weight=None, _freeze=False,
device=None, dtype=None) [SOURCE]
```

embedding is actually a OHE then feed into linear layer

Lecture 02 - Language Model

LM - probabilistic model to find probability that an input sentence will occur

Full estimation: $p(w_{1:n}) = p(w_1)\prod_{i=1}^{n} p(w_i|w_{1:i-1})$

Rule-based LM

N-gram: $Pr(w_i|w_{1:i-1}) = Pr(w_i|w_{i-n+1:i-1}) = \frac{c(w_{i-n+1:i})}{c(w_{i-n+1:i-1})}$ $Pr(<s>) = 1$

Example: Bi-gram
- Estimating N-grams Probability
  - Uni-gram counting

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

  - Bi-grams counting (column given row) $Pr(i, want) = P(want|i) = \frac{827}{2533} = \frac{cnt(i, want)}{cnt(i)}$
    - "i want" → $c(prev, cur) = c(w_{i-1}, w_i) = c(want, i) = 827$

prev \ curr	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

Evaluation

Extrinsic: downstream task - require human

Intrinsic: Perplexity ---***

$$Perplexity(P) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i)}} = 2^{H(P)},\ H(P) = -\frac{1}{N}\sum_{i=1}^{N}\log_2 P(w_i)$$

P is n-gram

N-gram problem

1. Unknown words: OOVs => word that is not found in train set

handle ① Assign to normal word → unk approach
approach
- lower freq. than Min threshold → treat as <unk> (replace on training set)
- top n freq

e.g. training data has 1200 distinct words
→ use top 1000 highest frequency as a vocab
→ the rest (200) replace to <unk> token
→ and set $p(<unk>) = \frac{w(<unk>)}{|V|} = \frac{200}{1000}$

② set $p(<unk>) = \frac{1}{|V|} = \frac{1}{1200}$ (from above example)

2. Zero: => n-gram count = 0 => Perplexity is divided by 0

1. Laplace Smoothing

↳ add 1 to all words count.

$$P(w_i) = \frac{c_i}{N} \xrightarrow{\text{to}} P_{\text{Laplace}}(w_i) = \frac{c_i+1}{N+V}$$ ci = ngram count

	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

→

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

2. LM Interpolation

↳ example. for tri-gram there is no $w_{n-2}w_{n-1}w_n$
sol ↳ use bigram to estimate $P(w_n|w_{n-1})$ instead

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n|w_{n-2}w_{n-1}) + \lambda_0 C$$

$\sum \lambda_i = 1, \forall \lambda_i \ge 0$; C → <unk>

3. Stupid Backoff

back to n-1 gram until count != 0

4. Kneser-Ney Smoothing

backoff to unigram not making sense

d = abs discount (=> count in train & test not equal) => penalty = 0.75

- If our corpus contains these bigrams
  - { San Francisco, San Francisco, San Francisco, Sun glasses, Reading glasses, Colored glasses }
- P_{cont}(Francisco) = (1/4) = 0.25
- P_{cont}(glasses) = (3/4) = 0.75
- Now, a word like "Francisco" will have low $P_{continuation}$

type(., w) = unique bigram count that end w => fix col sum row
type(w, .) = unique bigram count that start w => fix row sum col
type(., .) = all unique bigram count

$$P_{\text{cont.}}(w) = \frac{\text{type}(., w)}{\text{type}(., .)}$$, all bigram end with w over all unique bigram

$$\lambda(w_{i-1}) = \frac{d}{c_{\text{unigram}}(w_{i-1})}\text{type}(w_{i-1}, .)$$, weight to make $\sum_{w \in W} P_{kn}(w, w_{i-1}) = 1$

$$P_{kn}(w_i|w_{i-1}) = \frac{\max(c_{\text{bigram}}(w_{i-1}, w_i) - d, 0)}{c_{\text{unigram}}(w_{i-1})} + \lambda(w_{i-1})P_{\text{cont.}}(w_i)$$

Example: a bigram Kneser-ney (cont.)

Compute the log-likelihood of the sentence "<s> am Sam </s>"

Pkn2(am|<s>)=(max(0-0.75,0)/4)+(0.75*2/4)*(1/11) =0.03409 [type(<s>,·)=2, type(·,am)=1]

Pkn2(Sam|am)=(max(2-0.75,0)/3)+(0.75*2/3)*(2/11) =0.5076 [type(am,·)=2, type(·,Sam)=2]

Pkn2(</s>|Sam)=(max(2-0.75,0)/3)+(0.75*2/3)*(3/11)=0.5530 [type(Sam,·)=2, type(·,</s>)=3]

LL = ln(0.03409) + ln(0.5076) + ln(0.5530) = -4.6492

$$P_{KN}(w_i|w_{i-1}) = \frac{\max(c(w_{i-1}w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_{\text{CONTINUATION}}(w_i)$$

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})}|\{w : c(w_{i-1}, w) > 0\}|$$

$$P_{continuation}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{\sum_{w'}|\{w'_{i-1} : c(w'_{i-1}, w') > 0\}|}$$ (denominator = 11)

#bigram type = type(·,·) = 11

	<s>	I	am	Sam	like	green	eggs	</s>
<s>	0	3	0	1	0	0	0	0
I	0	0	3	0	1	0	0	0
am	0	0	0	2	0	0	0	1
Sam	0	1	0	0	0	0	0	2
like	0	0	0	0	0	1	0	0
green	0	0	0	0	0	0	1	0
eggs	0	0	0	0	0	0	0	1
</s>	0	0	0	0	0	0	0	0

existence → sum row = type(c, ·)
sum column = type(·, c) (sum existence, 1 or 0)

training corpus:
<s> I am Sam </s>
<s> Sam I am </s>
<s> I am Sam </s>
<s> I like green eggs </s>

<s>	I	am	Sam	like	green	eggs	</s>
4	4	3	3	1	1	1	4

Neural LM

Cost function

$$J = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_{t,j}\log \hat{y}_{t,j}$$ → cost only matched label

Fact: e^J is Perplexity !! ~ our probabilistic model tried to minimize perplexity

RNN based - RNN, GRU, LSTM

Lecture 02 - Subword (BPE, Wordpiece, Unigram, Sentpiece)

Byte-Pair encoding (BPE) ~ Huffman Code like ⇒ merge most frequent token

training — Corpus (word, freq)
`("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)`

epoch 1
hu = 10+5, pu = 5+12, bu = 4, un = 12+4, gs = 5, ug = 10+5+5 (most frequent)
merge in corpus:
`("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)`

usage
```
("u", "g") -> "ug"
("u", "n") -> "un"
("h", "ug") -> "hug"
```
trained 3 epochs merging rule

How to use (test data)
- bug = ["b", "ug"] ("b" in dict)
- mug = ["UNK", "ug"] ("m" not in dict)
- thug = ["UNK", "hug"] ("t" not in dict)

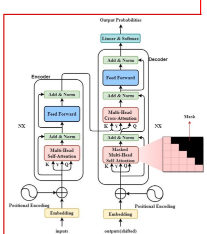
Wordpiece: same algor. as BPE but use score instead of frequency
- $WP_{score} = \frac{c(token_1, token_2)}{c(token_1)c(token_2)}$ ~ more like correlation (ρ)

** two types of tokens: start, continuing token (##x) ~ key dif: WP regards "start of word"
hug = (h, ##u, ##g)
<UNK>, unlike BPE, if word contain <UNK> it'll treat whole word as an <UNK>

`Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu", "hug"]`

How to use: "the longest subword"
- hugs = ["hug", "##s"]

If not possible to find subwords, tokenize the whole word as UNK.
- mug = ["UNK"]
- bum = ["UNK"] ~~(not ["b", "##u", UNK])~~



Unigram — Training algorithm

We'll reuse the corpus from the previous examples:
`("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)`

and for this example, we will take all strict substrings for the initial vocabulary: (Generate Vocab using all substring)
`["h", "u", "g", "hu", "ug", "p", "pu", "n", "un", "b", "bu", "s", "hug", "gs", "ugs"]`
(not include the full words)

Here are the frequencies of all the possible subwords in the vocabulary:
`("h", 15) ("u", 36) ("g", 20) ("hu", 15) ("ug", 20) ("p", 17) ("pu", 17) ("n", 16) ("un", 16) ("b", 4) ("bu", 4) ("s", 5) ("hug", 15) ("gs", 5) ("ugs", 5)`
$\sum freq = 210$ } tok

E-step
`("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)`

loss: $J = -\sum_{word \in corpus} c(word)\log(\text{word's score})$ (highest prob)

The tokenization of each word with their respective scores is:
```
"hug": ["hug"] (score 0.071428)
"pug": ["pu", "g"] (score 0.007710)
"pun": ["pu", "n"] (score 0.006168)
"bun": ["bu", "n"] (score 0.001451)
"hugs": ["hug", "s"] (score 0.001701)
```
} highest tokenizing prob. → use vertibi to find

```
10 * (-log(0.071428)) + 5 * (-log(0.007710)) + 12 * (-log(0.006168)) + 4 * (-log(0.001451)) + 5 * (-log(0.001701))
= 169.8
```

M Step
↳ remove top p% that will affect the loss
- remove 'pu' → loss not change
- but remove 'hug' → loss change → worse
} Thus we should rm 'pu'
```
"hug": ["hu", "g"] (score 0.006802)
"hugs": ["hu", "gs"] (score 0.001701)
```

Algorithm
for each token in Vocab:
  remove a token → re-compute loss w/o the token
  store top m loss affection
remove top p% that less affected the loss from Vocab.

→ TLDR; พยายามลบ token ที่ทำให้ loss เพิ่มขึ้นน้อยสุด

Sentencepiece — Solve 2 issues
- detokenization problem → replace " " whitespace placeholder w/ _ (U+2581) not underscore "_" (special chars)
- E2E avoid "need of language specification"

Lecture 03: RNN + attention mech.

m2m(syn) => Tokenizer, PoS
m2o => sent-analy
o2m => txt gen, autoregressive
m2m => MT

Autoregressive train
Teacher forcing
force to use teacher ground truth.

Attention mech. — enc-dec => forgot contxt

key: decoder desire to attend encoder's hidden states

$$c_i = \sum a_{ij} \cdot h_j, \quad a_{ij} = \text{softmax}(f_{attn}(s_{i-1}, h_j))$$

j - encoder index. i - decoder index
a_{ij} is softmax across all enc's hidden states i.e. softmax(ai)

type of attention mech. — s_{i-1}: previous hidden state

1. Additive attention
$f_{attn}(s_{i-1}, h_j) = \tanh(W_a[s_{i-1}; h_j])$ (learnable; multiply separately → split)
$= \tanh(W_1 s_{i-1} + W_2 h_j)$ ~ no operation (dot) between s_{i-1}, h_j

2. Multiplicative Attention
$f_{attn}(s_{i-1}, h_j) = s_{i-1}^T W_a h_j$ → faster, efficient
high var., small gradient
!! not good on larger dimension — mitigate: multiply by $\frac{1}{\sqrt{d_s}}$ (d_s = emb_dim)

3. Self Attention —** → improve embedding quality by adding context
$a_{(vect)} = \text{softmax}(w_a \tanh(W_s H^T))$ ↳ attend with itself, i.e. No additional data (encoder only)
learnable vector, learnable matrix, $H = (h_1, \dots, h_n)$

4. Key-Value attention —***
problem w/ old attention mech. a_{ij} and c_i both use h_j for calculation ~ redundant
split h_j into k_j and v_j => learn to be key and query

5. Scaled-dot product Attention (QKV Attention) —** ~ No recurrence → enable parallelism

Idea: $a_{ij} = \text{softmax}(f_{attn}(s_{i-1}, h_j))$ (dec Q, enc K)
$c_i = \sum_j a_{ij} h_j$ (enc V)
Note: like self attention, idea is splitting h_j into two part independently

Transformer { has more FLOPs (no bottleneck)

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$ ***

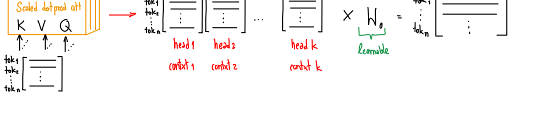
Note: $q_i = W_Q x_i$, $k_i = W_K x_i$, $v_i = W_V x_i$
→ QKV size() = (n, hidden dim)

softmax $\begin{bmatrix} q_1^T k_1 & \cdots & q_1^T k_n \\ \vdots & & \vdots \\ q_n^T k_1 & \cdots & q_n^T k_n \end{bmatrix}_{n \times n}$ = $\begin{bmatrix} w_{q_1k_1} & \cdots & w_{q_1k_n} \\ \vdots & & \vdots \\ w_{q_nk_1} & \cdots & w_{q_nk_n} \end{bmatrix}_{n \times n}$

$\frac{1}{\sqrt{d_k}}$ — query from dec, attend from enc: $\begin{bmatrix} \sum_j w_{q_1k_j} v_j \\ \vdots \\ \sum_j w_{q_nk_j} v_j \end{bmatrix}_{n \times hidden}$

self-attention - QKV from same part
cross-attention - Q and (K) from other
masked-sa - masked next word weight

Multihead-attention



positional encoding => same word = same embedding

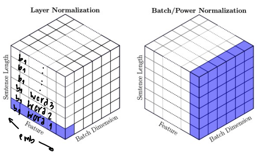
$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$ no positional information

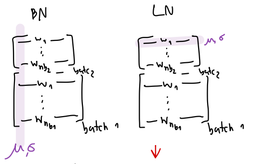
$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$ add positional information

pos is the token position, *i* is the dimension

type of PE: 1. absolute(fix contxt len) 2. relative

layer norm



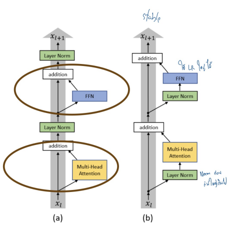


norm across emb dim (temporal norm)

norm across emb

use LN!!!!!

More transformer



transformer problem

1. emb_layer size -> big mem
V->H big
V->E->H (V,H >> E)

2. attn mat. size -> NxN attn too large
use sparse representation, Flash attn

3. contxt len -> XL-tran., RoPE

4. GPU Util -> Group Query

5. PreLN - solve gradient problem.

L04 - Token clf

e.g., PoS Tag

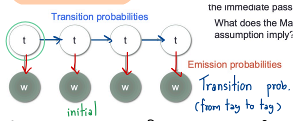
Traditional method

1. Seq method

T = list of tag, W = list of word, |T|=|W|=n

Objective $\arg\max_T P(T|W)$, i.e. find tags for each pos.

Modeling P(T, W) instead (generative) w/ Markov Assumption



$P(T,W) = P(t_1)P(t_2|t_1)P(t_3|t_2)..P(t_n|t_{n-1})P(w_1|t_1)P(w_2|t_2)...P(w_n|t_n)$

How to search for T that maximize P(T, W) => Vertibi Algor (DP)

Define $\Pi[i,t]$ as the maximum log probability of the tag sequence from position 1 to i with $t_i = t$.

Initialization:

$$\Pi[0, <s>] = 0$$

$$\Pi[0, t] = -\infty \quad \forall t \neq <s>$$

Recursion:

$$\Pi[i,t] = \max_{t' \in T}\left(\Pi[i-1,t'] + \underbrace{\log p(t|t')}_{\text{transition}} + \underbrace{\log p(w_i|t)}_{\text{emission}}\right)$$

$$\text{backpointer}[i,t] = \arg\max_{t' \in T}\left(\Pi[i-1,t'] + \log p(t|t') + \log p(w_i|t)\right)$$

example

A_{ij}	To N	To NN
From N	0.6	0.4
From NN	0.5	0.5

B_{ik}	I	eat	Chinese
State N	0.8	0.01	0.19
State NN	0.1	0.45	0.45

π[i, t]	I	eat	Chinese
State N	0.8	0.005	
State NN	0.1		

B[i, t]	I	eat	Chinese
State N	-	N	
State NN	-		

argmax

Store in back pointer

0.8 * 0.6 * 0.01 vs 0.1 * 0.5 * 0.01
0.0048 > vs 0.0005

Recursive Step:

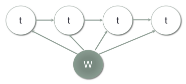
$$\pi[i,t] = \max_{t'}\{\pi[i-1,t'] + logP(t|t') + logP(w_i|t)\}$$

Not guarantee max

Vertibi Problem -> Too slow O(T^2W) => use beam search is fine

Modeling P(T|W) directly : Condition Random Field (CRF)

Truly discriminative => better for classification
add dependencies across all words



Model: $P(T|W) = \prod_t P(t_i | t_{i-1}, W)$

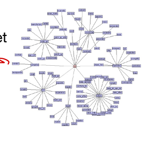
But W is too large => We calculate it via logit and softmax instead

w/ feat function ...

NN of PoS => BERT based, use NN w/ CRF

L05 - Word Representation

Symbolic - based on some rule(human labor) e.g. wordnet



Distributed: computed from word around it

sparse

1. Term Frequency (Raw frequency)
- doc info retrieval
2. Co-occurence -> small contxt (windowing)

3. Positive pointwise mutual info (PPMI)

measure informative ? -> PMI

$PMI(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)}$ → occur together / occur independently

target word, context word

$PPMI(w,c) = \max(PMI(w,c), 0)$

independent => not cnt

4. TF-IDF

!!designed for only docs similarity (not for word)

$$TF(w, doc) = \frac{freq(w \in doc)}{wordcnt(doc)}$$

$$IDF(w) = \log \frac{\#\text{docs in corpus}}{\#\text{docs that contain } w}$$

$$TF-IDF(w, doc) = TF(w, doc) \cdot IDF(w)$$

IDF
occur so frequent
= no sig. meaning

TF-idf variant: BM-25 => designed for RAG

dense

handle typos + capture synonyms

How 2 train = train on target task, LM task, supervised task

Simple NN model

CBOW: give context predict target word -> just avg embds
SKIP-gram: give target word predict context

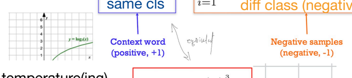
New idea => negative sampling => idea to constrastive learn.

use some sample as loss (instead of whole vocab)

o = otherword c = center(target) word

$$J_t(\theta) = \log\sigma(u_o^T v_c) + \sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)}\left[\log\sigma(-u_j^T v_c)\right]$$

same cls — diff class (negative)



temperature(ing) (first era)

$$P^{\frac{3}{4}}(w) = \frac{(w)^{\frac{3}{4}}}{\sum_{w'}(w')^{\frac{3}{4}}}$$

Spead out prob to rare word (like temperature)

Pretrained-Word2Vec => fastText ~ handle typo

add character n-gram in embedding

<aple> => f(<unk>) + w_1f(<ap) + ... ≈ <apple>

equal — make them unequal

<orenge> => f(<unk>) + w_1f(<or) + ... ≈ <orenge>

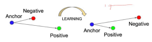
if not add, they are treat as same embd => not good

L05 - Text reprsentation

Latent(emb) space -> lower dim representation
embd -> trained on supervised or self-supervised

Constrastive learning similar thing = similar embedding (close angle)

Triplet loss -> pick anchor, positive, negative



$$\sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+$$

minimize distance on positive; maximize distance on negative; margin; max(0, x); negative sampling

f(x) is expensive
compute all pair

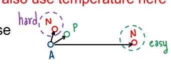
InfoNCE -> 1 anchor, 1 postive, many negative

$$InfoNCE\ Loss_a = \frac{-1}{B \cdot P} \sum_{i=1}^{B} \sum_{p \in P(i)} \log \frac{e^{\cos(f_i, f_p)}}{e^{\cos(f_i, f_p)} + \sum_{n \in N(i)} e^{\cos(f_i, f_n)}}$$

cosine similarity; select anchor; negative; Softmax across mini batch

Might also use temperature here

negative mining => choose only hard case



MUSE - sentence encoding

DAN = average context + feed to MLP

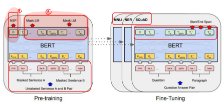
USE - universal sentence encoder

Skip-thought(sentence level skip-gram), res-pred, NLI

use attention or DAN to emb sent. Then using pretrained (multitask)

MUSE - Multilingual USE

Bert-based embedding



Bert as sent. representation
1. use NSP(sent. summary emb)
2. use Pooled embedded

emb on all task => shared encoder

Consert => contrastive self-supervised
the positive embedding = augemented embedding, other is negative

SimCSE=> from Consert apply only dropout is sufficient

L06 - Text Classification

task: sentiment, authorship id, spam filtering

Set of docs: D
Set of classes: C

BOW(Naïve Bayes) -> no positional (just wc)

count only buzzword (human defined)

$$\arg\max_C P(C|D) = \arg\max_C P(D|C)P(C)$$

(generative = MAP) $P(x) = \frac{\text{how many } x \text{ occurs in all docs}}{\# \text{ docs}}$

sampleID	hair color	eye color	weight	apply lotion	sun burn
S1	black	dark	overweight	no	-
S2	red	dark	normal	no	+
S3	blonde	light	overweight	no	+
S4	red	light	underweight	no	+
S5	black	dark	overweight	yes	-
S6	blonde	dark	overweight	no	+
S7	red	light	underweight	yes	-
S8	black	dark	normal	no	-
S9	blonde	dark	normal	yes	+
S10	red	light	normal	yes	+
S11	black	light	normal	yes	+
S12	blonde	light	underweight	no	+
S13	red	dark	normal	yes	-
S14	black	light	underweight	no	+

$x_z = \langle \text{red, dark, OW, No} \rangle$

$P(+|x) = P(+) \cdot P(\text{red}|+) \cdot P(\text{dark}|+) \cdot P(\text{OW}|+) \cdot P(\text{No}|+)$

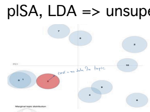
$= \frac{9}{14} \times \frac{3}{9} \times \frac{3}{9} \times \frac{2}{9} \times \frac{6}{9} \approx 0.01$

$P(-|x) = \frac{5}{14} \times \frac{2}{5} \times \frac{3}{5} \times \frac{4}{5} \times \frac{2}{5} \approx 0.018$

BOW > log. reg when data is small OOV same as <unk> in Ngram

pISA, LDA => unsupervised for insight, like K-mean

TLDR; just clustering for text



Lect 07 - MT, **Evaluation** mBART, NLLB

back translation: x → BART → ŷ → BART → x̂ (Loss)

pretrain - denoise(fill mask) like BERT for enc-decs of each language

m2m-100 - group of languages and specify group token

NLLB-200 - from m2m100 each group of Token use Mix. of Expert(MoE) to weight

Evaluation

3+1 types - syntatic (no semantic), semantic(embedding), human-based + LLM judge

Syntatic wordcount metric => weakness ~ typos + synonym

1.BLEU precision score with clipped(max) n-gram

precision - divide by len(pred)
recall - divide by len(ref)

use weighted geo-mean for each n-gram + penalty if short length

BLEU(W) = Brevity Penalty × Geo mean prec(W) → ** Brevity Penalty = $e^{1-r/c}$ if r > c; 1, otherwise

a sentence

- **Reference Sentence:** The cat is on the mat. → • Precision: $P_1 = \frac{5}{6}$ unigram
- **Candidate Sentence:** The cat is on a mat. → • Precision: $P_2 = \frac{3}{5}$ bigram

• BP = 1. r=c

$$BLEU = 1 \times \exp\left(0.5\log\frac{5}{6} + 0.5\log\frac{3}{5}\right)$$

assume equal weight = 0.5

$$= \exp(0.5(-0.182) + 0.5(-0.222))$$

$$= \exp(-0.202) \approx 0.817$$

2. ROUGE - F1-Score $F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$ like BLEU

Compute F1-score, prec, recall =overlap_cnt/len(pred or ref)

if multirefs use a most overlap as the ref.

ROUGE-N - match in N-gram level => no order just word count
ROUGE-L - use LCS instead of N-gram cnt. => have order
ROUGE-LSUM - ROUGE-L but sentence
ROUGE-W - ROUGE-L with consecutive score
ROUGE-S - ROUGE + preseved order skip-gram

3.METEOR

each word (in sentence) apply: stemming/lemmatization + synonym(via wordnet)

Weighted F1: recall:precision = 9:1 $F_{mean} = \frac{10P}{R + 9P}$

chunking (penalty on non-con matches) -> deal with shuffle order got same score

$$p = 0.5\left(\frac{c}{u_m}\right)^3,$$

c = chunk matches, u_m = **ref** word count

→ c must be the lowest as possible

Final score

$$M = F_{mean} \cdot (1 - p)$$

Reference	the	cat		sat	on	the	mat
Hypothesis	the	cat	was	sat	on	the	mat

c1, c2

Score	$0.9654 = \underbrace{0.9836}_{\text{Fmean}} \times (1 - \underbrace{0.0185}_{\text{Penalty}})$	=> M
Fmean	$0.9836 = 10 \times \underbrace{0.8571}_{\text{Precision}} \times \frac{\overset{\text{Recall}}{1.0000}}{\underbrace{1.0000}_{\text{Recall}} + 9 \times \underbrace{0.8571}_{\text{Precision}}}$	=> Fmean
Penalty	$0.0185 = 0.5 \times \underbrace{0.3333^3}_{\text{Fragmentation}}$	=>p
Fragmentation	$0.3333 = \frac{\overset{\text{Chunks}}{2.0000}}{\underset{\text{Matches}}{6.0000}}$	=> c/um

(chunk case be match across the position
e.g.,
ref: c1 c2 c3
co: c3 x c1
= 2 chuk match
)

4.TER = edit_distance / #ref (recall)

5.

chrF = (char n-gram)
chrF++ = chrF + (word n-gram)

$$\text{CHRF}\beta = (1 + \beta^2)\frac{\text{CHRP} \cdot \text{CHRR}}{\beta^2 \cdot \text{CHRP} + \text{CHRR}}$$

FB-score (instead of F1)

Semantic

BertScore => use bert (similarity score => dot prod) instead

$R_{\text{BERT}} = \frac{1}{|x|}\sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} x_i^\top \hat{x}_j, \quad P_{\text{BERT}} = \frac{1}{|\hat{x}|}\sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} x_i^\top \hat{x}_j, \quad F_{\text{BERT}} = 2\frac{P_{\text{BERT}} \cdot R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}}.$

COMET => use a model to predict score (gt is human score DA or HTER)

hypothesis-h, ref-r, source-s → use encoder then pool into 1 vect + feed to NN

COMET Variant: Comet marginal(Contrastive L), COMET w/o Ref (just rm ref from original comet)

BLEURT => BLEURT = BERT + FT w/ human rating

Human Judgement

HTER - TER but human editted prediction as a ref
Direct Assessment (DA) - Satisfication score (e.g., 0-100)
Multi Quality Metric(MQM) - multicriteria score

LLM as a JUDGE(For only QA/generation not MT)

QA Train
Atlas: Dec + Contriever
Replug: Contriver
GPT: Dec → integrate w/ RL

MT-Bench - multiturn (score each turn after prompt)
ChatbotArena - Let LLM fight with human preferences score

To Judge Prompt(using LLM): ***Must have score criteria

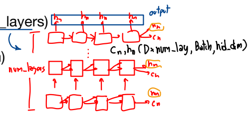
CHIE(Aj.เอก) - correctness, helpfulness, irrelevance, extraneousness(Binary score)

Torch

emb => Embedding(vocab_size, emb_dim)
=> in (*) (assert 0 < each entry < vocab size)
=> out(*, emb_dim)

Linear => Linear(in, out)
=>in (*, in)
=>out (*, out)

LSTM => LSTM(emb_dim, hidden_dim, num_layers)
=> in (Batch, Length, emb_dim)
=> out (Batch, Length, D * hidden_dim)
// D = bidirect(1 or 2)



Conv => Conv1d(Cin,Cout,kernel_sz,pad)
=> in(Batch, Cin, Lin) can be any
=> out(Batch, Cout, Lout)

Lout = Lin + 2*pad-kernel_sz + 1
(if kernel_size=2pad => Lin=Lout)