Natural Language Processing and Machine Translation

Language Models

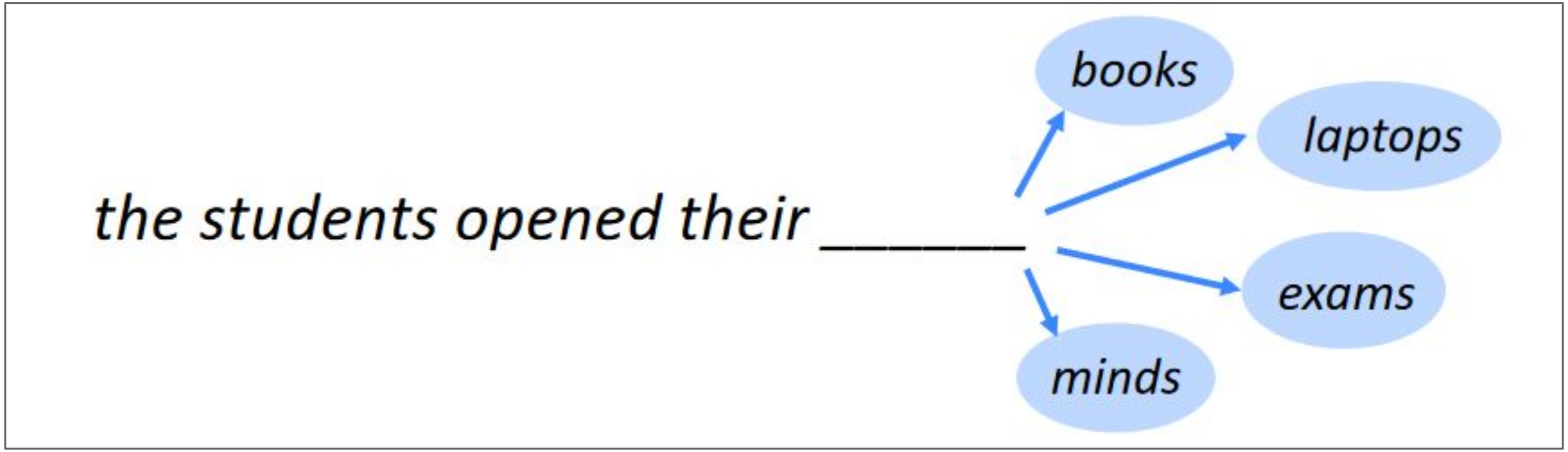
Abhishek Koirala

M.Sc. in Informatics and Intelligent Systems Engineering

THAPATHALI CAMPUS
INSTITUTES OF ENGINEERING

Introduction

- Use of various statistical and probabilistic techniques to determine the probability of a given sequence of words occurring in a sentence
- Analyze bodies of text data to provide a base for word predictions



https://medium.com/@antonio.lopardo/the-basics-of-language-modeling-1c8832f21079

N-gram

The cow jumps over the moon

Unigram/ 1-gram	Bigram/2-gram	3-gram	4-gram
The cow jumps over the moon	The cow cow jumps jumps over over the the moon	The cow jumps cow jumps over jumps over the over the moon	The cow jumps over cow jumps over the jumps over the moon

If X=Num of words in a given sentence K, the number of n-grams for sentence K would be:

$$Ngrams_K = X - (N - 1)$$

N-gram Language Models

Its water is so transparent that ....

$$P(the|its\ water\ is\ so\ transparent\ that).$$

One approach to calculate this using frequency approach

$$P(the|its\ water\ is\ so\ transparent\ that) = \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)}$$

Will this give us a good estimate in all possible scenarios ??

THAPATHALI CAMPUS
INSTITUTES OF ENGINEERING

N-gram Language Models

Another way to do this is using **chain rule of probability**

p(w1...ws) = p(w1) . p(w2 | w1) . p(w3 | w1 w2) . p(w4 | w1 w2 w3) ..... p(wn | w1...wn-1)

But this is again computationally expensive

We make this more simpler with an assumption:

- We approximate the context of the word wk by looking at the last word of the context. (**Markov Assumption**)

Eg. for bigram

$$p(w) = \prod_{i=1}^{k+1} p(w_i | w_{i-1})$$

N-gram language models

<s> I am a human </s>
<s> I am not a stone </s>
<s> I live in Lahore </s>

P(I|<S>)=C(<s>|I)/C(<s>)=3/3=1

P(am|I)=C(I|am)/C(I)=⅔

P(a|am)=C(am|a)/C(a)=½

P(human|a)=C(a|human)/C(a)=½

P(</s>|human)=C(human|</s>)/C(human)=1

P(not|am)=C(am|not)/C(am)=½

P(a|not)=C(not|a)/C(not)=1

P(stone|a)=C(a|stone)/C(a)=½

P(</s>|stone)=C(stone|</s>)/C(stone)=1

P(live|I)=C(I|live)/C(I)=⅓

P(in|live)=C(live|in)/C(live)=1

P(Lahore|in)=C(in|Lahore)/C(in)=1

P(</s>|Lahore)=C(Lahore|</s>)/C(Lahore)=1

P(I am a human)
= P(I|<s>) P(am|I) P(a|am) P(human|a) P(</s>|human)
= 1 * ⅔ * ½ *½ * 1
= ⅙

P(I am human)
= P(I|<s>) P(am|I) P(human|am) P(</s>|human)
= 1 * ⅔ * 0 * 1
= 0 => Does this seem correct?

THAPATHALI CAMPUS
INSTITUTES OF ENGINEERING

Laplace Smoothing

<s> I am a human </s>
<s> I am not a stone </s>
<s> I live in Lahore </s>

P(I|<S>)=C(<s>|I)/C(<s>)=3/3=1

P(am|I)=C(I|am)/C(I)=⅔

P(a|am)=C(am|a)/C(a)=½

P(human|a)=C(a|human)/C(a)=½

P(</s>|human)=C(human|</s>)/C(human)=1

P(not|am)=C(am|not)/C(am)=½

P(a|not)=C(not|a)/C(not)=1

P(stone|a)=C(a|stone)/C(a)=½

P(</s>|stone)=C(stone|</s>)/C(stone)=1

P(live|I)=C(I|live)/C(I)=⅓

P(in|live)=C(live|in)/C(live)=1

P(Lahore|in)=C(in|Lahore)/C(in)=1

P(</s>|Lahore)=C(Lahore|</s>)/C(Lahore)=1

The solution to the problem of unseen N-grams is to re-distribute some of the probability mass from the observed frequencies to unseen N-grams. This is a general problem in probabilistic modeling called **smoothing**.

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

Using laplace smoothing (Vocab = 11)

P(I am human)
= P(I|<s>) P(am|I) P(human|am) P(</s>|human)
= (3+1)/(3+11) * (2+1)/(3+11) * (0+1)/(2+11) * (1+1)/(1+11)
= 4/14 * 3/14 * 1/13 * 2/12
= 0.00078

THAPATHALI CAMPUS
INSTITUTES OF ENGINEERING

Good Turing Discounting

- Re-estimate the amount of probability mass to assign N-gram with zero or low counts by looking at the number of N-grams with higher counts
- Use the count of things which are seen once to help estimates the count of things never seen.
- Let Nc be number of N-grams that occur c times
    - For bigrams, No, is the number of bigrams of count 0, N1, is the number of bigrams with count 1, etc
- Revised count

$$\boxed{c^* = (c+1)\frac{N_{c+1}}{N_c}}$$

THAPATHALI CAMPUS
INSTITUTES OF ENGINEERING

Kneser Ney Smoothing

- Let the count assigned to each unigram be the number of different words that it follows. Define:

$$N_{1+}(\bullet\ w_i) = |\{w_{i-1} : c(w_{i-1}w_i) > 0\}|$$

$$N_{1+}(\bullet\ \bullet) = \sum_{w_i} N_{1+}(\bullet\ w_i)$$

- Let lower-order distribution be:

$$p_{KN}(w_i) = \frac{N_{1+}(\bullet\ w_i)}{N_{1+}(\bullet\ \bullet)}$$

- Put it all together:

$$p_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{\max\{c(w_{i-n+1}^{i}) - \delta, 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} + \frac{\delta}{\sum_{w_i} c(w_{i-n+1}^{i})} N_{1+}(w_{i-n+1}^{i-1}\ \bullet) p_{KN}(w_i|w_{i-n+2}^{i-1})$$

THAPATHALI CAMPUS
INSTITUTES OF ENGINEERING

Evaluating Language Models

2 types of evaluation

- Extrinsic evaluation
- Intrinsic evaluation

Extrinsic Evaluation

- Model metrics compared with respect to applications implemented in

Intrinsic Evaluation

- Single model evaluation
    - Perplexity

THAPATHALI CAMPUS
INSTITUTES OF ENGINEERING

Perplexity

I always order burger with ....

fries
drinks
sausage
....
....
burgers
with burgers

A good language models with add words like fries, drinks or sausage to the above sentence while a bad language model could add completely random words

A better model of text is the one that assigns a higher probability to the words that actually occurs.

Perplexity is the probability of test set normalized by the number of words

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

THAPATHALI CAMPUS
INSTITUTES OF ENGINEERING

Perplexity as average branching factor

How hard is the task of recognizing digits ‘0’, ‘1’,‘2’,‘3’,‘4’,‘5’,‘6’,‘8’,‘8’,‘9’ ?

There were …….. people in the room.

Assuming that the above space can be filled with any digit from 0-9 and probability of all these digits are equally likely

P(any digit) = 1/10

PP(any digit)= (1/10)^-1

= 10

Lower the perplexity, better the model

THAPATHALI CAMPUS
INSTITUTES OF ENGINEERING

Perplexity as average branching factor

How hard is the task of recognizing digits ‘0’, ‘1’,‘2’,‘3’,‘4’,‘5’,‘6’,‘8’,‘8’,‘9’ ?

There were …….. people in the room.

Assuming that the above space can be filled with any digit from 0-9 and probability of all these digits are equally likely

P(any digit) = 1/10

PP(any digit)= (1/10)^-1

= 10

Lower the perplexity, better the model

THAPATHALI CAMPUS
INSTITUTES OF ENGINEERING

Backoff

- Non linear method
- Estimate for an n-gram is allowed to back off through progressively shorter histories
- Trigram version

$$\hat{P}(w_i \mid w_{i-2}w_{i-1}) = \begin{cases} P(w_i \mid w_{i-2}w_{i-1}), & \text{if } C(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha_1 P(w_i \mid w_{i-1}), & \text{if } C(w_{i-2}w_{i-1}w_i) = 0 \\ & \text{and } C(w_{i-1}w_i) > 0 \\ \alpha_2 P(w_i), & \text{otherwise.} \end{cases}$$

THAPATHALI CAMPUS
INSTITUTES OF ENGINEERING

Word Representation

How do we represent the meaning of a wod?

⇒ One common NLP solution: Use a taxonomic resource(eg. **WordNet**), a thesaurus containing a list of synonyms set and hypernyms set (“is a” relationships)

E.g., synonyms set containing “good”:

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

E.g., hypernyms of “panda”

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

THAPATHALI CAMPUS
INSTITUTES OF ENGINEERING

Word Representation

Problems with such discrete representation

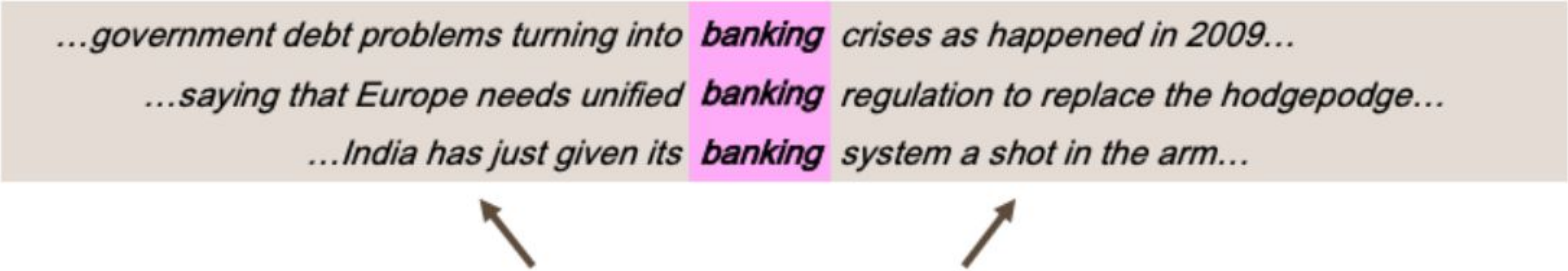
- Missing context and nuances
  - Eg. “There is a stone in river ***bank***”, “The **bank** should be closed by now”.
- Missing new meanings of words
  - Eg. wicked, badass, wizard, ninja
  - Impossible to keep up-to-date forever
- Requires human labor to maintain
- Cannot compute accurate word similarity

Word Similarity

- Naively, words can be represented by one-hot vectors
  - motel = [0 0 1 0 0 0 0 0 0 0 0]
  - hotel = [0 0 0 0 0 0 0 0 1 0 0]
- Dot products of these two vectors = 0 (orthogonal)
- Thus, there is no natural notion of similarity for one-hot vectors
- Solution?
  - Rely on WordNet to get similarity?
    - Incompleteness, inconsistency, difficult to maintain
  - **Actual Solution: Learn to encode similarity in the vectors**

Word Similarity

- How to encode similarity?
  - Distributional semantics: **A word's meaning is given by the words that frequently appear close-by**
    - "*You shall know a word by the company it keeps*" (J.R Firth 1957)
    - One of the most successful ideas of modern NLP
  - When a word *w* appears in a text, its context is the set of words that appear nearby (within a fixed size window)



These context words will represent ***banking***

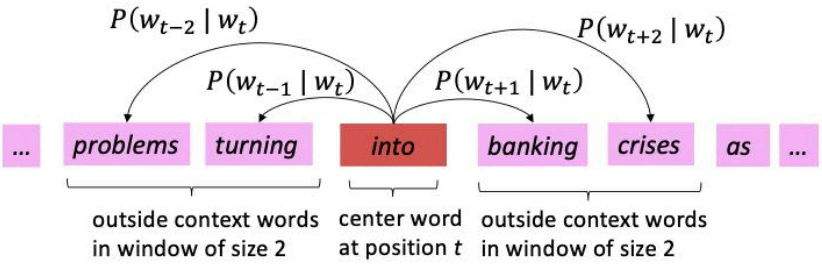
Word Vectors

$$\textit{expect} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

Word2Vec

Word2Vec is an initial framework for learning word vectors

- We got large corpus of text
- Go through each position *t* in the text, which has a **center word c**, and **context ("outside") words o**
- Calculate the **probability** of *o* given *c* (or vice versa)
- **Gradient descent** to maximize this probability
- Example windows for $P(w_{t+j}|w_t)$



Efficient Estimation of Word Representations in Vector Space, Mikolov et al., 2013, https://arxiv.org/pdf/1301.3781.pdf

Word2Vec: Objective Function

For each position t = 1,...T, predict context words within a window of fixed size m, given center word w_j, the likelihood is

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

The objective function $J(\theta)$ is the average negative log likelihood:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

THAPATHALI CAMPUS
INSTITUTES OF ENGINEERING

Word2Vec

Two flavors of Word2Vec algorithm

- **CBOW (Continuous Bag of words)**
  - Uses the context words to predict current word
- **Skip-gram**
  - Use the current word to predict its context

Word2Vec

The probability of a predicted word occurring given a center word:

$$P\left(predictedWord_n | centerWord\right) = \frac{e^{dot(predictedWord_n,\ centerWord)}}{\sum_i e^{dot(predictedWord_i,\ centerWord)}}$$

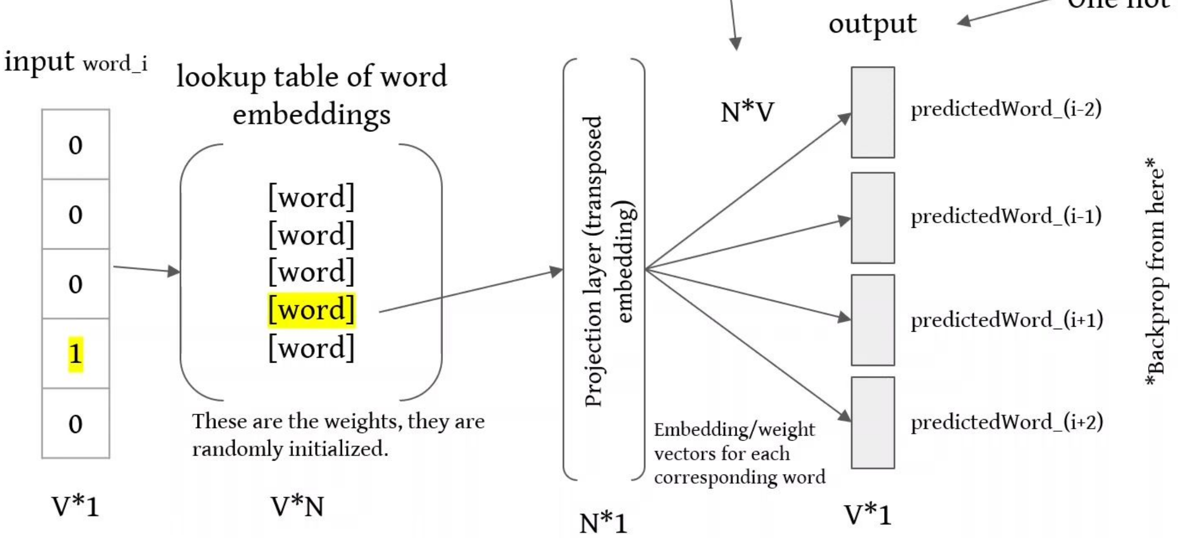
A closer look...

Activation function:

$$softmax\left(predictedWord_n\right) = \frac{e^{predictedWord_n}}{\sum_i e^{predictedWord_i}}$$

One hot vector in:

One hot vector out



The softmax activation normalizes the outputs as a probability distribution. This means a percentage is associated with each predicted word.

V: # of words in the corpus, N: # of values in our vectors

The weight vector is actually what becomes your word embedding!

THAPATHALI CAMPUS
INSTITUTES OF ENGINEERING

Word2Vec

```
[12] from gensim.models import Word2Vec
     import numpy as np
```

```
[13] sentences=[['this','is','a','sentence','about','school'],
               ['school','has','students','and','teachers'],
               ['the','students','learn'],
               ['the','teachers','make','money']]
```

```
[28] model=Word2Vec(sentences, min_count=1, window=3)
```

```
first=model['students']
target=model['learn']
second=model['teachers']

print("First: ", np.linalg.norm(target-first))
print("Second: ", np.linalg.norm(target-second))
```

```
First:  0.04220174
Second:  0.03774856
```

```
model.most_similar('school')
```

```
[('has', 0.17184367775917053),
 ('the', 0.15330740809440613),
 ('this', 0.12881389260292053),
 ('students', 0.08287020027637482),
 ('a', 0.06733693182468414),
 ('sentence', 0.05654553323984146),
 ('and', 0.008189082145690918),
 ('money', -0.0024816691875457764),
 ('teachers', -0.029205819591879845),
 ('learn', -0.06788718700408936)]
```

sg parameter in Word2Vec( ):

sg →0 (CBOW)

sg → 1 (Skipgrams)

Word2Vec : Skip-gram vs CBOW

- **CBOW** (Predict center word given outside words) and **Skip-gram** (Predict context ("outside") words given center word) uses the **same training procedure.**
- **CBOW** is much simpler, this implies a **much faster convergence** for CBOW than for Skip-gram, in the original paper, CBOW took hours to train, Skip-gram 3 days.
- CBOW learns better **syntactic relationships** between words while Skip-gram is better in capturing better **semantic relationships**. For the word 'cat':
  - CBOW would retrieve as closest vectors morphologically like plurals, i.e. **'cats'**
  - Skip-gram would consider morphologically different words (but semantically relevant) like **'dog' much closer to 'cat'** in comparison.
- Because Skip-gram rely on single word input, it is **less sensitive to overfit frequent words** (and it's also the reason of the better performances of Skip-gram in capturing semantic relationships).


рахмат
danke
謝謝
spas
ngiyabonga
شكراً جزيلاً
Баярлалаа
mersi
kia ora
barka
welalin
tack
teşekkür ederim
mahalo
tapadh leat
спасибо
faafetai lava
vinaka
спасибі
blagodaram
dank je
misaotra
matondo
paldies
grazzi
gracias
хвала
kiitos
dankie
dhanyavad
nanni
nandri
mauruuru
köszönöm
thank you
asante
manana
obrigada
tenki
bedankt
bayarlalaa
gràcie
hvala
enkosi
akun
dankon
aciū
djiere dieuf
tau
mochchakkeram
murakoze
дякую
mamnun
dziękuję
chnorakaloutioun
gratias ago
gràcies
sulpáy
go raibh maith agat
chokrane
sobodi
děkuji
taiku
obrigado
mèsi
didi madloba
sagolun
sukriya
kop khun krap
ありがとう
arigatô
takk
dakujem
trugarez
najis tuke
tanemirt
grazie
kam sah hamnida
rahmat
terima kasih
rahmet
diolch
dhanyavadagalu
shukriya
merce
мерси
তোমাকে ধন্যবাদ
감사합니다
xièxie
ευχαριστώ
merci