Deep Learning and NLP

Introduction and Word Embeddings

Joseph Le Roux

14/11/25

Outline

1
Introduction

1 NLP

Various tasks that *process data encoded in natural language* (NL)

speech recognition, text classification, NL understanding, and NL generation...

At the crossroad of :

- Linguistics
- Computer Science (formal languages, automata, graphs...)
- Logic
- Machine Learning (probability/optimization/statistics)
- Artificial Intelligence
- Cognitive Sciences

1 Language as a Symbolic System

Language

a system that allows a speaker (writer) to communicate (among other functions)

Elements of this system are discrete (symbols)

- speech/gestures/writings are transcriptions of this system
- if they were continuous, no writing system (discretization) possible without major loss of meaning

Problems for Machine Learning

- Requires many different symbols, some very frequent some very rare
- Discrete units make gradient descent impractical

1 Why NLP is difficult: Ambiguity

Ambiguity at all levels

- *saw, duck, her, round, like*, (lexical ambiguity)
- *I saw her duck* (lexical ambiguity, in context)
- *John eats a pizza with a fork* vs. *John eats a pizza with an egg* (syntactic ambiguity, attachment)
- *A computer that understands you like your mother* (syntactic ambiguity, attachment)
- *I seek a unicorn* (semantic ambiguity, existential quantification)
- *avocat pourri* → *dirty lawyer, rotten avocado* (semantic ambiguity, expressions in translation)

Yes but... we understand each other!

- Context may help, but how? Focus of these lectures
- World knowledge, social convention, statistics may also help
- Other modalities (vision, touch...)

Great variability

Sometimes very ambiguous... in general simple (simple enough for humans to learn!)

1 What we will discuss this semester

Sessions 1 and 2: Word Embeddings

- What are the units? how do we represent them?
- Word2Vec and ML for learning word representations

Session 3: Convolution for text

- How do we represent words in context
- Also, how do we train a NN and build batches of sentences

Session 4 and 5: Language Models and Transformers

- How can we represent sequences and generate texts
- Attention mechanism

Session 6: Pretraining for Language Models

- Go back to Word2Vec and adapt it to Language Modelling and Tranformers
- Learn Contextual Word Representations

2
Word Vectors

2 Words / Vocabulary / Lexicon / Index

Definitions

- word (token) = atomic element : *dog*, *car*, *the*, *Trump* ...
  - 🤔 what about letters? we will get back to this later (subwords, BPE...)
- Vocabulary V finite set of all accepted words
- add a *special token* `UNK`
  - representing all the things in texts that are not words (*ie qwerw3y*)
  - representing all the words that may be missing in V
- Lexicon L_V finite set of words or `UNK` : $L_V = V \cup \{\texttt{UNK}\}$
- Language $\mathscr{L}_V$ is the set of strings on the lexicon $L_V^* = \bigcup_{i=0}^{\infty} L_V^i = \{\varepsilon\} \cup L \cup LL \cup ...$
- To each element of the lexicon L correponds a unique integer, its *index*, between 0 (for `UNK`) and |V|.
  - bidirectional mapping
    1. $w(i) \in L_V$ returns the word from integer i (fails if $i \geq V$)
    2. $i(w) \in \mathbb{N}$ returns the index of token w (return 0 if $w \notin L$)

2 Word sense – Word meaning

Fundamental issue in NLP: assign a meaning to words

Different point of views:

- Linguistics: link *signifier* (sound/writing) and the *signified* (the thing/the meaning)
  - not really mechanizable
- Ontology (data/knowledge base): organize a hierarchy of concepts and terms on a semantic basis
  - do not really say anything about usage (how to use words)
- CS/NLP: pragmatic approach, meaning depends on contexts defined by application
  - EX: if a vocal assistant must perform the same when user says *play the Beatles* or *put the Beatles on* or *read the Beatles*, in this context *play,read,put on* must have the same meaning/representation
  - In other contexts these words are not synonyms

2 Multi-dimensional representation (word vectors)

Assume contexts are countable (and sparse)

Then, if meaning depends on context, and if context can change, we can represent words as vectors with one dimension by context. 🤯

Synonymity between two words

- Compute *distance* / *inner product* between their vectors.
- *Partial* synonyms, depend on contexts 🤔
- Geometrical Meaning!

Geometrical Meaning

- Represent words in dense vectors $\mathbb{R}^d$
- Distance/similarity interpreted as semantic relatedness (*eg* synonyms)

Eg. assuming that *play* and *launch* are closer semantically than *play* and *clean*, we have:

$$\|V(\text{play}) - V(\text{launch})\|_2^2 \leq \|V(\text{play}) - V(\text{clean})\|_2^2$$

2 (A long time ago...)

Very different representations in NLP/CL!

Lexicon elements used to be considered as unique by default. Equivalently, they were represented by *one-hot* vectors in $\{0,1\}^{|L_V|}$. For instance:

play	$[000000000010000000]$
read	$[001000000000000000]$
stop	$[000000000000000100]$

All tokens were orthogonal, (inner product zero, distance $\sqrt{2}$), whether they had the same meaning or not. Geometry meant nothing

2 Contextual (or Distributional) Representation

Contexts as neighbouring words

You shall know a word by the company it keeps. (J. R. Firth, 1957)

An old idea in linguistics, used in NLP since 90s/00s: word represented by a set of contexts,
Meaning of a word can be inferred from its *usage*, i.e. the sequence of token in which it appears.

Example

For instance for *théorie* in a French text corpus:

… Ezion qui le mettaient en	théorie	à l' abri de soucis …
… essais assez pointus sur une	théorie	mathématique quelques articles réunis trois …
… poète météo dépendant pas de	théorie	.
…	théorie	…
	…	

théorie = { … Ezion qui le mettaient en, à l' abri de soucis …, … essais assez pointus sur une, mathématique quelques articles réunis trois …, …}

Observation

- The more contexts 2 words have in common, the more they share meaning.

2 Word Vectors x Distributional Representation = Word Space Models

Vector similarity is the only information present in Word Space: semantically related words are close, unrelated words are distant. (H. Schütze, 1993)

In the 90s several methods based on word co-occurrences, and dimension reduction for efficiency.

Big Picture

1. Associate each lexicon elt e to a vector count v_e of length $|L_V|$, initialized to zero
2. From a set of documents
   - count $n_{e,e'}$ for each lexelt e, the number each e' occurs in the neighbourhood of e
   - filter: discard tokens such as *a*, *the*..., limit to a window of k tokens around e,
   - set $v_e[id(e')] = n_{e,e'}$
3. Concatenate all vectors $v_e, \forall e$ into a co-occurrence matrix M
   - perform some normalization (*eg* TF/IDF or a variant)
4. M is big and sparse
   - use (Truncated) Singular Value Decomposition to get matrix M'
   - such as M'_e is a dense vector of dimension $d << |L_V|$

2 Word Vectors x Distributional Representation = Word Space Models (2)

Word vector distance is unstable due to differences in frequency and magnitude, and is unbounded from above.

Similarity between word vectors

To decide whether two words are similar semantically, DR usually uses *normalized* inner product also called *cosine similarity*:

$$sim(x, y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}} = \frac{x \cdot y}{(\sqrt{x^2})(\sqrt{y^2})} = \frac{\langle x, y \rangle}{|x| \times |y|}$$

(🤔 what are the min/max values of sim)

Issues with distributional approach

- normalisation, scaling → compression (matrix dimension reduction)
- not incremental → need to retrain from scratch for new data

3
Word2Vec

3 Word2Vec [Mik+18]

Algorithm to parametrize word vectors

- implements Firth's principle, related to SVD methods [LG14]
- simple idea (but implementation may be *tricky*)

Not a *Neural Network* but

- the resulting vectors will be used as input by NNs later
- learning performed via gradient descent
- probabilistic modelling (and approximation)

A little old...

- More powerful recent methods
- ... but `Word2Vec` is simple and implements Firh's idea quite directly

3 WordVec Skip-gram Model

Principle

1. Assume we have a corpus of texts (big, >1M words) for training;
2. We set vocabulary V, out of vocabulary (OOV) words are replaced by token `UNK`;
3. At each position t of a text, we write token c at position t and its neighbouring tokens O_c (words in a window of size m around t). We assume a factorized probability to generate neighbouring tokens: $p(O_c|c;\theta) = \prod_{o \in O_c} p(o|c;\theta)$
4. From the computation of similarity between center word vector v_c and context word vectors v_o, we will define probability $p_\theta(o|c)$;
5. Training objective: maximize parameters for likelihood: $\prod_c p(O_c|c;\theta)$.

Loss

Maximizing the log-likelihood amounts to the following loss:

$$\begin{aligned}\max_\theta \log \prod_t \prod_{o \in O_t} p(o|c_t;\theta) &= \max_\theta \sum_t \sum_{o \in O_t} \log p(o|c_t;\theta) \\ &= \min_\theta \sum_t \sum_{o \in O_t} -\log p(o|c_t;\theta)\end{aligned}$$

3 Word2vec: Neighbours and Probability

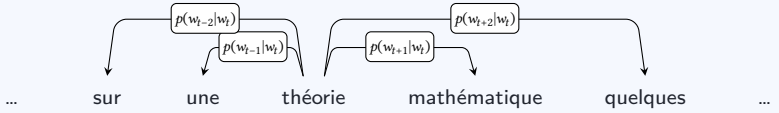
Window

- Given a position t in text, we write w_t, the token at this position
- A window of size m centered at position t, noted O_t consists of all words $w_i, \forall (t-m) \leq i \leq (t+m)$

Word2vec defines probability distributions

- Probability for all words w' to be in a window (of predefined size m) of a specific word w
- In the remainder, we call $p(w'|w)$ *neighbourhood probability*

Example



3 Word2vec : Maximum Likelihood Estimation (MLE)

General method to cast a ML problem (*eg* optimized by SGD) into a probabilistic framework. Probabilities are computed from scores given by a network with parameters θ

Definition (Likelihood)

A function returning a corpus probability (assume iid observations) from given parameters:

$$V(\theta) = \prod_{t \in \mathcal{T}} p(O_t|w_t;\theta) = \prod_{t \in \mathcal{T}} \prod_{t-m \leq j \leq t+m} p(w_j|w_t;\theta)$$

Intuition

Observations must be more probable that any event that we did not observed, and because observations *happened* they must have a probability 1. We want to *maximize* V:

$$\theta^* = \arg\max_\theta V(\theta) = \arg\max_\theta \prod_t \prod_{t-m \leq j \leq t+m} p(w_j|w_t;\theta)$$

3 Word2vec: From MLE to continuous optimization

Product → Sum

$V(\theta)$ is a product: difficult to optimize. $\log$ is monotone so we can write:

$$\theta^* = \arg\max_\theta \log V(\theta) = \arg\max_\theta \sum_t \sum_{t-m \le j \le t+m} \log p(w_j|w_t;\theta)$$

Minimization

We prefer minimizing a loss function:

$$\theta^* = \arg\min_\theta -\log V(\theta) = \arg\min_\theta \sum_t \sum_{t-m \le j \le t+m} -\log p(w_j|w_t;\theta)$$

Finally $-\log V(\theta)$ is a loss function

- positive
- zero when no error

3 Parametrization of probabilities (1/2)

Similarity between words

- We want to measure similarity between token w_t and context token w_o
- Similarity of their associated vectors $\longrightarrow$ inner product
- Issue with inner product: $v_1^\top v_2$ is maximal when $v_1 = v_2$. we want to avoid trivial solution: $v_i = v_j \forall i, j$
- Solution: define 2 vectors per token mot m:
  1. v_m when m is the center position of the window
  2. u_m when m is a context position in the window.

From similarity to proability

- in $p(w'|w)$, use inner product $u_{w'}^\top v_w$
- we use softmax:

  for a vector $a = [a_1 \dots a_n]$ softmax return a vector:

$$\text{softmax}(a)[i] = \frac{\exp(a[i])}{\sum_j \exp(a[j])}$$

3 Parametrization of probabilities (2/2)

Inner product + softmax =

$$p(w'|w) = \frac{\exp(u_{w'}^\top v_w)}{\sum_{w''} \exp(u_{w''}^\top v_w)}$$

Softmax implements the idea that the more context w' is similar to center w, that the higher $u_{w'}^\top v_w$ the more probable it is to appear in a window.
How to implement?

3 First implementation (1/2)

In lab, you will code an efficient batched version

```
import torch

class Word2Vec(torch.nn.Module):
    def __init__(self, corpus, lexicon_size, vec_size):

        # 2 tables to define from data:
        #word2idx table token -> index
        #idx2word table index -> token

        U = torch.nn.Embedding(lexicon_size, vec_size)
        V = torch.nn.Embedding(lexicon_size, vec_size)

    # returns the sim. score for all token as contexts of w_c
    def forward(self, idx_c):
        # inner product with all vectors in U
        logits = self.U.weight @ self.V(idx_c)
        return logits
```

3 First implementation (2/2)

```
import torch

def train(net, opt, train_set, nb_epoch):

    loss_fn = torch.nn.CrossEntropyLoss()
    # to adapt for lab sessions
    # to deal with batched input
    for epoch in range(nb_epoch):
        train_set.shuffle()
        for (o,c) in train_set:
            logits = net(c)
            loss = loss_fn(logits, o)
            loss.backward()
            opt.step()
            opt.zero_grad()

w2v = Word2Vec(corpus, 10000, 20)
# TODO: conversion between data and training set
#optimizer = ...
train(w2v, optimizer, train_set, 20)
```

3 Some issues with this version

Softmax on vocabulary

$$p(w'|w) = \frac{\exp(u_{w'}^\top v_w)}{\sum_{w''} \exp(u_{w''}^\top v_w)}$$

Will not scale for large vocabularies:

- complexity (time/memory) (denominator)
- rounding errors

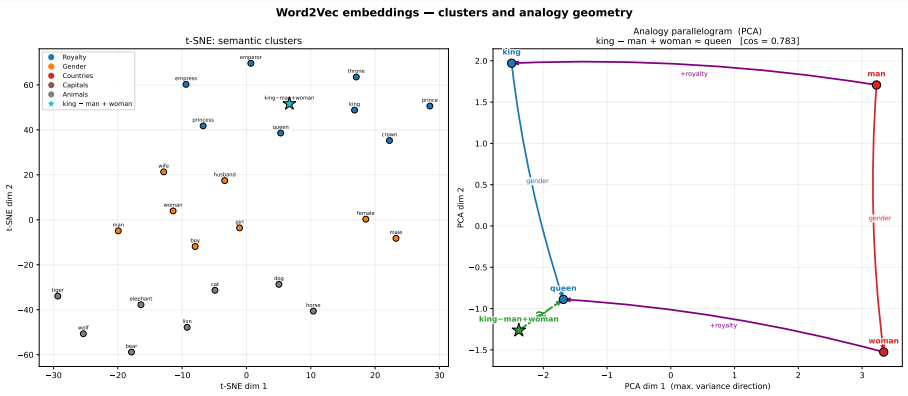
In the *real* Word2vec

- many modifications to improve efficiency
- less rounding errors

Coming up next

A very different take on similarity-based word vectors

3 What can we do with Word Vectors?



A test for the Geometry of Meaning: Analogy

- *X* is to *Y* what *U* is to *V*. Can you find *X* given *(U,V,Y)* ? Here *X is to woman, what king is to man* $\Rightarrow X =$ queen
- Semantic relation, equivalent of shifting/translating meaning semantic translation ≈ geometric vector translation!

3 Learning Example (1/3)

Corpus

...	Sgt.	Pepper	...	Lucy	...
	i	i+1		j	

- *Pepper* is in the context of *Sgt.* $\rightarrow$ (*Sgt.*, *Pepper*) is in the training corpus
- *Lucy* is not in the context of *Sgt.*
- We suppose (for this example!) that
  1. we only have 3 words in our vocabulary and word vectors have one dimension only
  2. for each word w $u_w = v_w$ (actually this is the case in some implementation)

Before/After Word2Vec Update: Objective

- Before $v_{Sgt.} = 0.3 = S$, $v_{Pepper} = 0.1 = P$, $v_{Lucy} = -0.4 = L$
- new example: (*Sgt.*, *Pepper*) objective is $\min_\theta -\log p(Pepper|Sgt.)$

$$O = -\log p(Pepper|Sgt.) = -\log \frac{\exp\langle P, S\rangle}{\exp\langle P, S\rangle + \exp\langle L, S\rangle}$$
$$= -\log\langle P, S\rangle + \log(\exp\langle P, S\rangle + \exp\langle L, S\rangle)$$

Learning Example (2/3)

Before/After Word2Vec Update: Gradients

Compute gradients wrt to each vector (here simple differentials...)

$$\begin{aligned}\frac{\partial O}{\partial S} &= -P + \frac{(P\exp\langle P,S\rangle + L\exp\langle L,S\rangle)}{(\exp\langle P,S\rangle + \exp\langle L,S\rangle)}\\ &= -P + P \times p(Peppers|Sgt.) + L \times p(Lucy|Sgt.)\\ &= -P + \mathbb{E}_{w\sim p(\cdot|Sgt.)}[v_w]\\ &= -0.1 + 0.1 \times 0.54 - 0.4 \times 0.46 = -0.23\end{aligned}$$

$$\begin{aligned}\frac{\partial O}{\partial P} &= -S + \frac{S\exp\langle P,S\rangle}{(\exp\langle P,S\rangle + \exp\langle L,S\rangle)}\\ &= -S + S \times p(Peppers|Sgt.)\\ &= -0.3 + 0.3 \times 0.54 = -0.14\end{aligned}$$

$$\begin{aligned}\frac{\partial O}{\partial L} &= \frac{S\exp\langle L,S\rangle}{(\exp\langle P,S\rangle + \exp\langle L,S\rangle)}\\ &= L \times p(Lucy|Sgt.)\\ &= 0.4 \times 0.46 = 0.18\end{aligned}$$

3 Learning Example (3/3)

Before/After Word2Vec Update: SGD Updates

Apply SGD update ($\alpha = 0.1$)

$$
\begin{aligned}
S &= S - \alpha \frac{\partial O}{\partial S} \\
&= 0.3 - 0.1(-0.23) \\
&= 0.3 + 0.023 \\
&= 0.32
\end{aligned}
\qquad
\begin{aligned}
P &= P - \alpha \frac{\partial O}{\partial P} \\
&= 0.1 - 0.1(-0.14) \\
&= 0.1 + 0.014 \\
&= 0.11
\end{aligned}
\qquad
\begin{aligned}
L &= L - \alpha \frac{\partial O}{\partial L} \\
&= -0.4 - 0.1(0.18) \\
&= -0.4 - 0.018 \\
&= -0.42
\end{aligned}
$$

Observed data updated in the same direction as condition, non-observed data in the opposite direction

4
Complements

4 MLE and Cross Entropy

In `Pytorch` learning with MLE uses a function called *cross entropy* 🤔

Definition (Entropy of a distribution)

a measure of the *randomness* of a distribution

$$H(p) = -\sum_{c \in \mathscr{C}} p(c) \log p(c)$$

- $H(c) \geq 0$
- If p uniform (*ie* random) $H(p) = \log(|\mathscr{C}|)$ (max value for H)
- If p deterministic ($p(c_i) = 1$), $H(p) = 0$
- (Prove above equalities as exercise)

4 MLE and Cross Entropy (2)

Definition (Conditional Entropy / Cross Entropy)

Allows to compute a kind of *distance* between 2 distribution 🤔

$$\texttt{CE}(q, p) = -\sum_{c} q(c) \log p(c)$$

Definition (Empirical distribution p_e)

- *distribution* of observed data $s = c_i$ always 1 (obervations are certain)
- for a multinomial, if for $p_e(c_i) = 1$ then $\forall j \neq i, p(c_j) = 0$

Equivalence

Let us compute the cross-entropy between:

- q the empirical distribution
- p the distribution computed by our model

4 MLE and Cross Entropy (3)

$$
\begin{aligned}
\mathrm{H}(q, p) &= -\sum_{c} q(c) \log p(c) \\
&= -q(c_i) \log p(c_i) - \sum_{c \neq c_i} q(c) \log p(c) \\
&= -1 \times \log p(c_i) - \sum_{c \neq c_i} 0 \times \log p(c) \\
\mathrm{H}(q, p) &= -\log p(c_i)
\end{aligned}
$$

We recover $-\log p(x)$, the loss we defined for MLE 🤯

4 Cross-Entropy in Pytorch (1)

```
# classifier for input of size 10, 5 classes
# 8 hidden neurons (cf. DATA MINING)
net = MLP(10,[8],5)

# let us pretend the following tensor contains
# the input for 3 examples
inputs = torch.rand(3,10)

# we obtain the 5 scores for all 3 examples
preds = net(inputs)

# let us suppose that the correct classes were:
empirical = torch.tensor([0,2,1], dtype=torch.long)

loss_function = torch.nn.CrossEntropyLoss()

# note that we do not compute log softmax (inside CE)
loss = loss_function(preds, empirical)

loss.backward() # and the rest...
```

4 Cross-Entropy in Pytorch (2)

After training, prediction:

```
# let us pretend the following tensor contains
# the input for 3 examples
inputs = ...

# we obtain the 5 scores for all 3 examples
predictionss = net(inputs)

#apply the argmax for each line:
outputs = torch.argmax(predictionss, dim=1)
```

no *cross-entropy*, no *softmax*: why?

5 From MLE to Contrastive Learning

5 Constrastive Sampling (Mikolov et al.) (1/7)

- With $|V| = 50,000$, softmax probability requires $50,000$ dot products per position ...
- Softmax inefficient $\longrightarrow$ alternative method to train word vectors.

From *word* probability to *class* probability

Define probability $p(D = d|w', w)$ for 2 classes d:

- $d = 1$ if w' is a word that belongs to a possible context for w
- $d = 0$ otherwise
- of course we have: $p(D = 0|w', w) = 1 - p(D = 1|w', w)$

How to model binomial distribution? (2 classes)

- softmax possible for 2 classes but... amounts to logistic (sigmoid) $\sigma(x) = \frac{1}{1+\exp(-x)}$
- combine sigmoid and (unnormalized) similarity:

$$p(D = 1|w', w) = \frac{1}{1 + \exp(-u_{w'} \cdot v_w)}$$

- 🤔 What is p when w', w are similar ? dissimilar?

5 Constrastive Sampling (Mikolov et al.) (2/7)

What we want to do

- train vector for tokens from a corpus organised in sentences...
- ... by using a loss function that involves a *similarity* beetween word vectors
- efficient: we want to avoid iterating through the lexicon

Maximum Likelihood Estimation

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{d\sim p(.)}[\log p_\theta(D = d)]$$

we want to correctly predict class (d 0 or 1) for all pairs (w', w)

- using our predictor $p_\theta(D = d|W_n = w', W_c = w)$
- w is a centre word, w' is a context candidate (neighbour)

5 Constrastive Sampling (Mikolov et al.) (3/7)

Maximum Likelihood Estimation: Decompose on positive/negative class expectations:

Make centre word appear in the objective

$$max_\theta \ \mathbb{E}_{d\sim p(.)}[\log p_\theta(D=d)]$$

$$= \max_\theta \ \mathbb{E}_{d\sim p(.)}[\log \mathbb{E}_{w\sim p_E(w)} \ p_\theta(D=d|W_c=w)] \quad p_E \text{ is the empirical distribution (corpus)}$$

$$\geq \max_\theta \ \mathbb{E}_{d\sim p(.)}\mathbb{E}_{w\sim p_E(w)}[\log p_\theta(D=d|W_c=w)] \quad \text{by Jensen's inequality}$$

$$= \max_\theta \mathbb{E}_{w\sim p_E(w)}\mathbb{E}_{d\sim p(.)}[\log p_\theta(D=d|W_c=w)] \quad \text{rearrange independent expectations}$$

$$= \max_\theta \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{d\sim p(.)}[\log p_\theta(D=d|W_c=w_t)] \quad \text{where } w_t \text{ the } t^{th} \text{ word in corpus}$$

5 Constrastive Sampling (Mikolov et al.) (4/7)

Maximum Likelihood Estimation: Decompose on positive/negative class expectations:

Make neighbour candidate appear in the objective

$$\max_\theta \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{d\sim p(.)}[\log p_\theta(D=d|W_c=w_t)]$$

$$= \max_\theta \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{d\sim p(.)}\big[\log \mathbb{E}_{w'\sim p(\cdot|d,w_t)}[p_\theta(D=d|W_n=w', W_c=w_t)]\big]$$

$$\geq \max_\theta \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{d\sim p(.)}\big[\mathbb{E}_{w'\sim p(\cdot|d,w_t)}[\log p_\theta(D=d|W_n=w', W_c=w_t)]\big] \quad \text{Jensen}$$

Final Objective

$$\theta^* = \operatorname*{argmax}_\theta \frac{1}{T}\sum_{t=1}^{T}\sum_{d=0,1} p(D=d)\mathbb{E}_{w'\sim p(\cdot|D=d,w_t)}[\log p_\theta(D=d|W_n=w', W_c=w_t)]$$

Now make some simplifications and assumptions

5 Constrastive Sampling (Mikolov et al.) (5/7)

Add the following assumptions:

Negative neighbours more probable

- $\exists \kappa = 1, 2, \ldots, N$ such that $P(D = 0) = \kappa \times P(D = 1)$
- in other words, the negative class is κ times more likely than the positive class
- $p(D = 1) = \frac{1}{\kappa+1}$ and $p(D = 0) = \frac{\kappa}{\kappa+1}$, `word2vec` sets κ to 5

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \frac{1}{6T} \sum_{t=1}^{T} \{\mathbb{E}_{w' \sim p(\cdot|D=1,w_t)}[\log p_\theta(D = 1|W_n = w', W_c = w_t)] + 5\mathbb{E}_{w' \sim p(\cdot|D=0,w_t)}[\log p_\theta(D = 0|W_n = w', W_c = w_t)]\}$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{t=1}^{T} \{\mathbb{E}_{w' \sim p(\cdot|D=1,w_t)}[\log p_\theta(D = 1|W_n = w', W_c = w_t)] + 5\mathbb{E}_{w' \sim p(\cdot|D=0,w_t)}[\log p_\theta(D = 0|W_n = w', W_c = w_t)]\}$$

Can also be interpreted as:

- negative class is κ times more important/difficult to learn than positive class

5 Constrastive Sampling (Mikolov et al.) (6/7)

We also need to define distributions for w': $P^+ = p(w'|D = 1, w_t)$ and $P^- = p(w'|D = 0, w_t)$

For the positive class

draw randomly a token from the window around position t

$$P_t^+(w) = p(w|D = 1, w_t) \begin{cases} \frac{1}{|O_t|} & \text{if } w \in O_t, \\ 0 & \text{otherwise} \end{cases}$$

For the negative class

Use the frequency of a word in training set as probability:

$$P_t^-(w) = p(w|D = 0, w_t) = \frac{\#w}{\sum_{w'} \#w'} = f(w)$$

In Mikolov's implementation, use a *flattened* version:

$$P_t^-(w) = \frac{\#w^{\frac{3}{4}}}{\sum_{w'} \#w'^{\frac{3}{4}}}$$

5 Constrastive Sampling (Mikolov et al.) (7/7)

Inefficiency of Expectations: Monte-Carlo Approximation

P^+, P^- have been defined such as they are simple to sample from. Objective becomes:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{t=1}^{T} \log p_\theta(D = 1|W_n = w^+, W_c = w_t)] + \sum_{k=1}^{\kappa} \log p_\theta(D = 0|W_n = w^k, W_c = w_t)]$$

where:

- w^+ is sampled from P^+
- w^k sampled from P^-

Final simplification

$\log p_\theta$ has a simple form:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{t=1}^{T} \log \left(1 + \exp(-\langle u_{w^+}, v_{w_t} \rangle)\right) + \sum_{k=1}^{\kappa} \log \left(1 + \exp(\langle u_{w^k}, v_{w_t} \rangle)\right)$$

5 Constrastive Sampling (Mikolov et al.) (subsampling)

A small trick to get a better model:

Subsampling

Do not take into account all the words in the training set:

- remove all words that appear less than n times (set $n = 2, 3, 4, 5 ...$)
- remove at random very frequent words in contexts. For each token w in the lexicon, eliminate w from O_t with probability:

$$P_d(w) = ReLU(1 - \sqrt{\frac{t}{f(w)}})$$

with $t = 10^{-5}$

- in words, words with frequency equal to or above t are always discarded.
- beware if we remove a word from context O_t, we still have to keep the size of the window ($2m$): need to extend window boundaries

5 Learning Example (1/3)

Recall Corpus

... Sgt. Pepper ... Lucy ...
i i+1 j

- *Pepper* is in the context of *Sgt.* → (*Sgt.*, *Pepper*) is in the training corpus
- *Lucy* is not in the context of *Sgt.*
- We suppose (for this example!) that
  1. we only have 3 words in our vocabulary and word vectors have one dimension only
  2. for each word w $u_w = v_w$ (actually this is the case in some implementation)
  3. $\kappa = 1$ (as many observed samples as unobserved samples)

Before/After Word2Vec Update: Objective

- Before $v_{Sgt.} = 0.3 = S$, $v_{Pepper} = 0.1 = P$, $v_{Lucy} = -0.4 = L$
- new example: (*Sgt.*, *Pepper*) objective is:

$$Q = \log\left(1 + \exp(-\langle P, S\rangle)\right) + \log\left(1 + \exp(\langle L, S\rangle)\right)$$

Before/After Word2Vec Update: Gradients

Compute gradients wrt to each vector (here simple differentials...)

$$\begin{aligned}
\frac{\partial O}{\partial S} &= -P\frac{\exp(-\langle P,S\rangle)}{1+\exp(-\langle P,S\rangle)} + L\frac{\exp(\langle L,S\rangle)}{1+\exp(\langle L,S\rangle)} \\
&= -P\frac{1}{\exp(\langle P,S\rangle)+1} + L\frac{1}{\exp(-\langle L,S\rangle)+1} \\
&= -Pp(D=0|Pepper,Sgt.) + Lp(D=1|Lucy,Sgt.) \\
&= -P(1-p(D=1|Pepper,Sgt.)) + Lp(D=1|Lucy,Sgt.) \\
&= -0.1(1-0.51) - 0.4\times 0.47 \\
&= -0.24
\end{aligned}$$

$$\begin{aligned}
\frac{\partial O}{\partial P} &= -S\frac{\exp(-\langle P,S\rangle)}{1+\exp(-\langle P,S\rangle)} \\
&= -S(1-p(D=1|Peppers,Sgt.) \\
&= -0.3(1-0.51) \\
&= -0.3(1-0.51) \\
&= -0.15
\end{aligned}$$

$$\begin{aligned}
\frac{\partial O}{\partial L} &= S\frac{\exp(\langle L,S\rangle)}{1+\exp(\langle L,S\rangle)} \\
&= S\times p(D=1|Lucy,Sgt.) \\
&= -0.3\times 0.47 = 0.14
\end{aligned}$$

5 Learning Example (3/3)

Before/After Word2Vec Update: SGD Updates

Apply SGD update ($\alpha = 0.1$)

$$
\begin{aligned}
S &= S - \alpha \frac{\partial O}{\partial S} \\
&= 0.3 - 0.1(-0.24) \\
&= 0.3 + 0.02 \\
&= 0.32
\end{aligned}
\qquad
\begin{aligned}
P &= P - \alpha \frac{\partial O}{\partial P} \\
&= 0.1 - 0.1(-0.15) \\
&= 0.1 + 0.01 \\
&= 0.11
\end{aligned}
\qquad
\begin{aligned}
L &= L - \alpha \frac{\partial O}{\partial L} \\
&= -0.4 - 0.1(0.14) \\
&= -0.4 - 0.01 \\
&= -0.41
\end{aligned}
$$

Lucy and Sgt. are moved further apart, Pepper and Sgt. moved in the same direction.

6
The End

6 Conclusion

Word Vectors

- the basis of modern NLP, represent basic units of meaning

Word2vec

- Simple Idea, implements Firth's principle in an efficient way
- Geometric Meaning (e.g. analogy as vector translation)
- Probabilistic modelling : MLE/Contrastive
- limitations: no information to specialize meaning (e.g. river *bank* versus financial *bank* should have different representations)

Extensions

- *Dynamic* Vectors: Word Vectors + Functions (NNs) to adapt Word vectors to each context (EIMO, BERT…)
- Modern word vectors are built from subwords vectors that are pooled (max/mean/avg) (BPE)

We review these extensions in sessions 4–6

7 Bibliography

[LG14] Omer Levy and Yoav Goldberg. "Neural Word Embedding as Implicit Matrix Factorization." In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/feab05aa91085b7a8012516bc3533958-Paper.pdf.

[Mik+18] Tomas Mikolov et al. "Advances in Pre-Training Distributed Word Representations." In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Ed. by Nicoletta Calzolari et al. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. URL: https://aclanthology.org/L18-1008.