Human Language Technologies

Federico Matteoni

A.A. 2021/22

Index

0.1 Introduction

Prof. Giuseppe Attardi
Prerequisites are: proficiency in Python, basic probability and statistics, calculus and linear algebra and notions of machine learning.

What will we learn Understanding of and ability to use effective modern methods for **Natural Language Processing**. From traditional methods to current advanced ones like RNN, Attentions...
Understanding the difficulties in dealing with NL and the capabilities of current technologies, with experience with **modern tools** and aiming towards the ability to build systems for some major NLP tasks: word similarities, parsing, machine translation, entity recognition, question answering, sentiment analysis, dialogue system...

Books Speech and Language Processing (Jurafsky, Martin), Deep Learning (Goodfellow, Bengio, Courville), Natural Language Processing in Python (Bird, Klein, Loper)

Exam Project (alone or team of 2-3 people) with the aim to experiment with techniques in a realistic setting using data from competitions (Kaggle, CoNLL, SemEval, Evalita...). The topic will be proposed by the team or chosen from a list of suggestions.

Experimental Approach

1. Formulate hypothesis
2. Implement technique
3. Train and test
4. Apply evaluation metric
5. If not improved:

   Perform error analysis

   Revise hypothesis
6. Repeat!

Motivations Language is the most distinctive feature of human intelligence, **it shapes thought**. Emulating language capabilities is a scientific challenge, a **keystone for intelligent systems** (see: Turing test)

Structured vs unstructured data The largest amount of information shared with each other is unstructured, primarily text. Information is mostly communicated by e-mails, reports, articles, conversations, media... and attempts to turn text to structured (HTML) or microformat only scratched the surface.
Problems: requires universal agreed **ontologies** and additional effort. Entity linking attempts to provide a bridge.

0.2 State of the Art

Early History During 1950s, up until AI winter.

Resurgence in the 1990s Thanks to statistical methods, novelty, to study language. Challenges arise: NIST, Netflix, DARPA Grand Challenge...
During 2010s: deep learning, neural machine translation...

Statistical Machine Learning Supervised training with **annotated** documents.
The paradigm is composed of the following:

Training set $\{x_i, y_y\}$

Representation: choose a set of features to represent data $x \mapsto \phi(x) \in R^D$

Model: choose an hypothesis function to compute $f(x) = F_\Theta(\phi(x))$

Evaluation: define the cost function on error with respect to examples $J(\Theta) = \sum_i (f(x_i) - y_i)^2$

Optimization: find parameters Θ that minimize $J(\Theta)$

It's a generic method, applicable to any problem.

Traditional Supervised Learning Approach Freed us from devising algorithms and rules, requiring the creation of annotated training sets and imposing the tyranny of feature engineering.
Standard approach for each new problem:

Gather as much labeled data as one can

Throw a bunch of models at it

Pick the best

Spend hours hand engineering some features or doing feature selection/dimensionality reduction

Rinse and repeat

Technological Breakthroughs Improved ML techniques but also large annotated datasets and more computing power, provided by GPUs and dedicated ML processors (like the TPU by Google).
ML exploits parallelism: stochastic gradient descent can be parallelized (asynchronous stochastic gradient descent). No need to protect shared memory access, and low (half, single) precision is enough.

Deep Learning Approach Was a big breakthrough.

Design a model architecture

Define a loss function

Run the network letting the parameters and the data representations **self-organize** as to minimize the loss

End-to-end learning: no intermediate stages nor representation

Feature representation Use a vector with each entry representing a feature of the domain element
Deep Learning represents data as vectors. Images are vectors (matrices), but words? **Word Embeddings**: transform a word into a vector of hundreds of dimensions capturing many subtle aspects of its meaning. Computed by the means of **language model**.
From a discrete to distributed representation. Words meaning are dense vectors of weights in a high dimensional space, with algebraic properties.
Background: philosophy, linguistics and statistics ML (feature vectors).

Language Model Statistical model which tells the probability that a word comes after a given word in a sentence.

Dealing with Sentences A sentence is a sequence of words: build a representation of a sequence from those of its words (compositional hypothesis). Sequence to sequence models.
Is there more structure in a sentence than a sequence of words? In many cases, tools forgets information when translating sentences into sequences of words, discarding much of the structure.

0.3 Language Modeling

Probabilistic Language Model The goal is to assign a probability to a sentence.

Machine Translation: P(high winds tonight) $>$ P(large winds tonight)

Spell Correction: P(about fifteen minutes from) $>$ P(about fifteen minuets from)

Speech Recognition: P(I saw a van) $>$ P(eye saw a van)

Language Identification: s from unknown language (italian or english) and Lita, Leng language models for italian and english $\Rightarrow$ Lita$(s) > L$eng(s)

Summarization, question answering...

We want to compute

$P(W) = P(w_1, w_2, \ldots, w_n)$ the probability of a sequence

$P(w_4 \mid w_1, w_2, w_3, w_4)$ the probability of a word given some previous words

The model that computes that is called the **language model**

Markov Model and N-Grams Simplify the assumption: the probability of a word given all the previous is the same of the probability of that word given just few (one, two...) previous words. So $P(w_i \mid w_{i-}, \ldots, w_1) = P(w_i \mid w_{i-1})$ (First order Markov chain).
With a N**-gram**: $P(w_n \mid w_1^{n-1}) \simeq P(w_n \mid w_{n-N+1}^{n-1})$
In general it's insufficient: language has **long distance dependencies**, but we can often get away with N-gram models. For example:

"The **man** next to the large oak tree near the grocery store on the corner **is** tall."

"The **men** next to the large oak tree near the grocery store on the corner **are** tall."

Or even semantic dependencies:

"The **bird** next to the large oak tree near the grocery store on the corner **flies** rapidly."

"The **man** next to the large oak tree near the grocery store on the corner **talks** rapidly."

So more complex models are needed to handle such dependencies.

Maximum likelihood estimate

$$P(w_n \mid w_{n-N+1}^{n-1}) = \frac{\text{count}(w_{n-N+1}^{n-1}, w_n)}{\text{count}(w_{n-N+1}^{n-1})}$$

Maximum because it's the one that maximize $P(\text{Training set} \mid \text{Model})$

Shannon Visualization Method Generate random sentences:

Choose a random bigram $(\langle s \rangle, w)$ according to its probability

Choose a random bigram (w, x) according to its probability

Repeat until we pick $\langle /s \rangle$

Shannon Game Generate random sentences by selecting the words according to the probabilities of the language models.

```
def gentext(cpd, word, length=20):
  print(word, end = ' ')
  for i in range(length):
    word = cpd[word].generate()
    print(word, end = ' ')
  print("")
```

Perils of Overfitting N-grams only word well for word prediction if the test corpus looks like the training corpus. In real life, this is often not the case.
We need to train robust models able to adapt.

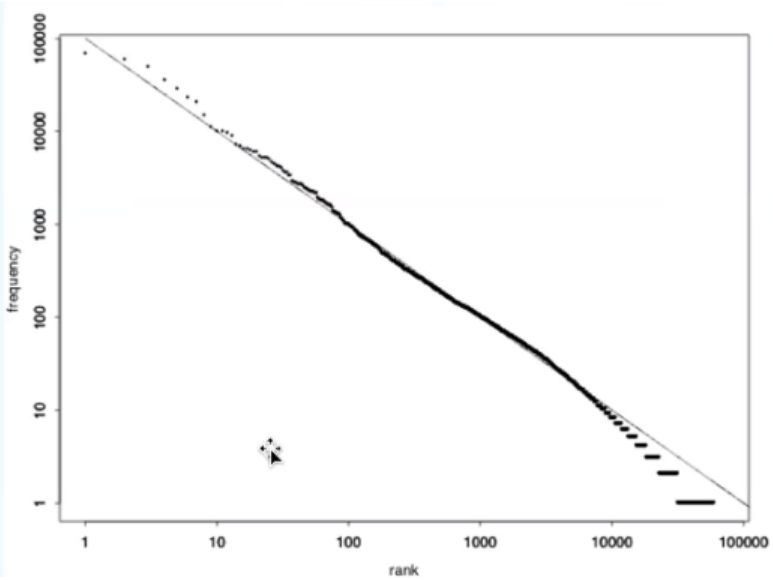
Smoothing Many rare (but not impossible) combinations of word sequences never occur in training, so MLE incorrectly assigns zero to many parameters (sparse data).
If a new combination occurs during testing, it is given a probability of zero and the entire sequence gets a probability of zero (infinite perplexity).
Parameters are **smoothed**/regularized to reassign some probability mass to unseen events (meaning removing probability from seen ones in order to maintain the joint distribution that sums to 1)

Zipf's Law A small number of events occur with high frequency, while a large number of events occur with small frequency. You can quickly collect statistics on the high frequency events, but you may have to wait an arbitrarily long time to get valid statistics on low frequency events.
The result is that our estimates are sparse. No counts at all for the vast bulk of things we want to estimate. Some of the zeroes in the table are really zeroes, but others are simply low frequency events you haven't seen yet: after all, anything can happen. How to address? By estimating the likelihood of unseen N-grams.

Word	Freq. (f)	Rank (r)	$f \cdot r$
the	3332	1	3332
and	2972	2	5944
a	1775	3	5235
he	877	10	8770
but	410	20	8400
be	294	30	8820
there	222	40	8880
one	172	50	8600
about	158	60	9480
more	138	70	9660
never	124	80	9920
Oh	116	90	10440
two	104	100	10400

Word	Freq. (f)	Rank (r)	$f \cdot r$
turned	51	200	10200
you'll	30	300	9000
name	21	400	8400
comes	16	500	8000
group	13	600	7800
lead	11	700	7700
friends	10	800	8000
begin	9	900	8100
family	8	1000	8000
brushed	4	2000	8000
sins	2	3000	6000
Could	2	4000	8000
Applausive	1	8000	8000

Result: f is proportional to $\frac{1}{r}$, meaning that there is a constant $k \mid f \cdot r = k$



0.3.1 Evaluation and Perplexity

Evaluation Train parameters of our model on a training set. For the evaluation, we apply the model on new data and look at the model's performance: **test set**. We need a metric which tells us how well the model is performing: one of such is **perplexity**.

Extrinsic Evaluation Evaluating N-gram models.

Put model A in a task (language identification, speech recognizer, machine translation system...)

Run the task, get an accuracy for A

Put model B in the task, get accuracy for B

Compare the accuracies

Language Identification Task Build a model for each language and compute probability that text is of such language.

$$lang = arg\max_{l} P_l(text)$$

Difficulty of Extrinsic Evaluation Extrinsic evaluation is time-consuming, can take days to run an experiment. Recent benchmarks like GLUE have become popular due to the effectiveness.
Intrinsic evaluation us an approximation called **perplexity**: it's a poor approximation unless the test data looks just like the training data.

Perplexity The intuition is the notion of surprise: how surprised is the language model when it sees the test set? Where surprise is a measure of "I didn't see that coming". The more surprised is, the lower the probability it assigned to the test set, and the higher the probability the less surprised it is.
So perplexity measures how well a model "fits" the test data. It uses the probability that the model assigns to the test corpus and normalizes for the number of words in the test corpus, taking the inverse.

$$PP(w) = \sqrt[N]{\frac{1}{P(w_1 \dots w_N)}}$$

Measures the weighted average branching factor in predicting the next word (lower is better).
In the chain rule

$$PP(w) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_1 \dots w_{i-1})}}$$

And for bigrams

$$PP(w) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_{i-1})}}$$

Lower perplexity $\Rightarrow$ better model

0.4 Representation of Words

Word Meaning Definition of **meaning**:

> Idea represented by a word, phrase...
>
> Idea that a person wants to express by using words, signs...
>
> Idea expressed in a work of writing, art...

Signifier (symbol) $\Leftrightarrow$ **Signified** (idea or concept) = **denotation**

Linguistic Solution Dictionary or lexical resources provide word definitions as well as synonyms, hypernums, antonyms... example: **Wordnet**.

Problems with lexical resources

> Imperfect, sometimes there are errors
>
> Missing new meanings of words, hard to keep up-to-date
>
> Subjective
>
> Require human labor to create and adapt
>
> Hard to compute accurate word similarity

Vector Space Model The **VSM** is a representation for text used in information retrieval. A document is represented by a vector in a n-dimensional space

$$v(d_1) = [t_1, \dots, t_n]$$

Each dimension corresponds to a separate term.
The VSM considers words as discrete symbols.

One Hot Representation A word occurrence is represented by one-hot vectors, with the vector dimension being the number of words in a vocabulary and 1 in correspondence of the position of the represented word. Useful for representing a document by the sum of the word occurrence vector, and document similarity given by **cosine distance** (angle between vectors).

tf*idf Measure Term frequency: frequency of the word.
Inverse document frequency:

Classical VSM Vector is all zeroes with idf_t instead of 1. The vector of a document is the sum of the vectors of its terms occurrence vectors.
Doesn't capture similarity between terms.

Problems with Discrete Symbols All vectors are orthogonal, with no notion of similarity. Search engines try to address the issue using WordNet synonyms but it's better to encode the similarity in the vectors themselves.

Intuition Model the meaning of a word by "embedding" it in a vector space. The meaning of a word is a vectors of numbers (vector models are also called **embeddings**), so a **dense vector space**. Constrast: word meaning is represented in many computational

Word Vectors/Embeddings Build dense vector for each word chosen so that it is similar to vectors of words that appear in similar contexts.
Four kinds:

Sparse Vector Representations

1. Mutual-information weighted word co-occurence matrices

Dense Vector Representations

2.
3.
4.

Distributional Hypothesis A word's meaning is given by the words that frequently appear close-by. Old idea but successful just with modern statistical NLP.
When a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window). Use of the many contexts of w to build a representation of w.

Word Context Matrix Is a $|V| \times |V|$ matrix X that counts the frequencies of co-occurrence of words in a collection of contexts (i.e. text spans of a given length).

Co-Occurrence Matrix Words $\equiv$ Context words. Rows of X capture similarity yet X is still high dimensional and sparse. One row per word with counts of occurrencies of any other word.
We can compute the distance vectors between words, but neighboring words are not semantically related.

0.4.1 Word Embeddings

Dense Representations Project word vectors $v(t)$ into a low dimensional space R^k with $k << |V|$ of continuous space word representations (a.k.a. **embeddigns**)

$$Embed : R^{|V|} \to R^k$$

$$Embed(v(t)) = e(t)$$

Desired properties: remaining dimensions

Collobert Build embeddings and estimate whether the word is in the proper context using a neural network. Positive examples from text, and negative examples made replacing center word with random one. The loss for the training is

$$Loss(\Theta) = \sum_{x \in X} \sum_{w \in W} \max(0, 1 - f_\Theta(x) + f_\Theta(x^{(w)}))$$

with $x^{(w)}$ obtained by replacing the central word of x with a random word.

Word2Vec Framework for learning words, much faster to train. Idea:

Collect a large corpus of text

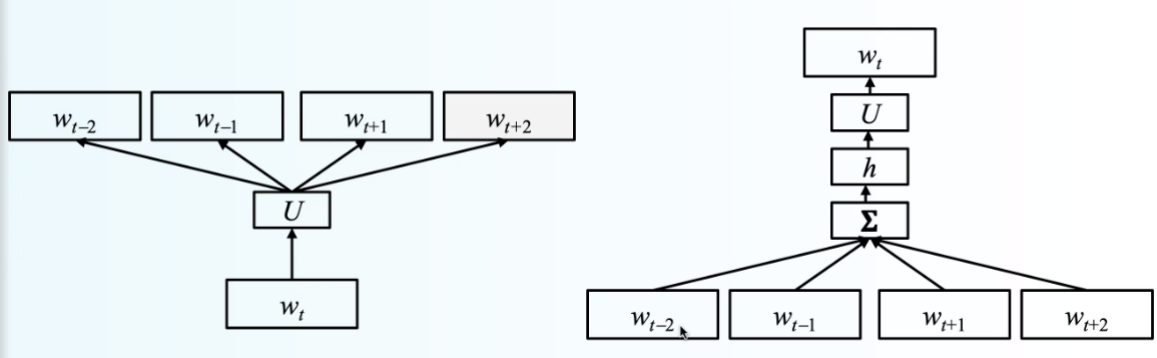
Every word in a fixed vocabulary is represented by a vector

Go through each position t in the text, which has a center word c and context ("outside") words o

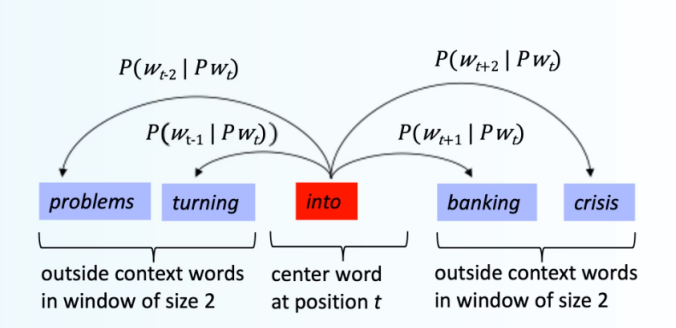
Use the similarity of the word vectors for c and o to calculate the probability of o given c

Keep adjusting word vectors to maximize this probability

Two kinds:



Skip-gram: predict context words within window of size m given the center word w_t



CBoW: predict center word w_t given context words within window of size m

Embeddings are a by-product of the word prediction task. Even though it's a prediction task, the network can be trained on any text (no need for human-labeled data!).
Usual context size is 5 words before and after. Features can be multi-word expressions. Longer windows can capture more semantics and less syntax.
A typical size for h is 200-300.

Skip-Gram

Inputs are one-hot representation of the word

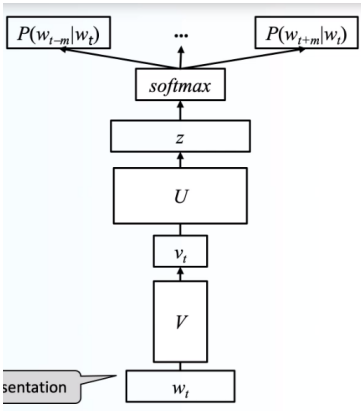
$w \in R^{|Vocabulary|}$ are high-dimensional

$v \in R^d$ is low dimensional: size of the embedding space d

$V \in R^{|Voc| \times d}$ input word matrix

row t of V is the input vector, representation for **center** word w_t

$U \in R^{d \times |Voc|}$ output matrix



column o of U is the output vector, representation for **context** word w_o

$v_t = w_t V$ embedding of word w_t
$z = v_t U$ z_i is the similarity of w_t with w_i
Softmax converts z to a probability distribution p_i

$$p_i = \frac{e^{z_i}}{\sum_{j \in V} e^{z_j}}$$

Procedure

Lookup the embedded word vector for the center word $v_c = V[c] \in R^n$

Generate score vector $z = v_c U$

Turn the score vector into probabilities $\hat{y} = \text{softmax}(z)$

$\hat{y}_{c-m}, \ldots$ Those are the estimates
So the training iterates through the whole corpus predicting sorrounding words given the center word.

Objective Function For each position $t = 1, \ldots, T$ predict context words within a windows of fixed size m given each center word w_t

$$\text{Likelihood} = L(\Theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m,\ j \neq 0} P(w_{t+j} \mid w_t, \Theta)$$

While the objective function $J(\Theta)$ is the average negative log likelihood

$$J(\Theta) = -\frac{1}{T} \log L(\Theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m,\ j \neq 0} \log(P(w_{t+j} \mid w_t, \Theta) =$$

To compute $P(o \mid c, \Theta)$ we use two vectors per word w: v_w when w center word and u_w when w context word. For a center word c and a context word o:

$$P(o \mid c) = \frac{e^{u_o v_c}}{\sum_{w \in V} e^{u_w v_c}}$$

With the dot product $u \cdot v = \sum_i u_i v_i$: larger product $\Rightarrow$ larger probability. Normalize over entire vocabulary to give probability distribution.
$J(\Theta)$ is a function of all windows in the corpus, potentially billions: too expensive to compute. The solution is the stochastic gradient descent, sampling windows randomly and update after each one.

Softmax

Soft because still assign some probability to smaller x_i

Max because amplifies the probability of largest x_i

Can we really capture the concept represented by a word? Philosophical debate.

Negative Sampling $\log \sum_{j \in F} e^{u_j}$ has lots of terms, costly to compute. The solution is to compute it only on a small sample of negative samples, i.e. $\log \sum_{j \in E} e^{u_j}$ where words E are a few (e.g. 5)

CBoW Continuous Bag of Words
Mirror of the skip-gram, where context words are used to predict target words.
h is computed from the average of the embeddings of the input context, z_i is the similarity of h with the words embedding of w_i from U.

Which Embeddings V and U both define embeddigns, which to use? Usually just V. Sometimes average pairs of vectors from V and U into a single one or append one embedding vector after the other, doubling the length.

GloVe Global Vectors for Word Representation. Insight: the ratio of conditional probabilities may capture meaning.

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \dots$$

fastText Similar to CBoW, word embeddings averaged to obtain good sentence representation. Pretrained models.

Co-Occurrence Counts

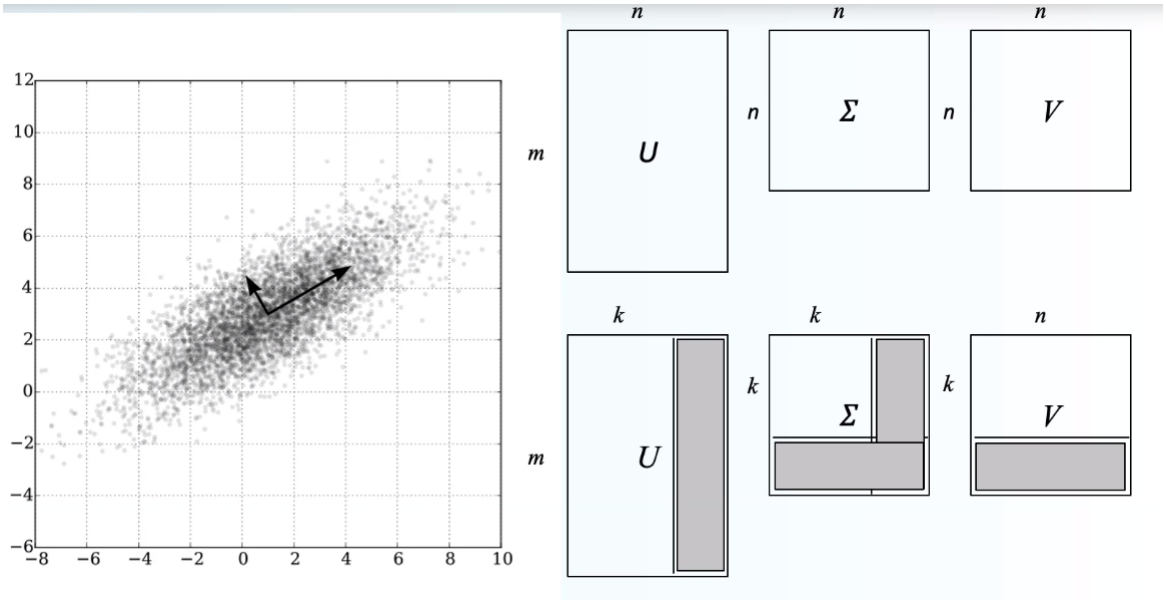
$$P(w_t, w_{t-i}, \dots, w_{t-1}) = \frac{P(w_t, w_{t-i}, \dots, w_{t-1})}{P(w_{t-i}, \dots, w_{t-1})}$$

It's a big matrix, of $|V| \times |V| \simeq 100\text{k} \times 100\text{k} \Rightarrow$ dimensionality reduction: principal component analysis, Hellinger PCA, SVD... trying to reduce to size to 100k×50, 100k×100 or something similar, assigning each word a vector of 50, 100 or similar features.

Weighting Weight the counts using corpus-level statistics to reflect co-occurrence significance: **Pointwise Mutual Information**

$$PMI(w_t, w_{t-i}, \dots, w_{t-1}) = \frac{P(w_t, w_{t-i}, \dots, w_{t-1})}{\log P(w_t) P(w_{t-i}, \dots, w_{t-1})} = \log \frac{\#(w_t, w_{t-i}, \dots, w_{t-1}) \cdot |V|}{\#(w_{t-i}, \dots, w_{t-1}) \#(w_t)}$$

Skip-gram model implicitly factorizes a shifted PMI matrix.
Idea of Singular Value Decomposition:



Which One? No clear winner. Parameters play a relevant role in the outcome of each method. Both SVD and SGNS performed well on most tasks, never underperforming significantly.
SGNS is suggested to be a good baseline: faster to compute and performs well.

Parallel word2vec How to synchronize access to V and U, given multicore CPU to run SGD in parallel? No synchronization is good, because computation is stochastic hence it is approximate anyhow. Parameters are huge: low likelihood of concurrent access to the same memory cell. The effect is a very fast training.

Computing embeddings The training cost of word2vec is linear in the size of the input. The training algorithm works well in parallel, given sparsity of words in contexts and use of negative sampling. It can be halted and restarted at anytime.

Gensim Cython

Fang Uses PyTorch

0.4.2 Evaluation

Polysemy Word vector is a linear combination of its word senses.

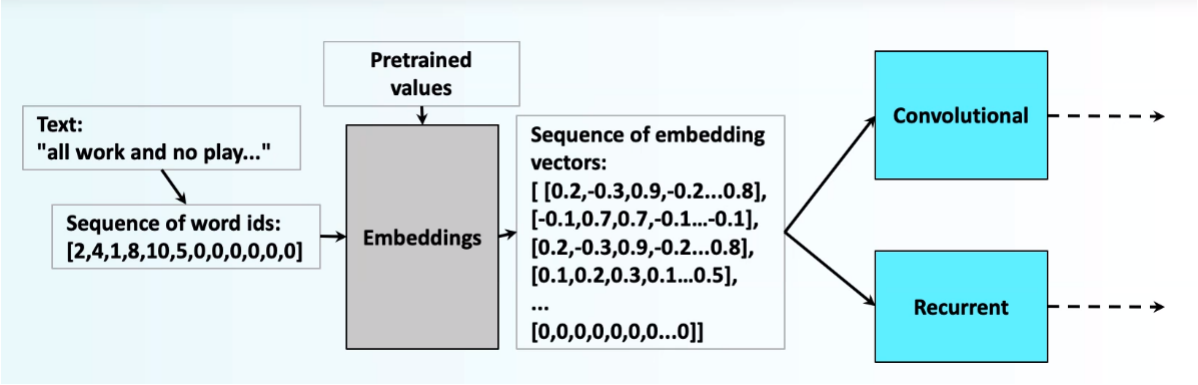
$$v_{pike} = \alpha_1 v_{pike_1} + \alpha_2 v_{pike_3} + \alpha_3 v_3$$

with $\alpha_i = \frac{f_i}{f_1+f_2+f_3}$ for the frequencies f_i.
It's intrinsic evaluation.

Extrinsic Vector Evaluation The proof of the pudding is in the eating. Test on a task, e.g. NER (Named Entity Recognition)

Embeddings in Neural Networks

An embedding layer is often used as fist layer in a neural network for processing text.
It consists of a matrix W of size $|V| \times d$ where d is the size of the embedding space. W maps words to dense representations.
It is initialized either with random weights...



Limits of Word Embeddings

Polysemous words

Limited to words (neither multi words nor phrases)

Represent similarity: antinomies often appear similar.
Not good for sentiment analysis or polysemous words. Example:

The movie was **exciting**

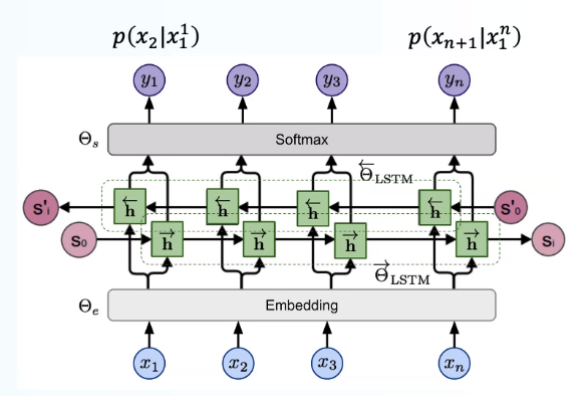
The movie was **boring**

Word Senses and Ambiguity

Sentiment Specific

Context Aware Word Embeddings

ELMo Embeddings from Language Model



Given a sequence of n tokens $(x_1, \ldots, x_n)$

OpenAI GPT-2

BERT Semi-supervised training on large amounts of text, or supervised training on a specific task with a labeled dataset.

0.5 Text Classification

For example: positive/negative review identification, author identification, spam identification, subject identification...

Definition The classifier $f : D \to C$ with

$d \in D$ input document

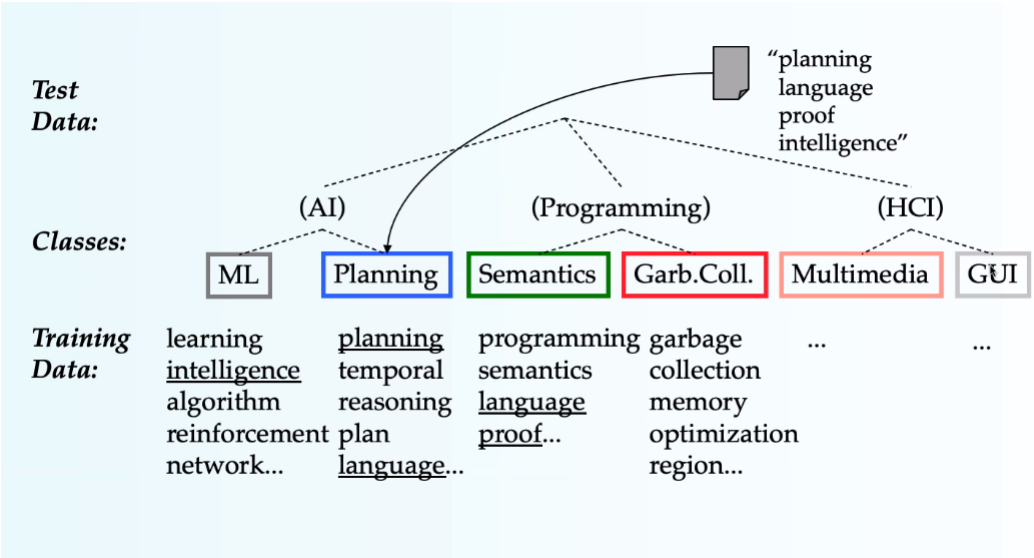
$C = \{c_1, \ldots, c_K\}$ set of classes

$c \in C$ predicted class as output

The learner has

Input: a set of N hand-labeled documents $T = \{(d_1, c_1), \ldots, (d_N, c_N)\}$

Output: a learned classifier $f : D \to C$



Hand-Coded Rules Often very high accuracy, but building and maintaining these rules is expensive. For example: assign category if a document contains a given boolean combination of words (e.g. a blacklist of words for spam classification).

Supervised Machine Learning Input

A document $d \in D$

A fixed set of classes $C = \{c_1, \ldots, c_K\}$

A training set of N hand-labeled documents $T = \{(d_1, c_1), \ldots, (d_N, c_N)\}$

As output

A learned classifier $\gamma : D \to C$

0.5.1 Naive Bayes Intuition