Natural Language Understanding

Doruk Çetin

September 13, 2019

Contents

1 Basics

- Morphology: studies the structure of words.
- Morpheme: the smallest grammatical unit in a language.
- Inflectional morphology: the study of the processes (such as affixation and vowel change) that distinguish the forms of words in certain grammatical categories.
- Derivational morpheme: an affix that's added to a word to create a new word or a new form of a word.
- Corpus: a computer-readable collection of text or speech.
- Lemma: a set of lexical forms having the same stem, the same major part-of-speech, and the same word sense.
- Word form: the full inflected or derived form of the word.
- Tokenization: the task of segmenting running text into words.
- Normalization: the task of putting words/tokens in a standard format.
- Lemmatization: the task of determining that two words have the same root, despite their surface differences.
- Stemming: a simpler but cruder method, which mainly consists of chopping off word-final affixes.
- Minimum edit distance between two strings is defined distance as the minimum number of editing operations (operations like insertion, deletion, substitution) needed to transform one string into another. It can be computed by dynamic programming.
- Levenshtein distance between two sequences is the simplest weighting factor in which each of the three operations has a cost of 1
- Word embeddings: latent vector representation of words.

Basic text normalization:

- Segmenting/tokenizing words from running text
- Normalizing word formats
- Segmenting sentences in running text.

The Viterbi algorithm is a probabilistic extension of minimum edit distance. Instead of computing the "minimum edit distance" between two strings, Viterbi computes the "maximum probability alignment" of one string with another.

The Porter algorithm is a simple and efficient way to do stemming, stripping off affixes. It does not have high accuracy but may be useful for some tasks.

2 Language Models

Language Models (LMs): models that assign probabilities to sequences of words. We always represent and compute language model probabilities in log format as log probabilities.

- Use cases (Redundancy): speech recognition, handwriting recognition, spelling correct.

- Use cases (Synthesis): machine translation, text summarization

The assumption that the probability of a word depends only on the previous word is called a **Markov assumption**. Estimating the probabilities by chain rule without Markov assumption is ill-fated as there are too many possible sentences.

k-th order Markov assumption is also called **n-gram model** ($n = k + 1$). In general, n-gram modeling is insufficient as languages have long-term dependencies. n-grams only work well for word prediction if the test corpus looks like the training corpus.

How to deal with huge web-scale n-grams: pruning w.r.t. some threshold, entropy-based pruning, efficient data structures like tries, Huffman coding, probability quantization...

2.1 Perplexity

An intrinsic evaluation metric is one that measures the quality of a model independent of any application.

The **perplexity** of a discrete probability distribution p is defined as

$$2^{H(p)} = 2^{-\sum_x p(x) \log_2 p(x)}$$

The perplexity is the exponentiation of the entropy, which is a more clear cut quantity. In the special case where p models a fair k-sided die (a uniform distribution over k discrete events), its perplexity is k (equal to branching factor). For perplexity over words this means the size of the vocabulary.

The more information the model gives us about the word sequence, the lower the perplexity. Minimizing perplexity is the same as maximizing probability.

Perplexity of a model (as the normalized inverse probability):

$$\begin{aligned} PP(X) = 2^{H(p)} &= 2^{-\sum_{i=1}^{N} p(x_i) \log_2 q(x_i)} \\ &= 2^{-\sum_{i=1}^{N} \frac{1}{N} \log_2 q(x_i)} \\ &= q(x_1)^{-\frac{1}{N}} q(x_2)^{-\frac{1}{N}} \dots q(x_N)^{-\frac{1}{N}} \\ &= \prod_{i=1}^{N} q(x_i)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{q(x_1)q(x_2)\dots q(x_N)}} \end{aligned}$$

where $p(x_i)$ is the real distribution (x_i are the words here) and $q(x_i)$ is our prediction. Here we assumed that all words will have the same probability (1 / # of words) in p.

The difference between the perplexity of a (language) model and the true perplexity of the language is an indication of the quality of the model.

Cross entropy $H(p, \widetilde{p})$, where $\widetilde{p}$ is the language model, is an upper bound on true sequence entropy $H(p)$:

$$\begin{aligned}
H(p, \widetilde{p}) &= \sum_w p(w) \log(\widetilde{p}) \\
&= \sum_w p(w) \log\left(\widetilde{p}(w) \frac{p(w)}{p(w)}\right) \\
&= \sum_w p(w) \log p(w) + \sum_w p(w) \log\left(\frac{\widetilde{p}(w)}{p(w)}\right) \\
&= H(p) + D_{KL}(p \parallel \widetilde{p}) \\
&\geq H(p)
\end{aligned} \tag{1}$$

2.2 Zeros and Smoothing

Smoothing: best effort to estimate **non-zero probabilities**, rather than using raw counts.

These zeros— things that don't ever occur in the training set but do occur in the test set—are a problem for two reasons. First, their presence means we are underestimating the probability of all sorts of words that might occur, which will hurt the performance of any application we want to run on this data.

Second, if the probability of any word in the test set is 0, the entire probability of the test set is 0. By definition, perplexity is based on the inverse probability of the test set. Thus if some words have zero probability, we can't compute perplexity at all, since we can't divide by 0!

Laplace (add-one) smoothing: add one to all probabilities before normalization, e.g. for bigrams:

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}, \quad P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Add-1 estimation is a blunt instrument, not used for n-gram l language modeling in practice but used in other domains (text categorization) where the number of zeroes is not so huge.

add-k smoothing: add a fractional count, instead of adding 1:

$$P_{Add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV}$$

2.3 Backoff and Interpolation

Idea: sometimes it helps to use less context. Condition on less context for contexts you have not learned much about:

- Backoff: use trigram if you have good evidence, otherwise bigram, otherwise unigram (we only "back off" to a lower-order n-gram if we have zero evidence for a higher-order n-gram)
- Interpolation: always mix unigram, bigram, trigram etc. (works better than backoff)

Linear interpolation: set the lambdas by maximizing probability of held-out data

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

Lambdas conditional on context:

$$\hat{P}(w_n \mid w_{n-1}w_{n-2}) = \lambda_1(w_{n-2}^{n-1}) P(w_n \mid w_{n-1}w_{n-2}) + \lambda_2(w_{n-2}^{n-1}) P(w_n \mid w_{n-1}) + \lambda_3(w_{n-2}^{n-1}) P(w_n)$$

Out of vocabulary (OOV) words can occur in open vocabulary tasks. Create an unknown word token ⟨UNK⟩ and change any word that is not in the lexicon to ⟨UNK⟩.

Stupid backoff: it is very simple but works well in large scale. Use MLE if the count is greater than zero, if not backoff to the lower order n-gram with some constant weight (e.g. if trigram occurs use trigram, if not use bigram probability times 0.4). Stupid backoff does not produce probabilities.

A related way to view smoothing is as discounting (lowering) some non-zero counts in order to get the probability mass that will be assigned to the zero counts. Both backoff and interpolation require discounting to create a probability distribution. The sharp change in counts and probabilities may occur if too much probability mass is moved to all the zeros.

2.4 Good-Turing Smoothing

Add-k smoothing, alternative formulation:

$$P_{Add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV} = \frac{c(w_{i-1}, w_i) + m(\frac{1}{V})}{c(w_{i-1}) + m}$$

Unigram prior smoothing (basically interpolation):

$$P_{UnigramPrior}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + mP(w_i)}{c(w_{i-1}) + m}$$

Idea of the **Good-Turing Smoothing** is to use the counts of things we have seen once to help estimate the count of things we have never seen.

Define N_c as the frequency of frequency c (the count of things we have seen c times). Probability mass to be assigned to k-frequent types is

$$P_{GT}^{k} = \frac{(k+1)N_{k+1}}{N}$$

compare with the observed probability $\hat{P}^k = kN_k/N$. Discount factor is $\frac{k^*}{k} = \frac{k+1}{k}\frac{N_{k+1}}{N_k}$.

We cannot always use the $n+1^{st}$ word to do the estimation Simple Good-Turing: use Good-Turing for lower orders and estimate the higher order terms by a best fit power law:

$$\frac{k^*}{k} = \frac{(k+1)k^\alpha}{k(k+1)^\alpha} = \left(\frac{k}{k+1}\right)^{\alpha-1} < 1 \text{ for } N_k \propto k^{-\alpha},\ \alpha > 1$$

Approximate discount “law” is empirically found to be $k^* = k - 0.75$

2.5 Kneser-Ney Smoothing

Absolute discounting formalizes this intuition by subtracting a fixed (absolute) discount d from each count. Basic idea is to free up probability mass from n-gram counts and redistribute it to (n-1)-gram model via interpolation. Example: probability of an unseen bigram $(w', w) \propto p(w) \approx \#(w)/N$

Absolute Discounting Interpolation:

$$P_{AbsoluteDiscounting}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w)$$

Maybe keeping a couple extra values of d for counts 1 and 2. Interpolation weights are for normalizing the probability distribution (for the probability mass we have discounted). We apply this recursively, e.g. smoothing trigrams with bigrams.

Kneser-Ney discounting augments absolute discounting with a more sophisticated way to handle the lower-order unigram distribution. The Kneser-Ney intuition is to base our estimate of $P_{Continutation}$ on the number of different contexts word w has appeared in, that is, the number of bigram types it completes. Every bigram type was a novel continuation the first time it was seen. The unigram is useful exactly when we have not seen the bigram of interest. A frequent word (Francisco) occurring in only one context (San) will have a low continuation probability. Context probability replaces naive unigrams.

$$P_{Continutation}(w) = \frac{\text{distinct bigrams with } w}{\text{distinct bigrams}} = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|}$$

where the numerator counts how many different bigrams does the word create by appearing after another word. The same ratio can be expressed as number of wordtypes preceding w over number of wordtypes preceding all words.

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_{Continuation}(w)$$

Kneser-Ney smoothing recursive formulation:

$$P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1})P_{KN}(w_i \mid w_{i-n+2}^{i-1})$$

$$c_{KN}(\bullet) = \begin{cases} \text{Count}(\bullet) & \text{for the highest order} \\ \text{ContinuationCount}(\bullet) & \text{for the lower orders} \end{cases}$$

where ContinuationCount($\bullet$) is the number of unique single word contexts for $\bullet$.

2.6 Comparison

n-gram model:

- Sparsity problem (Increasing n makes sparsity problems worse. Typically we can't have n bigger than 5.)

- Storage problems (Increasing n or increasing corpus increases model size.)
- Surprisingly grammatical but incoherent (We need to consider more than three words at a time if we want to model language well)

Window-based model:

- No sparsity problem, don't need to store all observed n-grams
- Fixed window is too small, window can be never large enough, enlarging window enlarges W
- Input words are multiplied by completely different weights in W, no symmetry in how the inputs are processed

RNN model:

- Can process any length input, model size doesn't increase for longer input
- Same weights applied on every step, there is symmetry on how inputs are processed
- Computation in step t can (in theory) use information from many steps back
- Recurrent computation is slow
- In practice, difficult to access information from many steps back

3 PoS Tagging

Part-of-speech tagging is the process of assigning a part-of-speech label to each of a sequence of words.

- Open classes: nouns, verb, adjectives, adverbs
- Closed classes: prepositions, particles, determiners, conjunctions, pronouns, auxiliary verbs, numerals
- An important tagset for English is the 45-tag Penn Treebank tagset, which has been used to label many corpora.

Summary:

- Baseline is already 90%: tag every word with its most frequent tag, tag unknown words as nouns
- The change from generative to discriminative model does not by itself result in great improvement. One profits from models for specifying dependence on overlapping features of the observation such as spelling, suffix analysis, etc.
- Higher accuracy of discriminative models comes at the price of much slower training.

3.1 Hidden Markov Models (HMMs)

Hidden Markov Models can be formalized as $HMM = (Q, O, A, B, \Pi)$, where

- $Q = q_1 q_2 \dots q_N$: a set of N states (PoS tags)
- $A = a_{11} \dots a_{ij} \dots a_{NN}$: a transition probability matrix
- $O = o_1 o_2 \dots o_T$: a sequence of T observations (words)

- $B = b_i(o_t)$: a sequence of observation likelihoods, also called emission probabilities
- $\Pi = \pi_1, \pi_2, \dots, \pi_N$: an initial probability distribution (priors)

Two simplifying assumptions:

- Markov assumption: $P(q_i \mid q_1 \dots q_{i-1}) = P(q_i \mid q_{i-1})$
- Output independence: $P(o_i \mid q_1, \dots, q_i \dots, q_T, o_1, \dots o_i, \dots, o_T) = P(o_i \mid q_i)$

Three fundamental problems:

1. Likelihood: Given an HMM and an observation sequence, determine the likelihood
2. Decoding: Given an observation sequence and HMM, discover best hidden state sequence (most probable explanation)
3. Learning: Given an observation sequence and the set of states in the HMM, learn the HMM parameters A and B

Likelihood problem: we cannot compute the total observation likelihood by computing a separate observation likelihood for each hidden state sequence and then summing them. Instead of using such an extremely exponential algorithm, we use an efficient $O(N^2T)$ algorithm called the **forward algorithm**. The forward algorithm computes the observation probability by summing over the probabilities of all possible hidden state paths that could generate the observation sequence, but it does so efficiently by implicitly folding each of these paths into a single forward trellis.

Each cell of the forward algorithm trellis at $\alpha_t(j)$ represents the probability of being in state j after seeing the first t observations, given the automaton λ:

$$\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j \mid \lambda) = \sum_{i=1}^{N} \alpha_{t-1}(i)\alpha_{ij} b_j(o_t)$$

Decoding problem: the most common decoding algorithms for HMMs is the **Viterbi (max-product) algorithm**. Like the forward algorithm, Viterbi is a kind of dynamic programming algorithm that makes uses of a dynamic programming trellis. Note also that the Viterbi algorithm has one component that the forward algorithm doesn't have: backpointers (for the so-called Viterbi backtrace).

The value of each cell is computed by recursively taking the most probable path that could lead us to this cell. Note that we represent the most probable path by taking the maximum over all possible previous state sequences. We compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell:

$$v_t(j) = \max_{q_1, \cdots, q_{t-1}} P(q_1 \dots q_{t-1}, o_1, o_2 \dots o_t, q_t = j \mid \lambda)$$

$$v_t(j) = \max_{i=1}^{n} v_{t-1}(i) a_{ij} b_j(o_t)$$

Learning problem: The standard algorithm for HMM training is the **forward-backward (Baum-Welch) algorithm**, a special case of the Expectation-Maximization algorithm. To understand the algorithm, we need to define a useful probability related to the forward probability and called the backward probability. The backward probability β is the probability of seeing the observations from time $t+1$ to the end, given that we are in state i at time t (and given the automaton *lambda*):

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \dots, o_T \mid q_t = i, \lambda) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$$

For PoS tagging, we don't actually need Baum-Welch or an EM algorithm, because all of your HMM "hidden" states are actually observed (we have all our PoS tags for all our sentences in our dataset). Hence we can use MLE (maximum likelihood estimation) directly. For a comparison, see slide 40 (MLE) vs. slide 41 (EM).

3.2 Inference in Temporal Models

- **Filtering** (state estimation): $[P(X_t|e_{1:t})]$ the task of computing the belief state —the posterior distribution over the most recent state— given all evidence to date, solved by **Forward** algorithm
- **Prediction**: $[P(X_{t+k}|e_{1:t})$ for $k > 0]$ the task of computing the posterior distribution over the future state, given all evidence to date, solved by **Forward** algorithm
- **Smoothing**: $[P(X_k|e_{1:t})$ for $0 \leq k < t]$ the task of computing the posterior distribution over a past state, given all evidence up to the present, solved by **Forward-Backward** algorithm
- **Most likely explanation** (decoding): $[\arg\max_{X_{1:t}} P(X_{1:t}|e_{1:t}]$ the task of finding the sequence of states that is most likely to have generated a given sequence of observations, solved by **Viterbi** algorithm
- **Learning**: the task of learning transition and sensor models, if not yet known, from observations. Learning requires smoothing, rather than filtering.

The posterior distributions computed by smoothing are distributions over single time steps, whereas to find the most likely sequence we must consider joint probabilities over all the time steps. The results can in fact be quite different.

3.3 Information Extraction

- **Information extraction** (IE): the task of turning the unstructured information embedded in texts into structured data, for example for populating a relational database to enable further processing
- **Named entity recognition** (NER): the task of finding each mention of a named entity in the text and label its type. Its standard evaluation is per entity, not per token.
- Metrics like recall and precision behave a bit funnily for IE/NER when there are boundary errors (which are common).
- The **IOB format** (short for inside, outside, beginning) is a common tagging format for tagging tokens in a chunking task in computational linguistics (ex. named-entity

recognition). Another similar format which is widely used is **IOB2 format**, which is the same as the IOB format except that the B- tag is used in the beginning of every chunk (i.e. all chunks start with the B- tag).

- **Co-reference resolution**: the task of finding all expressions that refer to the same entity in a text.
- **Relationship extraction**: the the task of finding and classifying semantic relationships among text entities.
- **Entity linking** (EL): the task of determining the identity of entities mentioned in text.

4 Vector Semantics

Also see related section on both CIL and ML4H notes. Notes on word embeddings are currently collected in a separate file.

Synonymy: equivalence of near-equivalence in meaning, **polysemy**: multitude of meanings

Raw dot-product, has a problem as a similarity metric: it favors long vectors. The dot product is higher if a vector is longer, with higher values in each dimension. The simplest way to modify the dot product to normalize for the vector length is to divide the dot product by the lengths of each of the two vectors. This normalized dot product turns out to be the same as the **cosine** of the angle between the two vectors.

Pointwise mutual information (is one of the most important concepts in NLP. It is a measure of how often two events x and y occur, compared with what we would expect if they were independent: $I(x, y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$. It is more common to use **Positive PMI** (called PPMI) which replaces all negative PMI values with zero.

Distribution similarity: Compare words by comparing their distributions over context words, e.g. KL divergence, Jensen-Shannon divergence

Two words have **first-order cooccurrence** (sometimes called **syntagmatic association**) if they are typically nearby each other. Thus wrote is a first-order associate of book or poem. Two words have **second-order cooccurrence** (sometimes called **paradigmatic association**) if they have similar neighbors. Thus wrote is a second-order associate of words like said or remarked.

Word embeddings are based on the distributional assumption that words appearing within similar context possess similar meaning.

Word2Vec tends to embed both syntactical and semantic information and it is very effective for the compositionality.

Although pre-trained word vectors work very well in many practical downstream tasks, in some settings it's best to continue to learn (i.e. 'retrain') the word vectors as parameters of our neural network. Explain why retraining the word vectors may hurt our model if our dataset for the specific task is too small: See TV/television/telly example. Word vectors in training data move around; word vectors not in training data don't move around. Destroys structure of word vector space. Could also phrase as an overfitting or generalization problem.

Evaluating word vectors:

- Intrinsic: Word Vector Analogies ; Word vector distances and their correlation with human judgments.
- Extrinsic: Name entity recognitions: finding a person, location or organization and so on.

4.1 skip-gram architecture

The word2vec family of models, including skip-gram and CBOW, is a popular efficient way to compute dense embeddings.

The **intuition of word2vec** is that instead of counting how often each word w occurs near, say, apricot, we'll instead train a classifier on a binary prediction task: "Is word w likely to show up near apricot?" We don't actually care about this prediction task; instead we'll take the learned classifier weights as the word embeddings. The revolutionary intuition here is that we can just use running text as implicitly supervised training data for such a classifier. Skip-gram intuition:

- Treat the target word and a neighboring context word as positive examples.
- Randomly sample other words in the lexicon to get negative samples
- Use logistic regression to train a classifier to distinguish those two cases
- Use the regression weights as the embeddings

Softmax in log-bilinear model needs the computation of partition function. Noise contrastive estimation or negative sampling allows an alternative.

The noise words are chosen according to their weighted unigram frequency $p_\alpha w = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$, where α is a weight.

For $k = 1$ (no oversampling) and $p_n(w_i, w_j) = p(w_i)p(w_j)$ negative sampling yields pointwise mutual information.

word2vec vs count-based methods:

+ scale with corpus size; capture complex patterns beyond word similarity
− slower than occurrence count based model; efficient usage of statistics

4.2 CBOW vs. Skip-Gram

Reference website

In the "skip-gram" mode alternative to "CBOW", rather than averaging the context words, each is used as a pairwise training example. That is, in place of one CBOW example such as [predict 'ate' from average('The', 'cat', 'the', 'mouse')], the network is presented with four skip-gram examples [predict 'ate' from 'The'], [predict 'ate' from 'cat'], [predict 'ate' from 'the'], [predict 'ate' from 'mouse']. (The same random window-reduction occurs, so half the time that would just be two examples, of the nearest words.)

- Skip-gram: works well with small amount of the training data, represents well even rare words or phrases.
- CBOW: several times faster to train than the skip-gram, slightly better accuracy for the frequent words

5 Sentiment Analysis

Sentiment classification related tasks:

- Multiple-aspect classification (e.g. restaurant: food, ambience, service, price/value)
- Contextual polarity disambiguation: classify marked phrase
- Sentiment towards a topic/entity
- cf. SemEval, International Workshop on Semantic Evaluation [link]

Unsupervised Learning of Polarity: Turney & Littman bootstrapped a lexicon through seed words of high polarity, computing association via PMI over context windows (size 10) and respectively assigning polarity scores:

$$\text{PMI}(w, S) = \log p(w \ \& \ v \in S) - \log p(w) - \log p(v \in S)$$

$$\text{Polarity}(w) = \text{PMI}(w, +) - \text{PMI}(w, -)$$

They estimate PMI through querying search engine with NEAR operator:

$$\text{Polarity} = \log \left(\frac{\#hits(w \text{ NEAR pos}) \cdot \#hits(\text{neg})}{\#hits(w \text{ NEAR neg}) \cdot \#hits(\text{pos})} \right)$$

Basic equations:

$$\text{Pointwise mutual info: } \text{PMI}(w_1, w_2) = \log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)} = \log_2 \frac{\#hits(w_1 \text{ NEAR } w_2)}{\#hits(w_1)\#hits(w_2)}$$

Distant Supervision: automatically generate noisy training data based on strong cues (e.g. hashtags, emoticons) to be used in sentiment classification tasks.

Negations are a major challenge as it can reverse the polarity:

- Naive approach: detect negations, determine their scope and reverse sign of polarity
- Take grammar more seriously, do syntactic analysis (cf. Stanford Sentiment Tree Bank)
- Use sequence based neural networks

5.1 Recursive Neural Networks

Recursive neural networks comprise a class of architecture that can operate on structured input. The idea is to have a composition function $g : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ that models the mapping $(\mathbf{x}_{\text{child1}}, \mathbf{x}_{\text{child2}}) \mapsto \mathbf{x}_{\text{parent}}$.

- Generalized linear model:

$$g(\mathbf{x}_l, \mathbf{x}_r) = \tanh\left(\mathbf{W}\begin{bmatrix}\mathbf{x}_l \\ \mathbf{x}_r\end{bmatrix}\right)$$

- Multi-linear (tensor) model:

$$g(\mathbf{x}_l, \mathbf{x}_r) = \tanh\left(\begin{bmatrix}\mathbf{x}_l \\ \mathbf{x}_r\end{bmatrix}^T \mathbf{V}\begin{bmatrix}\mathbf{x}_l \\ \mathbf{x}_r\end{bmatrix} + \mathbf{W}\begin{bmatrix}\mathbf{x}_l \\ \mathbf{x}_r\end{bmatrix}\right)$$

5.2 Text Summarization

Text summarization is used to generate a short and precise summary of a reference document (e.g. automatic creation of abstract or headline for news article). A simple baseline could be to take the first sentence of a document.

Approaches to summarization:

- Extractive Summarization: Select parts (typically sentences) of original text to form a summary.
  - Easier to solve (reduces to assigning scores to sentences)
  - Too restrictive (no paraphrasing possible)
- Abstractive Summarization: Generate new sentences using natural language generation techniques.
  - Harder to solve and evaluate
  - More natural summaries through paraphrasing

Three stages of summarization:

- Computing document/sentence representations.
- Sentence scoring: how important is each sentence?
- Selecting and sequencing sentences (also sentence realization: cleaning up sentences)

SumBasic is based on the intuition that summaries should maintain word distribution of the article:

1. Compute the probability distribution over the words w_i appearing in the input, $p(w_i)$ for every i; $p(w_i) = n/N$, where n is the number of times the word appeared in the input, and N is the total number of content word tokens in the input.
2. For each sentence S_j in the input, assign a weight equal to the average probability of the words in the sentence, i.e.

$$weight(S_j) = \sum_{w_i \in S_j} \frac{p(w_i)}{|\{w_i \mid w_i \in S_j\}|}$$

3. Pick the best scoring sentence that contains the highest probability word.

4. For each word w_i in the sentence chosen at step 3, update their probability $p_{new}(w_i) = p_{old}(w_i) \cdot p_{old}(w_i)$.
5. If the desired summary length has not been reached, go back to Step 2.

KLSum replaces SumBasic's greedy selection and update driterion. KLSum criterion for summary sentences S, collection distribution P_D and summary distribution P_S:

$$\text{score}(S) = D_{KL}(P_D \parallel P_S)$$

Smoothing of P_D is necessary. Joint sentence selection.

Abstractive summarization:

- Seq2seq for abstractive summarization limitations: nonsensical output, factual mistakes, repetitions, non-fluent language, UNKs
- Pointer-Generator Network: combine copying (extractive) with generation (abstractive). Rare words are extracted rather than generated. P-Gen more accurate but generates repetitions.
- Reducing repetition with coverage: Coverage = cumulative attention = what has been covered so far. Coverage is used as an extra input to the attention mechanism. Idea is to penalize attending to things that have already been covered.

ROUGE-N compares the machine-generated summary to the human-written reference summary and counts co-occurrence of N-grams. It is based on BLEU and recall oriented. It calculates what percentage of the bigrams from the reference summaries S appear in the automatic summary X.

$$\text{ROUGE-N} = \frac{\sum_{S \in RefSummaries} \sum_{gram_N} \min(Count(gram_N, S), Count(gram_N, X))}{\sum_{S \in RefSummaries} \sum_{gram_N} Count(gram_N, S)}$$

ROUGEsal: ROUGE + weighted salient terms

6 Machine Translation

Human languages vary along: word ordering (SVO, SOV, VSO), morphology (stems, affixes), word boundaries, inferential load, lexical divergences, etc.

Three classical approaches for MT:

- Direct: proceed word-by-word, translating each word, reorder if necessary
- Transfer: syntactically parse source language, convert the parse tree through rules into parse for target language, generate target sequence
- Interlingua: parse source sentence into meaning representation, generate target sequence from meaning

6.1 Statistical Machine Translation

Aims to model crucial components as **faithfulness and fluency**, which decouples to a translation model and a language model:

$$\hat{e} = \arg\max_e P(e \mid f) = \arg\max_e \frac{P(f \mid e)P(e)}{P(f)} = \arg\max_e P(f \mid e)P(e)$$

IBM Models: seminal works in SMT, 5 models of incremental complexity, EM parameter estimation

IBM Model 1, word-based $P(f \mid e) = \sum_a p(f, a \mid e)$:

$$P(f, a \mid e) = P(a \mid e)P(f \mid e, a) = \frac{\epsilon}{(l_e + 1)^{l_f}} \prod_{j=1}^{l_f} t(f_j \mid e_{a(j)})$$

where ϵ is a normalization constant (i.e. probability of choosing the length: $P(l_f)$) and $(l_e+1)^{l_f}$ is the number of possible alignments. It makes the simplifying assumption that each alignment is equally likely. Generative story: choose length of the foreign sentence, choose alignment, choose foreign words from each aligned foreign words. It is limited in the sense that it only allows many-to-one mappings but not one-to-many or many-to-many.

EM Algorithm for word alignment:

If we had Model 1 parameters already, we could re-estimate the parameters by using the parameters to compute the probability of each possible alignment and then using the weighted sum of the alignments to re-estimate the model 1 parameters.

1. Initialize the model typically with uniform distributions (all translations and word alignments are equally likely), input is sentence aligned corpus $(f, e)^N$
2. Repeat until convergence
   - E Step: Use the current model to compute the probability for all possible alignments of the training data

$$\mathbb{E}[count(f_j, e_i)] = \sum_{(f,e), f_j \in f \wedge e_i \in e} P(a \mid f, e), \text{ where } P(a \mid f, e) = \frac{P(f, a \mid e)}{\sum_a P(f, a \mid e)}$$

   - M Step: Use the alignment probability estimates to re-estimate values for all the parameters

$$t(f_j \mid e_i) = \frac{\mathbb{E}[count(f_j, e_i)]}{\sum_j \mathbb{E}[count(f_j, e_i)]}$$

Phrase-based translation model: many-to-many modeling, more context, improve with more data, standard model before NMT. Generative story: group words into phrases, translate each phrase into foreign language, reorder them if necessary. Decoding is NP-complete, so decode through greedy algorithms, beam search.

Learning the translation phrase table:

1. Get a bitext (parallel corpus) and align the sentences ($e - f$ sentence pairs)

2. Use IBM Model 1 to learn word alignments ($e \rightarrow f$ and $f \rightarrow e$)
3. Symmetrize the alignments (using heuristics to add points from the union, on top of intersection)
4. Extract phrases and assign scores (then pruning)

$$\phi(\bar{f} \mid \bar{e}) = \frac{count(\bar{e}, \bar{f})}{\sum_i count(\bar{e}, \bar{f}_i)}$$

Best translation:

$$\hat{e} = \arg\max_e \prod_i^l \phi(\bar{f} \mid \bar{e}) d(s_i - e_{i-1}) \prod_i^{|e|} P_{LM}(e_i \mid e_1, \ldots, e_{i-1}), \text{ where } d(x) = \alpha^{|x|}$$

6.2 BLEU

BLEU (bilingual evaluation understudy) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another.

BLEU's output is always a number between 0 and 1. This value indicates how similar the candidate text is to the reference texts, with values closer to 1 representing more similar texts. Few human translations will attain a score of 1, since this would indicate that the candidate is identical to one of the reference translations. For this reason, it is not necessary to attain a score of 1. Because there are more opportunities to match, adding additional reference translations will increase the BLEU score.

Basic precision measure is not a useful measure, instead BLEU uses a **modified precision** where the values are clipped w.r.t. maximum word (n-gram) occurrences in the references.

$$\text{BLEU} = \text{BP} \left(\prod_n^4 P_n \right)^{\frac{1}{4}}, \text{ where BP} = \min\left(1, \frac{\text{output-length}}{\text{reference-length}}\right)$$

$$P_n = \frac{\sum\limits_{\mathcal{C} \in \{\text{Candidates}\}} \sum\limits_{\text{n-gram} \in \mathcal{C}} Count_{clip}(\text{n-gram})}{\sum\limits_{\mathcal{C}' \in \{\text{Candidates}\}} \sum\limits_{\text{n-gram}' \in \mathcal{C}'} Count(\text{n-gram}')}$$

BLEU offers a surface-level evaluation, as it is based on n-grams. If you have a system that is very semantic and it generates high quality translations that express the same translation in very different words, it may get penalized by BLEU. Another problem with BLEU scores is that they tend to favor short translations, which can produce very high precision scores, even using modified precision.

Possible suggestions for BLEU:

- Penalize very long outputs as well
- Sometimes short outputs might be desirable, e.g. if a language issues the statement using less tokens (words). So there could be a language-specific factor.

- Short but expressive output is actually ok. Usually, less common words/n-grams are more expressive, hence we could put more weight on them. This is what an extension of BLEU, the so-called NIST score does.

6.3 Neural Machine Translation

Guest lecturer: **Christian Buck**

- Tall towers analogy of Warren Weaver, concept of interlingua
- Need a way to model sequences of data as vectors: RNNs
- RNNs does not explicitly keep a state, they rather input and output state vectors, applying same matrix multiplications in each step.
- Backprop through time leads to vanishing gradients, to which LSTMs are offered as a solution.
- Initial LSTM cell was a very lucky choice and it is very hard to improve upon that.
- When translating from one language to another one, we would like to condition on the base language. In RNNs, initial state vector would be a good place to condition on prior knowledge.
- Encoder-decoder architecture works surprisingly good at producing the vector representations of (presumably long) sentences.
- In practice, it works a bit better if you patch this output vector that you got from your encoder to every single decoder input because decoder is not very good at carrying along this information.
- Encoder-decoder architecture solves fundamental shortcoming of phrase-based MT: all decisions depend on whole source sentence.
- Problem: bottleneck is too small to fit every sentence
- Attention: compute weighted average of encoder state, solves bottleneck problem
- Bottom-up meta character grouping for sub-word segmentation
- Multi-lingual models
- Zero-shot translation: Google's NMT model uses an additional "token" at the beginning of the input sentence to specify the required target language to translate to. [link]
- Self-attention
- Multi-headed attention
- Non-autoregressive MT

7 Parsing

- Subcategorization: certain kinds of relations between words and phrases (e.g. want to ✓, find to $\times$)
- Constituency: grouping together of words. External, internal evidences, movement...

- Formal definition of context-free grammars (CFGs): terminals, non-terminals, rules and start symbol
- Agreement: related to inflectional morphology (e.g. he does ✓, he do ×, three cats ✓, three cat ×, gender agreement ...)

Ambiguity: A sentence is structurally ambiguous if the grammar assigns it more than one possible parse. Common kinds of structural ambiguity include PP-attachment, coordination ambiguity and noun-phrase bracketing ambiguity.

PP-attachment: compare "I saw the burglar with a telescope" (VP) with "I saw the burglar with a gun" (NP).

Coordination: "old men" and women vs old "men and women"

CFGs can be used for:

- Generation: of strings in the language
- Accepting: identify strings in the language
- Parsing: associating syntactic trees to strings in the language

CFG has a problem in the sense that if they are not specific enough they can overgenerate (e.g. via subcategorization, agreement). We can patch the grammar to make it more specific but it explodes the grammar, creates sparsity and it is not an elegant solution.

Two search strategies:

- Top-down (goal-directed, emphasizes prior knowledge): Never wastes time exploring trees that cannot result in an S etc. but spends effort on trees not consistent with the input.
- Bottom-up: data-directed, emphasizes empirical data

Exponential number of possible parse trees, so we need efficient algorithms.

7.1 CKY Algorithm

CKY (Cocke-Kasami-Younger) algorithm needs the grammar to be in Chomsky Normal Form: $A \to B\ C$, $A \to w$. Any CFG can be rewritten into CNF automatically, without any loss in expressiveness. Goal is to avoid redoing work via DP. DP parsing is $O(n^3|G|)$. Solve ambiguity (due to multiple hypotheses) through priors and context.

Conversion to CNF:

1. START: Eliminate the start symbol from right-hand sides (create a new start symbol)
2. TERM: Eliminate rules with nonsolitary terminals (convert terminals within rules to dummy non-terminals)
3. BIN: Eliminate right-hand sides with more than 2 nonterminals (binarize rules)
4. DEL: Eliminate ε-rules (null production)
5. UNIT: Eliminate unit rules (unit-productions)

On its complexity, from stackoverflow:

The runtime of CYK depends on the length of the longest production rule, since the algorithm considers all possible ways of decomposing a string into k parts for a production of length k. This means that the runtime per phase is $O(n^k)$, where k is the length of the longest production. Since there are $O(n)$ phases, the runtime of CYK on a grammar with maximum production length k is $O(n^{k+1})$.

CYK will work correctly on grammars that aren't in CNF, but the runtime may not end up being cubic in the length of the string. The CNF requirement just forces $k = 2$ and therefore guarantees an $O(n^3)$ overall runtime.

7.2 PCFGs and Statistical Parsing

Probabilistic context-free grammars (PCFGs): a probabilistic augmentation of context-free grammars in which each rule is associated with a probability, which comes as useful for disambiguation. They can be parsed with probabilistic CKY algorithm.

A PCFG is said to be consistent if the sum of the probabilities of all sentences in the language equals 1. Certain kinds of recursive rules cause a grammar to be inconsistent by causing infinitely looping derivations for some sentences.

Parameter estimation:

- Supervised estimation via treebank:

$$P(\alpha \to \beta \mid \alpha) = \frac{\text{Count}(\alpha \to \beta)}{\sum_\gamma \text{Count}(\alpha \to \gamma)} = \frac{\text{Count}(\alpha \to \beta)}{\text{Count}(\alpha)}$$

- Inside-outside algorithm: use a non-probabilistic parse to estimate probabilistic parser to estimate probabilities, weight counts and repeat (special case of EM)

PCFG independence assumptions miss structural dependencies between rules (rules and probabilities ignore context, solutions: non-terminal splitting, split-and-merge) and they lack of sensitivity to lexical dependencies (e.g. depending on how the probabilities are set, a PCFG will always either prefer NP attachment or VP attachment).

The Collins Model 1 introduces the idea of a generative story for each of the lexicalized rules given some independency assumptions. Rather than computing MLE probabilites for each of the rules (this is not feasible as the rules are very speci

c due to the lexicalization process and it would be impossible to have good counts), the probability of a rule is obtained as the product of smaller independent probabilities.

7.3 Dependency parsing

Dependency grammars: based on lexical relations rather than constituent ones, syntactic structure of a sentence is described purely in terms of words and binary semantic or syntactic relations between these words. They have strong predictive parsing power and can handle languages with relatively free word order.

The importance of this distinction is that if one acknowledges the initial subject-predicate division in syntax is real, then one is likely to go down the path of phrase structure grammar, while if one rejects this division, then one must consider the verb as the root of all structure, and so go down the path of dependency grammar.

Constituency parsing outputs a parse tree (a derivation from the CFG) whereas a dependency parser outputs a directed tree (no grammar). However the differences, functional relations can be extracted from CFG trees and phrases identifiable in DGs.

Advantages: no grammar, faster parsing, free word order languages, easier to use, easier to generate data in multiple languages.

Free word order: A phrase-structure grammar would need a separate rule for each possible place in the parse tree where a specific phrase type could occur. A dependency-based approach would just have one link type representing this particular relation. Thus, a dependency grammar approach abstracts away from word-order information, representing only the information that is necessary for the parse.

Three restrictions that apply to dependency trees:

1. There is a single designated root node that has no incoming arcs.
2. With the exception of the root node, each vertex has exactly one incoming arc.
3. There is a unique path from the root node to each vertex in V .

The arrow connects a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate).

Projectivity: An arc from a head to a dependent is said to be projective if there is a path from the head to every word that lies between the head and the dependent in the sentence. A dependency tree is then said to be projective if all the arcs that make it up are projective. In other words, a dependency tree is projective if it can be drawn with no crossing edges.

The reason why projectivity is important is twofold:

1. Transition-based approaches can only produce projective trees. Hence, any sentences with nonprojective parses will necessarily contain some errors.
2. The most widely used English dependency treebanks were automatically derived from phrase-structure treebanks through the use of head-finding rules. Trees generated in such a fashion are guaranteed to be projective.

We can use graph-based or transition-based parsers. On one hand transition-based approaches are fast (linear times) but by design they are greedy (however this can be mitigated with techniques such as beam search). On the other hand, graph-based approaches are slower but they do an exhaustive search.

A **shift-reduce parser** is a class of efficient, table-driven bottom-up parsing methods for computer languages and other notations formally defined by a grammar. A Shift step advances in the input stream by one symbol. That shifted symbol becomes a new single-node parse tree. A Reduce step applies a completed grammar rule to some of the recent parse trees, joining them together as one tree with a new root symbol.

The result of the algorithm will depend on how we (greedily) chose the action at time t. The choice can be anything, from totally random, to having an "oracle", to having some ML model tell us which action to pick.

Disadvantages of dependency parsing:

- Graph-based dependency parsers are slower (quadratic time instead of linear in the length of the sentence).

- Graph-based dependency parsers are not constrained to produce valid parse trees, so they may be more likely to produce invalid ones.
- Graph-based dependency parsers make parsing decisions independently so they can't use features based on previous parsing decisions.

8 Question Answering

Paradigms for QA:

- IR-based (Information Retrieval) approaches: answer a question by finding a short text, on the web or other corpus, that contains the answer
  (a) Question processing: detect question type, answer type, focus, relations and formulate queries to send to a search engine
  (b) Passage retrieval: retrieve ranked documents and break into suitable passages, then rank
  (c) Answer processing: extract candidate answers, then rank them using evidence from the text and external sources
- Knowledge-based approaches: build a semantic representation of the query (e.g. times, dates, locations, entities, numeric quantities) and map from this semantics to query structured data or resources:
  - Geospatial databases
  - Ontologies (Wikipedia infoboxes, dbPedia, WordNet, Yago)
  - Restaurant review sources and reservation services
  - Scientific databases
- Hybrid approaches: e.g. use IR to find candidates and use knowledges-based approaches to rank them

Factoid questions are questions with simple answers with single phrase or identity

Question headword is the the headword of the first noun phrase after the wh-word.

A **triplestore** or RDF store is a purpose-built database for the storage and retrieval of triples through semantic queries. A triple is a data entity composed of subject-predicate-object, like "Bob is 35" or "Bob knows Fred".

Active question reformulation: We propose an agent that sits between the user and a black box QA system and learns to reformulate questions to elicit the best possible answers. The agent probes the system with, potentially many, natural language reformulations of an initial question and aggregates the returned evidence to yield the best answer. For example, reformulate through a seq2seq and aggregate using a CNN.

Common evaluation metrics are accuracy (does answer match the gold-labeled answer?) and mean reciprocal rank:

- Return a ranked list of N candidate answers

- Score is 1/rank of the first right answer

$$\text{MRR} = \frac{\sum_{i=1}^{N} \frac{1}{\text{rank}_i}}{N}$$

Answering complex questions:

- The (bottom-up) snippet method: find a set of relevant documents, extract informative sentences from the documents (using tf-idf, MMR), order and modify the sentences into an answer
- The (top-down) information extraction method: build specific answerers for different question types (e.g. definition questions, biography questions, certain medical questions)

9 Appendix

9.1 tf-idf

Term frequency (tf) weighting schemes:

- binary: $0, 1$
- raw count: $f_{t,d}$
- term frequency: $f_{t,d}/\sum_{t' \in d} f_{t',d}$
- log normalization: $\log(1 + f_{t,d})$
- double normalization K: $K + (1 - K)\frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

Inverse document frequency (idf) weighting schemes:

- unary: 1
- inverse document frequency: $\log \frac{N}{n_t}$
- inverse document frequency smooth: $\log(\frac{N}{1+n_t})$
- inverse document frequency max: $\log(\frac{\max_{\{t' \in d\}} n_{t'}}{1+n_t})$
- probabilistic inverse document frequency: $\log \frac{N-n_t}{n_t}$

9.2 BERT

BERT (Bidirectional Encoder Representations from Transformers) is a method of pre-training language representations, meaning that we train a general-purpose "language understanding" model on a large text corpus (like Wikipedia), and then use that model for downstream NLP tasks that we care about (like question answering). BERT outperforms previous methods because it is the first **unsupervised, deeply bidirectional** system for pre-training NLP.

Fine-tuning is inexpensive. All of the results in the paper can be replicated in at most 1 hour on a single Cloud TPU, or a few hours on a GPU, starting from the exact same pre-trained model.

History of Contextual Representations:

- Semi-Supervised Sequence Learning: Train LSTM Language Model, Fine-tune on Classification Task
- ELMo: Deep Contextual Word Embeddings: Train Separate Left-to-Right and Right-to-Left LMs, Apply as "Pre-trained Embeddings"
- Improving Language Understanding by Generative Pre-Training: Train Deep (12-layer) Transformer LM, Fine-tune on Classification Task

Masked LM: Mask out k% of the input words (so that they cannot see themselves), and then predict the masked words. Masked LM takes slightly longer to converge because we only predict 15% instead of 100%, but absolute results are much better almost immediately. Too mach masking reduces context and too little masking is too expensive to train (15% is empirical).

Input Representation: Each token is sum of three embeddings (token + segment + position), tokenization is done to level word-piece (think of somewhere between word and character level)

Empirical advantages of **Transformer vs. LSTM**:

- Self-attention == no locality bias, Long-distance context has "equal opportunity"
- Single multiplication per layer == efficiency on TPU, Effective batch size is number of words, not sequences

The model must be learning more than "contextual embeddings", predicting missing words (or next words) requires learning many types of language understanding features (syntax, semantics, pragmatics, coreference, etc).

9.3 Conversational Agents

In time series analysis, **dynamic time warping (DTW)** is one of the algorithms for measuring similarity between two temporal sequences, which may vary in speed. For instance, similarities in walking could be detected using DTW, even if one person was walking faster than the other, or if there were accelerations and decelerations during the course of an observation.