Natural Language Processing Summary

1. Text Preprocessing

Definition: words: sequence of characters with meaning; sentence: sequence of words with reasonable meaning; document: one or more sentences; corpus: a collection of documents; word token: an instance of a word; word type: the type of a word.

Why preprocessing: Because language is compositional, it needs to decompose into smaller parts.

Steps:

1. Remove unwanted formatting (e.g. HTML)
2. **Sentence segmentation**: break documents into sentences
3. **Word tokenisation**: break sentences into words
4. **Word normalisation**: transform words into canonical forms
5. **Stopword removal**: delete unwanted words

Sentence Segmentation: Naïve approaches: break on punctuation (U.S. dollar not a sentence) or capital (Mr. Brown is just a name); Better approaches: lexicons or machine learning based model. The features can be uppercase, lowercase, number, character length or POS tags.

Word Tokenization: For English, separate by whitespace, but for other languages, such as Chinese or Germany, tokenization is a tough problem. For Chinese, we can use maxmatch algorithm to greedily tokenize the longest word in the vocabulary, but not always work.

Byte-pair Encoding: The core idea is iteratively merge frequent pairs of characters. The advantages are: ① data-informed tokenization; ② works for different languages; ③ can handle unknown words.

In practice, the frequent words are tokenized as full words, while the rare words will be broken into sub-words.

Word Normalization: The main goal is to reduce vocabulary and map words into the same type.

- Lower casing (Australia → australia)
- Removing morphology (cooking → cook)
- Correcting spelling (definately → definitely)
- Expanding abbreviations (U.S.A → USA)

Inflective Morphology: grammatical variants.

English inflects nouns, verbs, and adjectives

- Nouns: *number* of the noun (*-s*)
- Verbs: *number* of the subject *(-s)*, the *aspect* (*-ing*) of the action and the *tense (-ed*) of the action
- Adjectives: *comparatives* (*-er*) and *superlatives* (*-est*)

Lemmatization: normalize grammatical variant words into normalized words.

- poked → poke (not pok)
- stopping → stop (not stopp)
- watches → watch (not watche)
- was → be (not *wa*)

Derivational Morphology: use suffix and prefix to form a word where the type or the meaning of it changes.

- -ly (personal → personally)
- -ise (final → finalise)
- -er (write → writer)
- write → rewrite
- healthy → unhealthy

Stemming: A process of strip off all suffixes and prefixes and leaves a stem.

Port Stemmer: A rule-based stemmer for English.

Other Normalizations:

- Normalising spelling variations
  - Normalize → Normalise (or vice versa)
  - U r so coool! → *you are so cool*
- Expanding abbreviations
  - US, U.S. → United States
  - imho → in my humble opinion

Stop Words Removal: A list of words to be removed from the document.

- All *closed-class* or *function* words
  - E.g. *the, a, of, for, he, …*
- Any high frequency words
- NLTK, spaCy NLP toolkits

2. Language Models

Why Language Models: Good measurements for fluency of a sentence using probabilities; A method for language generation.

Application: query completion, OCR, machine translation, summarization, dialogue systems.

Language Model: Decompose full joint distribution $P(w_1, w_2, \dots, w_n)$ to $\prod_{i=1}^{m} P(w_i)$ (unigram), $\prod_{i=2}^{m} P(w_i|w_{i-1})$ (bigram) or $\prod_{i=n}^{m} P(w_i|w_{i-1}, w_{i-2}, \dots, w_{i-n+1})$. For calculate the probability, we can count the occurrence:

For unigram models,

$$P(w_i) = \frac{C(w_i)}{M} \qquad \frac{C(\text{barks})}{M}$$

For bigram models,

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})} \qquad \frac{C(\text{dog barks})}{C(\text{dog})}$$

For n-gram models generally,

$$P(w_i \mid w_{i-n+1} \dots w_{i-1}) = \frac{C(w_{i-n+1} \dots w_i)}{C(w_{i-n+1} \dots w_{i-1})}$$

In language model, we often use special token <s> and </s> to denote the start and end of a sentence and use <UNK> to denote unknown word token. Since language has long-term effect, some n-gram model will have very small probability, when n is large. Thus we need some smoothing mechanism to handle zero probability issue.

Smoothing:

- Laplacian smoothing: Add 1 smoothing.

For unigram models (**V**= the vocabulary),

$$P_{add1}(w_i) = \frac{C(w_i) + 1}{M + |V|}$$

For bigram models,

$$P_{add1}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + |V|}$$

- Lidstone smoothing: Add k smoothing:

$$P_{addk}(w_i \mid w_{i-1}, w_{i-2}) = \frac{C(w_{i-2},\ w_{i-1}, w_i) + k}{C(w_{i-2}, w_{i-1}) + k|V|}$$

- Absolute discounting: borrow a fixed probability for each seen n-grams and redistributes it to unseen n-grams (typically evenly).
- Kakz backoff: redistribute the probability mass lower order model.

$$P_{katz}(w_i|w_{i-1}) = \begin{cases} \frac{C(w_{i-1},w_i)-D}{C(w_{i-1})}, & \text{if } C(w_{i-1}, w_i) > 0 \\ \alpha(w_{i-1}) \times \frac{P(w_i)}{\sum_{w_j:C(w_{i-1},w_j)=0} P(w_j)}, & \text{otherwise} \end{cases}$$

unigram probability for w_i
e.g. P(infirmity)

sum unigram probabilities for all words that do not co-occur with context w_{i-1}
e.g. P(infirmity) + P(alleged)

the amount of probability mass that has been discounted for context w_{i-1}
((0.1 x 5) / 20 in previous slide)

- Kneser-Ney smoothing: redistribute the probability mass borrowed based on the continuation probability.
  - High versatility -> co-occurs with a lot of unique words, e.g. glasses
    - men's glasses, black glasses, buy glasses, etc
  - Low versatility -> co-occurs with few unique words, e.g. francisco
    - san francisco

$$P_{KN}(w_i \mid w_{i-1}) = \begin{cases} \frac{C(w_{i-1}, w_i) - D}{C(w_{i-1})}, & \text{if } C(w_{i-1}, w_i) > 0 \\ \beta(w_{i-1})P_{cont}(w_i), & \text{otherwise} \end{cases}$$

the amount of probability mass that has been discounted for context w_{i-1}

$$P_{cont}(w_i) = \frac{|\{w_{i-1} : C(w_{i-1}, w_i) > 0\}|}{\sum_{w_i} |\{w_{i-1} : C(w_{i-1}, w_i) > 0\}|}$$

- Interpolation: combine different orders of n-gram models, i.e. weighted sum of probabilities across progressively shorter contexts.

$$P_{IN}(w_i \mid w_{i-1}, wi-2) = \lambda_3 P_3(w_i \mid w_{i-2}, w_{i-1}) + \lambda_2 P_2(w_i \mid w_{i-1}) + \lambda_1 P_1(w_i)$$

Learned based on held out data

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

Generation: Given an initial word, choose the next word according to the probability distribution produced by the language model. The selection technique can be Argmax, sample based on probability distribution or beam search.

3. Part of Speech Tagging

Part of Speech: A word class, namely syntactic class or morphological class.

Why: Useful for many applications, such as information retrieval, sentiment analysis.

POS open class: nouns, verbs, adjectives and adverbs.

POS closed class: prepositions, particles, determiners, pronouns, conjunctions, modal verbs.

Ambiguity: The POS tag or a word depends on context.

Tagset:

NN noun	VB verb
JJ adjective	RB adverb
DT determiner	CD cardinal number
IN preposition	PRP personal pronoun
MD modal	CC coordinating conjunction
RP particle	WH wh-pronoun
TO *to*	

Rule-based Tagging: Start with a list of possible tags for each word, often includes other lexical information, and apply rules to narrow down to a single tag.

Unigram Tagger: Assign the most common tag to each word type.

Tagging Classifier: Use a standard discriminative classifier with features (target word, surrounding words, tags that are already classified). However, it will suffer from error propagation, which propagates the wrong predictions from previous steps to the next ones.

Hidden Markov Model: A better approach to handle POS tag problem. The basic idea is to decompose the sentence into individual words and tag each word separately and takes into account the whole sequence when learning and predicting in order to avoid error propagation.

- Two assumptions: We use Bayes rules to form the question: $\hat{\boldsymbol{t}} = arg\max_{\boldsymbol{t}} p(\boldsymbol{w}|\boldsymbol{t})p(\boldsymbol{t})$ and we decompose it into the product of the individual probabilities: $p(\boldsymbol{w}|\boldsymbol{t}) = \prod_{i=1}^{n} p(w_i|t_i)$ is the emission probabilities and $p(\boldsymbol{t}) = \prod_{i=1}^{n} p(t_i|t_{i-1})$ is the transition probabilities.
- Greedy approach: During inference, we take the tag that maximizes $p(t_i|t_{i-1})p(w_i|t_i)$, which will cause error propagation problem.
- Complete Search: Consider each combination of a sequence of tags and take the maximum one. It will always find a solution but it is infeasible, the complexity will be $O(T^N)$.
- Viterbi Algorithm: Use dynamic programming approach to reduce the complexity. The main idea is for each word tagging, we consider the scores of all tags from the previous word and take the maximum tag in the current word. After training, we backpropagate to get the best tag sequence for each word. The complexity is $O(T^2N)$, where T is the number of tagset and N the length of the sequence. A better practice is we can use log probabilities to handle numerical underflow and use vectorization instead of for-loop.
- Discriminative Model: CRF, MEMM.

4. Deep Learning

Feed-forward NN: A multilayer perceptron, where each neuron is the linear combination of features from the previous layer following a non-linearity, aka activation function, e.g. $tanh\left(\sum_j w_j x_j + b\right)$ or matrix-vector notation: $tanh(W^T x + b)$. The output layer is often sigmoid function for binary classification and softmax for multiclass classification. The optimization can be performed by gradient descent and typically we often use regularization term (L1 or L2) or Dropout to generalize our model.

Model Classification: Each word is represented by a fixed low-dimensional dense vector called embeddings, instead of a sparse occurrence matrix, which has higher dimension. The difference between two similar words embeddings will be small. The input of the n-gram language model is the concatenation of n word embeddings.

Advantages of FFNN Language Model:

- Count-based *N*-gram models (lecture 3)
  - cheap to train (just collect counts)
  - problems with sparsity and scaling to larger contexts
  - don't adequately capture properties of words (grammatical and semantic similarity), e.g., film vs movie
- FFNN *N*-gram models
  - automatically capture word properties, leading to more robust estimates

Disadvantage of FFNN Language Model: Relatively slow to train; Still only capture limited context; Still unable to handle unknown words.

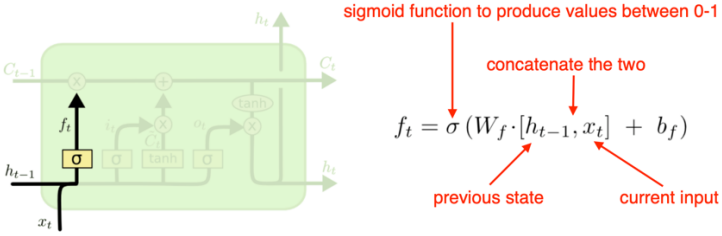
FFNN POS Tagging: Use both word embeddings and tag embeddings as the input.

CNN: convolutional filter as the linear combination of the local field instead of FFNN where each neuron is fully connected with each neuron in the next layer. We often use MaxPooling layer to reduce dimensions.

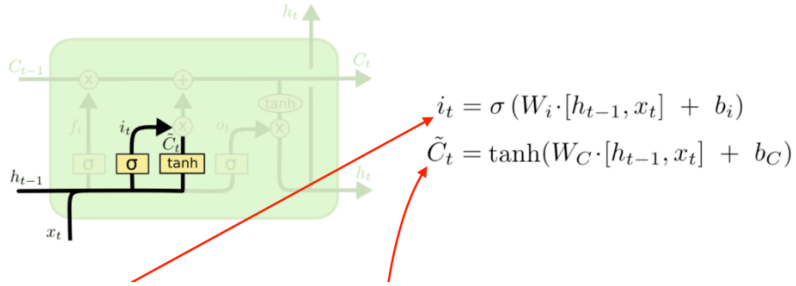
RNN: It allows for the representations of the arbitrary sized inputs. Each state is based on the previous state and the current input: $s_i = tanh(W_s^T s_{i-1} + W_x^T x_i + b)$, and output formula: $y_i = \sigma(W_y^T s_i)$, where $W_s, W_x, b,$ and W_y are shared across all time steps. The backpropagation will down to all matrix for each time step, called backpropagation through time. The training procedure is using target word for each time step as the label, while in prediction step, the generated word embedding in the current step is fed into the input of the next step.

LSTM: RNN cannot handle very long sequence, due to the vanishing gradient problem, we introduce the LSTM which uses gate control and memory cells to control the flowing of the each time step. The gate is a matrix or vector evaluated between 0 and 1, and is multiplied with hidden state element-wisely to control whether the previous information should be kept, whether the new input information should be brought and whether it should be sent to the next hidden state.

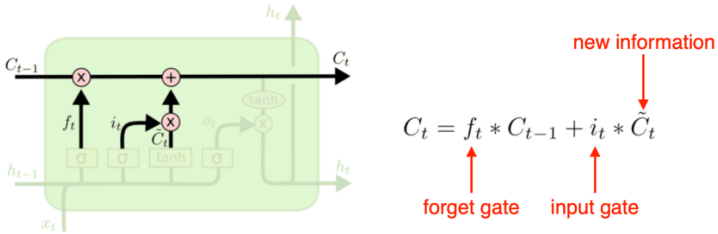
- Forget Gate: Controls how much information to forget in the memory cell.



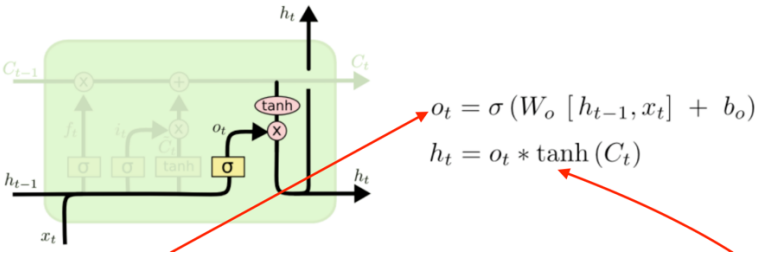
- Input Gate: Controls how much new information to put into the memory cell.



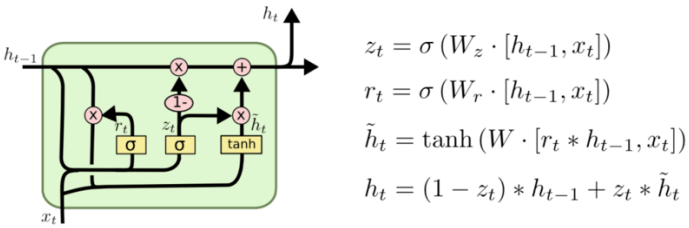
The combination of $\tilde{C}$ is the distilled information to be added, and use the combination of the forget gate and the input gate to update memory cell.



- Output Gate: Control how much the distilled memory cell to add into the next state.



GRU: A simple version of LSTM variant, which only uses 2 gates:



RNN Variant: multilayer RNN, bidirectional RNN

5. Lexical Semantics

What: How the meanings of words connect to one another.

Word Sense: Describes one aspect of the meaning of the world. Some words may have multiple meanings known as polysemous. Synonymy is two words with identical meanings, e.g. big and large; Antonymy is two words with opposite meanings, e.g. large and small; Hypernymy is the is-a relation, e.g. cat and animal; meronymy is the part-whole relation, e.g. leg and chair.

WordNet: A database of lexical relations in most major language (NLTK for English).

Synsets: The nodes of WordNet which represent senses but not words and lemmas.

Word Similarity: Use lexical database to eliminate word similarity. Given WordNet, the similarity between two senses is the inverse of path length, which is the 1 plus the edge length in the shortest path between sense c_1 and c_2

$$\text{simpath}(c_1, c_2) = \frac{1}{\text{pathlen}(c_1, c_2)}$$

The similarity between two words is the maximum similar path between two words among all its corresponding senses.

$$\text{wordsim}(w_1, w_2) = \max_{c_1 \in \text{senses}(w_1), c2 \in \text{senses}(w_2)} \text{simpath}(c_1, c_2)$$

Include depth information: use path to find the LCS.

$$\text{simwup}(c_1, c_2) = \frac{2 \times \text{depth}(\text{LCS}(c_1, c_2))}{\text{depth}(c_1) + \text{depth}(c_2)}$$

Concept Probability Node: The more general of a node, the higher concept probability. To compute them, we sum up unigrams of all children nodes of a parent node. So when calculating word similarity, it will use concept probability.

use IC instead of depth (simwup)

general concept = small values
narrow concept = large values

$$\text{IC} = -\log P(c)$$

$$\text{simlin}(c_1, c_2) = \frac{2 \times \text{IC}(\text{LCS}(c_1, c_2))}{\text{IC}(c_1) + \text{IC}(c_2)}$$

high simlin when:
- concept of parent is narrow
- concept of senses are general

$$\text{simlin}(\text{hill}, \text{coast}) = \frac{2 \times -\log P(\text{geological-formation})}{-\log P(\text{hill}) - \log P(\text{coast})}$$

Word Sense Disambiguation: Select the more appropriate word sense in a sentence. The supervised model still cannot handle that because of limited context window. Unsupervised method, such as Lesk, choose a sense whose WordNet overlaps most with the context.

5. Distributional Semantics

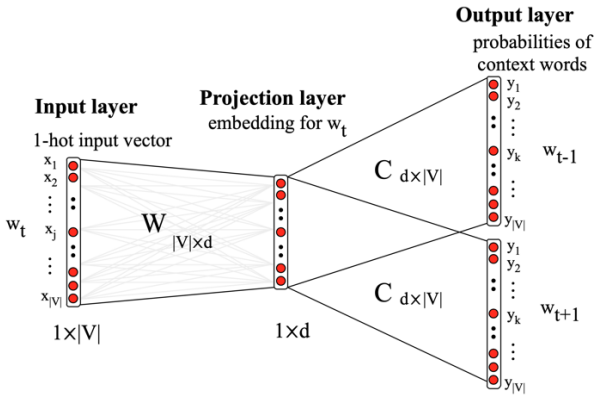
Word Vectors: The core idea is to represent a word meaning as a vector. For sparse vector, we use TF-IDF, which is term frequency (the probability of a word in a document), and inverse document frequency (the inverse of the occurrence of a document) to penalize the common words, such as a, the, this, etc. The very sparse vector can be decomposed into 3 parts (SVD decomposition), where U is the new term matrix, and it can be truncated into the first k dimensional matrix. If we want to represent a word with other context, we can use PMI as the metrics. The similarity metric is typically the cosine similarity.

$$\text{PMI}(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

However, it is very biased towards rare word pairs, we can use normalized PMI:

$$\text{Normalised PMI} \left(\frac{\text{PMI}(x, y)}{-\log_2 P(x, y)} \right)$$

Word2Vec: Predict word using the context words. Skip-gram: predict the surrounding words using the target word; CBOW: predict target word using the surrounding words. The model has two matrices to be learnt.



Negative Sampling: Since there are so many negative examples, the normalization term is hard to compute. We should sample some negative examples, and consider it as a binary classification problem.

$$P(+|t,c) = \frac{1}{1+e^{-t\cdot c}}$$ ← **maximise** similarity between target word and real context words

$$P(-|t,c) = 1 - \frac{1}{1+e^{-t\cdot c}}$$ ← **minimise** similarity between target word and non-context words

$$L(\theta) = \log P(+|t,c) + \sum_{i=1}^{k} \log P(-|t,n_i)$$

7. Contextual Representation

ELMO: Train a neural model to represent context embeddings. ELMO is a 2-layer bidirectional LSTM, each with dimension of 4096 and use character CNN to create word embeddings. The contextual embedding is the weighted combination of each hidden layer. The lower layer captures syntax while the higher layer captures semantics.

BERT: Use self-attention transformer to capture dependencies between words, but it cannot generate language. There are two objectives during training: One is masked language model, which masks $k\%$ of tokens at random and predict the masked words. Another is next sentence prediction, which predicts whether the sentence B is the next sentence A. BERT can be fine-tuned for downstream tasks, such as POS tagging, sentiment analysis, etc. BERT simply adds a classification layer for downstream tasks, compared with ELMO, which needs a task-specific model needed. BERT updates all parameters during fine-tuning, while ELMO parameters are fixed.

Self-Attention:

$$A(q,K,V) = \sum_i \frac{e^{q\cdot k_i}}{\sum_j e^{q\cdot k_j}} \times v_i$$

($e^{q\cdot k_i}$ ← query and key comparisons; $\sum_j e^{q\cdot k_j}$ ← softmax)

- Multiple queries, stack them in a matrix

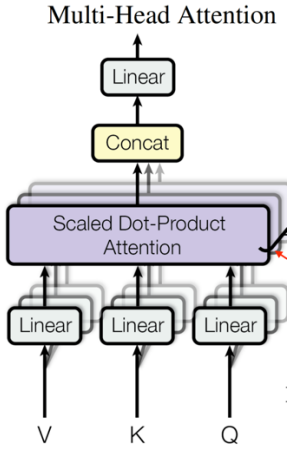
$$A(Q,K,V) = \text{softmax}(QK^T)V$$

- Uses **scaled dot-product** to prevent values from growing too large

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

(d_k ← dimension of query and key vectors)

Multi-headed Attention: Use multiple attention mechanism to allow multiple interactions.



8. Discourse

What: Discourse is to understand how sentences relate to each other in a document.

Discourse Segmentation: e.g. introduction, related works, experiments in a scientific articles.

- A document can be viewed as a sequence of segments
- A segment: a span of cohesive text
- Cohesion: organised around a **topic** or **function**

TextTiling Algorithm: Find the gap where the sentence between it has low lexical cohesion (unsupervised method).

For each sentence gap:

- Create two BOW vectors consisting of words from *k* sentences on either side of gap
- Use cosine to get a similarity score (*sim*) for two vectors
- For gap *i,* calculate a depth score, insert boundaries when depth is greater than some threshold *t*

$$depth(gap_i) = (sim_{i-1} - sim_i) + (sim_{i+1} - sim_i)$$

Supervised Discourse Segmenter:

- Apply a binary classifier to identify boundaries
- Or use sequential classifiers
- Potentially include classification of section types (introduction, conclusion, etc.)
- Integrate a wider range of features, including
  - distributional semantics
  - discourse markers (*therefore*, *and*, etc)

Discourse Parsing: To identify discourse units and the relations between them. Typically, we use rhetorical structure theory to hierarchically analyze the structure of the document.

Discourse Unit: Typically clauses of a sentence. The relation between two discourse units are elaboration or conjunction.

RST Tree: An RST relation that combines two or more discourse units into a composite discourse unit.

RST Parsing: Given a document, recover the RST tree. The method can be rule-based, bottom-up or top-down. Some discourse markers (although, but, etc.) can also indicate relations

Discourse Features:

- Bag of words
- Discourse markers
- Starting/ending *n*-grams
- Location in the text
- Syntax features
- Lexical and distributional similarities

Anaphor: Linguistic expressions that refer back to earlier elements in the text. Anaphor have an antecedent in the discourse, and not always nouns. Pronouns are the most common anaphor, sometimes others (demonstratives). There are some restrictions: number, gender and reflexive, sometimes the antecedents of pronouns should be recent.

Centering Theory:

- A unified account of relationship between discourse structure and entity reference
- Every utterance in the discourse is characterised by a set of entities, known as **centers**
- Explains preference of certain entities for ambiguous pronouns
- When resolving entity for anaphora resolution, choose the entity such that the **top foward-looking center** matches with the **backward-looking center**
- Why? Because the text reads more fluent when this condition is satisfied

Supervised Method:

- Convert restrictions and preferences into features
  - Binary features for number/gender compatibility
  - Position of antecedent in text
  - Include features about type of antecedent
- With enough data, can approximate the centering algorithm
- But also easy to include features that are potentially helpful
  - words around anaphor/antecedent

9. Formal Language Theory

Language: set of strings.

String: Sequence of elements from a finite alphabet.

Major Problem:

- Membership
  - Is the string part of the language? Y/N
- Scoring
  - Graded membership
  - "How acceptable is a string?" (language models!)
- Transduction
  - "Translate" one string into another (stemming!)

Regular Language: A formal language that can be expressed using a regular expression.

- Formally, a regular expression includes the following operations/definitions:
  - Symbol drawn from alphabet, Σ
  - Empty string, ε
  - Concatenation of two regular expressions, RS
  - Alternation of two regular expressions, R|S
  - Kleene star for 0 or more repeats, R*
  - Parenthesis () to define scope of operations

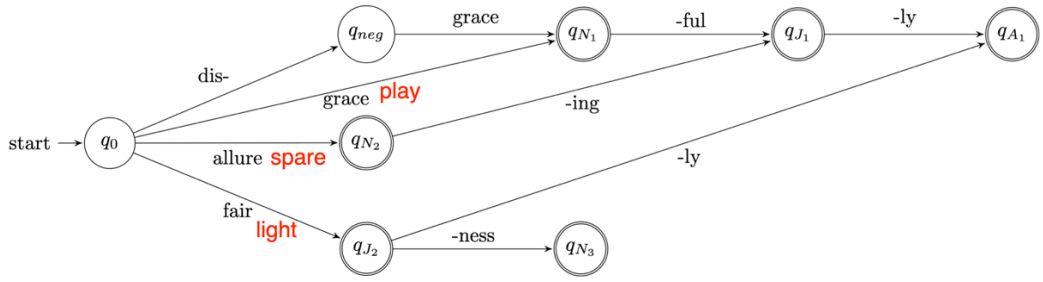
Properties:

- RLs are closed under the following:
  - **concatenation and union**
  - **intersection**: strings that are valid in both L1 and L2
  - **negation**: strings that are not in L

Finite State Acceptor: Provide a method for membership checking.

- FSA consists:
  - alphabet of input symbols, Σ
  - set of states, Q
  - start state, $q_0 \in Q$
  - final states, $F \subseteq Q$
  - transition function: symbol and state → next state
- Accepts strings if there is **path** from q_0 to a final state with transitions matching each symbol
  - Djisktra's shortest-path algorithm, O(V log V + E)

FSA can represent derivational morphology, e.g. grace→graceful, which is acceptable or allure→ *allureful, which is not acceptable.



Weighted FSA: Since some words are more plausible than others, we assign start state, transition and final as a probability.

- start state weight function, λ: Q → $\mathbb{R}$
- final state weight function, ρ: Q → $\mathbb{R}$
- transition function, δ: (Q, Σ, Q) → $\mathbb{R}$

Total score:

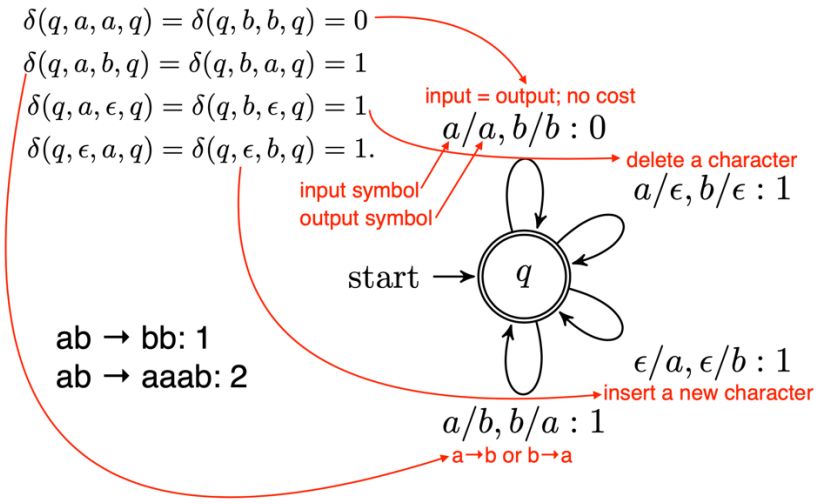
Total score of a path $\pi = t_1, \ldots, t_N$

$$\lambda(t_0) + \sum_{i=1}^{N} \delta(t_i) + \rho(t_N)$$

- t is an edge

Finite State Transducer: To translate one string to another, which support add, replace and remove a character. And they can also be weighted.

Finite state **acceptor**: allure + ing = allureing
Finite state **transducer**: allure + ing = alluring



Context-free Grammar: A more general approach than regular expression. Natural language is not context-free, can have cross-serial dependencies.

- Symbols: Can be terminal (individual word) or non-terminal (symbol such as POS tag).
- Production rules: Exact one non-terminal on the left-hand side and an ordered list of symbols on the right-hand side. It only depends on the left-hand side, not neighbors and ancestors.
- CFG parsing: Given a sentence and production rule, produce a valid tree.
- Constituents: The parts that sentences are broken into. Can have certain key properties: movement, substitution and coordination.

  (i) Movement

  - Constituents can be moved around sentences
    - Abigail gave [her brother] [a fish]
    - Abigail gave [a fish] to [her brother]
  - Contrast: [gave her], [brother a]

  (ii) Substitution

- Constituents can be substituted by other phrases of the same type
  - Max thanked [his older sister]
  - Max thanked [her]
- Contrast: [Max thanked], [thanked his]

(iii) Coordination

- Constituents can be conjoined with coordinators like *and* and *or*
  - [Abigail] and [her young brother] brought a fish
  - Abigail [bought a fish] and [gave it to Max]
  - Abigail [bought] and [greedily ate] a fish

We often use phases to describe a constituent, and a phase is determined by its head word.

- Sentence generation: Given terminal symbols, non-terminal symbols and production rules, generate a sentence based on the above rules.
- CFG tree: Terminals are leaves and non-terminals are internal nodes
- CYK algorithm:

- Bottom-up parsing
- Tests whether a string is valid given a CFG, without enumerating all possible parses
- Core idea: form small constituents first, and merge them into larger constituents
- Requirement: CFGs must be in **Chomsky Normal Forms**
- Convert grammar to Chomsky Normal Form (CNF)
- Fill in a parse table (left to right, bottom to top)
- Use table to derive parse
- S in top right corner of table = success!
- Convert result back to original grammar

- Chomsky normal form:
- Change grammar so all rules of form:
  - A → B C
  - A → a
- Convert rules of form A → B c into:
  - A → B X
  - X → c

- Convert rules A → B C D into:
  - A → B Y
  - Y → C D
  - E.g. VP → VP NP NP
    for ditransitive cases, "*sold [her] [the book]*"
- CNF disallows unary rules, A → B.
- Imagine NP → S; and S → NP … leads to infinitely many trees with same yield.
- Replace RHS non-terminal with its productions
  - A → B, B → *cat*, B → *dog*
  - A → *cat*, A → *dog*

- Parsing retrieval:

- S in the top-right corner of parse table indicates success
- To get parse(s), follow pointers back for each match

Probabilistic Context-free Grammar: Each production rule has a probability value to scale the importance of each production, aka conditional probability of the RHS given LHS. Thus, we can compare the likelihood of different parse trees by comparing probability.

- NP → DT NN [p = 0.45]
- NN → cat [p = 0.02]
- NN → leprechaun [p = 0.00001]
- …

- Stochastic Generation:

1. Start with S, the sentence symbol
2. Choose a rule with S as the LHS
   - **Randomly select a RHS** according to P(RHS | LHS)
     e.g., S → VP
   - Apply this rule, e.g., substitute VP for S
3. Repeat step 2 for each non-terminal in the string (here, VP)
4. Stop when no non-terminals remain

Gives us a tree, as before, with a sentence as the yield

- CYK algorithm: The idea is for each parsing step, if there are multiple possibilities, we always choose the one with larger probability.
- Parsing Retrieval:

- S in the top-right corner of parse table indicates success
- Retain back-pointer to best analysis
- To get parse(s), follow pointers back for each match
- Convert back from CNF by removing new non-terminals

Problems of CFG: Independence assumptions (solution is parent conditioning); Lack of lexical conditioning (solution is head lexicalization)

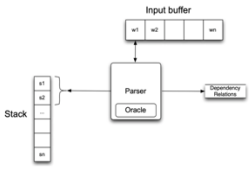
Dependency Grammar:

- Dependency grammar offers a simpler approach
  - describe **relations** between pairs of words
  - namely, between **heads** and **dependents**
  - e.g. (*prefer*, dobj, *flight*)
- Why:
- Deal better with languages that are morphologically rich and have a relatively free word order
  - CFG need a separate rule for each possible place a phrase can occur in
- Head-dependent relations similar to semantic relations between words
  - More useful for applications: coreference resolution, information extraction, etc
- Dependency Relations: Capture the grammatical relation between head word and supporting word
- Universal Dependency: A framework to create a set of dependency relations that are computationally useful and cross-lingual.

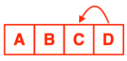
Clausal Argument Relations	**Description**
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

- Properties:
- Each word has a single head (parent)
- There is a single root node
- There is a unique path to each word from the root
- All arcs should be **projective**
- Dependency trees generated from constituency trees are **always projective**
- Projectivity: An arc is projective if there is a path from head to every word that lies between the head and the dependent. A dependency tree is projective if all arcs are projective, i.e. without crossing edges.
- Transition-based parsing: Can only handle projective dependency trees.

- Processes word from left to right
- Maintain two data structures
  - **Buffer**: input words yet to be processed
  - **Stack**: store words that are being processed



- At each step, perform one of the 3 actions:
  - **Shift:** move a word from buffer to stack
  - **Left-Arc:** assign current word as head of the previous word in stack



  - **Right-Arc:** assign previous word as head of current word in stack

- Parsing as a classification: We can use an oracle that tells the right action for each step and build a supervised model to mimic the actions of the oracle.
- Input:
  - Stack (top-2 elements: s_1 and s_2)
  - Buffer (first element: b_1)
- Output
  - 3 classes: *shift*, *left-arc*, or, *right-arc*
- Features
  - word (w), part-of-speech (t)

- Graph-based parsing:
- Given an input sentence, construct a fully-connected, weighted, directed graph
- **Vertices**: all words
- **Edges**: head-dependent arcs
- **Weight**: score based on training data (relation that is frequently observed receive a higher score)
- **Objective**: find the maximum spanning tree (Kruskal's algorithm)

The advantage of graph-based is it can produce non-projective trees, and the global score of the entire trees, it also avoids making greedy local decisions and it can capture long dependencies.

10. Machine Translation

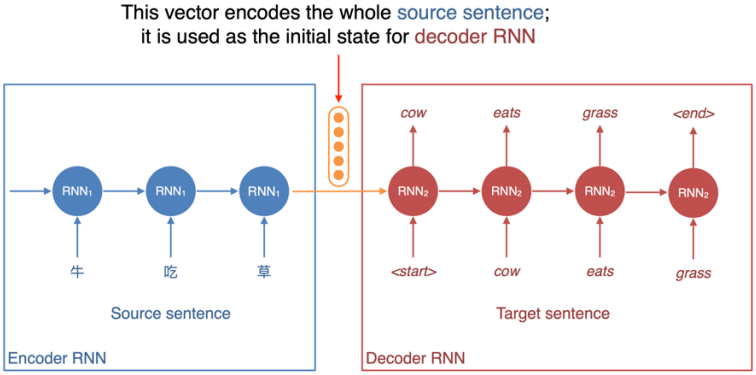
What: A task of translating from one source language to target language.

Why Difficult: Not just word-word mapping, but syntactical and semantic changes, idioms, inflection for gender, canse, etc.

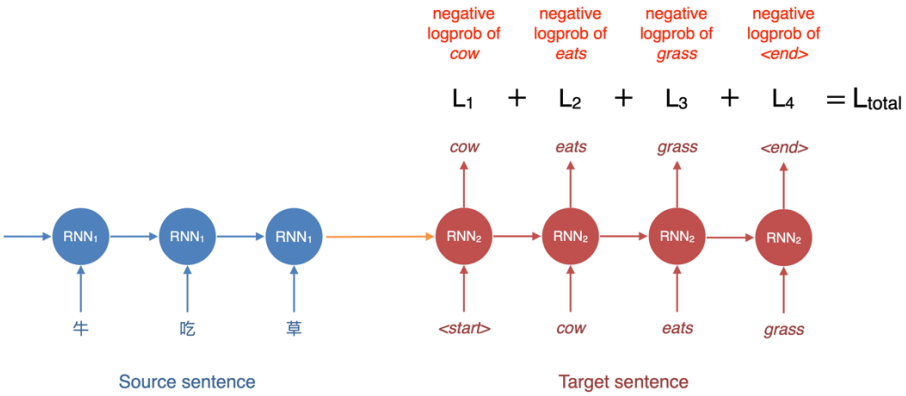
Statistical Machine Translation: Use the combination of language model $P(\boldsymbol{e})$ and translation model $P(\boldsymbol{f}|\boldsymbol{e})$. For translation model, we use parallel corpora, and words are not aligned in the parallel text. However, alignment is complicated. It includes the one-to-many alignment, many-to-one alignment, many-to-many alignment, dropped alignment. Statistical machine translation model is not accurate and less efficient than neural model.

Neural Machine Translation

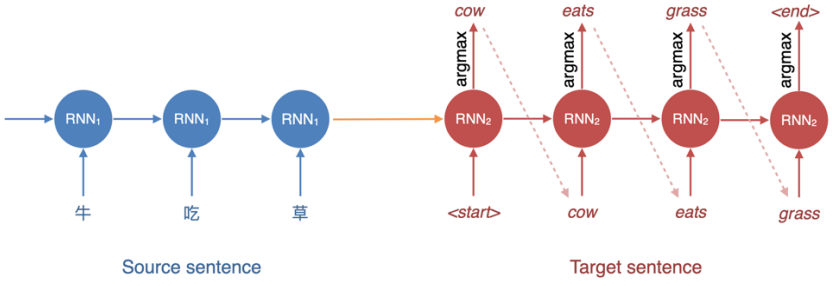
- Architecture: Two RNN. One is for encoding source language and another one is for decoding target language. The initial hidden state of the second RNN is the last hidden state of the first hidden state, known as Sequence-to-Sequence model. It is a conditional language model, where the first normal language model predicts the next word given the first word, and the second model is conditioned on the previous model.



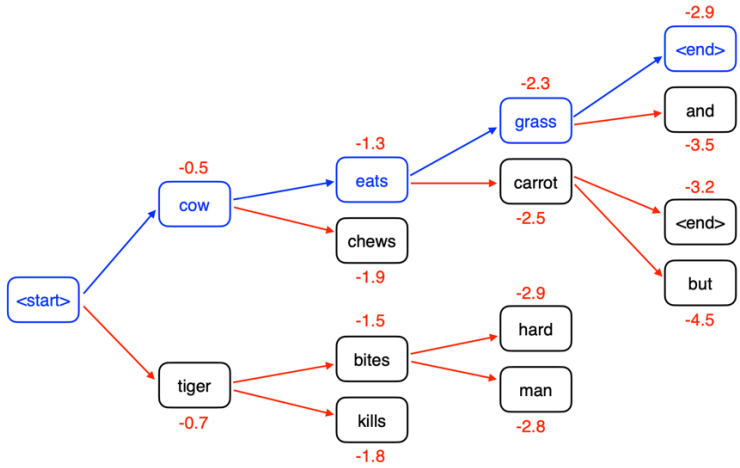
- Training: Training is just like language model, which is feed the right word, one step at a time, except the predicted word is conditioned on the source sentence.
- Loss: Same as the typical language model.



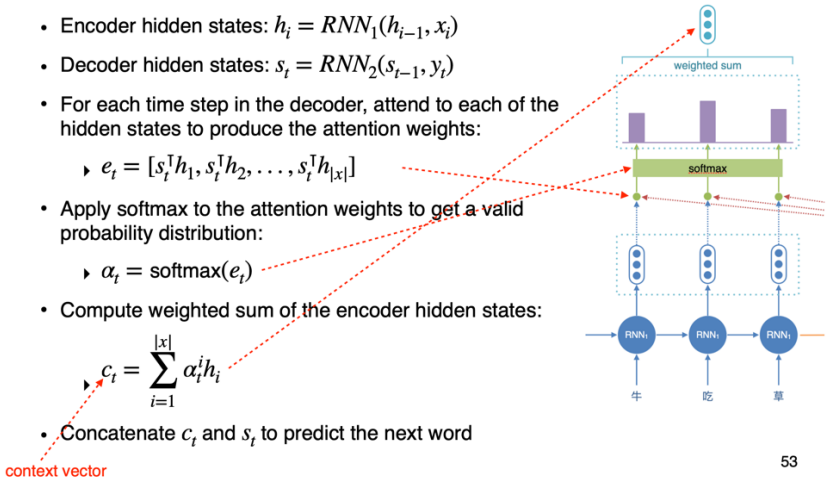
- Test: Test is also identical to language model.



- Beam Search: For each generation step, we keep track of the top-k words that produce the partial translation so far. The typical beam width is 5-10.



- Attention: For the decoder, it attends to the source language at every time step.



- Attention Variants:

- Dot product: $s_t^\intercal h_i$
- Bilinear: $s_t^\intercal W h_i$
- Additive: $v^\intercal \tanh(W_s s_t + W_h h_i)$

- BLEU
  - BLEU: compute n-gram overlap between "reference" translation and generated translation
  - Typically computed for 1 to 4-gram

$$\text{BLEU} = \text{BP} \times \exp\left(\frac{1}{N}\sum_{n}^{N} \log p_n\right)$$

$$p_n = \frac{\text{\# correct n-grams}}{\text{\# predicted n-grams}}$$

$$\text{BP} = \min\left(1, \frac{\text{output length}}{\text{reference length}}\right)$$

"Brevity Penalty" to penalise short outputs

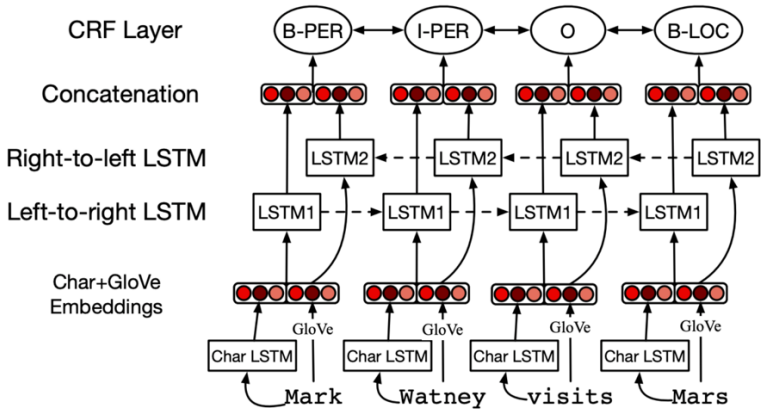
11. Information Extraction

- Given this:
  - *"Brasilia, the Brazilian capital, was founded in 1960."*
- Obtain this:
  - capital(Brazil, Brasilia)
  - founded(Brasilia, 1960)

Named Entity Recognition: Find entities in a sentence.

- Typical tags:
- **PER**: people, characters
- **ORG**: companies, sports teams
- **LOC**: regions, mountains, seas
- **GPE**: countries, states, provinces
  (in some tagset this is labelled as **LOC**)
- **FAC**: bridges, buildings, airports
- **VEH**: planes, trains, cars

- Problem: Some tags can be ambiguous, e.g. Washington can be a person or a location.
- IO tagging: B-XXX as the start of the entity, I-XXX as the subsequent entity, O as the non-entity token.
- Sequence labelling: Label whether the word is a named entity given the previous sequence. The features can be prefix/suffix or word shape
- Neural Method: Using LSTM.



Relation Extraction: To find the relations between named entities.

- Rule-based method: Use some key words as the evidence.

NP {, NP}* {,} (and|or) other NP_H	temples, treasuries, and other important civic buildings
NP_H such as {NP,}* {(or|and)} NP	red algae such as Gelidium
such NP_H as {NP,}* {(or|and)} NP	such authors as Herrick, Goldsmith, and Shakespeare
NP_H {,} including {NP,}* {(or|and)} NP	common-law countries, including Canada and England
NP_H {,} especially {NP}* {(or|and)} NP	European countries, especially France, England, and Spain

- Supervised method: The first step is to use a binary classifier to find if an entity pair is related or not. After finding all entity pair, use a multi-class classifier to obtain the specific relation. The features can be headword, word before/after the word, word type, bag-of-word in-between, etc.
- Semi-supervised method:

1. Given seed tuple: hub(Ryanair, Charleroi)
2. Find sentences containing terms in seed tuples
   - *Budget airline **Ryanair**, which uses **Charleroi** as a hub, scrapped all weekend flights out of the airport.*
3. Extract general patterns
   - [ORG], which uses [LOC] as a hub
4. Find new tuples with these patterns
   - hub(Jetstar, Avalon)
5. Add these new tuples to existing tuples and repeat step 2

- Semantic drift: Some generated pattern may be noisy or erroneous, so we should only accept patterns with high confidences.
- Distant supervision: Obtain new tuples directly from the source database, which has no risk of semantic drift.
- Evaluation: For NER, use F1-score; For relation extraction with known relation set, use F1-score, and for unknown relation set, use human evaluation.
- Other information extraction tasks: temporal expression extraction, event extraction, event ordering.

12. Question Answering

Definition: A task that automatically determines the answer for a neural language question. Typically, we only focus on factoid questions.

Factoid Questions: Questions with short precise answers.

- What war involved the battle of Chapultepec?
- What is the date of Boxing Day?
- What are some fragrant white climbing roses?
- What are tannins?

Non-factoid Questions: Questions which require a long answer, critical analysis, summary, calculations, etc.

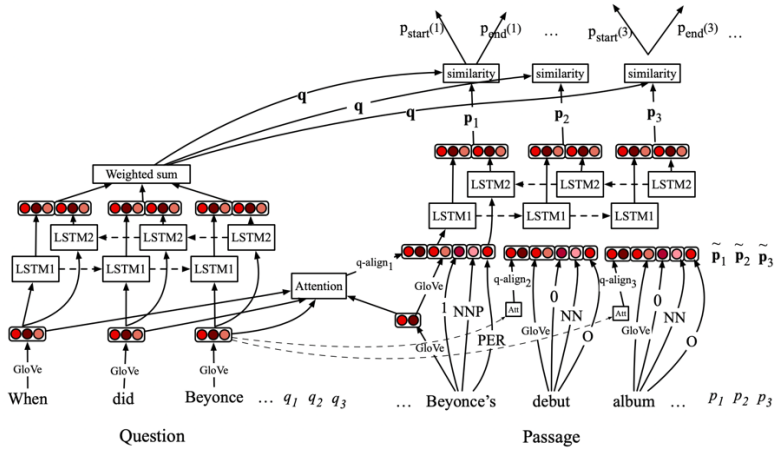
- Why is the date of Australia Day contentious?
- What is the angle 60 degrees in radians?

Information Retrieval-based QA: Given a query, search relevant document and extract answers with these documents.

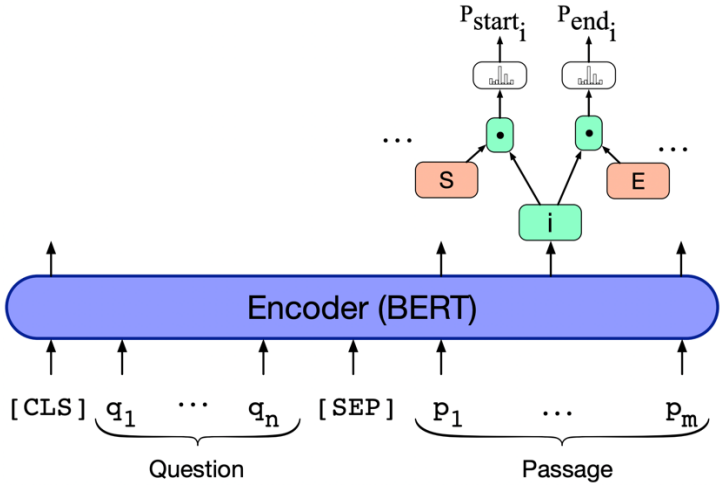
- Query processing: Discard non-context words (what, where, ?, etc); Formulate TF-IDF query; Identify entities and prioritize match.
- Answer types: Treat as a classification, which given a question, predict answer type.
- Retrieval: Find top-n documents matching query and find passages in these documents, ranked by scores. It should contain:
  - many instances of the question keywords
  - several named entities of the answer type
  - close proximity of these terms in the passage
  - high ranking by IR engine
- Answer Extraction: Find the span of the answer to the question.
- Reading Comprehension: Given a question and context passage, predict where the answer span starts and ends, i.e., compute the probability of token i being the start token and the end token.
- LSTM QA model:

(i) Feed the question tokens to a bi-directional LSTM, and aggregate outputs via the weighted sum to produce final question vector.

(ii) Passage embedding uses another bi-directional LSTM, but the input contains word embedding, POS tag feature, whether matching the question word feature, the attended question embedding where each passage token attends all words in questions. The passage token p_i and question token q is combined to form probability of start and end of the span $p_{start}(i)$, $p_{end}(i)$.



- BERT QA model:



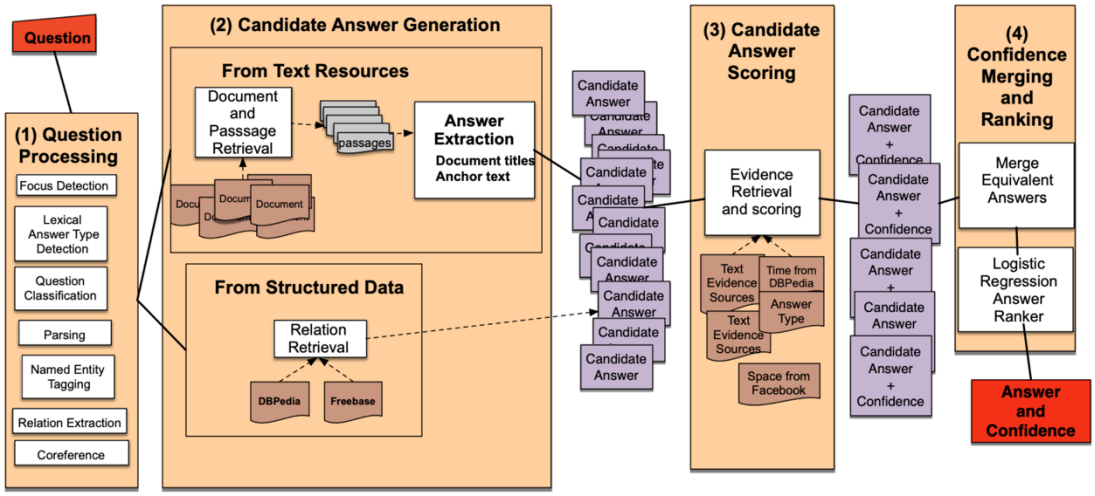
Knowledge-based QA: Build semantic representation of the query and query database of facts to find answers. Convert natural language sentence into a triple.

"When was Ada Lovelace born?" → `birth-year (Ada Lovelace, ?x)`

Then query knowledge base directly, by programming calculus or query language (SQL).

Hybrid QA:

- Waston:



QA Evaluation:

- IR: Mean Reciprocal Rank for systems returning matching passages or answer strings
  - E.g. system returns 4 passages for a query, first correct passage is the 3rd passage
  - MRR = ⅓
- MCTest: Accuracy
- SQuAD: Exact match of string against gold answer

13. Topic Modelling

What: Topic models learn common, overlapping themes in a document collection. The output of topic model is topics associated with each document and a list of words associated with each topic.

LSA Truncate: The vector U and V is truncated to k dimension. Each topic is the column vector of U matrix, and each document vector is the column vector of the V matrix. However, the issue is the vector has both positive and negative values which are hard to interpret.

Propabilistic LSA: The joint probability $P(w,d) = P(d)\sum_T P(w|t)P(t|d)$, where $P(w|t)$ is the word distribution for a topic and $P(t|d)$ is the topic distribution for a document, but the problem is that it is unable to infer topic distribution on new documents, and it needs to be retrained for new documents.

Latent Dirichlet Allocation: A Bayesian version of PLSA, which set the prior distribution of document-topic and topic-word distribution.

- Core idea: assume each document contains **a mix of topics**
- But the topic structure is **hidden (latent)**
- LDA infers the topic structure given the observed words and documents
- LDA produces soft clusters of documents (based on topic overlap), rather than hard clusters
- Given a trained LDA model, it can infer topics on new documents (not part of train data)

The input of LDA are collection of documents, bag-of-words, sometimes remove stop words, and do

lemmatization. The output is the distribution of words in each topic and distribution over topics of each document.

- Sample-based method

Training: Assign prior distribution α on topic-document distribution and β on topic-word distribution. Higher prior will have a flatter distribution and lower prior will have peaky distribution.

1. Randomly assign topics to all tokens in documents

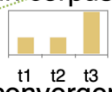
doc1	mouse: ?	cat: t3	rat: t2	chase: t1	mouse: t3
doc2	scroll: t1	mouse: t3	scroll: t3	scroll: t2	click: t2
doc3	tonight: t2	baseball: t1	tv: t2	exciting: t1	

2. Collect **topic-word and document-topic co-occurrence statistics** based on the assignments

	mouse	cat	scroll	tv	...
t1	1.01-1	0.01	1.01	0.01	
t2	0.01	0.01	1.01	1.01	
t3	2.01	1.01	1.01	0.01	

	t1	t2	t3
d1	2.1-1	1.1	2.1
d2	1.1	2.1	2.1
...			

3. Go through every word token in corpus and sample a new topic:
   - $P(t_i \mid w, d) \propto P(t_i \mid w)P(t_i \mid d)$



Need to de-allocate the current topic assignment and update the co-occurrence matrices before sampling

4. Go to step 2 and repeat until convergence

$$P(t_1 \mid w, d) = P(t_1 \mid \text{mouse}) \times P(t_1 \mid d_1)$$
$$\frac{0.01}{0.01 + 0.01 + 2.01} \times \frac{1.1}{1.1 + 1.1 + 2.1}$$

25

The hyperparameters to be tune is the number of topics. For broad topics, the heuristic is less than 10. For fine-grained topics, the heuristic is larger than 100.

Testing:

1. Randomly assign topics to all tokens in new/test documents

testdoc1	tiger: t2	cow: t1	cat: t3	tiger: t3	
testdoc2	football: t2	live: t2	men: t2	fun: t3	soccer: t1
testdoc3	news: t1	cnn: t3	tonight: t1		

2. Update document-topic matrix based on the assignments; but use the trained topic-word matrix (kept fixed)

from trained topic model

	mouse	cat	scroll	tv	...
t1	5.01	4.01	1.01	0.01	
t2	0.01	2.01	1.01	8.01	
t3	0.01	0.01	4.01	0.01	

	t1	t2	t3
td1	1.1	1.1	2.1
td2	1.1	3.1	1.1
...			

new matrix

3. Go through every word in the test documents and sample topics:
   - $P(t_i \mid w, d) \propto P(t_i \mid w)P(t_i \mid d)$

4. Go to step 2 and repeat until convergence

Evaluation: User perplexity metric.

$$\log L = \sum_W \sum_T \log P(w|t)P(t|d_w)$$

$$ppl = \exp(\frac{-\log L}{W})$$

- Issue: The issue is the perplexity is dependent on the number of topics. Often needs extrinsic evaluation, say downstream tasks.
- Topic Coherence: A intrinsic method that measures how coherent the generative topics are. Typically, a good topic model will generate more coherent topics.

- Word Intrusion: Insert a random word in a topic words and asks users to guess which one is the intruded word. If it is correct, the model is good.
- PMI: If word pairs within a topic has high PMI, the topic is coherent, and if most topics have high PMI, the topic model is good.

$$\text{PMI}(t) = \sum_{j=2}^{N} \sum_{i=1}^{j-1} \log \frac{P(w_i, w_j)}{P(w_i)P(w_j)}$$

Normalised PMI

$$\text{NPMI}(t) = \sum_{j=2}^{N} \sum_{i=1}^{j-1} \frac{\log \frac{P(w_i, w_j)}{P(w_i)P(w_j)}}{-\log P(w_i, w_j)}$$

Conditional probability

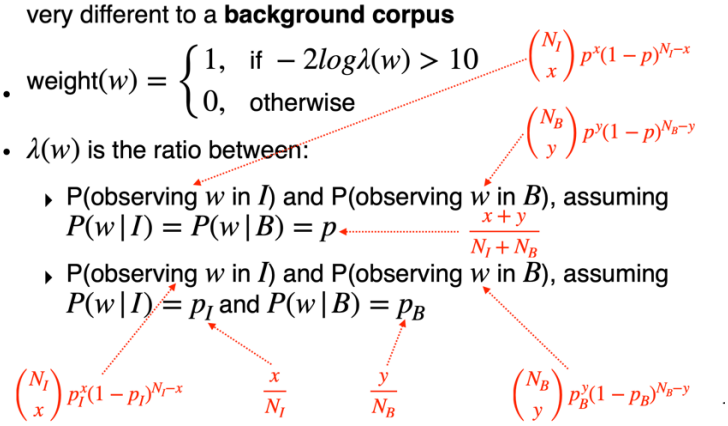
$$\text{LCP}(t) = \sum_{j=2}^{N} \sum_{i=1}^{j-1} \log \frac{P(w_i, w_j)}{P(w_i)}$$

14. Summarization

What: Distill the most important information from a text to produce shortened version. Can be extracted from single document or multiple document. The ways of summarization can be extractive, that is to select representative sentences from the document or abstractive, which summarize the content by paraphrasing. The goal of summarization can be generic or query-based.

Single Document Summarization:

- Content selection: Select what sentences to extract from the document. For single document summarization, information ordering is not necessary. The common method is supervised learning.
  (i) TF-IDF to find the most salient words in a documentation, but we need to remove function words and stop words.
  (ii) Log likelihood ratio: A word is salient if its probability in the input corpus is very different to a background corpus.
  - Intuition: a word is salient if its probability in the **input corpus** is very different to a **background corpus**
  - $\text{weight}(w) = \begin{cases} 1, & \text{if } -2log\lambda(w) > 10 \\ 0, & \text{otherwise} \end{cases}$
  - $\lambda(w)$ is the ratio between:
    - P(observing w in I) and P(observing w in B), assuming $P(w|I) = P(w|B) = p$ $\leftarrow \frac{x+y}{N_I+N_B}$
    - P(observing w in I) and P(observing w in B), assuming $P(w|I) = p_I$ and $P(w|B) = p_B$



$$\text{weight}(s) = \frac{1}{|S|} \sum_{w \in S} \text{weight}(w)$$

Only consider non-stop words in S

(iii) Sentence centrality: Choose sentences that are closer to other sentences. Use TF-IDF to represent a sentence, and use cosine similarity to measure distance.

$$\text{centrality}(s) = \frac{1}{\#\text{sent}} \sum_{s'} \cos_{tfidf}(s, s')$$

(iv) RST parsing to find the central sentence.

Multi-Document Summarization:

- Content selection: Similar to single-document, but ignore any redundant sentences. The maximum marginal relevance is a metric to evaluate the novelty of an added sentence, and penalize a candidate sentence, if it is similar to expected sentence.

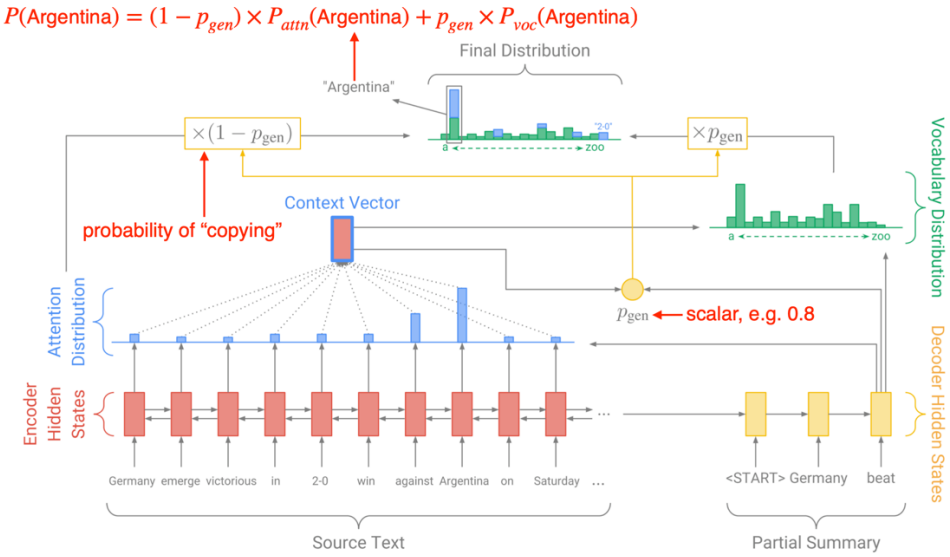
$$\text{MMR-penalty}(s) = \lambda \max_{s_i \in \mathcal{S}} sim(s, s_i)$$

- Information ordering: Two method: chronological ordering, and coherence ordering, where the latter order the extracted sentences in a way that makes adjacent sentences similar.
- Sentence realization:

- Make sure entities are referred coherently
  - Full name at first mention
  - Last name at subsequent mentions
- Apply coreference methods to first extract names
- Write rules to clean up

Abstract Summarization: Use encoder-decoder model, but it may cause information bottleneck problem, so we can use attention mechanism, to attend each word in the source sentence, but out-of-vocabulary is also an issue to be handled.

Encoder-decoder with Attention and Copying: The probability of a word is composed of two probabilities: one is the probability of copying with attention mechanism, and another one is the probability of generation to attached with an outside vocabulary.



Evaluation: Similar to BLEU, ROUGH evaluates the degree of word overlap between generated summary and reference summary.

$$\text{ROUGE-2} = \frac{\sum_{S \in \{ReferenceSummaries\}} \sum_{bigram \in S} \text{Count}_{\text{match}}(bigram)}{\sum_{S \in \{ReferenceSummaries\}} \sum_{bigram \in S} \text{Count}(bigram)}$$

- **Ref 1:** *Water spinach is a green leafy vegetable grown in the tropics.*
- **Ref 2:** *Water spinach is a commonly eaten leaf vegetable of Asia.*
- **Generated summary:** *Water spinach is a leaf vegetable commonly eaten in tropical areas of Asia.*
- *ROUGE-2* $= \frac{3 + 6}{10 + 9}$