Revision notes - CS4248

Ma Hongqiang

November 18, 2019

Contents

1 Regular Expressions and Automata

Definition 1.1 (Regular Expression)**.**
Regular Expression is a formula for specifying a set of strings.

Definition 1.2 (RE Patterns)**.**

- `*`: zero or more occurrences of the previous char or expression
- `+`: one or more occurrences of the previous char or expression
- `?`: exactly zero or one occurrence of the previous char or expression
- `{n}`: *n* occurrences of the previous char or expression
- `{n,m}`: from *n* to *m* occurrences of previous char or expression
- `{n,}`: at least *n* occurrences of the previous char or expression
- `.`: one occurence of any character
-
  : an asterisk
-
  `.`: a period
-
  `?`: a question mark
-
  `n`: a newline
-
  `t`: a tab
- `[ArrayOfChars]`: any of the char in ArrayOfChars
- `[StartingChar-EndingChar]`: any of the char from StartingChar to EndingChar, inclusive
- `[ÂSetOfChar]`: not any of the character in the SetOfChar
- `^`: Start of a line
- `$`: End of a line
-
  `d`: Expands to `[0-9]`

- `D`: Expands to `[Ô-9]`, any non-digit

- `s`: white space

- `S`: non-whitespace

Operator Precedence Hierarchy (highest to lowest):

- Parenthesis: `()`
- Counters: `* + ?{ }`
- Sequences/anchors
- Disjunction: `|`

Definition 1.3 (Finite State Automata)**.**

```
function D-RECOGNIZE(tape, machine) returns accept or reject
    index ← Beginning of tape
    current-state ← Initial state of machine
    loop
        if End of input has been reached then
            if current-state is an accept state then
                return accept
            else
                return reject
        elsif transition-table[current-state,tape[index]] is empty then
            return reject
        else
            current-state ← transition-table[current-state,tape[index]]
            index ← index + 1
end
```

Definition 1.4 (Finite State Automata)**.**

- Q: a finite set of N states $q_0, \ldots, q_{N-1}$
- Σ: a finite input alphabet of symbols
- q_0: the start state
- F: set of final states, $F \subseteq Q$
- $\delta(q, i)$: transition function between states. Given a state $q \in Q$ and input symbol $i \in \Sigma$, $\delta(q, i)$ returns a new state $q' \in Q$.

Definition 1.5 (Non-deterministic FSA)**.**
A non-deterministic FSA can have the following behavour:

- At a state q, transit to more than one state given the same symbol.
- At a state q, transit to another state without consuming any symbol: ϵ transition.

NFSA accepts an input string if there is **at least some** path in the NFSA that leads to an accepting state **and** exhausting the input string.

2 Words

Definition 2.1 (Morpheme)**.**
Morpheme refers to the minimal meaning-bearing unit in a language.

There are 2 classes of morphemes: stems and affixes. Affix can be prefix, suffix, infix and circumfix.
There are also 2 classes of behaviour in morphology:

- Inflection
  - Combine a stem and an affix to form a word in the same classes as stem.
- Derivation
  - Combine a stem and an affix to form a word in a different class
  - Harder to predict the meaning of the derived form

There are attempts in using algorithm to strip affixes from stems, like Porter Stemming Algorithm.

Definition 2.2 (Tokenization)**.**
Text tokenization converts text to a more convenient, standard form. It could involve:

- Clitic contractions: we're$\rightarrow$ we are.
- Choose a single normalized form for words with multiple forms.
- Case folding: convert all words to lower case.
- and more.

For Penn Treebank Tokenization Standard, it does the following

- Seperate out clitics: doesn't -¿ does n't
- Keep hyphenated words together
- Separate out all punctuation symbols

We may encounter spelling error in our text corpus, and we may do one of the following:

- Non-word error detection
- Isolated-word error correction
- Context-sensitive error detection and correction

It is recognized that most misspelled words in human typewritten text are single-error misspellings:

- Insertion

- Deletion
- Substitution
- Transposition

However, we also note that it is harder to detect cognitive error than typographic error, as cognitive error does not result in a non-word.
In the following, we assume that we will detect and correct non-word spelling errors, without using context. We will first propose candidates, and score them using

$$\hat{c} = \arg\max_{c \in C} P(c \mid o) = \arg\max_{c \in C} P(o \mid c)P(c)$$

where c is a class(word), C a set of classes and o observation, i.e. the misspelled word.
It is of our interest to evaluate $P(o \mid c)$ and $P(c)$.
$P(c)$ can be calculated by

$$P(c) = \frac{C(c)}{N}$$

if not smoothed, and

$$P(c) = \frac{C(c) + \lambda}{N + B\lambda}$$

if smoothed. Here $N = \sum_{i=1}^{B} C(c_i)$, where B is the number of distinct c's in C, and λ is a small number.
For $P(o|c)$, an simplistic approach is to use

$$P(o \mid c) = \begin{cases} \frac{sub[m,l]}{C(l)} & \text{if substitution} \\ \frac{trans[k,l]}{C(kl)} & \text{if transposition} \\ \frac{ins[l,m]}{C(l)} & \text{if insertion} \\ \frac{del[k,l]}{C(kl)} & \text{if deletion} \end{cases}$$

where

- $sub[m, l]$: number of times the correct letter l is typed as m
- $trans[k, l]$: number of times the correct letter sequence kl is typed as lk
- $ins[l, m]$: number of times the extraneous letter m was inserted after l
- $del[k, l]$: number of times the letter l was deleted from the correct letter sequence kl.

Theorem 2.1 (Minimum Edit Distance)**.**
Determine the number of edits(insertion/deletion).
Use dynamic programming to deal with this. Idea is

$\texttt{distance}[i, j] = \min(\texttt{distance}[i-1, j]+1, \texttt{distance}[i-1, j-1]+a[i] == b[j]?0 : 2), \texttt{distance}[i, k-1]+1$

3 N gram

For n-gram language model, we want to use the previous $n-1$ words to predict the next word, so we are concerned about the probability

$$P(w_n \mid w_1, \dots, w_{n-1})$$

and

$$P(w_1, \dots, w_n)$$

Definition 3.1 (Types, Tokens)**.**
Number of tokens N refers to total number of running words.
Number of types B refers to number of distinct words in a corpus. It is obvious that

$$N = C(c_1) + \dots + C(c_B)$$

Theorem 3.1 (Evaluating n-gram probability)**.**

$$\begin{aligned}
&P(w_1, w_2, \dots, w_N) \\
=&P(w_1)P(w_2 \mid w_1) \cdots P(w_N \mid w_1, \dots, w_{N-1}) \\
=&\prod_{k=1}^{N} P(w_k \mid w_1, \dots w_{k-1})
\end{aligned}$$

Theorem 3.2 (n-gram approximation)**.**
We can assume that w_k only depends on its previous $n-1$ words, so

$$P(w_k \mid w_1, \dots, w_{k-1}) \approx P(w_k \mid w_{k-(n-1)}, \dots, w_{k-1})$$

Especially, bi-gram(2-gram) approximation refers to

$$P(w_1, \dots, w_N) \approx \prod_{k=1}^{N} P(w_k \mid w_{k-1})$$

For such 2-gram estimation, we need to evaluate $P(w_k \mid w_{k-1})$ on the RHS. We can do

$$P(w_k \mid w_{k-1}) = \frac{C(w_{k-1}w_k)}{C(w_{k-1})}$$

More generally, we have

$$P(w_k \mid w_{k-(n-1)}, \dots, w_{k-1}) = \frac{C(w_{k-(n-1)}, \dots, w_k)}{C(w_{k-(n-1)}, \dots, w_{k-1})}$$

For training, we first choose a vocabulary. For any out-of-vocabulary word, we replace it with the unknown word $< UNK >$, and estimate the probability of $< UNK >$ like a regular word.

Definition 3.2 (Perplexity).
Let m be a language model. Perplexity(PP)is defined as

$$PP = m(w_1, \dots, w_n)^{-\frac{1}{n}}$$

A better language model has a lower perplexity.

Theorem 3.3 (Add-1 Smoothed Bigram Probability).
We can add-1 smooth the bigram probability

$$P(w \mid w_0) = \frac{C(w_0 w) + 1}{C(w_0) + V} := \frac{C^*(w_0 w)}{C(w_0)}$$

where $C^*(w_0 w) = (C(w_0 w) + 1)\frac{C(w_0)}{C(w_0)+V}$. We define discount d as

$$d := \frac{C^*(w_0 w)}{C(w_0 w)}$$

However, such smoothing is not ideal, since it will dilute the probability of existing bigrams by a lot, due to large value of V.

Theorem 3.4 (Witten-Bell Smoothed Bigram Probability).
Let $C(w_0) = \sum_{i=1}^{T} C(w_0 w_i)$, where T is the number of distinct word types *following* w_0.
Define $T + Z = V$, so Z is the number of unseen bigram types following w_0.
We further assume $T(w_0)$ is the total *count* of all unseen bigrams. Under this assumption,

$$P(w \mid w_0) = \frac{C(w_0 w)}{C(w_0) + T(w_0)} \text{ if } C(w_0 w) > 0$$

and

$$P(w \mid w_0) = \frac{T(w_0)}{Z(w_0)(C(w_0) + T(w_0))} \text{ if } C(w_0 w) = 0$$

Theorem 3.5 (Estimating n-gram probability using interpolation).
After we get $P(w_0)$ and $P(w_0 \mid w_{-1})$ from training set, we can use development set to get

$$\hat{P}(w_0 \mid w_{-1}) = \lambda_1 P(w_0 \mid w_{-1}) + \lambda_2 P(w_0)$$

where $\sum_i \lambda_i = 1$, and λ_i is determined by maximising $\hat{P}$ in development set.
However, we may encounter the problem of not seeing $w_{-1}w_0$, so $P(w_0 \mid w_{-1})$ is unknown.
We can estimate it by using **backoff**:

$$\hat{P}(w_0 \mid w_{-1}) = \begin{cases} \tilde{P}(w_0 \mid w_1) \text{ if } C(w_{-1}w_0) > 0 \\ \alpha(w_{-1}) \cdot \tilde{P}(w_0) \text{ if } C(w_{-1}w_0) = 0 \end{cases}$$

where $\tilde{P}$ is the discounted probability.
As such, we can determine $\alpha(w_{-1})$ as

$$\alpha(w_{-1}) = \frac{1 - \sum_{w_0 : C(w_{-1}w_0) > 0} \tilde{P}(w_0 \mid w_{-1})}{1 - \sum_{w_0 : C(w_{-1}w_0) > 0} \tilde{P}(w_0)}$$

Theorem 3.6 (Kneser-Ney Smoothing for Bigrams).

$$P_{KN}(w_0 \mid w_{-1}) = \begin{cases} \frac{C(w_{-1}w_0)-D}{C(w_{-1})} \ \textit{if}\ C(w_{-1}w_0) > 0 \\ \alpha(w_{-1}) \frac{|\{w'_{-1}:C(w'_{-1}w_0)>0\}|}{\sum_w |\{w'_{-1}:C(w'_{-1}w)>0\}} \ \textit{if}\ C(w_{-1}w_0) = 0 \end{cases}$$

Definition 3.3 (Entropy).
Entropy measures uncertainty. Entropy H of a random variable X is

$$H(X) = -\sum_{x\in X} p(x)\log_2 p(x)$$

Definition 3.4 (Entropy of a Sequence).
Entropy of a random variable ranging over all finite sequences of words of length n in a language L is

$$H(w_1,\ldots,w_n) = -\sum_{W_1^n\in L} p(W_1^n)\log_2 p(W_1^n)$$

And the entropy rate(per-word entropy) is $\frac{1}{n}H(W_1^n)$.

Definition 3.5 (Entropy of a Language).
$H(L) = \lim_{n\to\infty}\frac{1}{n}H(w_1,\ldots,w_n)$.
By SMB theorem,

$$H(L) = \lim_{n\to\infty} -\frac{1}{n}\log_2 p(w_1,\ldots,w_n)$$

Definition 3.6 (Cross Entropy).
Let p be actual probability distribution, and m a model of p. We have the cross-entropy of m on p to be $H(p,m) = \lim_{n\to\infty} -\frac{1}{n}\sum_{W\in L} p(w_1,\ldots,w_n)\log_2 m(w_1,\ldots,w_n)$.
By SMB theorem, we have

$$H(p,m) = \lim_{n\to\infty} -\frac{1}{n}\log_2 m(w_1,\ldots,w_n)$$

It is guaranteed that $H(p) \le H(p,m)$, and difference is a measure of accuracy of the model. We have, the perplexity of the model to be 2^H.

4 Part-of-speech Tagging

Part-of-speech(POS) gives information about a word and *its neighbours*. Also, the same word in different POS may be pronouced differently.
POS can help decide the correct morphological affixes, provide useful information to other NLP components, and help to determine the structure of a sentence.

Definition 4.1 (English POS)**.**
Classes of POS consists of open class and closed class.
Opem class consists of noun, verb, adjective and adverb.
Closed class include the rest, i.e. determiner, preposition, conjunction etc.

Definition 4.2 (Noun)**.**
Noun refers to people, places, things. It can occur with determiners, take possessives, and can occur in plural form.
There is proper nouns, which are capitalized names, and also common nouns.

Definition 4.3 (Verb)**.**
Verb refers to actions and processes. It can have the following morphological forms:

- Non-3rd-person-singular
- 3rd-person-singular
- Present participle
- Past participle

There is a subclass of verbs called **auxilliaries** which is of closed class.

Definition 4.4 (Adjective, adverb)**.**
Adjective describe properties or qualities. Adverbs can be modifying something, directional, locative, temporal, degree or manner.

Definition 4.5 (Closed Class POS)**.**

- Prepositions: on, under, over.
  Occur before noun phrases.
- Determiners: a, an, the. Often start a noun phrase.
  Articles is a subtype of determiner, which can be either indefinite or definite.
- Pronouns: she, I, others.
  An abbrevation for referring to some noun phrase or entit or event. There are personal pronouns, possessive pronouns and Wh-pronouns.
- Conjunctions: and, but, if.
  Join 2 phrases, clauses or sentences. There are coordinating conjunctions like mentioned above, and subordinating conjunctions like that.

- Auxiliary verbs: can, may, should
- Particles: off, out, on.
  Combine with a verb to behave as a semantic unit.
- Numerals: one, two first, second

There are also auxiliary verbs which are closed. They provide semantic features of the main verb. Examples are be, do, have, can etc.
Other closed classes include:

- Existential there
- Interjections
- Negatives
- Politeness markers
- Greetings

Definition 4.6 (POS Tagging)**.**
POS tagging is a task which inputs a sentence S, and output one single best POS tag for each word in S.

POS Tagging can be done using rule-based tagging, or stochastic HMM tagging.

Theorem 4.1 (Rule Based Tagging)**.**
Rule based tagging happens in 2 stages.

1. Use a dictionary to assign each word a list of potential POS.
2. Use a large list of hand-written disambiguation rules to narrow down to a single best POS for each word.

Theorem 4.2 (Stochastic POS Tagging)**.**
Stochastic POS Tagging tries to obtain the best tagging by

$$\hat{T} = \arg\max_T P(T \mid W) = \arg\max_T P(T, W)$$

where $P(T, W)$ is defined as

$$\begin{aligned} P(T, W) :&= P(\langle s \rangle, t_1, w_1, \ldots, t_T, w_T, \langle /s \rangle) \\ &= P(\langle s \rangle) P(t_1 \mid \langle s \rangle) P(w_1 \mid \langle s \rangle, t_1) \ldots \end{aligned}$$

Here, we do the following approximation

- $P(\langle s \rangle) = 1$.
- $P(t_1 \mid \langle s \rangle, t_1, w_1, \ldots, t_{i-1}, w_{i-1}) \approx P(t_i \mid t_{i-1})$

- $P(w_i \mid \langle s \rangle, t_1, w_1, \ldots, t_{i-1}, w_{i-1}, t_i) \approx P(w_i \mid t_i)$.
- $P(\langle /s \rangle \mid \langle s \rangle, t_1, w_1, \ldots, t_T, w_T) \approx P(\langle /s \rangle \mid t_T)$.

As such, we have

$$P(T, W) = (\prod_{i=1}^{T} P(t_i \mid t_{i-1}) \cdot P(w_i \mid t_i))P(\langle /s \rangle \mid t_T)$$

So we need to select t_1, tot_T to maximise the above.

Theorem 4.3 (Hidden Markov Models)**.**
Given the assumptions above, we can use Hidden Markov Model to compute the probability above by treating tags as states, and words as observations
We define $Q = q_1 \ldots q_N$ to be a set of N states. Let $A = (a_{ij})_{1 \leq i,j \leq n}$ be a transition probability matrix. Let $O = o_1 \ldots o_T$ to be a sequence of T observations, each one drawn from a vocabulary $V = v_1, \ldots, v_V$.
Let $B = b_i(o_t)$ to be a sequence of **obseration likelihood**, each expressing the probability of an observation o_t being generated from a state i.
Let q_0 and q_F to be a special start, and end state, together with $a_{01} \ldots a_{0n}$ out of start state, and $a_{1F} \ldots a_{nF}$ into the end state.
We clearly have

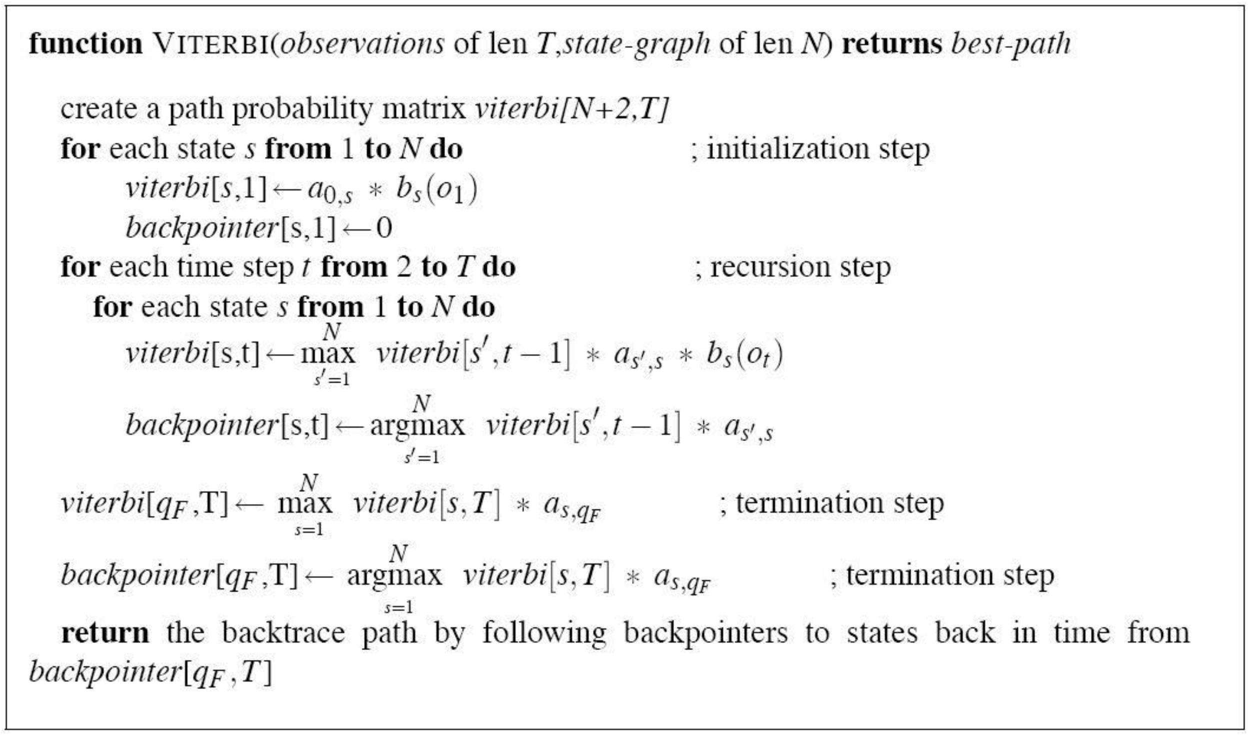
$$\sum_{j=1}^{n} a_{ij} = 1$$

and

$$\sum_{o_t \in V} b_i(o_t) = 1$$

Here, we want to determine the most probable state sequence. We can do so in $O(T \cdot N^2)$ using Viterbi algorithm. Viterbi algorithm relies on the fact:

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



```
function VITERBI(observations of len T,state-graph of len N) returns best-path

  create a path probability matrix viterbi[N+2,T]
  for each state s from 1 to N do                              ; initialization step
      $viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
      $backpointer[s,1] \leftarrow 0$
  for each time step t from 2 to T do                          ; recursion step
    for each state s from 1 to N do
      $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
      $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$
  $viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$              ; termination step
  $backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$     ; termination step
  return the backtrace path by following backpointers to states back in time from
$backpointer[q_F,T]$
```

5 Linear Models

Definition 5.1 (Linear Model)**.**
A linear model refers to $f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{W} + \mathbf{b}$, where $\mathbf{x}$ is the input, and $W \in \mathbb{R}^{d_{in} \times d_{out}}, b \in \mathbb{R}^{d_{out}}$ are parameter of the linear model.

Definition 5.2 (Binary Classification)**.**
If $b \in \mathbb{R}^1$, by letting $\hat{y} := \text{sign}(f(\mathbf{x}))$, we can use such a linear model for a binary classification. Note, $\text{sign}(x)$ returns 1 if $x \geq 0$ and -1 otherwise.

Definition 5.3 (Log-linear Binary Classification)**.**
Instead of such an absolute binary classification using sign function, we can use sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, and

$$P(\hat{y} = 1 \mid \mathbf{x}) = \sigma(f(\mathbf{x})) \quad P(\hat{y} = 0 \mid \mathbf{x}) = 1 - \sigma(f(\mathbf{x}))$$

Definition 5.4 (Multi-class Classification)**.**
We can classify an $\mathbf{x}$ to one of the many possible(> 2) classes.
To do so, we can use a linear classifier, by letting $\hat{\mathbf{y}} := f(\mathbf{x}) = \mathbf{x}\mathbf{W} + \mathbf{b}$, where $\mathbf{y}$ represents the probability of different classes.
Therefore, the prediction is $\arg\max_i \hat{\mathbf{y}}_i$.

We can also use **softmax** function defined as

$$\text{softmax}(\mathbf{x})_i = \frac{e^{\mathbf{x}_i}}{\sum_j e^{\mathbf{x}_i}}$$

as a generator of a probability from $\hat{y}$.
Let $\hat{\mathbf{y}} = \text{softmax}(\mathbf{x}\mathbf{W} + \mathbf{b})$. Then $\hat{\mathbf{y}}_i$ is the probability of class i.

Definition 5.5 (Training as Optimization)**.**
We hope to find a function f, such that predictions $\hat{\mathbf{y}} := f(\mathbf{x})$ is accurate over the training set. To do so, we need to have a loss function $L(\hat{\mathbf{y}}, \mathbf{y})$ to quantify the loss suffered due to inaccurate prediction $\hat{\mathbf{y}}$.
As such we need to find parameters Θ that minimize the sum of loss fucntion and regularization term:

$$\hat{\Theta} := \arg\min_{\Theta}(\frac{1}{n}\sum_{i=1}^{n} L(f(\mathbf{x}_i; \Theta), \mathbf{y}_i) + \lambda R(\Theta))$$

Regularization is to control the complexity of the parameter values and avoid overfitting. λ can be used to control the extent of regularization and can be set manually based on development set.

Following are some common loss functions

Definition 5.6 (Binary Cross-entropy loss)**.**
Binary Cross-entropy loss is used in binary classification with conditional probability output, where $y \in 0, 1$ and $\hat{y} \in (0, 1)$:

$$L_{\text{logistic}}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

This loss function maximizes the log conditional probability $\log P(y \mid \mathbf{x})$ for each training example $(\mathbf{x}, y)$.

Definition 5.7 (Categorical Cross-entropy Loss)**.**
Categorical cross-entropy loss is used on $\mathbf{y} := (\mathbf{y}_1, \ldots, \mathbf{y}_n)$, a vector representing the true multinomial distribution over the labels 1 to n, and $\hat{\mathbf{y}} := (\hat{y}_1, \ldots, \hat{y}_n)$, the classifier's output transformed by **softmax**:

$$L_{\text{cross-entropy}}(\hat{\mathbf{y}}, \mathbf{y}) := -\sum_i \mathbf{y}_i \log(\hat{\mathbf{y}}_i)$$

For hard classification problem with a single correct class, then we only will have 1 terms on the RHS.

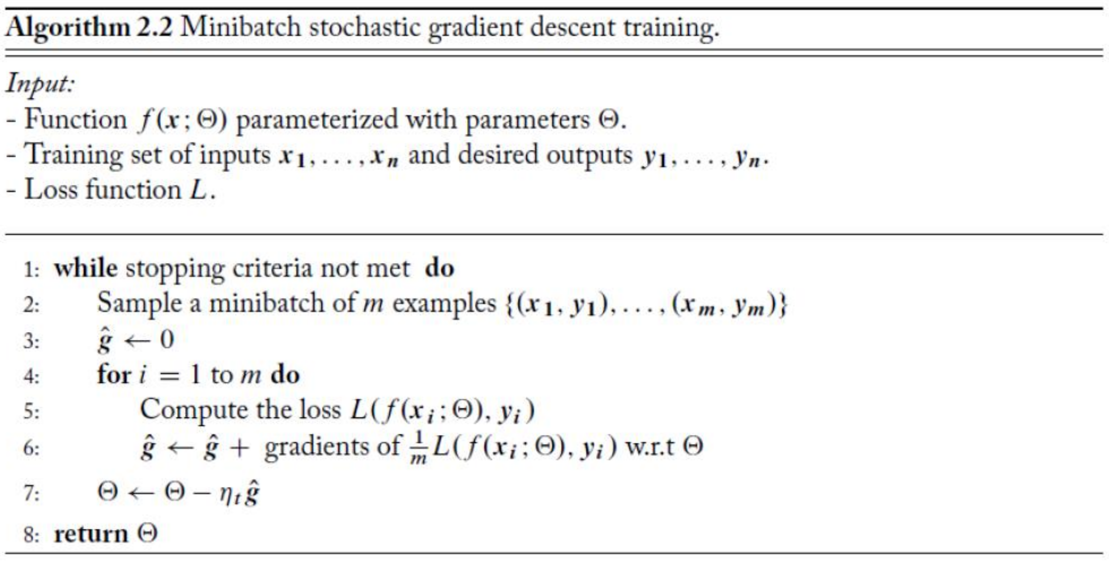
Definition 5.8 (Ranking Loss)**.**
In the event of only having positive training examples, and generating negative training examples by corrupting positive training examples, we need to aim for $f(\mathbf{x}) - f(\mathbf{x}') > 1$, where $\mathbf{x}$ positive and $\mathbf{x}'$ corrupted. We can define the loss function as such to achieve this training goal:

$$L_{\text{ranking}}(\mathbf{x}, \mathbf{x}') = \max\{0, 1 - (f(\mathbf{x}) - f(\mathbf{x}'))\}$$

And following are some regularisation technique.

Definition 5.9 (Common Regularization Functions)**.**

- *L_2 regularization:* $R_{L_2}(\mathbf{W}) = ||W||_2^2 := \sum_{i,j} \mathbf{W}_{ij}^2$
- *L_1 regularization:* $R_{L_1}(\mathbf{W}) = ||W||_1 = \sum_{i,j} |\mathbf{W}_{ij}|$
- *Elastic Net:* $R_{\textit{elastic-net}}(\mathbf{W}) = \lambda_1 R_{L_1}(\mathbf{W}) + \lambda_2 R_{L_2}(\mathbf{W})$



Algorithm 2.2 Minibatch stochastic gradient descent training.

Input:
- Function $f(x;\Theta)$ parameterized with parameters Θ.
- Training set of inputs $x_1, \ldots, x_n$ and desired outputs $y_1, \ldots, y_n$.
- Loss function L.

```
1: while stopping criteria not met do
2:     Sample a minibatch of $m$ examples $\{(x_1, y_1), \ldots, (x_m, y_m)\}$
3:     $\hat{g} \leftarrow 0$
4:     for $i = 1$ to $m$ do
5:         Compute the loss $L(f(x_i;\Theta), y_i)$
6:         $\hat{g} \leftarrow \hat{g} +$ gradients of $\frac{1}{m}L(f(x_i;\Theta), y_i)$ w.r.t $\Theta$
7:     $\Theta \leftarrow \Theta - \eta_t \hat{g}$
8: return $\Theta$
```

Definition 5.10 (Gradient Descent)**.**

6 Feed-forward Neural Network

Linear Model is not able to classify, without loss, a non-linear separable function. To resolve this work, we can apply to $\mathbf{x}$ a **non-linear input transformation** ϕ, such that

$$\hat{\mathbf{y}} = \phi(\mathbf{x})\mathbf{W} + \mathbf{b}$$

can be linearly separated. One of such function can be ReLU.

Definition 6.1 (Multilayer perceptron)**.**
A multilayer perceptron with k hidden layers is of the following form:

$$NN_k(\mathbf{x}) = \mathbf{h}^k\mathbf{W}^{k+1} + \mathbf{b}^3$$

with

$$\mathbf{h}^i = g^i(\mathbf{h}^{i-1}\mathbf{W}^i + \mathbf{b}^i)$$

and $h^0 = \mathbf{x}$.
We call g^i activation function. Common activation functions can be sigmoid, tanh $:= \frac{e^{2x}-1}{e^{2x}+1}$,
hard tanh $:= \begin{cases} -1 & x < -1 \\ 1 & x > 1 \\ x & \text{otherwise} \end{cases}$, ReLU$:= \max\{0, x\}$, etc.

Similarly, as linear model, feed-forward neural network can also have binary classification, k-class classfication and softmax classfication.

A neural network can fit towards any function. To prevent a neural network from overfitting, we can randomly dropping half of the neurons in the neural network during stochastic gradient descent, i.e. after each layer of activation function.

For pracicticalities, there are a few point to note:

- Initialization: We can initialize a random weight matrix $\mathbf{W} \sim U(-\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}, +\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}})$.
- We can have different random initializations and pick the best.
- We can combine multiple models in prediction, for example, through majority vote.
- For vanishing gradients, we can use specialized architecture such as LSTM.
- For exploding gradients, we can do $\hat{g} := \min(\hat{g}, \frac{\text{threshold}}{||\hat{g}||}\hat{g})$.
- We need to experiment with learning rate.

7 Neural Language Model

Language Modelling is to assign a probability to a sequence of words, or assign a probability of a word following a sequence of words.
N-grams model requires intricate and manually designed smoothing schemes, and it is harder to scale to larger n-gram models, as number of observed n-gram grows at least multiplicatively when n increases by 1. Also, n-gram has the problem with lack of generalization across contexts.

Definition 7.1 (Neural Language Model)**.**
The input to multilayer perceptron is a sequence of k words, and the output is a probability distribution over the next word.

For example, the below is a neural language model with 2 hidden layer: Let $\mathbf{x} = [v(w_1), \dots, v(w_k)]$ be the sequence of k words, where $v(\cdot)$ is the word embedding function. Let $\mathbf{h} = g(\mathbf{x}\mathbf{W}^1 + b^1)$, and $\hat{\mathbf{y}} = \text{softmax}(\mathbf{h}\mathbf{W}^2 + \mathbf{b}^2)$.
For neural language model, since we need to have a word embedding of finite dimension, we need V to be a finite vocabulary, including `UNK` for unknown words, `<s>` for sentence initial padding and `</s>` for end-of-sentence marking.

Training examples would be sequences of $k + 1$ words from a corpus, where the first k words are used as features, and last $(k + 1)$th word as target label for classification. Due to such classification is categorical, we can use categorical cross-entropy loss $= \log(\hat{\mathbf{y}}_t)$, where t is the index of the correct label.

Some drawbacks of neural network could be softmax, which is costly for very large vocabulary.

As you can see, we do need a word embedding function $v(\cdot)$ to kickstart our training. We can do one of the following to obtain an initial embedding:

- random initialization
- supervised task-specific pretraining
- unsupervised pre-training

For **random initialization**, we can intiialize word vectors randomly. Suppose d is the number of dimensions of a word vector after embedding, we can sample random numbers from $[-\frac{1}{2d}, \frac{1}{2d}]$ or other reasonable ranges.

For **supervised task-specific pre-training**, basically is to use the labelled training data in a large dataset in one task to pre-train word vector, and use that in training for a nother task.

For **unsupervised pre-training**, we use raw texts without human manual annotation. So we do need to predict a word based on its context, and also predict the context based on a word.

Theorem 7.1 (Collobert and Weston)**.**
The context is surrounding words, i.e. $P(w_3 \mid w_1 w_2 \cdot w_4 w_5)$, and the Collobert Weston assign a score to a word, so we need to ensure that the correct word is assigned a score higher tthan incorrect words.

$$s(w, c_{1:k}) g(xU) \cdot v$$

where $x = [v_c(c_1), \ldots, v_c(c_k); v_w(w)]$.
Here, w is target word, $c_{1:k}$ a sequence of context words. s is a score of a word-context pair. $v_c(\cdot)$ is an embedding vector.
Loss function is a margin-based ranking loss function, with correct word-context pair above incoorect pairs with a margin of at least 1:

$$L(w, c, w') = \max(0, 1 - (s(w, c) - s(w', c)))$$

where w' is a random chosen word.

Theorem 7.2 (Word2Vec)**.**
Training for Word2Vec distinguishes a set D of correct word-context pairs from a set $\bar{D}$ of incorrect word-context pairs.
The score is defined as $s(w, c) = \mathbf{w} \cdot \mathbf{c}$, where $\mathbf{w}, \mathbf{c}$ vectors of equal dimension.
The probability $P(D = 1 \mid w, c) = \frac{1}{1+e^{-s(w,c)}}$, and $P(D = 0 \mid w, c) = 1 - P(D = 1|w, c)$.
The training objective is to maximize the log-likelihood of the data $D \cup \bar{D}$. So loss function is

$$L = \sum_{(w,c) \in D} \log(P(D = 1|w, c)) + \sum_{(w,c) \in \bar{D}} \log(P(D = 0|w, c))$$

The positive examples $(w, c) \in D$ is from a raw text corpus, whereas negative sampling is done by, for each $(w, c) \in D$, sample k words and add each of these words (w_i, c) as a negative example to $\bar{D}$.
Negative words are sampled, according to their corpus frequency: $\frac{\#(w)^{0.75}}{\sum_{w'} \#(w')^{0.75}}$. This gives higher relative weight to less frequent words.
For **CBOW**, suppose the context has k words $c_{1:k}$, the context vector is computed by $\mathbf{c} := \sum_{i=1}^{k} \mathbf{c}_i$, which means that the order of context words is ignored.
In other words, we have

$$\log P(D = 1|w, c_{1:k}) = \log \frac{1}{1 + e^{-\sum_{i=1}^{k} \mathbf{w} \cdot \mathbf{c_i}}}$$

For **SkipGram**, We assume that each word in $c_{1:k}$ is independent of each other, so

$$P(D = 1|w, c_{1:k}) = \prod_{i=1}^{k} P(D = 1|w, c_i) = \prod_{i=1}^{k} \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{c}_i}}$$

After training, we will keep embeddings for the word, but not embeddings for the context.

Theorem 7.3 (Word Similarity)**.**
Word Similarity can be measured via cosine similarity:

$$\text{sim}_{\cos}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{||u||_2 ||v||_2}$$

Also, word analogy can be found via word vector algebra:

$$\text{analogy}(mLw \to k :?) = \arg\max_{v \in V \setminus \{m,w,k\}} \text{sim}_{\cos}(v, k - m + w)$$

8 CNN

It is ideal if neural network can be specialised for dealing with language data, which is

- Sequential
- of Long distance dependency

CNN and RNN are useful.
CNN, convolutional neural networks identify information n-grams, and consider local ordering patterns.
RNN, recurrent neural networks,

- Capture subtle patterns and regularities in sequences
- Model non-Markovian dependencies
- Consider infinite-window and zoom in on informative sequential patterns in the window.

CNN and RNN are used as **feature extractors**, each producing a vector, or a sequence of vectors that is fed into further part of the network that eventually leads to prediction.
The network is trained end-to-end, i.e. predicting part and CNN/RNN are trained together.

8.1 Convolutional Neural Networks

Naive solution to CBOW order-ignorance is to produce word vectors for bigrams, but it is problematic due to huge embedding matrix, and also different bigrams can be relevant, but are completely different bigrams.
CNN serves as a feature-extracting architecture, that extract meaningful substructure useful for the overall prediction task.
We focus on 1D convolutions:

- We apply an **nonlinear, learned function** (filter) over each k-word sliding window in the sentence.
- l filters are applied to produce an l-dimentional vector for each sliding window of k words.
- A pooling operation combines the vector from different windows into a single l-dimensional vector.
- This vector is fed into a neural network for prediction.

Filter functions learns to identify informative k-grams, whereas pooling enables focusing on the most important aspect of a sentence.
More formally, we have this formulation of CNN:

- A sentence is a sequence of words $w_{1:n}$.

- Word embedding matrix E is obtained for each word in $w_{1:n}$
- Sliding window size is k
- Define The window context $x_i = w_{i:i+k-1}$, where w_j represents its **embedding**.
- Let $U = [u_1, \dots, u_l]$ be l different filters, each applicable on a vector of dimension k.
- We will have output from filters $p_i = g(x_i \cdot U + b)$, where g is a non-linear activation function.
  p_i is a collection of l values representing the ith window, and each dimension of p_i captures a different kind of indicative information.
- In total, we will have m numbers of p_i vectors.
  - In narrow convolution, $m = n - (k-1)$.
  - In wide convolution, $m = n + (k-1)$, due to padding of $k-1$ words to each side.

CNN also could accept multiple channels, say sequence of words, and also sequence of POS tags, and we can have different p's for different channels, and we can combine the channels later arbitrarily, say by addition, or by vector.
As we can see, p is sensitive to the identity and order of the words within a k gram, but the same representation p is extracted for the same k-gram regardless of its position within the sentence.
For pooling, we will combine $p_1, \dots, p_m$ into a single vector c, where c captures the essence of the important information in the sentence, and is fed downstream for prediction.
Training will turn downstream NN, U, b and E such that c encodes information relevant to the prediction task.

Definition 8.1 (Max Pooling).

$$c_j = \max_{1 \le i \le m} (p_i)_j$$

Definition 8.2 (Average Pooling).

$$c = \frac{1}{m} \sum_{i=1}^{m} p_i$$

CNN support hierarchical convolutions, where there is a hierarchy of convolutional layers:

$$p^1_{1:m_1} = \mathrm{CONV}^{k_1}_{U^1, b^1}(w_{1:n})$$

$$\dots$$

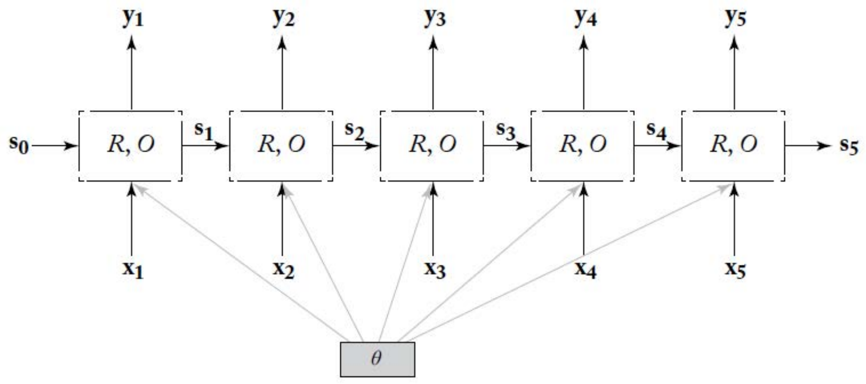
$$p^r_{1:m_r} = \mathrm{CONV}^{k_r}_{U^r, b^r}(p^{r-1}_{1:m_{r-1}})$$

Also, the sliding window can have custom **stride size** s, where window will be at $1, 1+s, 1+2s, \dots$.
One observation is that if window size k and stride size is $k-1$, there will be an exponential

growth in the effective window size as a function of number of layers, from top to bottom.
Parameter tying is a technique where same U, b are used in all the layers.
Skip connections is used in deep architectures, where ith layer is not only feed from $i-1$th layer but also from previous layers.

9 RNN

CNN has the problem of modelling of the word order is restricted to mostly local patterns. RNN can overcome this problem. We can represent RNN as



- $RNN^*(x_{1:n} : s_0) = y_{1:n}$
- $s_i = R(s_{i-1}, x_i)$
- $y_i = O(s_i)$

The functions R and O are the same across the sequence positions.
The last y_n encodes the entire input sequence and can be used for further prediction.

Definition 9.1 (Simple RNN)**.**

- $s_i = R_{SRNN}(s_{i-1}, x_i) := g([s_{i-1}; x_i]W + b)$
- $y_i = O_{SRNN}(s_i) = s_i$

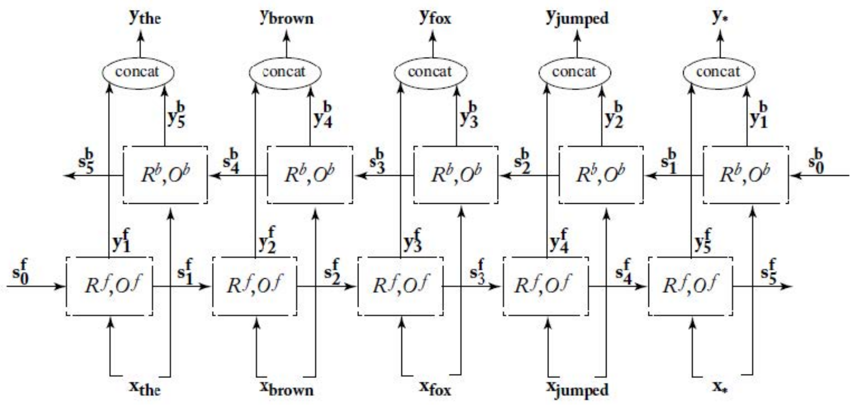
We can use RNN to predict an event e given sequence $x_{1:n}$:

$$P(e = j \mid x_{1:n}) = \text{softmax}(y_n \cdot W = b)_j$$

RNN allows conditioning on the entire history withotu making Markov assumption.
RNN Training can be done via backpropagation through time, by adding a loss node after the last y_i.
RNN can be used as acceptor (in prediction), or as encoder, or as a transducer.
For RNN as transducer, we will produce an output $\hat{t}_i$ after each input x_i is read. Therefore, we will have a local loss $L_{local}(\hat{t}_i, t_i)$, and the total loss

$$L(\hat{t}_{1:n}, t_{1:n}) = \sum_{i=1}^{n} L_{local}(\hat{t}_i, t_i)$$

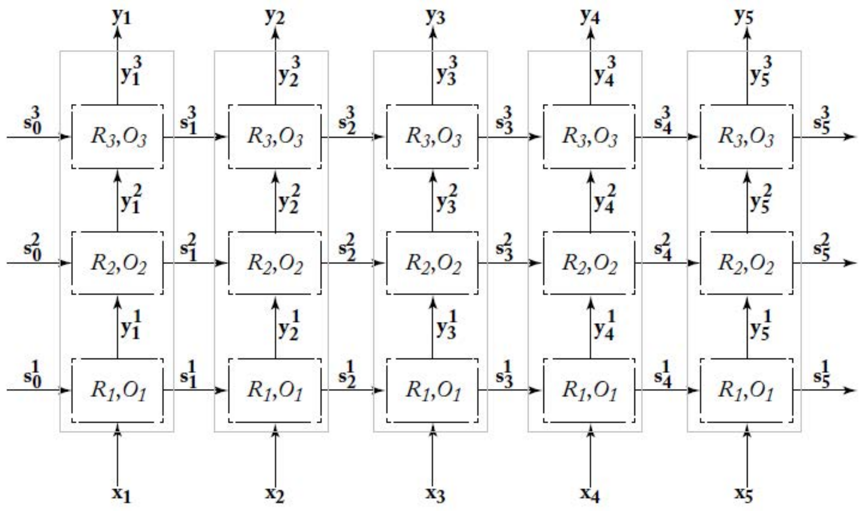
One example, is to use RNN as a pos tagger, where each word input will have a tag output.



Definition 9.2 (Bidirectional RNN).
Bidirectional RNN uses both the previous and following words (context).

BiRNN is very effective for sequence tagging tasks.

Definition 9.3 (Multilayered RNN).
We can stack multiple RNN together.



We can treat CBOW as RNN without taking into account order of words:

- $s_i = R_{CBOW}(s_{i-1}, x_i) = s_{i-1} + x_i$
- $y_i = O_{CBOW}(s_i) = s_i$

Simple RNN is hard to train effectively, due to vanishing gradients. Gated architectures are designed to overcome this deficiency.

Definition 9.4 (LSTM). *Long short-term memory is designed to solve vanishing problem.*

- $s_j = R_{LSTM}(s_{j-1}, x_j) = [c_j; h_j]$

- $i = \sigma(x_j W^{xi} + h_{j-1} W^{hi})$
- $f = \sigma(x_j W^{xf} + h_{j-1} W^{hf})$
- $o = \sigma(x_j W^{xo} + h_{j-1} W^{ho})$
- $z = \tanh(x_j W^{xz} + h_{j-1}^{hz})$
- $c_j = f \oplus c_{j-1} + i \oplus z$
- $h_j = p \oplus \tanh(c_j)$
- $y_j = O_{LSTM}(s_j) = h_j$

RNN can be used for POS tagging:

- s to be the input sentence
- Let $w_i = c_1, \dots, c_l$
- c_i is the character embedding vector for c_i
- $x_i = \phi(s, i) = [E_{w_i}; RNN^f(c_{1:l}); RNN^b(c_{l:1})]$
- $p(t_i = j \mid w_1, \dots, w_n) = \text{softmax}(MLP(biRNN(x_{1:n}, i)))_j$
- Trained with cross-entropy loss function

POS Tagging can include character embeddings, which take into account suffix, prefix, orthographic cues important for POS tagging.
POS tag prediction is conditioned on the entire input sentence.
RNN can be used as an acceptor, where it read an input sentence and produce a binary, or multiclass answer.
For example, given a sentence s, we can use RNN to classify s as positive or negative.

- $P(label = k \mid w_{1:n}) = \hat{y}_k, k = 1, 2$
- $\hat{y} = \text{softmax}(MLP(RNN(x_{1:n})))$
- $x_{1:n} = (E_{w_1}, \dots, E_{w_n})$

But we can also use a bidirectional RNN.

9.1 Formal Grammars of English

Definition 9.5 (Syntax)**.**
Syntax is relationship between words, including

- Order of words
- Grouping of words
- Structure of sentences

Definition 9.6 (Constituent)**.**
A constituent is a group of words that behaves as a single unit.

A noun phrase is a sequence of words surrounding at least one noun.

Definition 9.7 (Context-Free Grammar)**.**
It allows we to derive words from abstract data types. For example, $NP \to DetNominal \to DetNoun \to aNoun \to aflight$
The node NP dominates all the nodes: $Det, Nominal, Noun, a, flight$. The node NP immediately dominates the nodes $Det, Nominal$.

The formal definition is $G = (N, \Sigma, P, S)$

- N, a set of non-terminal symbols
- Σ, a set of terminal symbols
- P, a set of productions, each of the form $A \to \alpha$, where $A \in N$ and $\alpha \in (\sigma \cup N)^*$.
- A designated start symbol $S \in N$

If $A \to \beta$ is a production of P, and α and γ are any strings in $(\Sigma \cup N)^*$, then $\alpha A \gamma$ directly derives $\alpha\beta\gamma$
Suppose that $\alpha_1, \ldots, \alpha_m$ are strings in $(\Sigma \cup N)^*$, $m \geq 1$, and

$$\alpha_i \Rightarrow \alpha_{i+1}$$

for $i = 1, \ldots, m-1$, then, α_1 derives α_m, or $\alpha_1 \Rightarrow^* \alpha_m$.
The language generated by G, denoted by $L(G)$ is a set of strings composed of terminal symbols which can be derived from teh start symbol S of G:

$$L(G) = \{w \mid w \in \Sigma^* \text{ and } S \Rightarrow^* w\}$$

Syntactic parsing is a mapping from a strin gof words to its parse tree.

Definition 9.8 (Strong Equivalence)**.**
Two grammars G_1 and G_2 are strongly equivalent if they generate the same set of strings $L(G_1) = L(G_2)$ and they assign the same phrase structure to each string, allowing only for renaming of non-terminal symbols.

Definition 9.9 (Weak equivalence)**.**
Two grammar are weakly equivalent if the generate the same set of strings but do not assign the same phrase structure to each string.

Definition 9.10 (Chomsky Normal Form)**.**
The grammar should be ϵ-free, and each production of the grammar is either of the form $A \to BC$ or $A \to a$.

Any CFG can be converted into a weakly equivalent CFG in Chomsky Normal Form.

Definition 9.11 (Unit Production)**.**
A unit production is a production $A \to B$ where

- the RHS consists of a single non-terminal symbol B,

All other productions are non-unit productions.

Theorem 9.1 (Algorithm to convert CFG with no ϵproduction to CNF)**.**

1. Construct a new set of productions P' by first including all non-unit productions
2. For non-terminal symbols A and B, if $A \to A_1 \to \cdots \to B$, where A_i are non-terminal symbols, then for each non-unit production $B \to \alpha$, add a new production $A \to \alpha$ to P'.
3. Consider each production in P' of the form $A \to X_1 X_2 \ldots X_m$, where $m \geq 2$. If X_i is a terminal symbol a, introduce a new non-terminal symbol C and a production $C \to a$. THey replace X_i by C.
4. For each production in P' of the form $AB_1B_2B_3$, replace this production by $A \to B_1D_1$ and $D_1 \to B_2B_3$
5. Similar things should be done to $A \to$longer RHS sequences. D_i will be new non-terminal symbols.

There are four kinds of sentence-level constructions:

- Declarative: $S \to$ NP VP.
- Imperative: $S \to$ VP
- Yes-no-question: $S \to$ Aux NP VP
- Wh-question: $S \to$ Wh-NP VP or $S \to$ Wh-NP Aux NP VP.

Definition 9.12 (Clause)**.**
Clause form a complete thought, a node of a parser tree, below which the main verb of the clause has all of its arguments.

Definition 9.13 (Noun Phrase)**.**
Components of an NP:

- Head: Central Noun
- Pre-nominal modifiers
- Post-nominal modifiers
- $Nominal \rightarrow NominalNoun$

For Pre-head modifiers, it can be a determiner, a , the, or a possessive expression, like United's.
Determiners can be omitted, and there could be predeterminers coming before the determiners. NP $\rightarrow$ PreDet NP
Others can appear between the determiner and the head noun:

- Cardinal numbers,
- Ordinal numbers,
- Quantifiers,
- Adjective-phrase

So, we have

$$NP \rightarrow (Det)(Card)(Ord)(Quant)(AP)Nominal$$

For post-head modifiers, there could be

- Prepositional phrases(PP): Nominal $\rightarrow$ Nominal PP
- Non-finite clauses $-ed$, $-ing$, or infinitive Nominal $\rightarrow$ Nominal GerundVP, where
  - GerundVP $\rightarrow$ GerundV
  - GerundVP $\rightarrow$ GerundV NP
  - GerundVP $\rightarrow$ GerundV PP
  - GerundVP $\rightarrow$ GerundV NP PP
  - GerundV $\rightarrow$ $arriving|leaving|\ldots$
- Relative clauses: $Nominal \rightarrow NominalRelClause$, where
  - $RelClause \rightarrow whoVP$
  - $RelClause \rightarrow thatVP$

Definition 9.14 (Agreement)**.**
Verbs can appear in 2 forms in the present tense:

- For 3rd person, singular subjects

- For all other subjects

For yes-no questions, we have the following rules

- $S \rightarrow 3sg\,\mathrm{Aux}\,2sg\,\mathrm{NP}\,\mathrm{VP}$
- $S \rightarrow Non3sg\,\mathrm{Aux}\,Non3sg\,\mathrm{NP}\,\mathrm{VP}$
- $3sg\,\mathrm{Aux} \rightarrow does|has|can|\ldots$
- $Non3sg\,\mathrm{Aux} \rightarrow do|have|can|\ldots$
- $3sg\,\mathrm{NP} \rightarrow \mathrm{Det}\,Sg\,\mathrm{Nominal}$
- $Non3sg\,\mathrm{NP} \rightarrow \mathrm{Det}\,Pl\,\mathrm{Nominal}$
- $Sg\,\mathrm{Nominal} \rightarrow Sg\,\mathrm{Noun}$
- $Pl\,\mathrm{Nominal} \rightarrow Pl\,\mathrm{Noun}$

Note that determiner can also be singular(this) or plural(these).
Also, pronouns can be either nominative(I, he, she) or accusative(me, him, her).

Definition 9.15 (Verb Phrase)**.**
$VP \rightarrow \mathrm{Verb}\,|\,\mathrm{Verb\,NP}\,|\,\mathrm{Verb\,PP}\,|\,\mathrm{Verb\,NP\,PP}$. A VP can have different kinds of constituents, depending on the verb.
Complements of a verb is constituents following a verb. A verb subcategorises for certain complements(what to expect). We define subcategorization frame, the possible sets of complements of a verb.
So depends on teh subcategorization frame, we can have this more fine grained notation:

- $VP \rightarrow \mathrm{Verb}-with-no-comp$
- $VP \rightarrow \mathrm{Verb}-with-NP-comp\,\mathrm{NP}$
- $VP \rightarrow \mathrm{Verb}-with-S-comp\;S$

There are also auxiliary verbs:

- Modal verbs: can, ...
- Perfect auxiliary: have
- Progressive auxiliary: be
- Passive auxiliary: be

An auxiliary verb places constraint on the form of the following verb, so it can be viewed as subcategorization:

- Model-verb bare-stem

- Perfect-aux past-participle
- Progressive-aux presetn-participle
- Passive-aux past-participle

There are also orders between multiple auxiliary verbs: modal ¡ perfect ¡ progressive ¡ passive. A verb group is all auxiliary verbs plus the main verb.

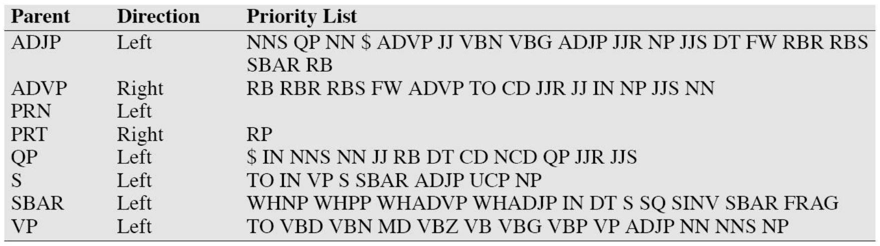
Definition 9.16 (Coordination)**.**
NPs VPs and sentences can be joined with conjunctions(and, or).

- NP $\rightarrow$ NP *and* NP
- VP $\rightarrow$ VP *and* VP
- $S \rightarrow S\ and\ S$.

Definition 9.17 (Treebank)**.**
Treebank is a corpus in which each sentence is syntactically annotated with a parse tree.

Penn Treebank has traces (-NONE- nodes) to mark long-distance dependencies or syntactic movements. It also have grammatical function tags, denoted as $-X$ after the main tag.

Theorem 9.2 (Heads, Head Finding)**.**
Head is the word in the phrase that is grammatically most important. Heads are passed up a parse tree.
Each non-terminal in a parse tree is annotated with a word, which his its lexical head.
We need to manually identify one RHS constituent of a CFG rule as the head child of that rule. Lexical head of the parent is the lexical head of the head child.
We can have a head percolation table: For a CFG rule $VP \rightarrow Y_1 \ldots Y_n$, start from left of



Parent	Direction	Priority List
ADJP	Left	NNS QP NN $ ADVP JJ VBN VBG ADJP JJR NP JJS DT FW RBR RBS SBAR RB
ADVP	Right	RB RBR RBS FW ADVP TO CD JJR JJ IN NP JJS NN
PRN	Left	
PRT	Right	RP
QP	Left	$ IN NNS NN JJ RB DT CD NCD QP JJR JJS
S	Left	TO IN VP S SBAR ADJP UCP NP
SBAR	Left	WHNP WHPP WHADVP WHADJP IN DT S SQ SINV SBAR FRAG
VP	Left	TO VBD VBN MD VBZ VB VBG VBP VP ADJP NN NNS NP

$Y_1, \ldots Y_n$ to look for the first Y_i of TO, and we try all the things on the priority list. If not found, return Y_1.

Definition 9.18 (Dependency Grammars)**.**
Dependency grammars captures bianry relations between words in a sentence. It ignores constituent sand phrase structure rules.
It allows strong predictive power that a word has on its dependents. It also handlw eith ease languages with relatively free word order.

We can transform a phrase structure parse into a untyped dependency parse using head percolation table.

10 Syntactic Parsing

We can parse using top-down parsing, by considering all possibilities starting S, and backtrack when failed.
We can also do bottom up parsing, and try combining them to get S.
There are pros and cons to both approach:

- Top-down parsing
  - Goal-directed search
  - Never wastes time exploring trees that cannot result in an S
- Bottom-up parsing
  - Data-directed search
  - Never suggests trees that are not grounded in the actual input sentence
- Need to incorporate features of both

The problems of parsing is **structural ambiguity**:

- Attachment ambiguity.
  There could be $VP(V, NP(NP, PP))$ or $VP(V, NP, PP)$, and PP is used to describe different objects here.
- Noun-phrase bracketing ambiguity
  There could be $VP(V, NP(PNoun), NP(Nom))$ or $VP(V, NP(Nom(PNoun, Nom)))$, and the verb can have 2 noun phrases attached to it(do somebody something) or 1 noun phrases attached to it (do something).
- Coordination ambiguity
  Example: old men and women

Parsing can be considered as search, which involves searching the space of parse trees.
Agenda-based backtracking search leads to reduplication of work, as parsing of subtrees can be repeated.

Definition 10.1 (Dynamic Programming Parsing Methods)**.**

- The CKY algorithm
- Earley Algorithm
- Chart parsing

Theorem 10.1 (CKY Algorithm).
Bottom-up dynamic programming algorithm, which requires CFG to be in Chomsky Normal Form(CNF).

```
function CKY-PARSE(words, grammar) returns table
for $j \leftarrow$ from 1 to Length(words) do
    table[$j-1$, $j$]$\leftarrow \{A \mid A \to$ words[$j$] $\in$ grammar$\}$
    for $i \leftarrow$ from $j-2$ downto 0 do
        for $k \leftarrow i+1$ to $j-1$ do
            table[$i,j$] $\leftarrow$table[$i,j$] $\cup \{A \mid A \to BC \in$ grammar$, B \in$ table$[i,k], C \in$ table$[k,j]\}$
```

To return all possible parses,

- Augment each entry such that each non-terminal is paired with pointers to the entries form which is was derived
- Permit multiple versions of the same non-terminal to be entered in an entry

Theorem 10.2 (Earley Algorithm).
Top-down dynamic programming algorithm. Uses a chart to store subtrees for each constituent.
Each possible subtree is only stored once only.
We have states, which is a CFG rule, with dots and indexes-pair. For example,

$$S \to \cdot \, \mathrm{VP}[0,0]$$

The presense of $S \to \alpha \cdot [0,N]$ in the list of states indicates a successful parse.
There are 3 operators:

- Predictor
- Scanner
- Completer

Predictor creates new states representing top-down predictions/expectations. For example,

- Given $S \to \cdot \, \mathrm{VP}[0,0]$,
- We add
  - $\mathrm{VP} \to \cdot \, \mathrm{Verb}[0,0]$
  - $\mathrm{VP} \to \cdot \, \mathrm{Verb\,NP}[0,0]$

Scanner uses the predicted POS to incorporate the next word. For example,

- Given state $\mathrm{VP} \to \cdot \, \mathrm{Verb\,NP}[0,0]$
- Add new state: $\mathrm{Verb} \to \mathrm{book} \cdot [0,1]$

```
function EARLEY-PARSE(words, grammar) returns chart
  ENQUEUE(($\gamma \rightarrow \bullet S$, [0,0]), chart[0])
  for i ← from 0 to LENGTH(words) do
    for each state in chart[i] do
      if INCOMPLETE?(state) and
              NEXT-CAT(state) is not a part of speech then
          PREDICTOR(state)
      elseif INCOMPLETE?(state) and
              NEXT-CAT(state) is a part of speech then
          SCANNER(state)
      else
          COMPLETER(state)
    end
  end
  return(chart)

procedure PREDICTOR(($A \rightarrow \alpha \bullet B \beta$, [i, j]))
    for each ($B \rightarrow \gamma$) in GRAMMAR-RULES-FOR(B, grammar) do
        ENQUEUE(($B \rightarrow \bullet \gamma$, [j, j]), chart[j])
    end

procedure SCANNER(($A \rightarrow \alpha \bullet B \beta$, [i, j]))
    if B ⊂ PARTS-OF-SPEECH(word[j]) then
        ENQUEUE(($B \rightarrow word[j]$, [j, j+1]), chart[j+1])

procedure COMPLETER(($B \rightarrow \gamma \bullet$, [j, k]))
    for each ($A \rightarrow \alpha \bullet B \beta$, [i, j]) in chart[j] do
        ENQUEUE(($A \rightarrow \alpha B \bullet \beta$, [i, k]), chart[k])
    end

procedure ENQUEUE(state, chart-entry)
    if state is not already in chart-entry then
        PUSH(state, chart-entry)
    end
```

Completer finds and advance all previously created states that were looking for this non-terminal at this position. For example

- Givern state NP $\rightarrow$ Det Nominal $\cdot [1, 3]$
- Find state VP $\rightarrow$ Verb $\cdot$ NP$[0, 1]$, and add the new state VP $\rightarrow$ Verb NP $\cdot[0, 3]$.

Theorem 10.3 (Chart Parsing).
Chart parsing allows a flexible and dynamic order of processing the chart entries through an explicit agenda.

```
function Chart-Parse(words, grammar, agenda-strategy) returns chart
Initialize(chart, agenda, words)
while agenda
    current-edge ← Pop(agenda)
    Process-Edge(current-edge)
return (chart)

**procedure** Process-Edge(edge)
    Add-to-Chart(edge)
**if** Incomplete?(edge)
    Forward-Fundamental-Rule(edge)
**else**
    Backward-Fundamental-Rule(edge)
Make-Predictions(edge)

**procedure** Forward-Fundamental-Rule($A \to \alpha B \beta, [i, j]$)
**for each** ($B \to \gamma \cdot, [j, k]$) **in** chart
    Add-to-Agenda($A \to \alpha B \cdot \beta, [i, k]$)

**procedure** Backward-Fundamental-Rule($B \to \gamma \cdot, [j, k]$)
**for each** ($A \to \alpha B \beta, [i, j]$) **in** chart
    **Add-to-Agenda**($A \to \alpha B \cdot \beta, [i, k]$)

**procedure** Add-to-Chart(edge)
**if** edge is not already in chart **then**
    add edge to chart

**procedure** Add-to-Agenda(edge)
**if** edge is not already in agenda **then**
    apply(agenda-strategy, edge, agenda)

**procedure** Make-Prediction(edge)
**if** Top-Down and Imcomplete?(edge)
    TD-Predict(edge)
**elseif** Bottom-up and Complete?(edge)
    BU-Predict(edge)

**procedure** TD-Predict($A \to \alpha \cdot B \beta, [i, j]$)
**for each**($B \to \gamma$) **in** grammar **do**
    Add-to-agenda($B \to \cdot \gamma, [j, j]$)

**procedure** BU-Predict($B \to \gamma \cdot, [i, j]$)
**for each**($A \to B \beta$) **in** grammar **do**
    Add-to-agenda($A \to \cdot B \cdot \beta, [i, j]$)
```

11 Statistical Parsing

Statistical parsing aims to resolve structural ambiguity, by choosing the most probable parse.

Definition 11.1 (Probablistic Context-Free Grammar)**.**
Let $G = (N, \Sigma, P, S, D)$ be the PCFG, where

- N is a set of non-terminal symbols
- Σ a set of terminal symbols, with $N \cap \Sigma = \varnothing$
- P, a set of productions, each of the form $A \to \alpha$, where $A \in N$ and $\alpha in (\Sigma \cup N)^*$
- $S \in N$, a designated start symbol
- D, a fucntion that assigns a probablity to each rule in P:

$$P(A \to \alpha) \text{ or } P(A \to \alpha \mid A)$$

The score of a parse tree is provided by

$$P(T, S) = \prod_{n \in T} p(r(n))$$

So

$$\hat{T}(S) = \arg \max_{T \text{ s.t. } S = \text{yield}(T)} P(T \mid S) = \arg\max P(T)$$

Definition 11.2 (Probabilitistic CKY ALgorithm)**.**
PCKY requires PCFG to be in Chomsky Normal Form $A \to BC$ or $A \to \alpha$.
table$[i, j, A]$ stores the max probability for a constituent A spanning word positions i to j.
So the probability of most probably parse is table$[0, n, S]$.

```
**function** PCKY(words, grammar) **returns** most probably parse and probability
**for** $j \leftarrow$ **from** 1 **to** length(word) **do**
    **for all** $\{A \mid A \to \text{words}[j] \in G\}$
        table$[j-1, j, A] \leftarrow P(A \to \textbf{words}[j])$
    **for** $i \leftarrow$ **from** $j-2$ **downto** 0 **do**:        **for** $k \leftarrow i+1$ **to** $j-1$ **do**
            **for all** $\{A \mid A \to BC \in G\}$ **and** table$[i,k,B] > 0$ **and** table$[k,j,C] > 0$
                **if** table$[i,j,A] < P(A \to BC) \times$ table$[i,k,B] \times$ table$[k,j,C]$ **then**
                    table$[i,j,A] \leftarrow P(A \to BC) \times$ table$[i,k,B] \times$ table$[k,j,C]$
                    back$[i,j,A] \leftarrow \{k, B, C\}$
**return** Build-Tree(back[0, length(words), $S$]), table[0, length(words), $S$].
```

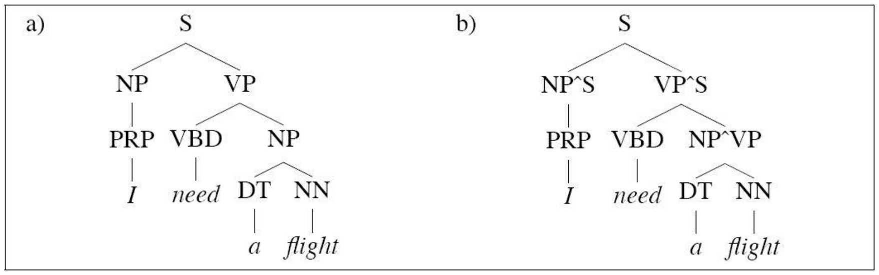
To learn PCFG Rule Probabilities, we can estimate from parsed sentences(treebank):

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

THe problem of PCFG include

- Poor independence assumptions
  The expansion of a non-terminal actually depends on its context. For example, NP can be either pronoun or non-pronoun, and the expansion probability will differ.
- Lack to sensitivity to words:
  - Attachment ambiguity: Some words may attach to some other words in a fixed manner, and knowing such words will help in disambiguation.
  - Coordination Ambiguity: It could have the same conditional probability due to the same set of grammar rules used, but many possibilities exist, in which only 1, in the correct structure, is correct.

To resolve coordination ambiguity, we use **parent annotation**, i.e. for each tag, we include parent's tag also, if it is ambiguity. WE can also split non-terminals and preterminals:



- Each non-terminal in a parse tree is annotated with a word(lexical head)
- Identify one RHS constituent of a PCFG rule as the head child of that rule
- Lexcial head of the parent is the lexical head of the head child

This forms a probablistic lexicalized CFG, where we can capture dependencies between words.
The Collins Parser uses such probablistic lexicalized CFG.

Definition 11.3 (Collins Parser Model 1)**.**
Each rule has the form $P(h) \to L_n(l_n) \cdots L_1(l_1)H(h)R_1(r_1)\dots R_m(r_m)$, where h, l_i, r_i is of the form $\langle \text{headword}, \text{headPOSTag} \rangle$.
$P(\text{RHS} \mid \text{LHS})$ is obtained by multiplying

- $P_h(H \mid P, h)$: generate the head constituent
- $P_r(R_i(r_i) \mid H, P, h, \text{distance}_r(i-1))$: generate each right modifier
- $P_l(L_i(l_i) \mid H, P, h, \text{distance}_l(i-1))$: generate each left modifier

Herer, P, L, H, R are the type.
$\text{distance}_\cdot(x)$ is a function of the surface string between head and the distance the tags crosses, and returns:

- whether string is of length 0
- and whether string contain a verb

To evaluate a parser, we can use Penn Treebank. We define

$$\text{labelled recall} = \frac{\#\text{correct constituents in parser's parse of } s}{\#\text{ constituents in treebank's parse of } s}$$

$$\text{labelled precision} = \frac{\#\text{correct constituents in parser's parse of } s}{\#\text{constituents in parser's parse of } s}$$

A constituent is correct if there is a constituent in the treebank's parse that spans the same words with the same non-terminal symbol.

Definition 11.4 (Human Parsing)**.**
Garden-path sentence, which are sentences constructed with 3 properties that combine to make them very difficult for people to parse:

- Temporarily ambiguous: the sentence itself is unambiguous, but its initial portion ambiguous
- one of the 2 or more parses in the initial portion is somehow preferable to human parsing mechanism
- Dispreferred parse could be the correct parse for the sentence

Human subjects spent significantly more time reading the word *was*.
Human may also have subcategorization preference.

12 Representation of Meaning

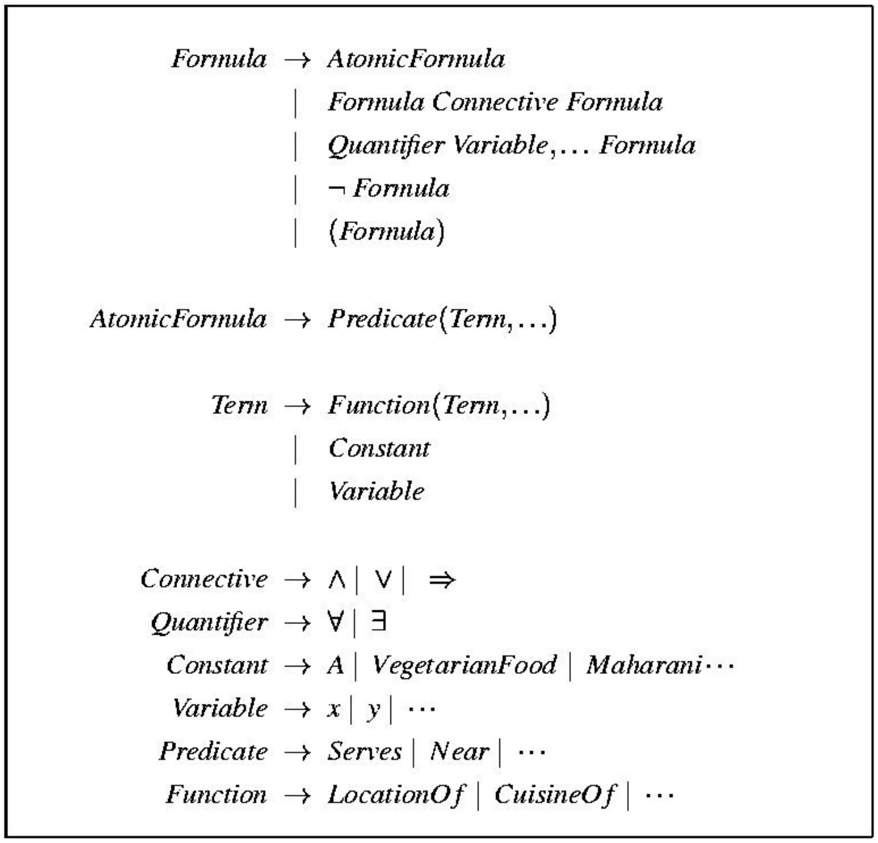
Meaning representation(semantic representation) is formal structures composed from symbols to represent the meaning of sentences.
Non-literal meaning require inference from literal meaning of the sentences.

Definition 12.1 (Meaning Representation Languages)**.**
Meaning Representation Languages are frameworks used to specify the syntax and semantics of meaning representations.
Frequently used meaning representation languages include:

- First order logic
- Semantic Network
- Conceptual dependency
- Frame Representation

Desirable properties of meaning representation language include

- Verifiability:
  - Able to determine the truth of the meaning representation of a sentence against the world modelled
- Unambiguous representations:
  - Semantic ambiguity to be resolved by semantic analysis, and should be unambiguous in menaing representation
- Canonical forms
  - Different sentences with the same meaning should be assigned the same meaning representation.
  - It is not that easy, since we may have different words, different sentence structures, etc.
  - But we need to have same representation for these different things in order to facilitates matching to facts in the knowledge base.
  - Therefore, we need to assign same meaning to different words, and assign same meaning to different syntactic structures.
- Inferences and variables
  - NLP program must be able to draw valid inferences not explicitly stated in the meaning represetnation of the input sentence, or program's knowledge base
- Expressiveness



$$
\begin{array}{rcl}
\textit{Formula} & \to & \textit{AtomicFormula} \\
 & | & \textit{Formula Connective Formula} \\
 & | & \textit{Quantifier Variable}, \ldots\ \textit{Formula} \\
 & | & \neg\ \textit{Formula} \\
 & | & (\textit{Formula}) \\
 & & \\
\textit{AtomicFormula} & \to & \textit{Predicate}(\textit{Term}, \ldots) \\
 & & \\
\textit{Term} & \to & \textit{Function}(\textit{Term}, \ldots) \\
 & | & \textit{Constant} \\
 & | & \textit{Variable} \\
 & & \\
\textit{Connective} & \to & \wedge \mid \vee \mid \Rightarrow \\
\textit{Quantifier} & \to & \forall \mid \exists \\
\textit{Constant} & \to & A \mid \textit{VegetarianFood} \mid \textit{Maharani} \cdots \\
\textit{Variable} & \to & x \mid y \mid \cdots \\
\textit{Predicate} & \to & \textit{Serves} \mid \textit{Near} \mid \cdots \\
\textit{Function} & \to & \textit{LocationOf} \mid \textit{CuisineOf} \mid \cdots
\end{array}
$$

Definition 12.2. *FOL*
First order logic is a flexible, expressive meaning representation language. It has the following grammar: FOL takes a model-theoretic semantics: a FOL formula is true with respect to a model and an interpretation.
Model contains a domain of objects and relations among objects.
An interpretation is a mapping:

- Constant symbol $\to$ object
- Predicate symbol $\to$ relation
- Function symbol $\to$ function relation

Definition 12.3 (Lambda Expression)**.**
A lambda expression is $\lambda x : P(x)$, where $P(x)$ is a FOL formula.
λ-reduction is to apply a lambda expression to a term.

Definition 12.4 (Inference)**.**
We perform inference by modus ponens: $\alpha \wedge \alpha \implies \beta$ so β.
We call α the antecedent of $\alpha \implies \beta$ and β the consequent.

Definition 12.5 (Categories)**.**
We use **reification**, i.e., we represent all concepts that we want to make statements about as full-fledged objects.

Definition 12.6 (Event)**.**
We use reification to elevate events to objects, and arguments of an event will apear as predicates.

13 Computational Semantics

Theorem 13.1 (Principle of Compositionality)**.**
The meaning of a sentence can be composed from the meaning of its parts.

The meaning of a setntence is dependent on the ordering, grouping and relations among the words in the sentence(syntactic structure).
We will pass inputs throw syntactic analysis and then semantic analysis to get meaning representations as output.

Definition 13.1 (Syntax-driven Semantic Analysis)**.**
Syntax-driven semantic analysis allows for multiple syntactic parses and multiple meaning representations. It leaves to discourse processing to select among these multiple meaning representations, by using knowledge of context and other domain-specific knowledge.

Definition 13.2 (Semantic Attachments)**.**
We augment CFG rules with semantic attachments:

$$A \to \alpha_1 \dots \alpha_n \quad \{f(\alpha_j \cdot \text{sem}, \dots, \alpha_k \cdot \text{sem})\}$$

A semantic attachment describes how to compute the meaning representation of the LHS constituent(A.sem) using the meaning representations of the RHS constituents($\alpha_i \cdot \text{sem}$).
It associates complex, function-like λ-expressions with lexical items. It copies semantic values from children to parents in non-branching rules.

However, **idioms** can violate the assumption of compositionality.

14 Lexical Semantics

Definition 14.1 (Lexeme)**.**
Lexeme is an individual entry in the lexicon, which is a pairing of a particular orthographic form and phonological form with a meaning representation.
Lexeme is represented by a lemma.

Definition 14.2 (Lemma)**.**
Lemma is the grammatical form used to represent a lexeme, often the base form.

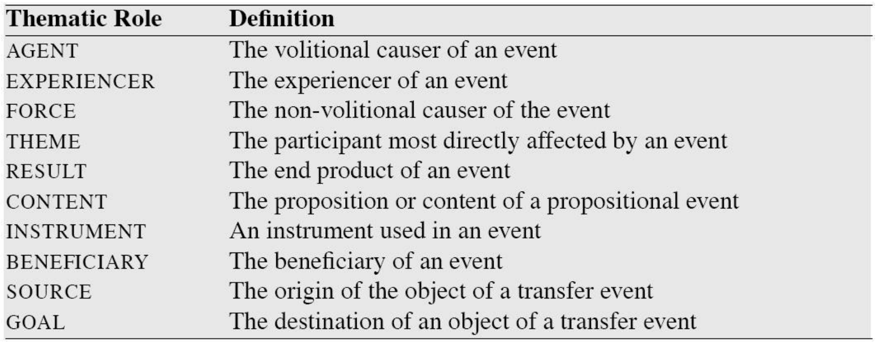
Lexicon is a finite list of lexemes; sense is the discrete representation of the meaning of a lexeme.
Some observations include:

- Circularity in the definitions, including direct or indirect self-references
- Dictionaries are for human consumption but not for computer use
- We need grounding in the real-word
- But relations among words are still valuable to be studied

Relations among lexemes and senses include

- Homonymy: a relation that holds between lexemes with the same form but *unrelated* meanings.
  For example, two meanings of the word *bank*
- Polysemy: A single lexeme with multiple *related* meanings.
  For example, meanings of the word *serve.*
- Metonymy: Denote a concept by naming some other concept closely related to it
- Synonymy: a relation that holds between different lexemes with the same meaning.
  Two lexemes are considered **synonyms** if they can be substituted for one another in *some* sentence without changing the meaning/acceptability of the sentence.
- Hyponymy: A subclass relation between two lexemes, with is-a hierarchy.

Definition 14.3 (Thematic Roles)**.**
Thematic roles are generalized across events, and characterize arguments of verbs.
Thematic roles include:



Thematic Role	**Definition**
AGENT	The volitional causer of an event
EXPERIENCER	The experiencer of an event
FORCE	The non-volitional causer of the event
THEME	The participant most directly affected by an event
RESULT	The end product of an event
CONTENT	The proposition or content of a propositional event
INSTRUMENT	An instrument used in an event
BENEFICIARY	The beneficiary of an event
SOURCE	The origin of the object of a transfer event
GOAL	The destination of an object of a transfer event

Diathesis alternations allows thematic roles to be ambiguous.

Definition 14.4 (Metaphor)**.**
Metaphor refer to and reason about a concept or domain using words and phrases whose meanings come from a completely **different** domain.

15 Computational Lexical Semantics

Word Sense disambiguation(WSD) determine the correct sense of a word in context.
WordNet is the most widely used online English lexical resources in NLP.
We can use Machine Learning to perform robust WSD:

- Supervised Learning
- Bootstraping
- Unsupervised

For **supervised learning** approaches, we take target word, which is the word to be disambiguated and context, in which the target word is embedded, as input.
We do preprocessing of context:

- POS tagging
- Stemming
- Parsing

and eventually convert into a feature vector representation.
We usually extract linguistic features, like collocation features and co-occurrence features. Collocation features are positive specific, and contain local information. Co-occurrence features are words occurring in a fixed size window around the target word, is position-independent and contain global/topical information.
Traidtional supervised learning will learn from a manually sense-tagged corpus, using algorithms like Naive Bayes:

$$\hat{s} = \arg\max_{s \in S} P(s) \prod_{j=1}^{n} P(v_j \mid s)$$

Deep Learning uses Bi-LSTM approach: $\text{biLSTM}(x_{1:n}, i) = y_i = [\text{LSTM}^f(x_{1:i}); \text{LSTM}^b(x_{n:i})]$ and $\text{NN}(y_i) = z = g(y_i W^1 + b^1)W^2 + b^2$, and $o = \text{softmax}(z)$, with categorical cross-entropy loss function. Note that W^2, b^2 are specific to each word type.
For **bootstrapping approach**, we have weakly supervised data, and we use Yarowsky algorithm:

- Start with supervised training on seed examples
- Use trained classifier to label all unlabelled examples
- Choose the most confidently labellled examples to add to the growing pool of training examples
- Iterate

For **unsupervised methods**,

- We discover word senses

- Group sentences into clusters of similar feature vectors according to some similarity metric
- We use agglomerative clustering

There are 2 kinds of WSD evaluation:

- Extrinsic evaluation:
  - In vivo evaluation
  - Task based evaluation
  - End-to-end evaluation
- Intrinsic evaluation
  - In vitro evaluation

16 Computational Discourse

Definition 16.1 (Discourse)**.**
Discourse is a coherent structured group of sentences.

- Monologue is communication that flows in one direction, from speaker to the hearer
- Dialogue allows each participant periodically to take turn to become a speaker/hearer

Note that a randomly ordered sequence of sentences does not form a discourse. Discourse must be **coherent**.

Theorem 16.1 (Rhetorical Structure Theory(RST))**.**
RST is based on 23 rhetorical relations. Most relations hold between two text spans, called a *nucleus* and a *satellite*.
Nucleus is the unit that is more central to the writer's purpose, and can be interpretable independently.
Satellite is less central and generally only interpretable wrt the nucleus.

DIscourse parsing extract a tree/graph representing an entire discourse, with a directed edge from a satellite to its nucleus:

1. Identify cue phrases
2. Segment a text into discourse segments, using cue phrases
3. Classify the relation betwween each consecutive discourse segment, using cue phrases.

However, problem with above algorithm include:

- Cue phrases are ambiguous
- Many coherence relations are NOT signaled by cue phrases.

Definition 16.2 (Reference)**.**
Reference is use of linguistic expression to denote an entity.
Reference resolution is to determine what entities are referred to by which linguistic expressions.
Referring expression is a natural language expression used to perform reference, whereas referrent is the entity referred to.
Two referring expressions *corefer* when they are used to refer to the same entity. The first of two is the *antecedent* and the second *anaphor*.
Anaphora is reference to an entity that has been previously introduced into the discourse.

There are 2 reference resolution tasks:

- Coreference resolution: finding referring expressions that refer to the same entity
- Pronominal anaphora resolution: finding the antecedent for a single pronoun

Coreference chain is the set of coreferring expressions.
Types of referring expresisons can be

- Indefinite noun phrases, which introduce entities new to the hearer
- Definite noun phrases, which refer to entities identifiable to the hearer.
- Pronouns, which require referents to have a high degree of salience; usually refer to entities introduced no further than one or two sentences back.
  There are 2 types of pronouns
  - Cataphora, which are pronouns mentioned before there referrent
  - Pronouns in quantified contexts, which behaves like a variable bound to a quantified expression
- Demonstratives: this, that, these those
- Names: names of people/organizations/locations
- Inferrables: a referring expression does not refer to any entity that has been explicitly mentioned but instead one that can be inferentially related to a mentioned entity
- Generics: a generic reference that does not refer back to an entity explicitly mentioned in the text
- Non-referential uses

For pronominal anaphora resolution, we check

- Number agreement
- Person agreement
- Gender agreement
- Binding theory constraints
- Selectional restrictions
- Recency
- Grammatical role: a salience hierarchy ordered by grammatical position
- Repeated mention: entities that have been focused on in the prior discourse are more likely to continue to be focused on in subsequent discourse.
- Parallelism
- Verb semantics

Task for pronominal anaphora resolution is, given a pronoun and potential antecedent NP, determine if pronoun refers to the NP. Training data is hand-labelled corpus in which each pronoun has been linked by hand to its antecedent.
Features are those criteria above. We can use LSTM. Let g_i to be span representation for span i, and input sentence with word embeddings to be $x_{1:T}$. Let output vector at position t to be biLSTM$(x_{1:T}, t) = x_t^* = [\text{LSTM}^f(x_{1:t}), \text{LSTM}^b(x_{T:t})]$. We use attentino mechanism to represent head word of the span:
let $\alpha_t = w_\alpha \cdot MLP(x_t^*)$, $a_{i,t} = \frac{\exp(\alpha_t)}{\sum_{k=\text{START}(i)}^{\text{END}(i)} \exp(\alpha_k)}$.
Let $\hat{x}_i = \sum_{t=\text{START}(i)}^{\text{END}(i)} a_{i,t} \cdot x_t$. Let $\phi(i)$ encode the length of span i, then $g_i = [x^*_{\text{START}(i)}, x^*_{\text{END(i)}}, \hat{x}_i, \phi(i)]$.
Pairwise score $s(i,j)$ for a coreference link between span i and j.

- $s_m(i) = w_m \cdot MLP(g_i)$
- $s_a(i,j) = w_a \cdot MLP([g_i; g_j; g_j \circ g_j; \psi(i,j)])$
- $\psi(i,j)$ encodes the distance between two spans
- $s(i,j) = \begin{cases} 0 & \text{if } j = \epsilon \\ s_m(i) + s_m(j) + s_a(i,j) & \text{if } j \neq \epsilon \end{cases}$

Suppose we have N spans,

$$P(y_1, \ldots, y_N) = \prod_{i=1}^{N} P(y_i) = \prod_{i=1}^{N} \frac{\exp(s(i, y_i))}{\sum_{y' \in y(i)} \exp(s(i, y'))}$$

where y_i is antecedent for span i, and $y(i)$ is set of possible assignments for y_i.