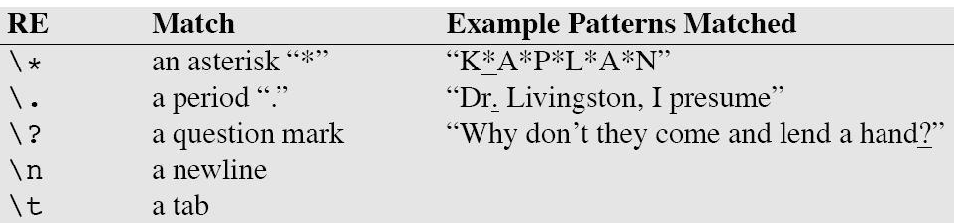
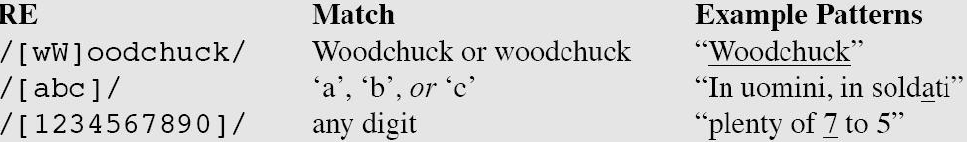
# Regular Expressions(RE)

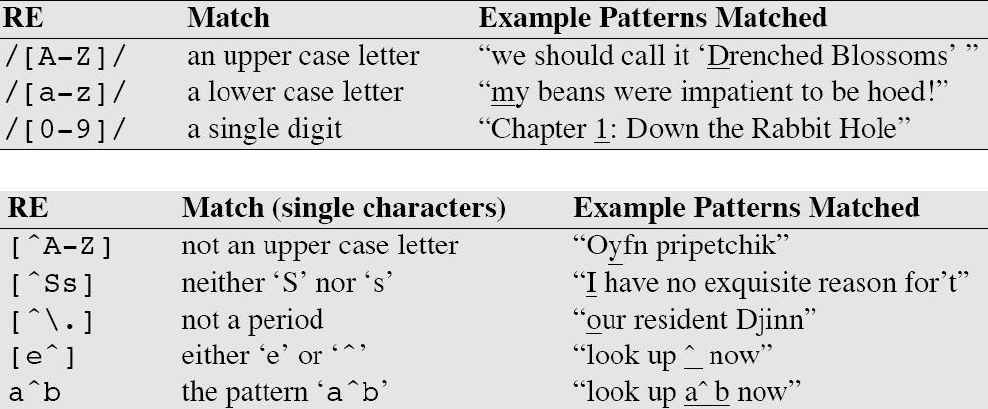
* **RE** is a formula to specify a set of strings
  + - A string is thereby a sequence of alphanumeric characters
      * letters, digits, punctuations, spaces, tabs
  + Matches the **first** occurrence of the specified string as part of a string in a larger text corpus
* **RE PATTERNS – Must be encased between ‘**/**… … …/’**
  + **\***
    - **0 or more** occurrences of the previous char/expression
  + **+**
    - **1 or more** occurrences of the previous char/expression
  + **?**
    - **exactly 0 or 1** occurrence of the previous char/expression
    - useful for **plural** (0 or 1 ‘s’) and **American/British** spelling
      * **woodchucks?** matches **woodchuck** and **woodchucks**
      * **colou?r** matches **colour** and **color**
  + **{ n }**
    - ***n*** occurrences of the previous char/expression
  + **{ n, m }**
    - **anywhere from *n* to *m*** occurrences of the previous char/expression
  + **{ n, }**
    - **at least *n*** occurrences of the previous char/expression
  + **.**
    - **1** occurrence of the **ANY** **char**
    - combo with **\***  to accept a variable length of any string
  + Escape characters with backslash



* + - \d
      * **any digit;** equivalent to **[0-9]**
    - \D
      * **any non-digit;** equivalent to **[^0-9]**
    - \s
      * **any form of white space,** equivalent to **[\_\r\t\n\f]**
    - \S
      * **any form of non-white space,** equivalent to **[^\s]**
  + **[<text>]**
    - Letters enclosed in square brackets signify **a choice of only one** of the letters within those brackets



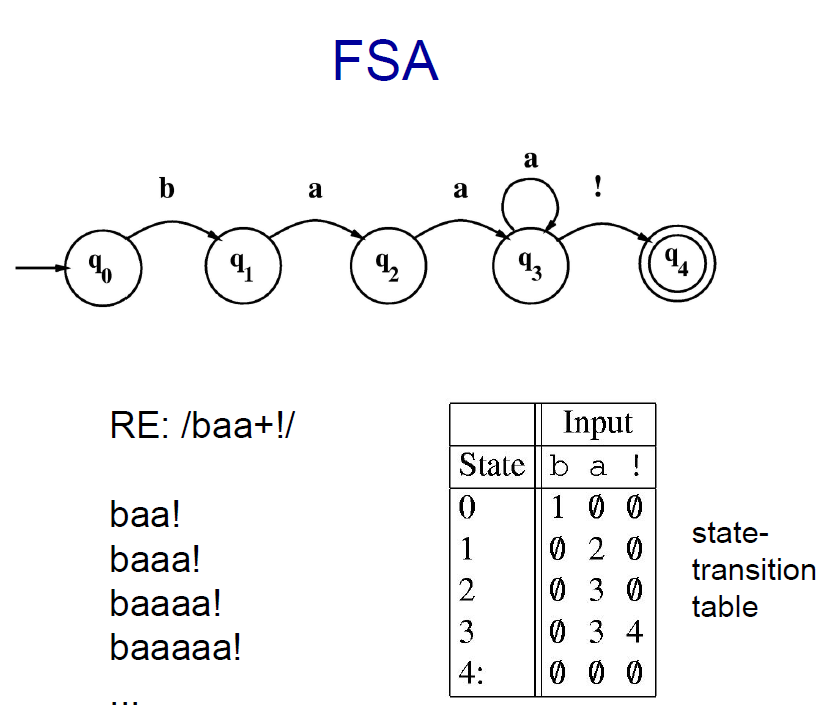
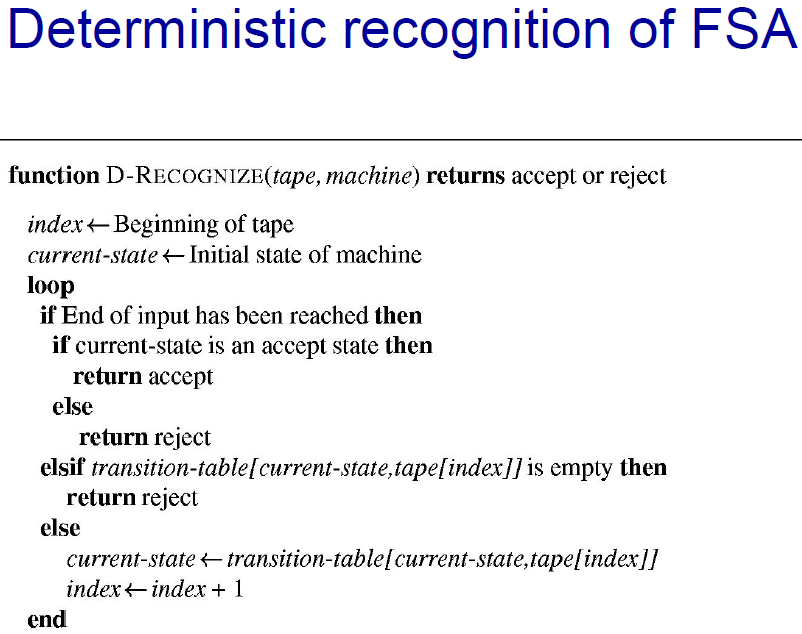
* + **^**
    - **within** square brackets **[]**, signifies a **not** **operator**
    - **outside** square brackets, see **anchors**

****

* + - **note**: the not operator appears at the front and applies to the **whole** expression
  + Anchors
    - **^**
      * Start of line aka <s>
    - **$**
      * End of line aka </s>
    - /^The dog\.$/
      * This regex matches a line that contains only ‘The dog.’
  + Precedence
    1. Parenthesis
    2. Counters, i.e. \* + ? { }
    3. Sequences/Anchors
    4. Disjunction
  + Examples:
    1. /cat|dog/ matches ‘cat’ or ‘dog’ since sequence > disjunction
    2. gupp(y|ies) matches ‘guppy’ or ‘guppies’

# Finite State Automata

* FSAs are said to accept an input string if the input string is fully expended **and** the FSA is still in an accepting state.
  + If either of the conditions are not met, the input string is rejected
  + Note that any regex can be expressed by an FSA

* + **/baa+!/** matches any string that starts with ‘baa’ followed by any number of ‘a’s, ended off by a single exclamation mark.
  + Start state q0, upon receiving b, transitions to state q1
  + **\*\*INDICATE START STATE WITH AN INITIAL ARROW (NOTICE INCOMING ARROW TO q0)\*\***
  + **\*\*INDICATE ACCEPTING STATE WITH DOUBLE CIRCLES\*\***
  + A fail state maybe added to signify a ‘sink state’ that has no outward arrows
    - hence FSA will never transition back into an accepting state and erroneously accept a string!
    - **\*\*REMEMBER TO INCLUDE LOOPBACK ARROWS FOR FAIL STATE\*\***
* Formal definition of FSA:
  + **Q**
    - a finite set of N states: q0, q1, q2, … qn-1, where q0 is the initial state
  + **Σ**
    - a finite input alphabet of symbols
  + **F**
    - set of final states; **F ⊆ Q**
  + **𝛿(q, i)**
    - the transition function between states, given an input of *i* at state *q*
    - new state *q’* is returned as a result of this function
* Definitions
  + Alphabet
    - finite collection of symbols
  + Words
    - finite sequence of letters/symbols from the alphabet
  + **Formal Language**
    - a set of strings(words) formed over a set of all alphabets.
      * it cherrypicks which words (permutations of alphabets) are to be included in this logical set of a language
    - can be characterized by a model (FSA)
      * FSA models that can both generate and recognize all and only the strings of a formal language act as definitions for that language
  + **Regular Language**
    - A formal language that is accepted by some DFSA (see below)
      * By extension, regular languages are equivalent to FSAs
    - Given regular languages ***L1*** and ***L2***, then
      * ***L1* ᛫ *L2*** (concatenation)
      * ***L1* ⋃ *L2***(union)
      * ***L1\**** (Kleene closure) are all regular languages.
* **Non-deterministic Finite State Automata (NFSAs)**
  + A NFSA accepts an input string ***X*** if and only if there is ***at least******some*** path in the NFSA transition graph from the start state that leads to some accepting final state while simultaneously exhausting the input string ***X***.
  + Since the above definition does not impose a restriction on the number of states/paths, given any input ***a*,** NFSAs can allow for transitions to several different states.
  + Epsilon transitions **ε** are permitted
    - Able to transit to another state **without consuming any symbols**
* **Deterministic Finite State Automata (DFSAs)**
  + These FSAs are a subset of a NFSAs, more formally, for any NFSA, there exists an equivalent DFSA
  + NFSAs can be converted to DFSA by keeping track of all possible states an NFSA can reach on a given input and mapping the combination of these states individually
    - a state in a DFSA represents multiple states in a NFSA
  + Being deterministic in nature, for each state ***S*** and input ***a***, there is **at most 1** edge labelled ***a*** leaving ***S***.

# Words

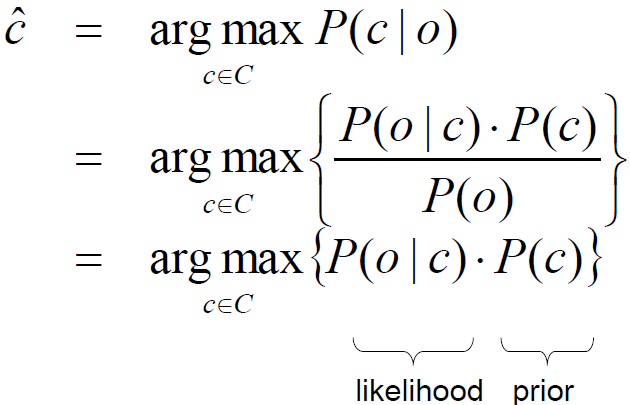
* **Morphemes**
  + Minimum meaning-bearing unit in a language
  + Classified into:
    - Stems
      * ‘Main’ piece of a word
      * e.g. cat, fox, eat,
    - Affixes
      * Built onto stems to modify meaning
        + Prefix (before)
        + Suffix (after)
        + Infix (inside the stem, Tagalog)
        + Circumfix (have both prefixes and suffixes, German)
      * e.g. cat**s**, fox**es**, eat**ing**, **un**buckle
      * A word can have multiple affixes
        + e.g. unbelievably – **un-**, believe, **-able**, **-ly**
* **Morphology**
  + the study of how words are built from morphemes, is important as listing all different morphological variants is inefficient
  + Affixes are efficient abstractions of common ideas and meanings that can be applied consistently to stems
    - Especially useful since these ideas can be applied to new stems like google, googling etc.
  + **Inflectional morphology**
    - combines a stem and an affix to result in a word in the **same** syntactic class as the stem
      * used for syntactic function of agreement; meaning is not changed hence easier to predict
  + **Derivational morphology**
    - combines a stem and an affix to result in a word in a **different** syntactic class as the stem
      * **harder to predict meaning of the derived form!**
        + cannot generalize the semantic transformation, since words like deploy retains meaning but depart morphs it!
      * clue a **noun**, is turned into clueless, an **adjective**!
* **Text processing**
  + **Stemming**
    - Stemming is an imperfect but fast and efficient way to approximate morphological analyses by stripping off affixes to retain only the stems of words.
    - **Porter Stemming Algorithm (PSA)**
      * PSA is used widely in information retrieval
      * **Intuition**: Implement a series of rewrite rules that runs in cascade
        + output of each rewrite transformation is fed as input to the next rule
  + **Text normalisation**
    - Tokenisation
      * Segment running text into words
        + Punctuations are dealt with as separate symbols to aid in expanding clitic contractions

Words that cannot exist in its own form such as “**ll**” in “we’**ll**” can be appropriately expanded

* + - Normalisation
      * Standardises words that have multiple representations
        + US vs USA
    - Case folding
      * Allows for comparability between strings by making everything lowercase
    - **Penn Treebank Tokenization Standard**
      * Separates out clitics
        + does 🡪 does n’t
      * Keep hyphenated words together
      * Separates out punctuation symbols

# Spelling Errors

* 3 classes of errors
  + Non-word error detection
    - butchering a word till it no longer is a word in the dictionary
  + Isolated-word error correction
    - correcting individual typos without caring for context
  + Context-sensitive error detection and correction
    - typos of word ***a*** led to actual legit word ***b***, hence context needs to be examined to identify that ***b*** should actually be ***a***
* Error patterns
  + Majority are single error, which are 1 of the following:
    1. **insertion**
       - additional character
    2. **deletion**
       - omission of a character
    3. **substitution**
       - replacing a character
    4. **transposition**
       - swapping the order of 2 (consecutive) characters
    - which is a result of errors of these 2 natures
      * typographic (legit typos)
      * cognitive (legit don’t know how spell)
* **Spelling error correction**
  + Single errors – assumptions
    - Only dealing with non-word spelling errors that are a result of only **1 of the 4 misspelling errors**
    - Disregards context
  + Solution
    - Propose candidates to replace incorrectly spelled words
      * need a large dictionary of correctly spelled words
    - Score the candidates, pick the best one!
      * given a misspelled word ***o***, use Bayesian classification to identify which candidate ***ci***amongst ***c1*** to ***cn*** has the highest chance to be the correct word
      * formally,



* + - * given the word is ***c***, what is the probability that it is misspelled as ***o***?
      * note that denominator ***P(o)*** can be removed since it is a constant and does not affect result of argmax
      * the probabilities can be calculated from large corpora of texts where misspelled words are identified and corrected
        + supervised ML
    - **Estimating prior probability P(c)**
      * **Maximum Likelihood Estimate**
        + ***P(c) = C(c) / N***

Probability of candidate = Count of candidate / Total candidates

* + - * + However, for more obscure words, large corpora of texts might still not suffice – the word might simply not show up at all in the finite corpus!

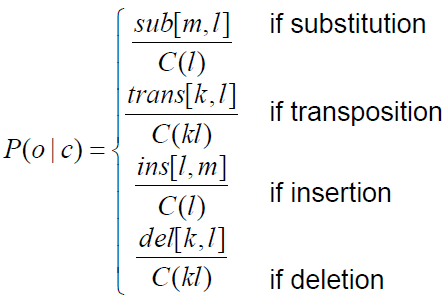
C(c) = 0 implies that the word will never show up, i.e. not be considered at all even if the context is different

* + - * Additive Smoothing
        + ***P(c) = [C(c) + λ] + [N + Bλ]***

Eliminates 0 probabilities, allowing every word to be considered!

***B*** is the number of unique candidates

* + - * + Jeffreys-Perks Law ***λ = 0.5***
        + Laplace’s Law ***λ = 1***
    - Estimating likelihood ***P(o|c)***
      * we do not use the exact typo to estimate the above likelihood because the chance of that happening in the entire corpus is crazy small!
        + the word count of that exact typo will result in a very small probability!
        + need a super large corpus to get a reasonable probability under this method
      * instead, we approximate likelihood by taking into account **all the occurrences of similar typos**
        + we will have a much larger dataset estimate to work with!
        + if deletion type typos are more common, it is more likely that ‘acress’ would have meant ‘actress’ instead of ‘acres’ or ‘caress’
        + sub[m, l]: # times correct letter l was typed as m
        + trans[k, l]: # times correct sequence kl was typed as lk
        + ins[l, m]: # times extraneous letter m was inserted after l
        + del[k, l]: # times letter l was deleted from correct sequence kl



* + Multiple errors
    - * Minimum-edit distance
        + minimum number of editing operations (ins, del, subs) needed to transform 1 string to another
        + Costs:

ins – 1

del – 1

subs – 2 if the 2 characters are different, else 0

* + - * + use dynamic programming

# n-Grams

* Sequences of n number of words
* **Language models**
  + statistical model of word sequences
  + n-grams uses previous ***n-1*** words to predict the next word, i.e. ***P(wn|w1, w2, …, wn-1)***
* Applications
  + Useful for word prediction
    - Contexts lent by word sequences contribute to likelihood and probabilities of future words in the sequence
    - **Helps in context-sensitive spelling error detection and correction**
  + Speech recognition
    - machine translation
      * Listen to audio, transcribe and run context checks to evaluate which set of n-grams make more sense given the context
* Dealing with issues of Word Counting
  + Leverage on corpora to expose the computer to large collections of words
  + Issues
    - Handling of punctuations
    - Case folding might confuse things like names with actual words (Bush vs bush)
    - Inflected forms
      * Should we count the stems or just the forms themselves?
* **Types vs. Tokens**
  + Tokens ***N*** are the number of words
  + Types ***B*** are the number of **distinct** words
  + ***N*** = ***C***(***c1***) + ***C***(***c2***) + … + ***C***(***cB***)
* n-gram approximation
  + Simple unsmoothed n-grams
    - Estimating probability of a word

1. No corpus available

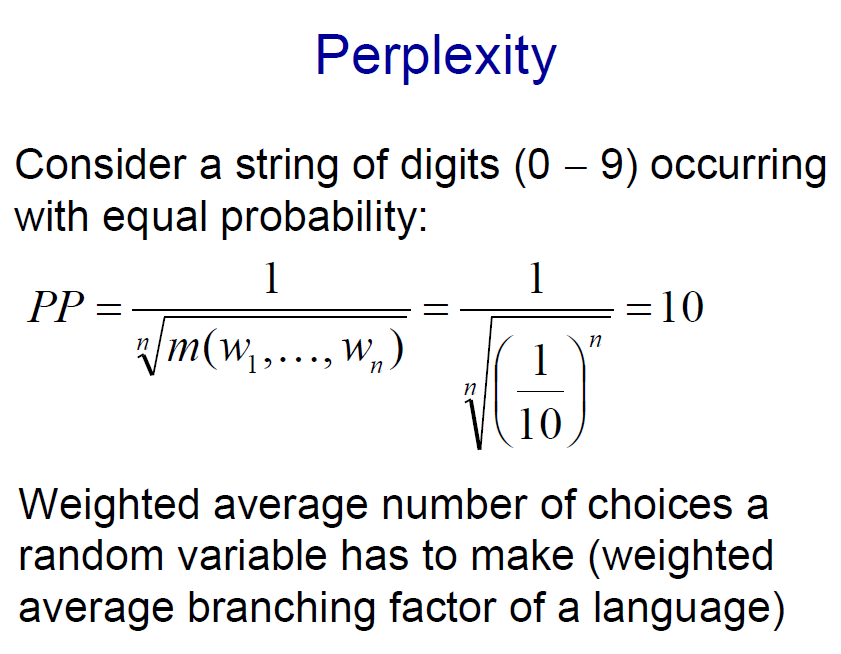
* just assume uniform distribution of words, i.e. ***P(w) = 1 / B***

2. Corpus available

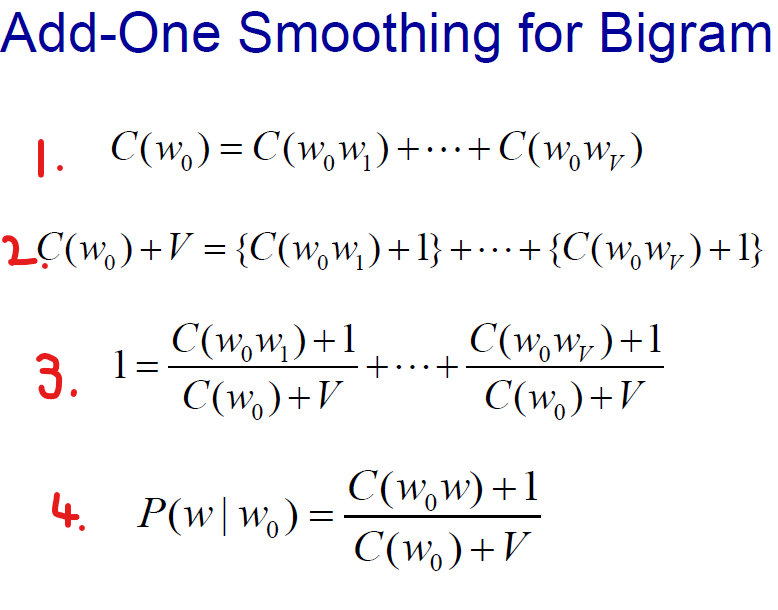
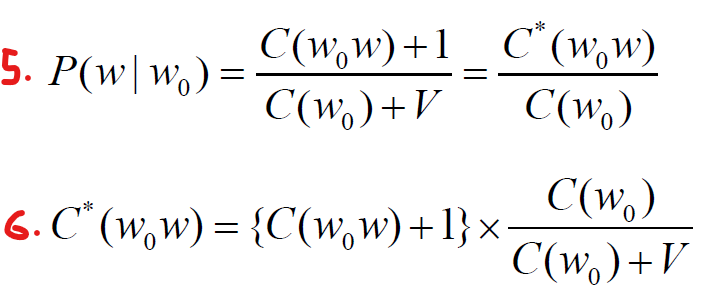
* just take # occurrence of word in corpus, i.e. ***P(w) = C(w) / N***
* context is not considered here

3. Corpus available

* assume word depends on context, specifically given by ***n-1*** previous words
  + ***P(wn|w1, w2, …, wn-1) = ∏nk=1******P(wk|w1***, ***…***,***wk-1)*** [chain rule]
    - * Markov assumption
        + ***wk*** only depends on previous ***n-1*** words!
        + knowing more words does not affect the probability of ***wk***
      * In reality, we only consider a few words due to complexity
        + bigram – nn; trigram – nn^n
  + Bi-gram approximation
    - add start and end of sentence markers as different words have different probabilities of appearing at the start and the end
      * you don’t normally end a sentence with ‘because’
      * ***P(<s> I want mala </s>) = P(I|<s>)P(want|I)P(mala|want)P(</s>|mala)***
    - If we use bigram estimates to approximate longer n-grams, then the probability of longer sequence of words will become extremely small 🡪 numerical underflow
      * Circumvent this by using logprob instead, ie don’t multiply probabilities, instead, **sum their logs!**
  + Estimating n-gram probability
    - **MLE**
      * **P(wk|wk-1) = C(wk-1wk)/C(wk-1)**
        + how many times **wk** appears after **wk-1** out of all occurrences of **wk-1**
* Training and Tests
  + Divide corpus into 3 sets (typical breakdowns, not necessary to be these values)
    - Training (80%)
      * main set on which model is trained on (this is where bigram estimates are formulated)
    - Development (10%)
      * this set is used to fine-tune the model after training
        + learning rate, batch size e.g.
    - Test (10%)
      * the test; of course if the test set is similar to training set (same genre), better performance
  + Issues with Unknown words
    - Closed vocabularies
      * vocabulary is fixed; test set will not have words that did not appear in the previous sets
      * unrealistic since new words and terms will appear (new company names, new person names)
    - Open vocabularies
      * test set will contain some words not yet seen by model
      * OoV (out of vocab) rate: % of OoV words in test set
      * To combat: assign OoV words to some pseudo-word in a list of pseudo-words
        + The list of pseudo-words are basically wildcard tokens that store the probabilities for OoV
        + To a machine, words, known or unknown are a bunch of tokens; just treating them as some wildcard will still allow it to approximate the probability of it appearing
* Evaluating n-grams – ideas of **Perplexity**
  + compare performance (through error-rate) of a model built by bigram/trigram approximations
  + Extrinsic evaluation
    - measure quality of a language model by **actually using it** 
      * although realistic and practical, it is expensive and time-consuming and requires all other components of the application that are coupled with it to be retrained and redeployed
        + e.g. the acoustic model which recognizes speech and translate it to text needs to be retrained and redeployed as well
  + Intrinsic evaluation
    - measure quality of a language model without using it
    - approximate its difficulty via the measure of **perplexity**
    - Improvements in perplexity (lower perplexity) correlates with improvements in performance!
  + A better language model better predicts test data and assigns higher probability to the correct words.
  + A better language model has a lower perplexity, which is given by
    - m is the language model built, e.g. bigram/trigram
    - calculate probability assigned to the sequence of words by language model
    - raise this probability to the power of -1/n
    - therefore, a better language model assigns a higher probability 🡺 denominator higher 🡺 PP lower
    - notice for bigram approximations, multiplying n numbers and taking the nth root 🡺 geometric mean!
    - e.g.:



* + - for each digit, each is equally likely to happen
  + Sparse data problem and evaluation
    - Finite corpus 🡪 some n-grams might not appear and hence zero probability (very bad!)
    - Methods to combat:
      1. **Smoothing**
         * re-evaluate zero probability n-grams and reassign them some non-zero probability



* + - * + 1: total count of w0 occurring (regardless of what ever word that follows)
        + 2: as some words do not appear, we just add 1 for each word type. V word types 🡺 total count of w0 occurring is c(w0) + V
        + 3: smoothened probabilities still sum up to 1 by total probability
        + 4: adjusted probability for a word w given w0 is now (the count of what w0w was before + 1 / count of all w0 + vocab size)
        + C\*(w0w) is the smoothened count
      1. **Discounting**
         * lower probability of non-zero probability n-grams in order to give some probability mass to zero probability n-grams
         * ‘spare them some probability’!
         * discount factor ***d*** is given by:
      * Smoothened probabilities will still sum up to 1!!
    - **Estimation method: Witten-Bell Smoothing**
      * **assumption:** the ‘would-have-been-sum-of-appearances’ of all unseen bigrams that do not appear in the corpus is equal to the number of distinct bigrams **Z(w0)**, and the would-have-been sum is evenly distributed amongst all unseen bigrams.
      * count of a word = count of all bigrams with that word
        + c(w0) = c(w0w1) + c(w0w2) + … + c(w0wv), v is size of vocab.
        + notice that some words wi do not appear, hence c(w0wi) = 0
      * Let **T(w0)**, **Z(w0)** be the total count of all seen and unseen bigrams respectively.
        + to smooth, we add T(w0)

c(w0) + T(w0) = c(w0w1) + c(w0w2) + … + c(w0wv) + T(w0)

* + - * + P(w|w0) = c(w0w)/( c(w0) + T(w0) ), if the bigram exists before smoothing
        + P(w|w0) = T(w0w)/[ Z(w0) \* (c(w0) + T(w0))], if bigram didn’t exist.

Even distribution of seen/unseen (T/W)

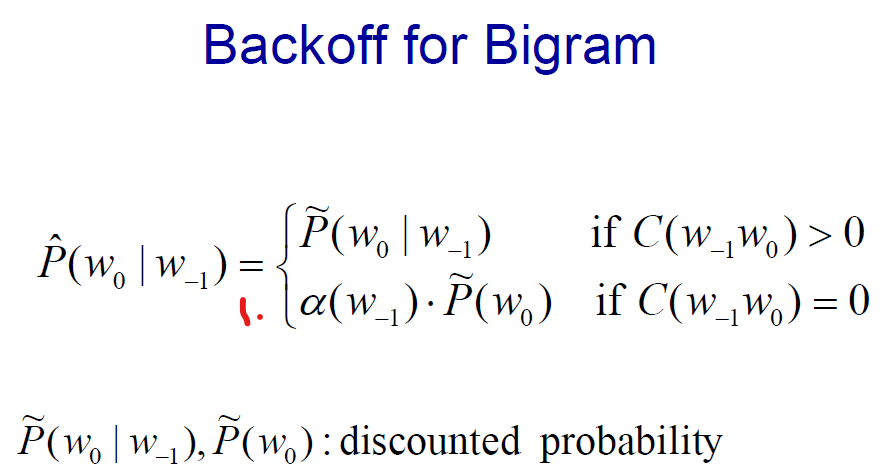
* + - **Estimation method: Interpolation (using lower order n-grams to build probabilities)**
      * For large corpora of texts, the count of seeing n-grams might be very small 🡪 near zero probability gives a bad estimate
      * Idea: use a shorter context to approximate the probability, instead of using probability to guess the next word
        + Assign weighted coefficients to the preceding ‘context-bearing’ words to help predict the next word.

Trigrams: P(w­0|w-2 w-1) = λ1P(w­0|w-2 w-1) + λ2P(w­0|w-1) + λ3P(w­0)

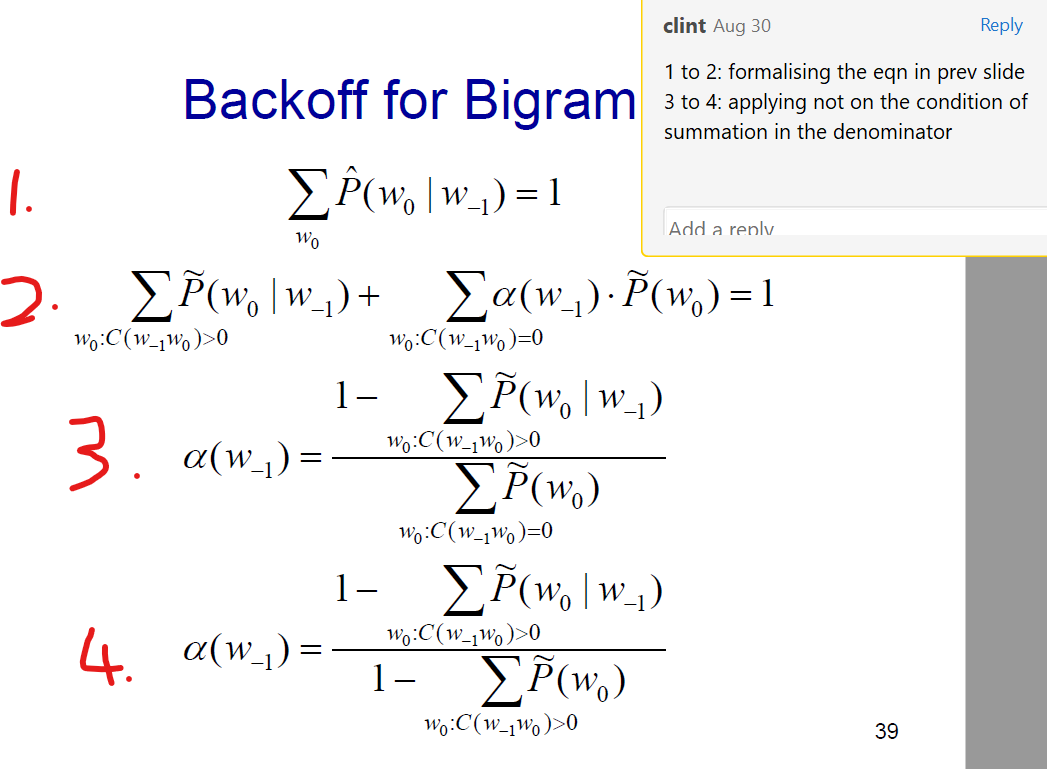
Bigrams: P(w­0|w-1) = λ1P(w­0|w-1) + λ2P(w­0)

λi all sum up to 1

* + - * Use training set to set λs
        + Finetune λs by selecting them to maximise probabilities over developmental set
    - **Estimation method: Backoff (use lower order n-grams since higher orders do not exist)**
      * Interpolation uses weighted sum of trigram, bigram and unigram estimates for probability estimates
      * Backoff is used when there is no k-order n-gram, causing the model to use lower order (1…k-1) n-grams probability estimates

****

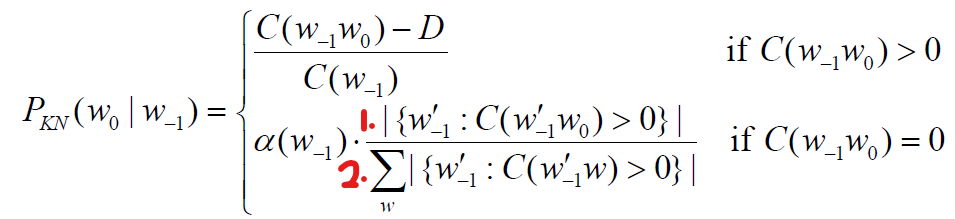
* + - * scaling factor alpha is needed here as the sum of probability estimates for all words must sum to 1.

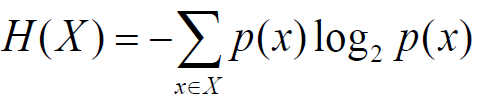


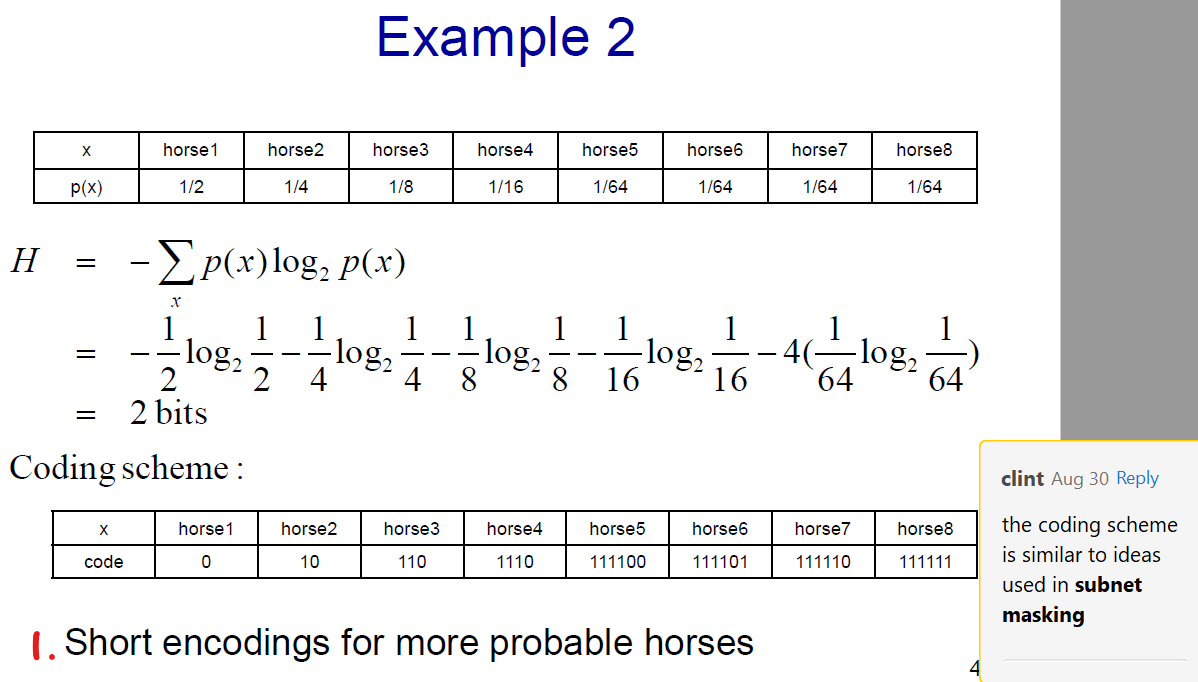
* + - * notice that 2 is simply:

sum of the **adjusted** **probability estimates** for bigrams **that do appear** + sum of **adjusted probability estimates** for bigrams **that do not appear** = 1 (total probability)

* + - * 2 to 3 is just algebra
    - **Estimation method: Kneser-Ney Smoothing for Bigrams (KNS)**
      * It uses **absolute discounting** for bigrams that do appear in the corpus
      * It also uses **backoff**, which uses a distribution that is based off the number of unique words that precedes w0
        + this helps to factor in context



* + - * this is just basically saying, for bigrams that do appear, perform absolute discounting
      * for bigrams that do not appear, we can factor in context by using the **number of unique word types that comes before w0 observed in the corpus** divided by **the number of unique bigrams**
        + division here is for normalisation
        + why this works: Given a very niche word ‘**Francisco’** that only appears after ‘**San’**, and that our text is from a tabloid in the city of San Francisco, then we can avoid over suggesting ‘**Francisco’** since it really only appears after ‘**San’** despite its prevalence compared to other words.
      * **Words that appear in many unique contexts are likely to appear more frequently, especially in new contexts.**
      * Empirically, KNS is the most effective smoothing method for n-gram language modelling
  + Entropy
    - Measure of uncertainty, measured in bits
      * Average number of bits to encode information in optimal coding scheme
    - Entropy H(x) is given by:



* + - 1: this is good as I will need less space to transmit information, since most of the time the information I need to transmit requires very little bits.
    - The entropy of a random variable over all finite sequences of words of length n for language L is given by:
    - and its rate, also known as per word entropy, is given by
    - Entropy of a language, H(L)

which, is equivalent to

* + - **Cross entropy**
      * cross entropy of a model ***m*** with respect to actual distribution ***p*** measures the average number of bits you need to encode the data ***m*** with the code that is optimal for ***p***
      * used to compare 2 language models
      * given by

where ***p*** is the actual probability distribution generated by some actual data and ***m*** is the model of ***p***.

* + - * ***H(p)*** ***H(p, m)***, since the coding scheme ***H(p)*** should be optimal
      * the difference between ***H(p)*** ***H(p, m)*** is a measure of how accurate ***m*** is in modelling the language
      * hence, more accurate models = lower cross entropy
      * **Perplexity = 2Entropy**, since
      * **In general**, lower entropy = lower perplexity = higher probability of getting predicted word right

# Part-of-Speech

* Also known as word or morphological classes, lexical tags that gives **contextual information about a word and its neighbours**
  + knowing the class of a word can lend a higher probability to the semantics of subsequent words
    - if the nth word is a **possessive pronoun**, it is highly likely that the n+1th word is a **noun**
    - same words with different POS tagging may also be pronounced differently
      * lives (noun) vs. lives (verbs); lead (noun) vs. lead (verb)
  + useful for text-to-speech analysis, parsing, word-sense disambiguation
  + helps to decide correct morphological affixes and context
    - chairs as a noun and a verb
  + can determine the structure of a sentence without even needing to see the words
    - noun phrase boundaries
* Definitions
  + Morphological:
    - words in the same POS have similar affixes
      * plural nouns share the suffix ‘-s’
  + Distributional
    - words in the same POS occur with similar surrounding words
      * appear in the similar sentences
* Classes
  + **Open class** (nouns, verbs, adjectives and adverbs)
    - Classes of words that might have new words added as time progresses
    - **Nouns**
      * concrete, abstract or verb-like references to people, places or things
      * properties
        + occurs with determiners (e.g. a, the)
        + take possessives
        + occurs with plural
      * classes:
        + proper nouns (pronouns)

Capitalised names

* + - * + common nouns

count nouns (countable)

mass nouns (uncountable e.g. snow, salt)

* + - **Verbs**
      * refers to actions and processes
      * morphological forms
        + non-3rd-person-singular
        + 3rd-person-singular
        + present participle
        + past participle
    - **Adjectives**
      * describes properties or qualities of nouns
      * modifies the following properties:
        + quality
        + quantity
        + numerical

difference is that quantity adjectives are subjective e.g. ‘some’, ‘plenty’ while numerical adjectives modifies it with a **number**

* + - * + demonstrative
        + interrogative
    - **Adverbs**
      * describe properties or qualities verbs, adjectives and other adverbs
      * modifies the following properties:
        + direction/location

downhill, there

* + - * + temporal

yesterday, Monday

* + - * + degree

very, extremely

* + - * + manner

slowly, delicately

* + **Closed class** (the rest)
    - sometimes termed **function words**
    - will never emit new words
    - **Prepositions** – on, under, over
      * occur **before noun phrases**
      * expresses spatial or temporal relations
    - **Particles** – off, out, on
      * resemble propositions that can combine with verbs to form phrasal verbs
      * phrasal verbs are a semantic unit – its meaning is not predictable from its constituent verb and particle
      * to differentiate particles and prepositions we can test for phrasal verbs
        + particles may appear after their objects but not prepositions

‘I can put off the meeting’ == ‘I can put the meeting off’

‘the horse went off its track’ != ‘the horse went its track off’

* + - **Determiners** – a, an, the
      * often **start a noun phrase**
      * articles are a subtype of determiners which are either indefinite or definite
        + indefinite: a, an
        + definite: the
    - **Pronouns** – she, I, others
      * abbreviation for referring to some noun phrase or entity or event
      * personal pronouns (you, she, I)
      * possessive pronouns (your, her, my)
        + indicative of possessive abstract relation between person and object
      * wh-prounouns
        + occurs at the start of a clause
    - **Conjunctions** – and, but, if
      * joins 2 phrases, clauses or sentences
      * Coordinating conjunctions
        + join 2 elements of equal statuses (and, or, but)
      * Subordinating conjunctions
        + join 2 elements where one element is of an embedded status (that)

I thought that you might like some milk

main clause vs. subordinate clause

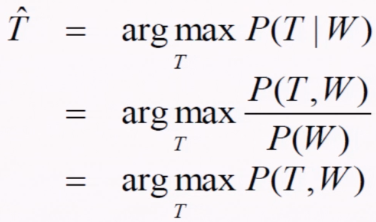
* + - **Auxiliary verbs** – can, may, should
      * provides semantic features of the main verb
        + tense
        + aspect

is it completed?

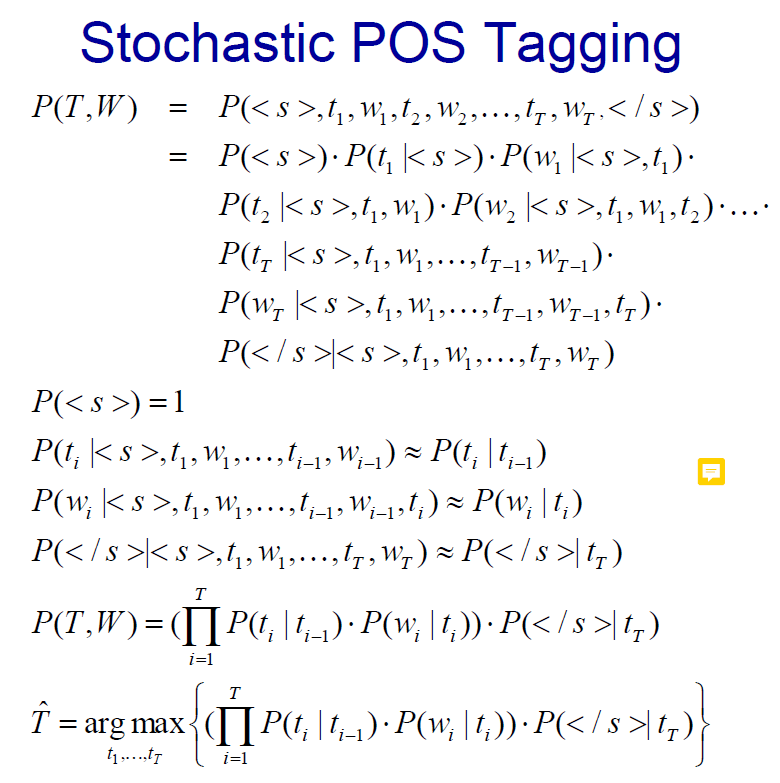
* + - * + mood (modal – can, may, must)

whether an action is necessary or possible

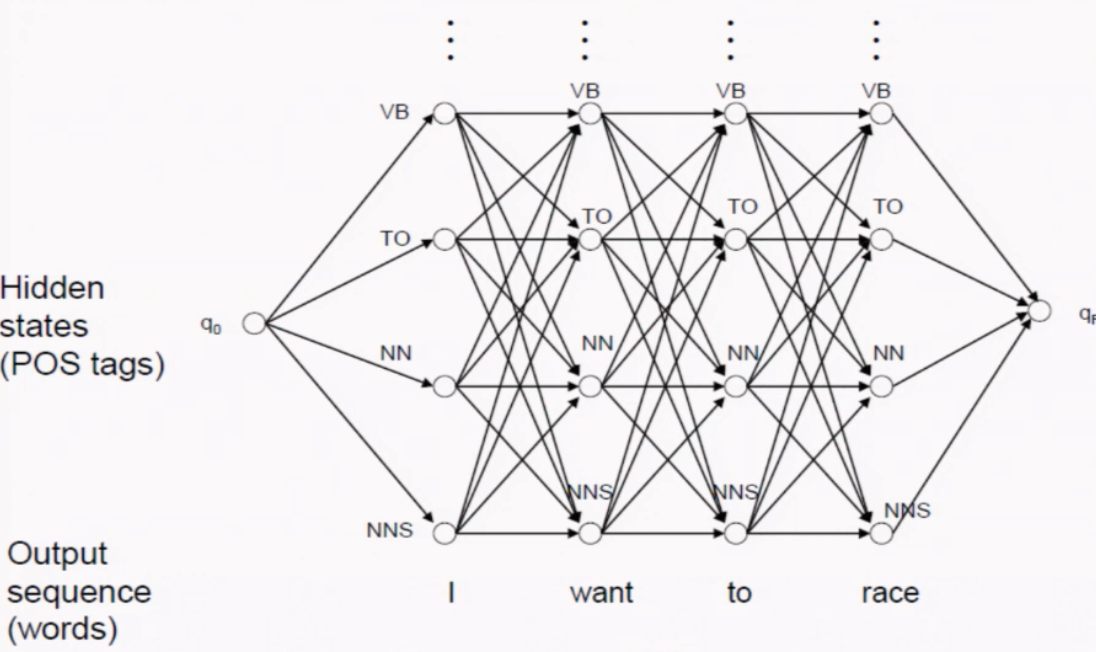
* + - **Numerals** – one, two, first, second
* **Part-of-Speech Tagging**
  + Input a sentence to return a sentence that is tagged with its most suitable part-of-speech tag for each word as output
  + helps to resolve ambiguity as a word has different semantics based on its context and POS tag
    - in this context, word ***x*** is a verb not a noun, hence the context should be so and so…
  + **Rule-based tagging**
    - Use a dictionary to assign each word a list of potential POS tags and then pass it through a set of disambiguation rules to narrow down to a single best POS for each word
    - Time consuming and difficult to generate the set of rules
    - Time complexity scales with the number of rules
    - Unable to account for colloquial speech since rules are not that strictly abided by
  + **Stochastic Hidden Markov Model**
    - Find the sequence of tags that maximises the conditional probability of a sequence of tags given the sequence of words



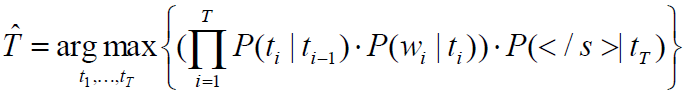
\*we can just remove 1/P(W) since it is a constant across all variable T



* + - That is given by the
      * probability that a tag is followed by another specific tag
      * probability that a word is tagged that specific tag
      * probability that the sentence is ended on that tag
    - implicit Markov assumption in finite horizon
      * scopes the dependencies of subsequent words to simplify probability computation
    - inefficient as it scales with number of words
    - 2 components required by tagging model:
      1. probability of a tag given a tag
      2. probability of a word given a tag
    - aij represents the probability of moving from state *i* to state *j*
    - bi(ot) represents the probability of an observation ot from state *i*

****

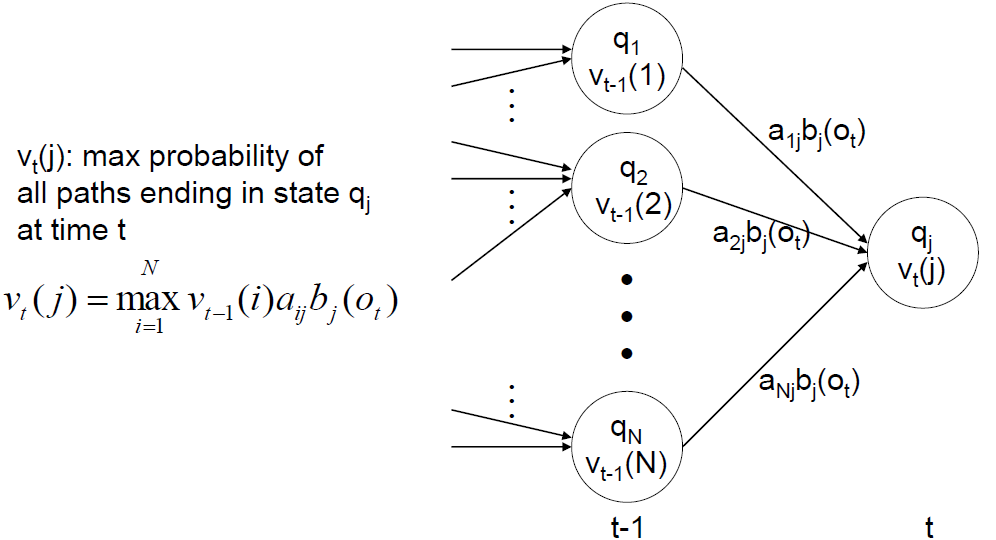
* + - **Direct evaluation method**
      * traverse the HMM lattice to find the most probable state sequence
      * O(T· NT), where T is # of words and N is # of POS tags
        + NT sequences since there are N POS possibilities for **each** word

****

* + - * + notice that each word in a sentence is now interlaced with a tag 🡺 number of tokens = 2T + 2 🡺 need 2T + 1 operations to multiply them together!

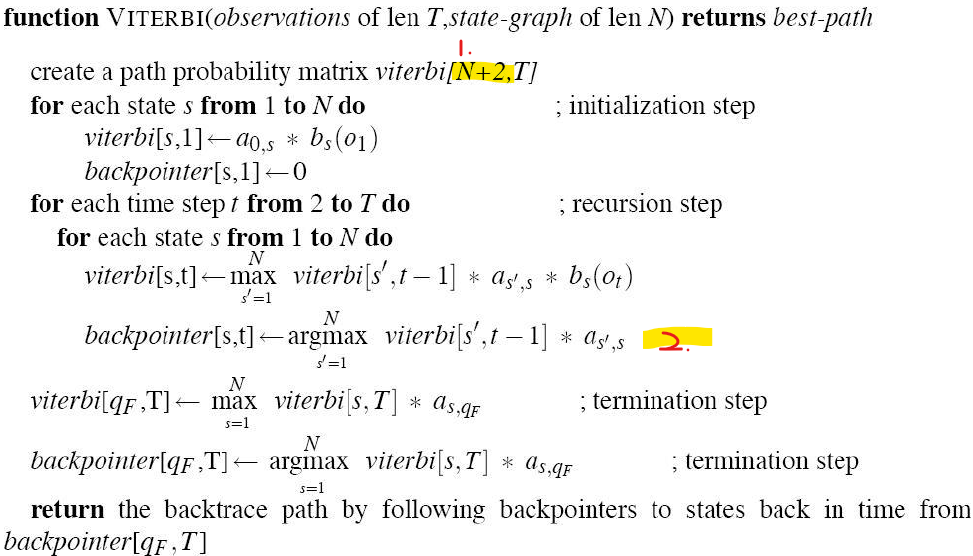
this simply grows in a factor of T

* + - * + inefficient since it grows exponential with T!!
    - **Viterbi Algorithm**
      * Dynamic algorithm with complexity that is linear with T = O(T· N2)
        + T columns, N rows 🡺 T \* N nodes
        + For each node, we need to do N operations
      * Computes exact POS tags as direct evaluation

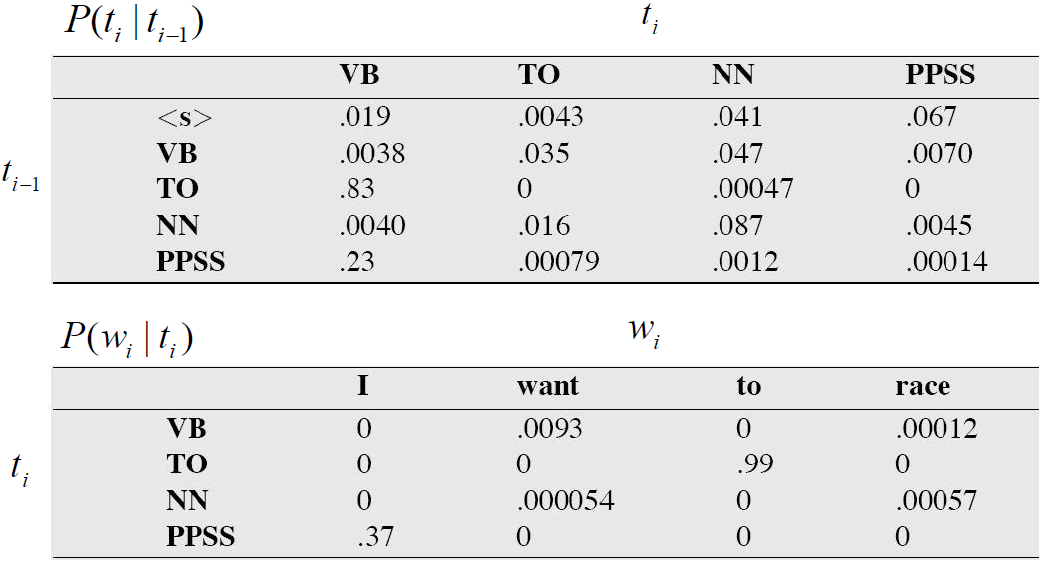
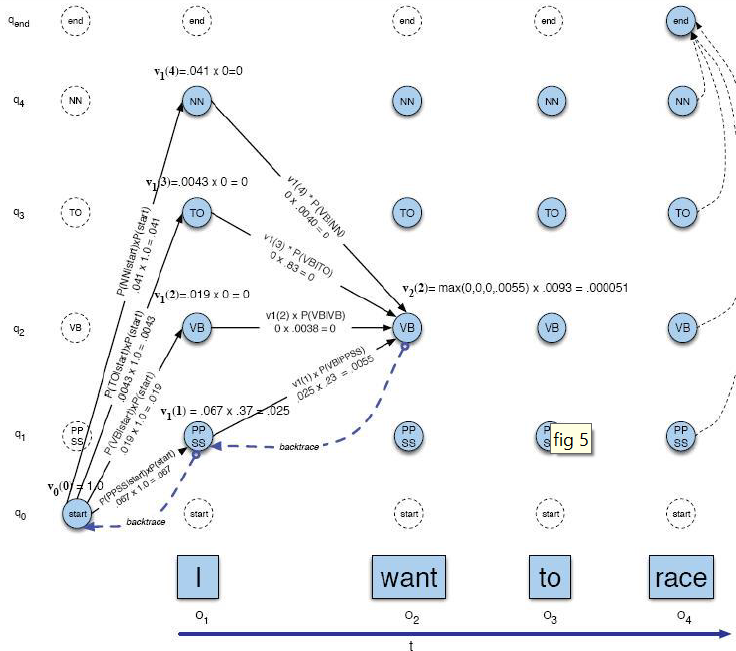
****

\****j*** refers to the ***jth*** state; ***vt***(***j***) is the probability of tagging tag ***t*** to state ***j***

* + - * vt(j) is simply the maximum probability of reaching the t-1th term (state ***i***) \* probability of transitioning from state ***i*** to state ***j*** \* probability of observing that tag ***t*** being assigned to state ***j***



* + - 1. +2 is required for the sentence tags <s> and </s>
      2. the term of bs(ot) is dropped because it is a constant
         * it does not matter from which state you came from for backpointer trace
         * the term is not dropped in the Viterbi calculation as it is required for subsequent calculations
      * trace



* + - * + we are always going to compute **P(ti|ti-1)** before **P(wi|ti)**

For the first word, what is the probability that a tag ***i*** is going to appear before the ***i-1*** tag, which is <s>

Then we multiply the probability that the first word, I, is tagged as such, to give us the probability **P(ti|ti-1)**· **P(wi|ti)**

For future transitions, we repeat:

we first multiply the accumulated probabilities with this transition probability **P(ti+1|ti)** and take the maximum

then we take the probability and multiply the probability that the 2nd word is tagged as the **i+1** tag

* + - * it is important to do smoothing to avoid 0 probabilities in the above example
      * unknown word modelling
        + treat them like words that appear exactly once in the training data
        + check word features

ratio of unknown words (open vs closed classes)

capitalization signifies pronoun

morphological affixes

if a word ends with -ing, high chance to be a verb, specifically tag VBG

* + - * + **P(wi|ti) = P(unknownword|ti)· P(capital|ti)· P(affixes|ti)**
    - generally matches human POS tagging rates (~97%) !
      * baseline ~90%
    - **Evaluating tagger performance**
      * N-fold cross-validation
        + Split the corpus into N equal parts, D1,D2, …, Dn
        + Iteratively, pick 1 set to be the test set while the rest is the training set.
        + Obtain the average accuracy after all ***n*** iterations

# Natural Language Processing – Linear Models

* Use supervised ML algorithms to infer usage patterns and regularities from pre-annotated input-output pairs
  + Neural networks learn parameterized differentiable mathematical functions while deep learning composes these functions together to learn them together
    - Need to learn how to represent input data – achieved via word embedding
  + **Word embedding (embedding a word into some dimensional space)**
    - Map discrete symbols (your character strings aka words) to continuous vectors in a low dimensional space
    - This representation of words as vectors is learned automatically during training by maximizing some objective like word prediction
    - Word distances == Vector distance
      * Synonyms have a smaller distance!
    - Addresses discreteness of words and sparsity of data sets!
  + **Linear Models**
    - ***f(x) = x· W + b (W is a din x dout matrix)***
      * ***x*** is the input data which is a row vector; we are trying to learn how to tweak parameters ***W*** and ***b*** to obtain ***f(x)***
    - Binary classification
      * dout = 1
      * based on the sign of f(x), classify into the 2 classes
    - Log-linear Binary classification
      * probability that a classifier assigns to a class
        + P(y = 1|x) = 1 / ( 1 + e-x) = 1 / (1 + e-(xW +b))
    - Multi-class classification
      * dout = the number of classes
      * take the max of the output vector to assign the class
    - Log-linear Multi-class Classification
      * softmax(x)[i] = ex[i]/( **Σj** ex[j])
        + softmax maps all values in the output matrix to a value between 0 and 1 to model a probability distribution
        + they will sum up to 1
    - Feature extraction
      * Map real world object to a vector of measurable quantities that provide the algorithm useful clues as to how to train and set **W** and **b**
  + **Loss Functions**
    - L(**ŷ**, **y**) maps 2 vectors, predicted label **ŷ** and true label **y**, to a scalar quantifying the loss due to inaccuracies in the prediction
    - Training finds parameters that minimizes the sum of loss function and regularization term

**Θ = argminθ**

* + - where n is the number of training examples, ***f(xi; θ)*** is the predicted vector of learned function, yi is the true function, and R is the regularization term
      * want to find the an output vector which minimizes the average loss
    - **R** helps to control complexity of parameter values and to prevent overfitting
      * weighted by ***λ*** and is a hyperparameter tuned based on development set
    - **Binary cross-entropy loss (logistic loss)**
      * Used in binary classification with conditional probability output
        + y { 0, 1 }
        + 0 < ŷ < 1, where ŷ = sigmoid(***f***(***x***)) = P(y = 1|x)
        + Llogistic(ŷ, y) = -y log(ŷ) – (1 – y) log(1- ŷ)

Maximises the log conditional probability for each training example

* + - **Categorical cross-entropy loss (negative log likelihood)**
      * Lcross-entropy(ŷ, y) =
      * for hard-classifications, simply Lcross-entropy(ŷ, y) = -log(ŷ[t]), since the rest of the inputs will be 0 and only the true class will be ‘counted’
    - **Ranking loss (margin based)**
      * If we only have positive training examples, we can generate negative examples by corrupting some of them to let us differentiate between the examples and learn which is positive which is negative
      * Given a positive example ***x*** and negative example ***x’***, we aim to get

***f(x) – f(x’) > 1, --* (1)**

to differentiate between positive and negative examples.

* + - * Lranking(margin)(***x***, ***x’***) = max(0, 1 – (***f***(***x***) – ***f***(***x’***)))
        + Hence, when **(1)** is achieved, Lranking(margin) will return 0, as we have adequately distinguished between positive and negative examples
    - Regularisation
      * R equates complexity with large weights
        + **L2**

RL2(***W***) = ***||W||22***= (sum of squares)

* + - * + **L1**

RL1(***W***) = ***||W||1***= (absolute difference)

* + - * + **Elastic Net (combination)**

Relastic-net(**W**) = λ1RL1(**W**) + λ2RL2(**W**)

* + - Gradient descent
      * To minimize loss functions in a series of successive approximations done in the right measured direction
        + L(***w1***) > L(***w2***) > … > L(***wmin***)
        + Given ***w***, find ***v*** such that L(***w***) > L(***w*** + ***v***) and **||*v||*** is small

small ***v*** 🡺 L(***w*** + ***v***) can be approximated by Taylor Series

L(***w*** + ***v***) L(***w***) + ***v*·** L(***w***)

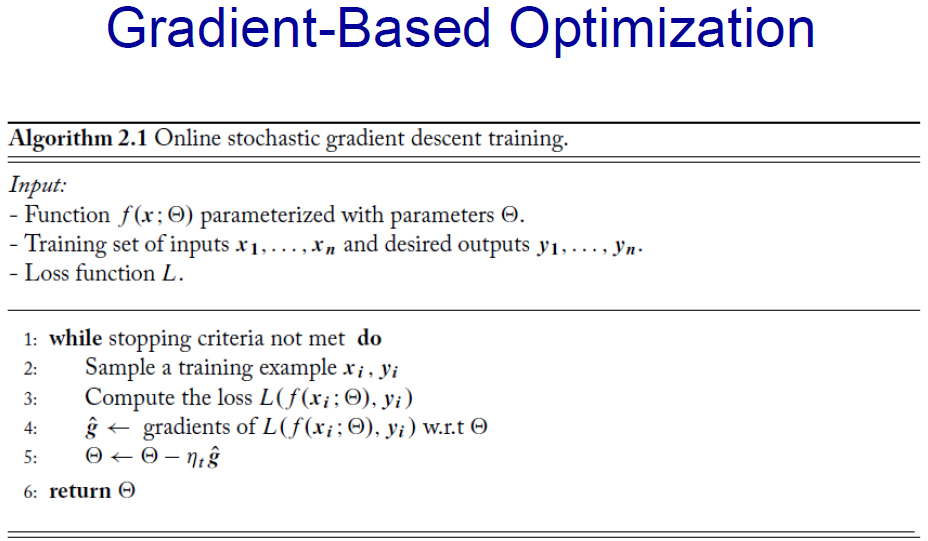
L(***w0***, ***w1***, … , ***wm-1***) = , , where L is simply the gradient

* + - * + We obtain the steepest descent when ***v*** is chosen collinear to L(***w***), i.e.

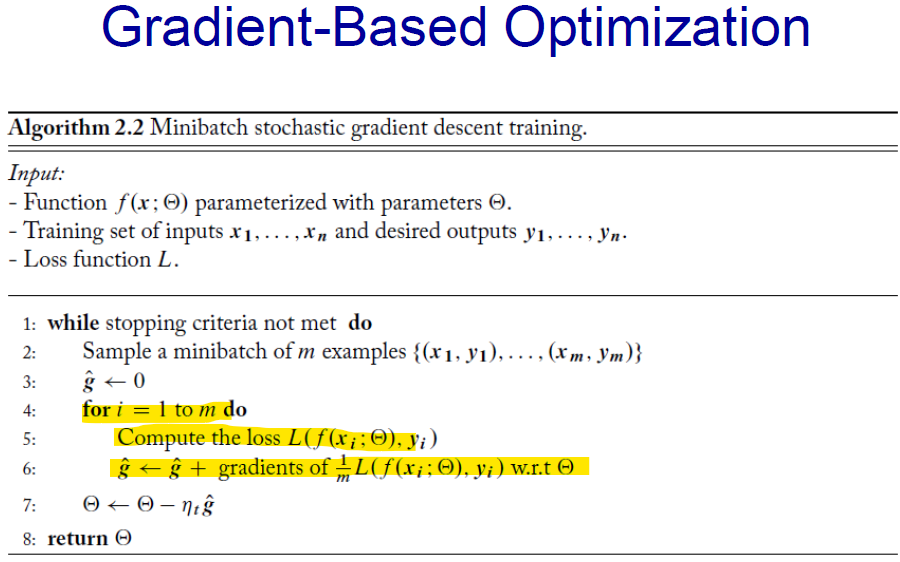
***v*** = -αL(***w***), where α > 0 (negative for descent), to give

L(***w*** – -αL(***w***)) L(***w***) -αL(***w***)2

* + - * αk is the learning rate (step size)
        + faster convergence if ak decreases over the iterations to prevent accidental ascending!
      * Iteration terminates when L(***w***) is less than a predefined threshold
      * If L(***w***) is a convex function, guaranteed to find global minimum, else can only guarantee a local minimum
      * Pseudo-code:



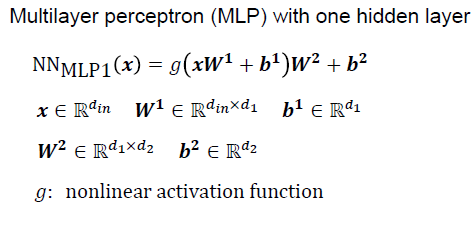
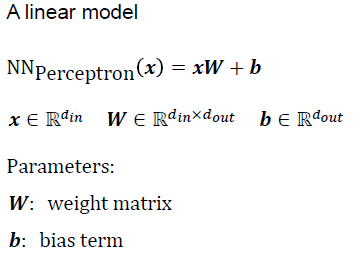
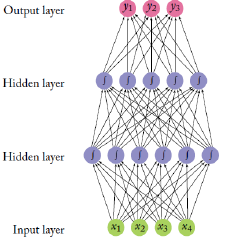
* + - * alternatively, batch-wise implementation

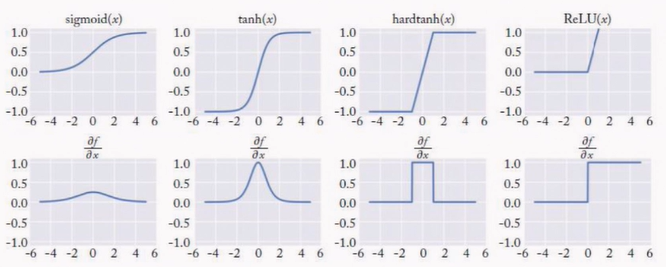


* + - * minibatch size ***m*** is a hyperparameter to be tuned with the development set
      * Higher values of ***m*** allow for
        + efficiency through parallelization of highlighted portion
        + better estimate for corpus-wide gradient
      * Smaller values of ***m*** allow for
        + more updates and faster convergence to target function

# Neural-Networks: Feed Forward NN

* Linear models are unable to capture simplistic logical constraints like even XOR
* **Solution:** Apply nonlinear input transformations to allow inputs to be linearly separable
  + this is what FF NN does



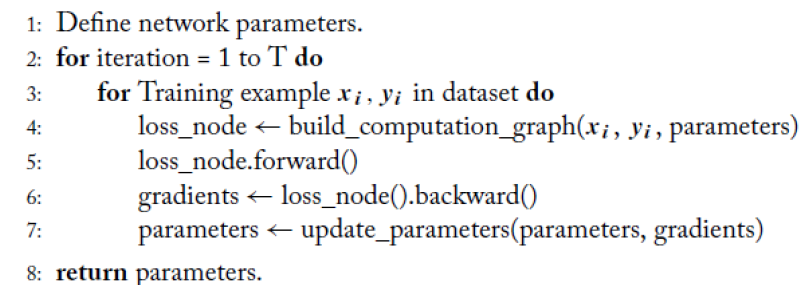
* embedding of the evaluation is a non-linear transformation of the inputs
* Often, the 1st layer transforms the data into a good representation g(xW1 + b1) for the 2nd layer to apply a linear classifier on
  + we can recursively do this non-linear transformation!
* NN outputs a dout dimensional vector
  + real values for regression
  + sign() for binary classification
  + k for k-class classification
    - can be softmaxed to turn output vector into a distribution
* While MLP1 theoretically is able to approximate any functions, it is difficult to set parameters to do so
  + does not guarantee right function can be found
  + does not state how large a hidden layer needs to be
    - layer size might grow exponentially large to approximate some NN
  + hence we bother with multiple hidden layers and other complex NN structures
* **Activation functions g()**
  + help introduce non-linearity to output of a neuron by evaluating inputs with bias
  + Types
    - Sigmoid
    - RELU
    - tanh:
    - hard tanh
  + Dropout regularization
    - Randomly drops half of the neurons in the NN to 0 during stochastic gradient descent
      * Prevents NN from overfitting training data
    - Introduce masking vectors that takes the output of 1 layer and mask them with 0 before feeding it to the next layer for each layer

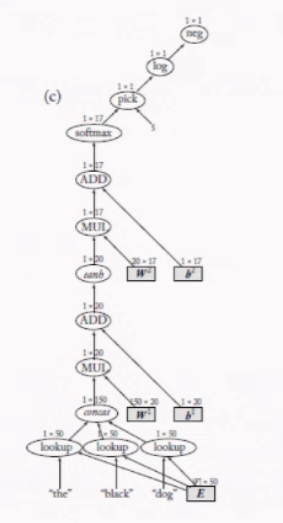
**h1 = g1(xW1 + b1)**

**m1 ~ Bernoulli(r1)**

**h1masked = m1· h1**

**h2 = g2(h1masked W2 + b2)**

* + - Notice that the Bernoulli trials will drop half of the neurons
* **Training Neural Networks**
  + Similar to gradient descent for linear models
  + Let us represent our NN as a mathematical expression, which can be represented as a directed acyclic computation graph
    - Nodes denote mathematical operations or bound variables
    - Edges denote the flow of intermediary values between nodes
    - traverse the graph from child to parent

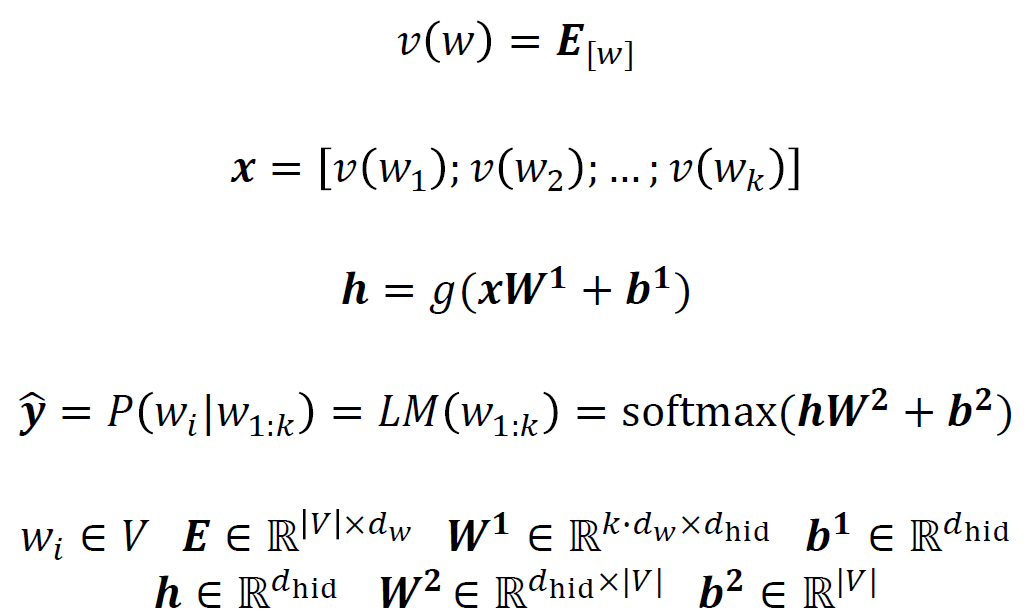


* + Forward computation
    - starting with children nodes, use the outputs from preceding nodes as arguments for your node’s function and return them as output to your parent
    - for i to N (starting with child nodes)

let a1, a2, …, am denote your m children

outputi = f(output(a1), …, output(am))

* + - nodes are ordered in topological order
  + Convergence
    - only guaranteed to converge on local minima unless function to be learnt is convex
      * good enough in practice
    - **Methods in practice**
      * Initialization of weight vector W
        + Sample points uniformly from U
      * Random restarts
        + avoids local minima by randomly restarting from different initializations
      * Ensemble of multiple models
        + Build different models using different set of random initializations
        + Combine models for predictions by averaging the output vectors
      * Dealing with vanishing and exploding gradients
        + Due to chain rule resulting in multiplication of many derivatives
        + Specialised architectures like LSTM to implement forget gates to specify what traits are to be learnt
        + Clipping gradients if they exceed a threshold by normalizing them
      * Tune learning rates
        + Too small too slow
        + Too large might not converge
        + Sample with range of learning rates
* Language modelling
  + Failures of n-gram models
    - requires implementation of manual smoothing schemes
    - computationally expensive to use larger order n-grams
      * grows **exponentially** to order power n
    - cannot generalize across contexts
      * this would have been done by using higher order n-grams but expensive
  + NN based models solves the above problems
    - allows for larger contexts sizes with only a **linear** increase in number of params
      * this context preservation is done via word vectors



* + - input to multilayer perceptron is simply a sequence of k words
    - output is a probability distribution of the next word
      * 1st line embeds the word as a vector
      * 2nd line concatenates the different words
      * 3rd line feeds this as input to the perceptron
      * 4th line illustrates the output as a probability distribution\
      * note that V here includes are pseudo tokens as well
        + UNK tokens, <s> </s> tags etc.
      * Training examples use 1st k words as features and k+1th word as target for classification
    - Since we are interested in predicting the next word, our loss function here is a **categorical cross-entropy loss**
      * Lcross-entropy(ŷ, y) = -log(ŷ[t]), where ***t*** is the index of the correct word
  + Features
    - raw texts can be used as input
    - word representations are obtained as a byproduct
    - however softmax operation is very costly for a large vocabulary
* Word embedding
  + Random initialization
    - choose from a uniform distribution
  + Supervised task-specific pre-training
    - Leverage on an auxiliary task’s access to a larger set of labeled training data and use the embedding trained from of it
  + Unsupervised pre-training
    - use raw texts and try to predict the next word based on the context given by the corpus
    - supervised from previous texts in the same corpus, but no manual labelling involved
  + **random initialization** used for commonly occurring features like POS tags
  + **pre-training methods** used for potentially rare features like individual words
  + note that the **pre-trained vectors** have features of the auxiliary task/corpus, such as frequency in that context
    - might want to normalize vectors to erase the frequency
    - can further tune vectors to fit specific task
      * might lose generalization capabilities and overfit training data!
  + Algorithms
    - Collobert and Weston
      * Idea
        + Discover vector by using context based off **surrounding words** (not receding!)
        + Assign scores instead or probabilities

avoids inefficient softmax thereby making complexity independent of vocabulary size

correct words need to be scored higher than incorrect ones

* + - * **Loss function**
        + L(w, c, w’) = max(0, 1 – (s(w,c1:k) – s(w’, c1:k)))

if margin is smaller than 1, L will return non-0 error