

Language ambiguity (has multiple precise meanings): - word 'bring me the file' (resolve w/ POS) - syntactic 'I shot an elephant in my pj's' - semantic 'the rabbit is ready for lunch' - referential 'Pavarotti is a big opera star' - non-literal 'it's raining cats and dogs'		Deep Learning learns/abstract functions instead of rules based on maintenance of intuitive linguistic rules.	Lexicon – morph(eme)ological analysis (stem and affix e.g. 'cat'+ 's') Word segmentation (tokenization) Word normalization (case/acronyms/spelling) Lemmatization 'sing, sung, sang' → 'sing' Stemming (common root, above 's') Part-Of-Speech (tag words with noun, verb...)	Context-Free Grammar: Derive sentence structure through a parse tree S → NP VP, NP → Det N, VP → V NP, VP → V, VP → V PP, PP → P NP Discourse: meaning of a text (relationship between sentences) Pragmatics: intentions/commands Corpus: a collection of documents Document: one item of corpus (sequence) Token: atomic word unit Vocabulary: unique tokens across corpus. One-Hot Encoding: sparse (wasted space), orthogonal vectors (every word is equidistant), cannot represent out of vocab well
Sigmoid (binary class): $\frac{1}{1+e^{-x}}$ , ReLU: $\max(0, x)$ , Tanh: $\frac{e^x - e^{-x}}{e^x + e^{-x}}$ Softmax (k-class): $\frac{e^{-\epsilon_k}}{\sum_k e^{-\epsilon_k}}$ MSE (regression) $\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$ , Binary cross-entropy: $-\frac{1}{N} \sum_{i=1}^N (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$ Categorical cross entropy (k-class): $-\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^C y_c^{(i)} \log(\hat{y}_c^{(i)})$	Euclidean Distance: $\sqrt{\sum_{i=1}^N (q_d - d_i)^2}$ Cosine Similarity: $\cos(\theta) = \frac{p_1 \cdot p_2}{\ p_1\  \ p_2\ }$ Analogy Recovery: offset of the vectors reflect their relationship. $a - b \approx c - d \iff d \approx c - a + b$	Window: window consists of target and context (surrounding), Window size = radius Continuous Bag Of Words: context → target, Skip-gram: target → context (give as one-hot, get word representation, map embedding to target words using weight matrix, apply softmax). Train with list of pairs (target, context) by sliding window over input. Loss: $p(w_{t+j} w_t) = \frac{\exp(u_{t+j} \cdot h_{w_t})}{\sum_{w' \in V} \exp(u_{t+j} \cdot h_{w'})}$ , the aim: $\max \prod_i \prod_j p(w_{t+j} w_t) \rightarrow \min_{\theta} - \sum_t \sum_j \log p(w_{t+j} w_t; \theta) \rightarrow \frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} w_t; \theta)$ over all elems in corpus. However, the bottom term in the $p(w_{t+j} w_t)$ is inefficient to compute across the entire corpus vocabulary. Therefore, train a Negative Sampling model to predict whether a word appears in the context of another: $\log p(D = 1 w_t, w_{t+1}) + k E_{\tilde{c} \sim P_{\text{context}}(\cdot w_t)} [\log p(D = 0 w_t, \tilde{c})]$ where $p(D = 1 w_t, w_{t+1})$ is a binary logistic regression probability of seeing the word $w_t$ in the context $w_{t+1}$ . Approximate the expectation by drawing random words from vocabulary, and on left choose positive pairs. Thus the equation is replaced: $p(D = 1 w_t, w_{t+1}) = \frac{1}{1 + \exp(-u_{w_{t+1}} \cdot h_{w_t})}$ . We can sample k (5-10 words) with frequency or random sampling.		
Byte-Pair Encoding: Instead of manually specifying rules for lemmatisation or stemming, lets learn from data which character sequences occur together frequently. 1) Start with a vocabulary of all individual characters 2) Split the words in your training corpus also into individual characters + '.' at end 3) Find which two vocabulary items occur together most frequently in the training corpus 4) Add that combination as a new vocabulary item 5) Merge all occurrences of that combination in your corpus 6) Repeat until a desired number of merges has been performed. For unknown words follow above and apply replacements in order discovered.	Classification: $\hat{y} = \arg \max_y P(y x)$ predict which y is most likely given input x. In the MultiNLI corpus we are given pairs of sentences (premise, hypothesis) with classification problem (Entailment: If hypothesis is implied by premise, Contradiction: If hypothesis contradicts the premise, Neutral: otherwise). Accuracy = $\frac{TP+TN}{TP+FP+FN+TN}$ , $f1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{2TP}{2TP+FP+FN}$ , Macro average: averaging of each class F1 scores: increases the emphasis on less frequent classes. Micro average: TPs, TNs, FNs. FPs are summed across each class e.g. $\frac{\sum_c TP_c}{\sum_c TP_c + \sum_c FP_c + \sum_c FN_c} = \text{Accuracy}$	Naive Bayes Classifier: $P(y x) = \frac{P(x y)P(y)}{P(x)}$ $\hat{y} = \arg \max_y P(y x) = \arg \max_y P(y) \prod_{i=1}^I P(x_i y)$	In a Bag of Words count the number of times a token appears in the vocabulary per class. In One smoothed Naive Bayes: $P(x_i y) = \frac{\text{count}(x_i, y) + 1}{\sum_{x' \in V} (\text{count}(x', y) + 1)}$ ( $= \frac{\text{count}(x_i, y) + 1}{(\sum_{x' \in V} \text{count}(x', y) +  V )}$ ) Binary Naive Bayes: only consider if a feature is present, rather than considering every time it occurs. Controlling for negation: pre-pend 'NOT'.	
Language Models: Assign probabilities to sequence of words, like predicting the next word in a sentence. Uni-directional: use information from left to generate predictions about words on the right. Bi-directional: use information from both sides to fill the target. N-gram: need because language is flexible, and a natural extension of a sentence may not appear in corpus. $P(w_n w_{n-1}^{n-1}) \approx P(w_n w_{n-1}^{n-1}) = \frac{C(w_{n-1}^{n-1}, w_n)}{C(w_{n-1}^{n-1})}$ . If certain words absent the probability is 0.	Change this with add-one smoothing: $P_{\text{add-1}}(w_n w_{n-1}) = \frac{C(w_{n-1}, w_n) + 1}{C(w_{n-1}) + V}$ (however, this influences the less frequent words (not more). Therefore, implement backoff smoothing. Interpolation: $P_{\text{interp}}(w_i w_{i-2}w_{i-1}) = \lambda_1 P(w_i w_{i-2}w_{i-1})$	Feed-Forward LM (FFLM): get word embeddings for words and concat them together embedding: $V \times E$ , concat: $c_k \in \mathbb{R}^{C \times E}$ with a output layer $CE \times V$ then softmax. RNN: $h_{t+1} = f(h_t, x_t) = \tanh(W_h h_t + U x_t)$ , $y_t = W_h h_t + B y_t$ , Teacher Forcing: if there is an incorrect label we force it to use the actual expected label. The ratio can be 100% (i.e. full teacher forcing), 50%, or you can even anneal it during training. This may cause Exposure Bias where it never actually uses its own predictions during training. Bidirectional RNN: When comparing the number of parameters in this vs a single directional rnn, it doubles. The output layer also doubles (because we are given a matrix $H \times O$ twice from each direction making the output layer have dimension $H \times H \times O$ ). This extends to multi-layered RNNs.	Naive translation: minimise negative log likelihood loss $-\sum_{t=1}^T \log p(\hat{y}_t y_{<t}, c)$ . Take hidden vector encoding from encoder RNN into the decoder. This limits amount of information it can retain for longer vectors, as more words get decoded the hidden layer continues through the decoder and loses its information.	
$S(w_i w_{i-2}w_{i-1}) = \begin{cases} \frac{C(w_{i-2}w_{i-1}, w_i)}{C(w_{i-2}w_{i-1})}, & \text{if } C(w_{i-2}w_{i-1}, w_i) > 0 \\ 0.4 \cdot S(w_i w_{i-1}), & \text{otherwise} \end{cases}$ $S(w_i w_{i-1}) = \begin{cases} \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}, & \text{if } C(w_{i-1}, w_i) > 0 \\ 0.4 \cdot S(w_i), & \text{otherwise} \end{cases}$ $s(w_i) = \frac{C(w_i)}{N}$ $+ \lambda_2 P(w_i w_{i-1}) + \lambda_3 P(w_i)$ , where $\lambda_1 + \lambda_2 + \lambda_3 = 1$ to combine evidence from multiple n-grams. To make a prediction $P(w_1, \dots, w_n) = \prod_{k=1}^n P(w_k w_1^{k-1})$ we switch into log space $\log P[\cdot] = \sum_{k=1}^n \log(P(w_k w_1^{k-1}))$ however the longer the sentence the lower its likelihood. Switch to Perplexity: where n is the number of words: $PPL(w) = \sqrt[n]{\prod_{i=1}^n P(w_i w_{1:i-1})}$ the higher the conditional probability of the word sequence, the lower the perplexity. Thus, minimizing perplexity is equivalent to maximizing the test set probability according to the language model. It is a measure of surprise in an LM when seeing new text. For a single word, the score is 1. If the goal of the language model is to support with another task, the best choice of language model is the one that improves downstream task performance the most (extrinsic evaluation). Perplexity is less useful in this case (intrinsic evaluation). Cross Entropy Loss: The cross-entropy is useful when we don't know the actual probability distribution p that generated some data. $H(T, Q) = -\sum_{i=1}^N \frac{1}{N} \ln q(x_i)$ . To convert to perplexity take the base of the logarithm and perform $PPL(M) = \text{base}^H$ .		 LSTM: $f_t$ : forget gate = $\sigma(W_{f_t} x_t + W_{h_f} h_{t-1} + b_f)$ $i_t$ : input gate = $\sigma(W_{i_t} x_t + W_{h_i} h_{t-1} + b_i)$ $o_t$ : output gate = $\sigma(W_{o_t} x_t + W_{h_o} h_{t-1} + b_o)$ $g_t$ : candidate cell = $\tanh(W_{g_t} x_t + W_{h_g} h_{t-1} + b_g)$ $c_t$ : memory cell state = $f_t \odot c_{t-1} + i_t \odot g_t$ $h_t$ : hidden state = $o_t \odot \tanh(c_t)$ then, $h_t$ : output of cell = $\phi_g(W_{h_t} h_t + b_h)$ simply. Struggles learn long-term deps. Less vanishing gradient because additive formulas means we don't have repeated multiplication. The forget gate controls when to preserve gradients or not. $W_{ii} = (H \times E)$ and $W_{hi} = (H \times H)$ since we go from embeddings to hidden representation.		
Statistical Machine Translation: a pipeline of Alignment model (responsible for extracting the phrase pairs) Translation model (contains phrases alongside their translation lookup table) Language model (contains the probability of target language phrases). The objective is $p(t s)$ given a source sentence predict a sentence t. $\arg \max p(t s) = p(t)p(s t)$ . Downsides: Sentence Alignment (In parallel corpora single sentences in one language can be translated into several sentences in the other and vice versa) Word Alignment (no clear equivalent in the target language) Statistical anomalies (Real-world training sets may override translations) Idioms (Only in specific contexts do we want idioms to be translated) Out-of-vocabulary words.	Implementing Attention+RNN: For each hidden state in the encoder, $c_t = \sum_{i=1}^I \alpha_i h_i$ combine these into a context vector, it is dynamic and contextualised representation. We then feed a decoder this context vector and the <init> token. This is the energy, and it is calculated with $e_t = a(s_{t-1}, h_t) = v^T \tanh(W s_{t-1} + U h_t)$ where $e_t \in \mathbb{R}^1$ is the unnormalized energy score, and a is a learnt neural network. $s_{t-1} \in \mathbb{R}^{D \times 1}$ is the previous decoder hidden state, $h_t \in \mathbb{R}^{2D \times 1}$ encoder hidden state for the ith word, $v^T \in \mathbb{R}^{D \times 1} \wedge v \in \mathbb{R}^{1 \times D}$ , $W \in \mathbb{R}^{D \times D}$ , $U \in \mathbb{R}^{D \times D}$ then the energy gets normalized with a softmax to form the $\alpha$ .	 Gated Recurrent Unit: $r_t$ : switch gate = $\sigma(W_{r_t} x_t + W_{h_r} h_{t-1} + b_r)$ $z_t$ : reset gate = $\sigma(W_{z_t} x_t + W_{h_z} h_{t-1} + b_z)$ $g_t$ : candidate state = $\tanh(W_{g_t} x_t + r_t * (W_{h_g} h_{t-1} + b_g))$ $h_t$ : hidden state = $(1 - z_t) * g_t + z_t * h_{t-1}$ no longer has input/output gate but maintains forgetting mechanism. GRU more efficient computing & less over-fitting but LSTM good default choice.		
BLEU: reports a modified precision metric for each level of n-gram Modified Precision score $p_n = \frac{\text{Total Unique Overlap}_n}{\text{Total n-grams}}$ where Total n-grams is the 'total n-grams' in the produced sentence (not unique), and 'total unique overlap' is the unique set of unique tokens appearing in the output have appeared in the union of the reference sentences. BLEU-4: $\text{BP}(\prod_{n=1}^4 p_n)^{\frac{1}{4}}$ , $\text{BP} = \min(1, \frac{\text{MT Output Length}}{\text{Reference Length}})$ where $p_n$ defined above. Used to mostly encourage the hypothesis to be of a similar length to the reference. A shortcoming of BLEU is that it focuses a lot on the precision between Hyp and Ref, but not the recall. Chr-f: character n-gram F <sub>1</sub> score. Balances character precision (percentage of n-grams in the hypothesis which have a counterpart in the reference) and character recall (percentage of character n-grams in the reference which are also present in the hypothesis). CHRRF <sub>1</sub> = $\frac{2 \times \text{CHRP} \times \text{CHRR}}{\text{CHRP} + \text{CHRR}}$ where CHRP: percentage of n-grams in the hypothesis which have a counterpart in the reference, CHRR: percentage of character n-grams in the reference which are also present in the hypothesis. Good for morphologically rich languages. TER translation error rate: Minimum # of edits required to change a hypothesis into one of the references. TER is performed at the word level, and the "edits" can be: Shift, Insertion, Substitution and Deletion. ROGUE F-1 score n-gram, ROGUE-L: F-score of longest common subsequence e.g. source: The cat is on the mat hyp: The cat and the dog so LCS: the cat the, precision = 3/5, recall = 3/6. METEOR: summarization and captioning, more robust than BLEU. Downside: cannot capture context of sentence to return a higher score. BERT-score: a trained language model that can give contextual representations of tokens/words. May return different scores when evaluated against different models.		 Transformers: notice that there are residual connections. The encoder receives an input of S words with encoded dimensions D. Complexity: time $Q = (q_1, \dots, q_N)^T \in \mathbb{R}^{N \times d_q}$ , $O(MN d_q + MN d_v)$ , space $K = (k_1, \dots, k_M)^T \in \mathbb{R}^{M \times d_k}$ , $O(MN \times d_k)$ , $O(MN + Nd_v)$ , # parameters (only key and value) so $V = (v_1, \dots, v_M)^T \in \mathbb{R}^{M \times d_v}$ , $K, V : O(Md_k + Md_v)$ and in self attention this is $O(Nd_v)$ i.e. $a(\cdot)$ activation function applied row-wise ( $N = M := D$ and $d_q = d_k := d_v$ ) $Q \cdot V^T = K$ . Hard attention is when $a = \text{onehot}(\arg \max x_i)$ and Soft attention is when $a = \text{softmax}(x)$ . Self-attention: The self-attention mechanism allows each input in a sequence to look at the whole sequence to compute a representation of the sequence. Multi-head attention $M \text{ multihead}(Q, K, V, a) = \text{concat}(\text{head}_1, \dots, \text{head}_N) W^O$ , where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V, a)$ and Attention is defined as before. This helps define different hidden similarity measures. Normalization: for each sample in a batch $LN(x_n) = \gamma \frac{x - \mu}{\sigma} + \beta$ with learnable params to scale. Residual connections: Residual connections help mitigate the vanishing gradient problem, where Vanishing gradients = tiny weight changes. Residual connections provide a shortcut for information to flow to layer layers of the network, where the output of an earlier layer is added directly to the output of a layer layer. Positional Encodings transformers are position invariant by default; positional encoding vector is independent of the word, but only to the position in the sequence it is either learnt or where pos=position of word, d=dimension of output space, i = column indices $PE_{pos, 2i} = \sin(\frac{pos}{10000^{\frac{2i}{d}}})$ and $PE_{pos, 2i+1} = \cos(\frac{pos}{10000^{\frac{2i}{d}}})$ smaller early on, larger later on (iterating i over d). Masked Attention: mask out some attention values by adding matrix with 0s and set upper triangle to $-\infty$ (useful for sequence prediction with a given ordering: in test time "future" is not available for the "current" to attend). Cross Attention: In self-attention, we work with the same input sequence. In cross-attention, we mix or combine two different input sequences. In the case of the original transformer architecture above, that's the sequence returned by the encoder module on the left and the input sequence being processed by the decoder part on the right. with i: current global step, warmup hyperparameter (4000), d: model dimensionality Decaying learning rate: $lr = \sqrt{\frac{1}{d}} \times \min(\sqrt{\frac{1}{d}}, i \times \text{warmup}^{-1.5})$		
Inference: Greedy Decoding: Outputs the most likely word at each time step (i.e. an argmax) fast but doesn't look into the future; can get weird later. Beam Search: Instead of choosing the best token to generate at each time-step we keep k possible tokens at each step. Maintain the log probability of each hypothesis in beam by incrementally adding logprob of generating each next token. Only the top k paths are kept. Temperature Sampling: problem with above is determinism. Therefore, divide logits by T and run softmax $\frac{e^{w_i/T}}{\sum_j e^{w_j/T}}$ . Multinomial sample over softmax probabilities.	Improving Performance: Back-translation: translate the source into one language and translate it back. The hope is, that once translated back the semantics is the same but syntactically it may be different. Synonym Replacement: Use dictionary & syntax trees to find appropriate synonyms/ Use word embeddings & nearest neighbours to find synonyms. Batching, Padding and Sequence Length: Group similar length sentences together in the batch or train your model on simpler/smaller sequence lengths first.			
TF-IDF: Problem: words in a query are weighted equally. Term Frequency – Measures how often a term occurs in a document. Inverse Document Frequency - Measures how common or rare a term is across all documents in the corpus. (Terms that appear in many different documents are less significant than those that appear in a smaller number of documents) $TF_{w,d} = \frac{\text{count}(w,d)}{\sum_{w'} \text{count}(w',d)}$ (frequency of w occurring together with d). $IDF_{w,d} = \log \frac{21}{ \{d \in D: w \in d\} }$ down-weights words that appear everywhere. $TF - IDF_{w,d,D} = TF_{w,d} IDF_{w,d}$	Contextual Word Representations: there is one word representation per word, even though in different contexts it may mean different things. RNNs capture this. ELMO: two directions: $p(t_1, t_2, \dots, t_N) = \prod_{k=1}^N p(t_k t_1, t_2, \dots, t_{k-1})$ and $p(t_1, t_2, \dots, t_N) = \prod_{k=1}^N p(t_k t_{k+1}, t_{k+2}, \dots, t_N)$ . Uses LSTMs with multiple layers of bidirectional slices. Intermediate representation + second layer allows for more complex second-level reasoning about words. Use ELMO to enhance. BERT: Bidirectional Encoder Representations from Transformers: Advantage from self-attention is that every word is just one hop away from every other word. In LSTMs or RNNs, if we wanted to get information from one word to another, we'd have to step through every word in the sequence and keep information in memory. Unlike Self-attention which combines directly information from every other word. Segment Embeddings, Position Embeddings, Token Embeddings. Train with Masked Language Modelling, too little masking = difficult to train, too much = not enough context. Next sentence prediction did these two sentences appear in this order? Using Pre-trained models: for sentence classification, insert <CLS> token and attach new layer to the output for this token. Token labelling: put classification pairs on each token. Pair classification: separate two sentences with a token + add classification head on top 'does sentence 1 entail sentence 2 or vice versa?' Question answering: We can structure the task with an input, and a paragraph of text which may contain some answers. Then train the model to label individual tokens to indicate which tokens are the answer, then simply extract these			

<ul style="list-style-type: none"> <li>• <b>BERT</b>: Trained with masked language modelling and next sentence prediction.</li> <li>• <b>RoBERTa</b>: Got rid of next sentence prediction, optimized hyperparameters.</li> <li>• <b>DeBERTa</b>: Focused on improvements to positional encodings</li> <li>• <b>SpanBERT</b>: mask contiguous tokens instead of random. Makes the task harder.</li> <li>• <b>DistilBERT</b>   <b>ALBERT</b>: training a small model to behave similarly to the bigger version</li> <li>• <b>BigBird</b>   <b>LongFormer</b>: Self attention has <math>O(N^2)</math> complexity. Models with sparse attention mechanisms have been proposed to extend the input length</li> <li>• <b>ClinicalBert</b>   <b>MedBert</b>   <b>PubMedBert</b>   <b>BEHRT</b>: BERT-like models trained on specific domains.</li> <li>• <b>ERNIE</b>: add special entity embeddings into transformer for added benefit.</li> <li>• <b>Multi-lingual Models</b> <ul style="list-style-type: none"> <li>◦ <b>Multi-modal Models</b>: combine text + visual</li> <li>◦ <b>ImageBERT</b>: encodes objects as additional tokens.</li> </ul> </li> <li>• <b>VitBERT</b>: two parallel BERT encoders, interact using co-attention module, Image features are extracted using pre-trained Faster R-CNN</li> <li>• <b>Masked Image modelling</b>: mask parts of image.</li> <li>• <b>Masked Protein modelling</b>: AlphaFold.</li> </ul>	<p><b>Pre-training encoder-decoder models</b>: Without parallel data: <b>prefix language modelling</b> give it half a sentence and predict the second half. <b>Shuffle words</b> and the task is to recover the original sentence <b>masking</b> just predict the full sentence. <b>Instruction Tuning</b>: took annotated datasets and phrased it as a Seq2Seq task. Place the query into a template into a conversational-sounding instruction.</p> <p><b>Pre-training decoder models</b>: train to optimize <math>p_{\theta}(w_i w_{1:t-1})</math> Great for tasks where the output has the same vocabulary as the pre-training data. <b>Fine tuning</b>: put a new layer on top and fine-tune the model for a desired task. <b>Zero-shot learning</b>: Give the model a natural language description of the task, have it generate the answer as a continuation. <b>One-shot learning</b>: In addition to the description of the task, give one example of solving the task. No gradient updates are performed. <b>Few-shot learning</b>: In addition to the task description, give a few examples of the task as input. No gradient updates are performed.</p>
	<div> <div> <p>Step 1 Collect demonstration data, and train a supervised policy.</p> </div> <div> <p>Step 2 Collect comparison data, and train a reward model.</p> </div> <div> <p>Step 3 Optimize a policy against the reward model using reinforcement learning.</p> </div> </div> <p><b>Advanced prompting and learning from human feedback</b>: Chain of thought to show reasoning. <b>Zero-shot chain-of-thought</b>: ask it to 'think step by step'.</p> <p><b>Retrieval Based language models</b>: NNs aren't a great place to store factual information because everything is distributed and smooth embeddings. Thus the language model acts as the controller, and the factual retrieval model finds relevant texts from any databases you have. This allows for citations.</p> <p><b>PROBLEMS</b>: language models trained with instruction finetuning use manually created ground-truth data to follow instructions. Therefore, 1) tasks like open-ended creative generation have no right answer, 2) language modelling penalizes all token-level mistakes equally, but some errors are worse than others. <b>Therefore</b> introduce human feedback by giving possible answer. However, human-in-the-loop is expensive. <b>Therefore, write a model to predict the scores of humans or to avoid noisy scores, write them to rank the answers instead.</b></p>

**Part of Speech Tagging**: tokenizing sentences with type (adj, noun, etc.). The task is to estimate  $P(T|W)$ ,  $T = t_1, \dots, t_n \wedge W = w_1, \dots, w_n$  as an instance of many to many classification.  $P(T|W) = \frac{p(W|T)P(T)}{P(W)} = P(W|T)P(T)$  where  $P(T) \approx P(t_1)P(t_2|t_1) \dots P(t_n|t_{n-1})$  by chainrule/bigram assumption and  $P(W|T) \approx P(w_1|t_n) \dots P(w_n|t_n)$  where we estimate  $P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} \wedge P(w_i|t_i) = \frac{C(w_i, t_i)}{C(t_i)}$ . Generate two tables with this data including tags <s> and <e> in the tag table. The table should be down: given I see Y, across what is the probability that I will see X. Then calculate with  $P(T|W) \approx \sum_i P(t_i|t_{i-1}) \cdot P(w_i|t_i)$  for each cell. This is greedy.

$Q = q_1 q_2 \dots q_N$ $A = a_{11} a_{12} \dots a_{NN}$ $O = o_1 o_2 \dots o_T$ $B = b_i(o_i)$ $\pi = \pi_1, \dots, \pi_N$	<p>A set of <math>N</math> states (tags)</p> <p>A <b>transition probability</b> matrix <math>A</math>, each <math>a_{ij}</math> representing the probability <math>P</math> of moving from state <math>i</math> to state <math>j</math>, s.t. <math>\sum_{j=1}^N a_{ij} = 1 \quad \forall i</math></p> <p>a sequence of <math>T</math> observations (words), each one drawn from a vocabulary <math>V = v_1, v_2, \dots, v_v</math></p> <p>a sequence of observation likelihoods (<b>emission probabilities</b>), each expressing the probability of an observation <math>o_i</math> being generated from a state <math>i</math></p> <p><math>\pi_i</math> is the probability that the chain will start in state <math>i</math> <math>\sum_{i=1}^N \pi_i = 1</math></p>	<p><b>Hidden Markov Model Tagger</b>: before, we don't consider all possible paths. How: <math>\hat{T} = \arg \max_x P(T W)</math>. Assume: <b>Markov</b> to predict the future tag all that matters is the current state. <b>Independence</b> the probability of an output observation (word) <math>o_i</math> depends only on the state that produced the observation <math>q_i</math></p> <p><b>Emission probability</b>: <math>P(w_i t_i)</math> and <b>Transition probability</b>: <math>P(t_{i+1} t_i)</math></p> <p><b>Viterbi algorithm</b>: <math>v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t) = \max_{i=1}^N v_{t-1}(i) P(q_j q_i) P(o_t q_j)</math> where <math>v_{t-1}(i)</math> is the previous viterbi path probably from the previous time step, <math>a_{ij}</math> is the transition probability from the previous state <math>q_i</math> to the current state <math>q_j</math> and <math>b_j(o_t)</math> is the emission probability.</p> <p>The number of possible paths grows exponentially with the length of the input Viterbi's running time is <math>O(SN^2)</math>, where <math>S</math> is the length of the input and <math>N</math> is the number of states in the mode. Maybe BeamSearch.</p>
<p><b>function</b> VITERBI(observations of len <math>T</math>, state-graph of len <math>N</math>) <b>returns</b> best-path, path-prob</p> <p>create a path probability matrix <b>viterbi</b>[<math>N, T</math>]</p> <p><b>for</b> each state <math>s</math> <b>from</b> 1 <b>to</b> <math>N</math> <b>do</b> ; initialization step</p> <p style="padding-left: 20px;"><b>viterbi</b>[<math>s, 1</math>] <math>\leftarrow \pi_s * b_s(o_1)</math></p> <p style="padding-left: 20px;"><b>backpointer</b>[<math>s, 1</math>] <math>\leftarrow 0</math></p> <p><b>for</b> each time step <math>t</math> <b>from</b> 2 <b>to</b> <math>T</math> <b>do</b> ; recursion step</p> <p style="padding-left: 20px;"><b>for</b> each state <math>s</math> <b>from</b> 1 <b>to</b> <math>N</math> <b>do</b></p> <p style="padding-left: 40px;"><b>viterbi</b>[<math>s, t</math>] <math>\leftarrow \max_{s'=1}^N \text{viterbi}[s', t-1] * a_{s',s} * b_s(o_t)</math></p> <p style="padding-left: 40px;"><b>backpointer</b>[<math>s, t</math>] <math>\leftarrow \arg \max_{s'=1}^N \text{viterbi}[s', t-1] * a_{s',s} * b_s(o_t)</math></p> <p style="padding-left: 20px;"><b>bestpathprob</b> <math>\leftarrow \max_{s=1}^N \text{viterbi}[s, T]</math> ; termination step</p> <p style="padding-left: 20px;"><b>bestpathpointer</b> <math>\leftarrow \arg \max_{s=1}^N \text{viterbi}[s, T]</math> ; termination step</p> <p><b>bestpath</b> <math>\leftarrow</math> the path starting at state <b>bestpathpointer</b>, that follows <b>backpointer</b>[] to states back in time</p> <p><b>return</b> bestpath, bestpathprob</p>	<p><b>Maximum entropy classifier</b>: a discriminative model to directly estimate the posterior. <math>\hat{T} = \arg \max_x \prod P(t_i w_i, t_{i-1})</math> by giving it contextual words surrounding the task.</p> <p><b>RNN for POS tagging</b>: input the word embedding, output the tag, train with cross-entropy loss and run inference by selecting the most likely tag from softmax at each step.</p> <p><b>SOTA</b>: use multilingual BERT.</p>	<p><b>Constituency Parsing</b>: Grammar checking, semantic analysis, parsing, downstream tasks. Challenge that in the lexicon many possible trees exist. Define a <b>Context free grammar</b>: <math>G=(T, N, S, R)</math> with Terminals, Nonterminals, Start Symbol, Rules from non-terminal <math>\rightarrow</math> sequence of terminals/non-terminals.</p> <p><b>The CKY algorithm</b>: <math>O(n^3 G )</math> : <math>n</math> is the length of the parsed string; <math> G </math> is the size of the CNF grammar <math>G</math>. Create a matrix <math>n \times n</math>, use upper triangular portion, start at diagonal and record rules word matches. Step up and use two surrounding cells below to see if it satisfies any other rules. Continue until the corner.</p> <p>Use it for <b>recognition</b> (does it contain a sentence?) <b>parsing</b> (include backpointers to see which rules stem from what other rules). Can also do statistical parsing</p>