Language ambiguity (has multiple precise meanings):
- word ‘bring me the file’(resolve w/ POS)
- syntactic ‘I shot an elephant in my pjs’
- 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
$\mathrm{S} \rightarrow \mathrm{NP\ VP}, \mathrm{NP} \rightarrow \mathrm{Det\ N}, \mathrm{VP} \rightarrow \mathrm{V\ NP}, \mathrm{VP} \rightarrow \mathrm{V}, \mathrm{VP} \rightarrow \mathrm{V\ PP}, \mathrm{PP} \rightarrow \mathrm{P\ NP}$
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 coropus. **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): $\frac{e^{z_i}}{\sum_k e^{z_k}}$
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: $\cos(\theta)=\frac{p_1\cdot p_2}{||p_1||\times||p_2||}$

Analogy Recovery: offset of the vectors reflect their relationship.
$a-b\approx c-d \iff d\approx c-a+b$

Byte-Pair Encoding: Instead of manually specfying 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 trianing 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.

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: $p(w_{t+j}|w_t)=\frac{\exp(u_{w_{t+j}}^\top h_{w_t})}{\sum_{w'=1}^W \exp(u_{w'}^\top h_{w_t})}$, the aim: $\max\prod_t\prod_j p(w_{t+j}|w_t)\rightarrow \min_\theta -\sum_t\sum_j \log p(w_{t+j}|w_t;\theta)\rightarrow \frac{1}{T}\sum_{t=1}^T\sum_{-c\le j\le c, j\neq 0}\log p(w_{t+j}|w_t;\theta)$ 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: $\log p(D=1|w_t,w_{t+1})+k\mathbb{E}_{\tilde{c}\sim P_{noise}}[\log p(D=0|w_t,\tilde{c})]$ 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: $p(D=1|w_t,w_{t+1})=\frac{1}{1+\exp -u_{w_{t+1}}^\top h_{w_t}}$. We can sample k (5-10 words) with frequency or random sampling.

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).

$Accuracy=\frac{TP+TN}{TP+FP+TN+FN}$, $f1=2\times\frac{precision\times recall}{precision+recall}=\frac{TP}{TP+0.5(FP+FN)}$, 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. $\frac{\sum_i^C TP_i}{\sum_i^C TP_i+\frac{1}{2}(\sum_i^C FP_i+\sum_i^C FN_i)}=Accuracy$

$\underbrace{P(y|x)}_{\text{Posterior}}=\frac{\overbrace{P(x|y)}^{\text{Likelihood}}\overbrace{P(y)}^{\text{Prior}}}{\underbrace{P(x)}_{\text{Evidence}}}$

$\hat{y}=\arg\max_y P(y|x)=\arg\max_y P(x|y)P(y)$ since evidence doesn’t change.
Naive Bayes Classifier:
$\hat{y}=\arg\max_y \overbrace{P(x_1,\dots,x_I|y)}^{P(x_1|y)\cdot\ldots\cdot P(x_I|y)}P(y)=\arg\max_y P(y)\prod_{i=1}^I P(x_i|y)$

In a **Bag of Words** count the number of times a token appears in the vocabulary per class. In **One smoothed Naive Bayes:** $P(x_i|y)=\frac{count(x_i,y)+1}{\sum_{x\in V}(count(x,y)+1)}=\frac{count(x_i,y)+1}{(\sum_{x\in V}count(x,y))+|V|}$ **Binary Naive Bayes**: only consider if a feature is present, rather than considering every time it occurs. **Controlling for negation**: pre-pend ‘NOT_’.

Discriminative algorithms directly learn P(Y|X) without considering likelihood. Generative: consider likelihood **Logistic Regression**: apply sigmoid/softmax with $s=w\cdot x+b$ with loss $H(P,Q)=-\sum_i P(y_i)\log Q(y_i)$.

RNN: $h_{t+1}=f(h_t,x_t)=\tanh(Wh_t+Ux_t), W\in\mathbb{R}^{H\times H}, U\in\mathbb{R}^{H\times E}$ The model is less able to learn from earlier inputs, better for long ranged **CNN**: CNNs can perform well if the task involves key phrase recognition

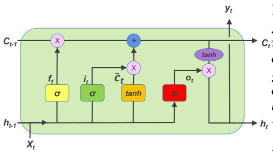
Language Models: Assign probabilities to sequence of words, like predicting the next word in a sentence. Uni-directional: use information from left to generate predictions about words on the right. Bi-directional: use information from both sides to fill the target.
N-gram: need because language is flexible, and a natural extension of a sentence may no appear in corpus. $P(w_n|w_1^{n-1})\approx P(w_n|w_{n-N+1}^{n-1})=\frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$. If certain words absent the probability is 0.

$$S(w_i|w_{i-2}w_{i-1})=\begin{cases}\frac{C(w_{i-2}w_{i-1}w_i)}{C(w_{i-2}w_{i-1})}, & if\ C(w_{i-2}w_{i-1}w_i)>0\\ 0.4\cdot S(w_i|w_{i-1}), & otherwise\end{cases}$$
$$S(w_i|w_{i-1})=\begin{cases}\frac{C(w_{i-1}w_i)}{C(w_{i-1})}, & if\ C(w_{i-1}w_i)>0\\ 0.4\cdot S(w_i), & otherwise\end{cases}$$
$$s(w_i)=\frac{C(w_i)}{N}$$

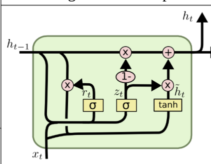
Change this with **add-one smoothing**: $P_{add-1}(w_n|w_{n-1})=\frac{C(w_{n-1},w_n)+1}{C(w_{n-1})+V}$ however, this influences the less frequent words (not more). Therefore, implement backoff smoothing.
Interpolation:
$P_{interp}(w_i|w_{i-2}w_{i-1})=\lambda_1P(w_i|w_{i-2}w_{i-1})+\lambda_2P(w_i|w_{i-1})+\lambda_3P(w_i),\quad where\ \lambda_1+\lambda_2+\lambda_3=1$ 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: $PPL(w)=\sqrt[n]{\frac{1}{\prod_{k=1}^n P(w_k|w_1^{k-1})}}$ 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. $\mathrm{H}(\mathrm{T},\mathrm{q})=-\sum_{i=1}^N\frac{1}{N}\ln q(x_i)$. To convert to perplexity take the base of the logarithm and perform $PPL(M)=base^H$.

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: $h_{t+1}=f(h_t,x_t)=\tanh(Wh_t+Ux_t), y_t=W_{hy}h_t+B_y$. **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 looses its information.



LSTM:
f_t : forget gate $=\sigma(W_{if}x_t+W_{hf}h_{t-1}+b_f)$
i_t : input gate $=\sigma(W_{ii}x_t+W_{hi}h_{t-1}+b_i)$
o_t : output gate $=\sigma(W_{io}x_t+W_{ho}h_{t-1}+b_o)$
g_t : candidate cell $=\tanh(W_{ig}x_t+W_{hg}h_{t-1}+b_g)$
c_t : memory cell state $=f_t\odot c_{t-1}+i_c\odot g_t$
h_t : hidden state $=o_t\odot\tanh(c_t)$
then, h_t : output of cell $=\phi_y(W_yh_y+b_y)$ 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.

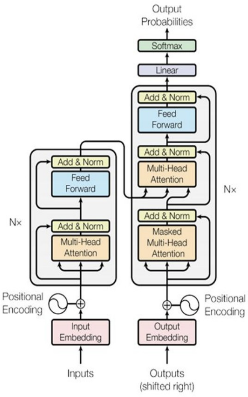


Gated Recurrent Unit:
r_t : switch gate $=\sigma(W_{ir}x_t+W_{hr}h_{t-1}+b_r)$
z_t : reset gate $=\sigma(W_{iz}x_t+W_{hz}h_{t-1}+b_z)$
g_t : candidate state $=\tanh(W_{ig}x_t+r_t*(W_{hg}h_{t-1}+b_g))$
h_t : hidden state $=(1-z_t)*g_t+z_t*h_{t-1}$
no longer has input/output gate but maintains forgetting mechanism. GRU more efficient computing & less over-fitting but LSTM good default choice.

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. $\arg\max_t p(t|s)=p(t)p(s|t)$. 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, $c_t=\sum_{i=1}^I\alpha_ih_i$ 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 α is the **energy**, and it is calculated with $e_i=a(s_{t-1},h_i)=v^T\tanh(Ws_{t-1}+Uh_i)$ where $e_i\in\mathbb{R}^1$ 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, $h_i\in\mathbb{R}^{2D\times 1}$ encoder hidden state for the ith word, $v^T\in\mathbb{R}^{D\times 1}\wedge v\in\mathbb{R}^{1\times D}, W\in\mathbb{R}^{D\times D}$

BLEU: reports a modified precision metric for each level of n-gram $\text{Modified Precision score } p_n=\frac{\text{Total Unique Overlap}_n}{\text{Total n-grams}}$ 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**: $\mathtt{BP}(\prod_n^4 p_n)^{\frac{1}{4}},\quad \mathtt{BP}=\min(1,\frac{\text{MT Output Length}}{\text{Reference Length}})$ where p_n defined above. Used to mostly encourage the hypothesis to be of a simlar 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_β 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). $CHRF_1=\frac{2\cdot CHRP\cdot CHRR}{CHRP+CHRR}$ 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 a: 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.
$$Attention(Q,K,V;a)=a\left(\frac{QK^\top}{\sqrt{d_q}}\right)V$$
$$Q=(\boldsymbol{q}_1,\dots,\boldsymbol{q}_N)^\top\in\mathbb{R}^{N\times d_q}$$
$$K=(\boldsymbol{k}_1,\dots,\boldsymbol{k}_M)^\top\in\mathbb{R}^{M\times d_q}$$
$$V=(\boldsymbol{v}_1,\dots,\boldsymbol{v}_M)^\top\in\mathbb{R}^{M\times d_v}$$
a(.) activation function applied row-wise ($N=M:=D$ and $d_q=d_v:=d_h$)

The encoder receives an input of S words with encoded dimensions D. **Complexity:** *time* $O(MNd_q+MNd_v)$, *space* $O(MN+Nd_v)$, # params (only key and value) so $K,V:O(Md_q+Md_v)$ and in self attention this is $O(Nd_v)$ i.e. Q=V=K. Hard attention is when $a=onehot(\arg\max x_i)$ and Soft attention is when $a=softmax(x)$. 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 $Multihead(Q,K,V,a)=concat(head_1,\dots,head_n)W^O$, where $head_i=Attention(QW_i^Q,KW_i^K,VW_i^V,a)$ 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 $PE_{pos,2i}=\sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$ and $PE_{pos,2i+1}=\cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$ 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: $\mathrm{lr}=\sqrt{\frac{1}{d}}\times\min\left(\sqrt{\frac{1}{d}},i\times warmup^{-1.5}\right)$

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 run softmax $\frac{e^{y_i/T}}{\sum_j^N e^{y_j/T}}$. 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) $TF_{w,d}=\frac{count(w,d)}{\sum_{w'}count(w',d)}$ (frequency of w occurring together with d). $IDF_{w,D}=\log\frac{|D|}{|\{d\in D:w\in d\}|}$ down-weighs words that appear everywhere.
$TF-IDF_{w,d,D}=TF_{w,d}IDF_{w,D}$