
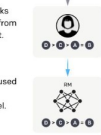
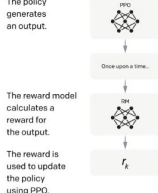


<p>Language ambiguity (has multiple precise meanings):</p> <ul style="list-style-type: none">- word 'bring me the file' (resolve w/ POS)- syntactic 'I shot an elephant in my pj's'- semantic 'the rabbit is ready for lunch'- referential 'Pavarotti is a big opera star'- non-literal 'it's raining cats and dogs'	<p>Deep Learning learns/abstract functions instead of rules based on maintenance of intuitive linguistic rules.</p>	<p>Lexicon – morph(eme)ological analysis (stem and affix e.g. 'cat'+ 's')</p> <p>Word segmentation (tokenization)</p> <p>Word normalization (case/acronyms/spelling)</p> <p>Lemmatization 'sing, sung, sang' → 'sing'</p> <p>Stemming (common root, above 's')</p> <p>Part-Of-Speech (tag words with noun, verb...)</p>	<p>Context-Free Grammar: <u>Derive</u> sentence structure through a <u>parse tree</u></p> <p>S → NP VP, NP → Det N, VP → V NP, VP → V, VP → V PP, PP → P NP</p> <p><u>Discourse</u>: meaning of a text (relationship between sentences) <u>Pragmatics</u>: intentions/commands</p> <p><u>Corpus</u>: a collection of documents <u>Document</u>: one item of corpus (sequence) <u>Token</u>: atomic word unit <u>Vocabulary</u>: unique tokens across corpus. One-Hot Encoding: sparse (wasted space), orthogonal vectors (every word is equidistant), cannot represent out of vocab well</p>
<p>Sigmoid (binary class): $\frac{1}{1+e^{-x}}$, ReLU: $\max(0, x)$, Tanh: $\frac{e^x - e^{-x}}{e^x + e^{-x}}$</p> <p>Softmax (k-class): $\frac{e^{-\epsilon_k}}{\sum_k e^{-\epsilon_k}}$</p> <p>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): $-\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^C y_c^{(i)} \log(\hat{y}_c^{(i)})$</p> <p>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. <u>For unknown words</u> follow above and apply replacements in order discovered.</p>	<p>Euclidean Distance: $\sqrt{\sum_{i=1}^N (q_d - d_i)^2}$</p> <p>Cosine Similarity: $\cos(\theta) = \frac{p_1 \cdot p_2}{\ p_1\ \ p_2\ }$</p> <p>Analogy Recovery: offset of the vectors reflect their relationship. $a - b \approx c - d \iff d \approx c - a + b$</p>	<p>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: $\log p(D = 1 w_t, w_{t+1}) + k E_{\tilde{c} \sim P_{\text{context}}} [\log p(D = 0 w_t, \tilde{c})]$ 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.</p>	
<p>Classification: $\hat{y} = \arg \max_y P(y x)$ predict which y is most likely given input x.</p> <p>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).</p> <p>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. $\frac{\sum_c TP_c}{\sum_c TP_c + \sum_c FP_c + \sum_c FN_c} = \text{Accuracy}$</p>	<p>Naive Bayes Classifier: $P(y x) = \frac{P(x y)P(y)}{P(x)}$</p> <p>$\hat{y} = \arg \max_y P(y x) = \arg \max_y P(y) \prod_{i=1}^I P(x_i y)$</p>	<p>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'.</p>	
<p>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.</p> <p>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.</p> <p>$S(w_{i-1} w_{i-2} w_{i-1}) = \begin{cases} \frac{C(w_{i-2} w_{i-1} w_i)}{C(w_{i-2} w_{i-1})}, & \text{if } C(w_{i-2} w_{i-1} w_i) > 0 \\ 0.4 \cdot S(w_{i-1} w_i), & \text{otherwise} \end{cases}$</p> <p>$S(w_{i-1} w_i) = \begin{cases} \frac{C(w_{i-1} w_i)}{C(w_{i-1})}, & \text{if } C(w_{i-1} w_i) > 0 \\ 0.4 \cdot S(w_i), & \text{otherwise} \end{cases}$</p> <p>$s(w_i) = \frac{C(w_i)}{N}$</p> <p>$+ \lambda_2 P(w_i w_{i-1}) + \lambda_3 P(w_i)$, where $\lambda_1 + \lambda_2 + \lambda_3 = 1$ to combine evidence from multiple n-grams. To make a prediction $P(w_1, \dots, w_n) = \prod_{k=1}^n P(w_k w_{k-1}^{k-1})$ we switch into log space $\log P[\cdot] = \sum_{k=1}^n \log(P(w_k w_{k-1}^{k-1}))$ however the longer the sentence the lower its likelihood. Switch to Perplexity: where n is the number of words: $PPL(w) = \sqrt[n]{\prod_{i=1}^n P(w_i w_{i-1}^{i-1})}$ 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.</p> <p>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).</p> <p>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 $PPL(M) = \text{base}^H$.</p>	<p>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.</p> <p>Interpolation: $P_{\text{interp}}(w_i w_{i-2} w_{i-1}) = \lambda_1 P(w_i w_{i-2} w_{i-1})$</p>	<p>RNN: $h_{t+1} = f(h_t, x_t) = \tanh(W h_t + U x_t)$, $W \in \mathbb{R}^{H \times H}$, $U \in \mathbb{R}^{H \times E}$ 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</p>	
	<p>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.</p> <p>RNN: $h_{t+1} = f(h_t, x_t) = \tanh(W h_t + U x_t)$, $y_t = W_h h_t + B_y$, 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. Bi-directional 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.</p>	<p>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.</p>	
	 <p>LSTM:</p> <ul style="list-style-type: none">f_t: forget gate = $\sigma(W_{f_t} x_t + W_{f_t} h_{t-1} + b_{f_t})$$i_t$: input gate = $\sigma(W_{i_t} x_t + W_{i_t} h_{t-1} + b_{i_t})$$o_t$: output gate = $\sigma(W_{o_t} x_t + W_{o_t} h_{t-1} + b_{o_t})$$g_t$: candidate cell = $\tanh(W_{g_t} x_t + W_{g_t} h_{t-1} + b_{g_t})$$c_t$: memory cell state = $f_t \odot c_{t-1} + i_t \odot g_t$$h_t$: hidden state = $o_t \odot \tanh(c_t)$ <p>then, h_t: output of cell = $\phi_y(W_{h_t} h_t + b_{h_t})$ simply.</p> <p>Struggles learn long-term deps. Less vanishing gradient because additive formulas means we don't have repeated multiplication. The forget gate controls when to preserve gradients or not. $W_{ii} = (H \times E)$ and $W_{hi} = (H \times H)$ since we go from embeddings to hidden representation.</p>		
	 <p>Gated Recurrent Unit:</p> <ul style="list-style-type: none">r_t: switch gate = $\sigma(W_{r_t} x_t + W_{r_t} h_{t-1} + b_{r_t})$$z_t$: reset gate = $\sigma(W_{z_t} x_t + W_{z_t} h_{t-1} + b_{z_t})$$g_t$: candidate state = $\tanh(W_{g_t} x_t + r_t * (W_{g_t} h_{t-1} + b_{g_t}))$$h_t$: hidden state = $(1 - z_t) * g_t + z_t * h_{t-1}$ <p>no longer has input/output gate but maintains forgetting mechanism. GRU more efficient computing & less over-fitting but LSTM good default choice.</p>		
	 <p>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 $K = (k_1, \dots, k_M)^T \in \mathbb{R}^{M \times d_k}$, $O(MN \times d_k)$, $O(MN + Nd_v)$, # params (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</p>		
	<p>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: $lr = \sqrt{\frac{1}{d}} \times \min\left(\sqrt{\frac{1}{d}}, i \times \text{warmup}^{-1.5}\right)$</p>		
<p>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 $\frac{e^{w_i/T}}{\sum_j e^{w_j/T}}$.</p> <p>Multinomial sample over softmax probabilities.</p> <p>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.</p>	<p>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 <u>sentence classification</u>, 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</p>		
<p>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). $IDF_{w,d} = \log\left(\frac{2D}{ \{d \in D: w \in d\} }\right)$ down-weights words that appear everywhere. $TF - IDF_{w,d,D} = TF_{w,d} IDF_{w,d}$</p>			

<ul style="list-style-type: none">• BERT: Trained with masked language modelling and next sentence prediction.• RoBERTa: Got rid of next sentence prediction, optimized hyperparameters.• DeBERTa: Focused on improvements to positional encodings• SpanBERT: mask contiguous tokens instead of random. Makes the task harder.• DistilBERT ALBERT: training a small model to behave similarly to the bigger version• BigBird LongFormer: Self attention has $O(N^2)$ complexity. Models with sparse attention mechanisms have been proposed to extend the input length• ClinicalBERT MedBERT PubMedBERT BEHRT: BERT-like models trained on specific domains.• ERNIE: add special entity embeddings into transformer for added benefit.• Multi-lingual Models<ul style="list-style-type: none">◦ Multi-modal Models: combine text + visual◦ ImageBERT: encodes objects as additional tokens.• VilBERT: two parallel BERT encoders, interact using co-attention module, Image features are extracted using pre-trained Faster R-CNN• Masked Image modelling: mask parts of image.• Masked Protein modelling: AlphaFold.	<p>Pre-training encoder-decoder models: Without parallel data: <u>prefix language modelling</u> give it half a sentence and predict the second half. <u>Shuffle words</u> and the task is to recover the original sentence <u>masking</u> just predict the full sentence. Instruction Tuning: took annotated datasets and phrased it as a Seq2Seq task. Place the query into a template into a conversational-sounding instruction.</p> <p>Pre-training decoder models: train to optimize $p_{\theta}(w_t w_{1:t-1})$ Great for tasks where the output has the same vocabulary as the pre-training data.</p> <p>Fine tuning: put a new layer on top and fine-tune the model for a desired task. Zero-shot learning: Give the model a natural language description of the task, have it generate the answer as a continuation. One-shot learning: In addition to the description of the task, give one example of solving the task. No gradient updates are performed. Few-shot learning: In addition to the task description, give a few examples of the task as input. No gradient updates are performed.</p>		
<div><p>Step 1 Collect demonstration data, and train a supervised policy.</p><p>A prompt is sampled from our prompt dataset.</p><p>Explain the moon landing to a 5 year old</p><p>A labeler demonstrates the desired output behavior.</p><p>Some people went to the moon.</p><p>This data is used to fine-tune GPT-3 with supervised learning.</p></div>	<div><p>Step 2 Collect comparison data, and train a reward model.</p><p>A prompt and several model outputs are sampled.</p><p>Explain the moon landing to a 5 year old</p><p>Some people went to the moon. The moon is a big ball in the sky. The moon is a big ball in the sky. The moon is a big ball in the sky.</p><p>A labeler ranks the outputs from best to worst.</p><p>This data is used to train our reward model.</p></div>	<div><p>Step 3 Optimize a policy against the reward model using reinforcement learning.</p><p>A new prompt is sampled from the dataset.</p><p>The policy generates an output.</p><p>A thing about frogs</p><p>The reward model calculates a reward for the output.</p><p>The reward is used to update the policy using PPO.</p></div>	<p>Advanced prompting and learning from human feedback: Chain of thought to show reasoning. Zero-shot chain-of-thought: ask it to 'think step by step'.</p> <p>Retrieval Based language models: 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>PROBLEMS: 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. <u>Therefore introduce human feedback by giving possible answer. However, human-in-the-loop is expensive. Therefore, write a model to predict the scores of humans or to avoid noisy scores, write them to rank the answers instead.</u></p>