Lecture 1: Word Vectors

- Language is context and meaning => choice of words
- Representing a word's meaning
  - signifier/symbol <=> signified (idea/thing)
  - Previous common solution for encoding meaning is WordNet thesaurus containing synonyms and hypernyms (is a relationship)
- Traditional NLP regards words as discrete symbols
  - Vector dimension = number of words in your dictionary
  - No notion of similarity between words (encoded as one-hot vectors)
- Distributional semantics - word's meaning is given by words that frequently appear close-by to it ("You shall know a word by the company it keeps")
- Word's context is important to define
  - Fixed window around word can be its context
- Word vectors/Word Embeddings/Word representation
  - Dense vector for each word so that similar vectors of words appear in similar contexts (similarity measured by dot product)
- Word2vec - framework for learning word vectors
  - Input: large corpus of words/text, every word is in a fixed vocabulary
  - Go through each position in text (center word c and context words)
  - Calculate similarity of word vector c and all context words by calculating P (context word | center word)
  - Adjust word vectors to maximize likelihood
- Minimizing Objective function

  Likelihood = $L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{t+j} \mid w_t; \theta)$

  θ is all variables to be optimized

  sometimes called a *cost* or *loss* function

  The objective function $J(\theta)$ is the (average) negative log likelihood:

  $$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P(w_{t+j} \mid w_t; \theta)$$

  -
- Calculating P(o | c)

  $$P(o|c) = \frac{\exp(\boxed{u_o^T v_c})}{\boxed{\sum_{w \in V} \exp(u_w^T v_c)}}$$

  ① Dot product compares similarity of *o* and *c*.
  $u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
  Larger dot product = larger probability

  ③ Normalize over entire vocabulary to give probability distribution

  -
  - Two vectors per word w, v_w when w is a center word, u_w when w is a context word

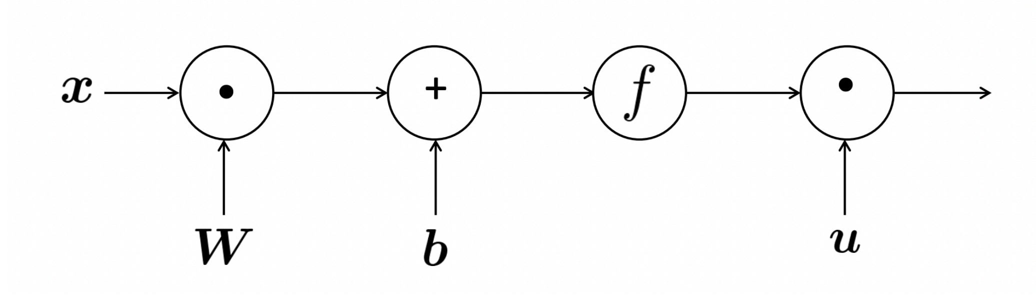
        - Average both of these at the end (technically you average matrices consisting of these row vectors stacked together)
    - Denominator is expensive because it scales with the size of the vocabulary
- dim(Theta) for optimization = 2dV, where d = dimension of vectors, V = number of words in corpus, 2 vectors per word

Lecture 2: Word Vectors/Senses, Neural Network Classifiers

- word2vec maximizes objective function by putting similar words nearby in space
- Word2Vec model variants
    - Skip-grams (SG)
        - Predict context words (position independent) given center word
    - Continuous Bag of Words
        - Predict center word from (bag of) context words p (c | o)
- skip-gram model w/ negative sampling
    - Normalization term of P (o | c) is expensive to calculate, so instead approximate it with negative sampling
    - Train binary logistic regressions to differentiate a true pair (center word w/ word in its context) vs random/noise pair
    - $$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)} \left[\log \sigma\left(-u_j^T v_c\right)\right]$$
        - Maximize probability of two words co-occurring (1st part) , minimize probability of k-sampled noise words (2nd part)
    - $$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta)$$
    - ^Objective function they maximize
- Co-occurrence matrices (symmetric, irrelevant whether left or right context)
    - Co-occurrence vectors are high dimensional, but sparse => subsequent classification models are less robust because of sparseness
    - Solution: create low-dimensional vectors using SVD
        - However, running an SVD on raw counts doesn't work well, because words have a long-tailed distribution (few common words, a lot of rare words)
        - Need to scale counts in cells with function words that are too frequent (the, he, has), smaller windows, use pearson correlations instead of counts + send negative correlations to 0
- GloVe = word vectors produced with fast training + scalable to huge corpora; alternative to word2vec
- Evaluating word vectors

    - Intrinsic evaluation
        - Fast to compute, not clear if helpful until correlation to real task is established
    - Extrinsic evaluation
        - Evaluation on real task, can take a long time to compute accuracy
- i.e. word analogies task or meaning similarity (intrinsic word vector evaluation) for evaluating word vectors
- Named entity recognition (identifying references to a person, organization, or location) - common extrinsic word vector evaluation
- Word sense ambiguity
    - Most words have lots of meanings <- can happen by linguistic sense extension (how new words are created)
    - Need "word sensors" to understand contextual meaning
    - Different senses of a word can be represented as a weighted sum in standard word embeddings
        - i.e. $v_{word} = a1 * v_{word1} + a2 * v_{word2} + ...$
- Surprising result of weighted sum approach: You can actually separate out the senses because you are working in high-dimensional sparse coding spaces
- Deep Learning Classification: Named Entity Recognition (NER)
    - Task: Find and classify names in text by label word tokens (i.e. person/location/date/etc.)
- Idea: one-hot encoded word vectors as input => multiply by embedding => makes it linearly separable in high dimension

Lecture 3: Neural Networks Math

- Common non-linearities: sigmoid, tanh (i.e. tanh(x) = 2logistic(2x) - 1; range: [-1, 1]), hard tanh, ReLU, Leaky RelU
    - ReLU - max(x, 0) <- good gradient backflow
    - Leaky/parametric Relu, make values less than 0 negative rather than zero.
- Use Cross Entropy Loss in PyTorch
    - H(p, q) = - sum over all classes p(c) * log q(c) where q(c) is our model's probability
- Backpropagation algorithm (below)
- Given NN with x (input) => hidden layer = f(Wx + b) => output = s = (u^T * h) modeled as graph
    - 

- In one activation layer, deriv(output) w.r.t to deriv(Weight matrix) is (delta)^T * (x)^T, where z = Weight * x + b; deriv (output) w.r.t to deriv (Bias term) is (h^T) * f'(z)
- Local gradient (associated with each Node in NN graph)
  - Gradient of OUTPUT w.r.t to INPUT (n * m) where n is # of inputs, m is outputs
  - Multiple inputs to a node leads to multiple local gradients
- Downstream gradient from node = local gradient at node * upstream gradient from node
- Gradients sum at outward branches during backpropagation!!
- Node intuition during Backprop
  - Addition in activation function distributes upstream gradient
  - Max activation "routes/only keeps" input gradient going upstream
  - Multiplication in activation "switches" upstream gradient
- Bprop summary

  2. Bprop:

     - initialize output gradient = 1
     - visit nodes in reverse order:

     Compute gradient wrt each node using gradient wrt successors

     $\{y_1,\, y_2,\, \ldots\, y_n\}$ = successors of x

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

  Done correctly, big O() complexity of fprop and bprop is **the same**

- Manual gradient checking can be done by Numeric Gradients
  - Good for debugging, but very slow

Lecture 4: Dependency Parsing

- Two views of linguistic structure of sentences
  - Context-free grammars (starting unit: words, words combine into phrases, phrases => bigger words)
    - For example, Det (the, this, etc), Adj, Noun can be a rule
  - Dependency structure: shows which words depend on which other words
- Model needs to understand sentence structure to interpret language correctly
- Prepositional phrase attachment ambiguity
  - San Jose cops kill man with knife (with knife can modify either man or kill)
- Coordination scope ambiguity
  - Doctor: No heart, cognitive issues

- Verb Phrase attachment ambiguity
- Dependency syntax
    - Syntactic structure (sentences) is made up of relations between lexical items/words that are connected by arrows/dependencies (arrows are typed by grammatical relation)
    - Dependencies form a tree (connected, acrylic, single-rooted graph)
- Drawing dependencies (arrow from head to dependent/child)
    - Initially add a fake ROOT so every word is a dependent of exactly 1 other node
- Treebank - caused by a rise of annotated data
    - way to evaluate NLP systems
    - broad converge, not just a few intuitions
    - frequencies + distributional information
- Sources of information for dependency parsing
    - Bilexical affinities (is dependency is plausible)
    - dependency distance (most are close)
    - intervening material (dependencies rarely span intervening verbs/punctuation)
    - Valency of heads (how many dependents on which side are usual for a head)
- Dependency parsing
    - Sentence is parsed by choosing for each word what other word (or ROOT) it's a dependent of
    - Constraints: only one word is a dependent of ROOT, no cycles of A -> B, B -> A
    - Last consideration is whether arrows can cross (non-projective or not)
- Projectivity
    - no crossing dependency arcs when the words are laid out in linear order w/ arcs over words
- Methodologies for dependency parsing
    - DP
    - Graph algos - MST
    - Constraint Satisfaction - eliminate edges that don't satisfy heuristics
    - Deterministic dependency parsing
- Greedy transition-based parsing
    - Parser has stack (starts with ROOT, top to the right), buffer (starts with input sentence, top to left), dependency arcs (initially empty), set of actions
    - Set of allowed actions
        - shift, left-arc, right-arc
    - End state: only root left on stack and the buffer is empty
- MaltParser: run greedy transition-based parsing with ML
    - No search, each action is predicted by a discriminative classifier over each legal move
- Evaluation metrics for dependency parsing
    - Unlabeled accuracy score, labeled accuracy score
- Neural dependency parser
    - Parts of speech + dependency labels are d-dimensional word vectors

    - Vector representation = word vector + POS vector + dependency vector concatenated

Lecture 5: Recurrent Neural Networks, Language Modeling

- L2 regularization isn't enough for large NN's, enough for SVMs/more linear models
    - Dropout - training randomly set input to 0 with probability p (except input layer), during testing: multiply all weights by 1-p
    - Dropout prevents feature co-adaptation: a feature cannot only be useful in the presence of other features
- Parameter Initialization
    - Normally initialize weights to ~ Uniform(-r,r) (i.e. not zero matrices)
    - Xavier initialization: Var(W_i) = 2 / (n_in + n_out)
- Optimizers
    - Plain SGD works fine, good results often requires hand-tuning LR
    - Complex nets/sophisticated optimizers: Adagrad (simplest, stalls early), RMSprop, Adam, NAdamW <- good for word vectors
- Language Modeling
    - Task of predicting what word comes next
    - Can be used to generate text
- n-gram Language Models
    - Markov assumption: current word only depends on the preceding n-1 words
    - Count P(n-gram/n-1 gram)
    - Sparsity problems
        - Solutions: smoothing (add small delta to the numerator count), backoff (condition on n-2 gram rather than n-1 gram)
- Neural Language Model
    - Fixed-window neural LM: concatenate words in window, take concatenated word embeddings, pass through hidden layer, send through softmax to get next words probability
    - Improvements over n-gram: No sparsity, don't need to store all observed n-grams
    - Problems: fixed window too small (W can never be large enough), no symmetry in how inputs are processed (different weights multiplied to different inputs)
- RNN
    - Sequence Modeling (apply the same weights W repeatedly)
    - Benefits: can process any length input, computation can use information from many steps back, model size doesn't increase for longer input, symmetry in how inputs are processed
    - Drawbacks: slow, in practice its difficult to access information from many words/steps back
    - Predicted output becomes next time steps input
- Teacher forcing to train RNN LM (loss at each step is summed)
- BPTT - sum gradients as you go backwards
    - Usually only for a finite number of steps (i.e. 20)
- LM evaluation metric = perplexity

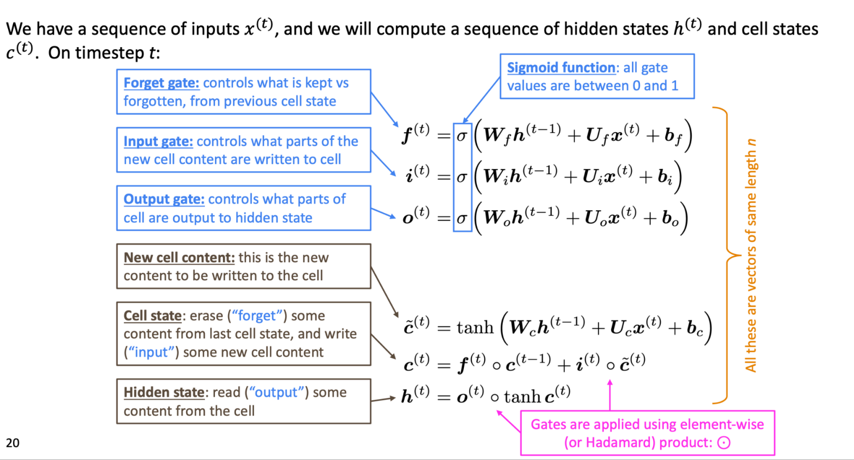
$$\text{perplexity} = \prod_{t=1}^{T} \underbrace{\left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)}_{\text{Inverse probability of corpus, according to Language Model}}^{1/T}$$

Normalized by number of words

-

Lecture 6: Seq2Seq, Machine Translation, Subword Models

- Problems with RNNs: Vanishing + Exploding Gradients
- Vanishing Gradients
  - Happens if chain rule gradients are small
  - Can't model long-distance language relationships in LM
- Exploding Gradients
  - If gradient is too big, SGD update is too big (can even result in Inf or NaN gradients)
  - Gradient clipping - used for exploding gradient problem
    - If ||g|| is > threshold, scale before applying SGD update
- Fixing vanishing gradient problem
  - vanilla RNN - hidden state is constantly being re-written
  - Can we introduce another weight to store memory for long-term use?
- LSTMs (Long Short-Term Memory RNNs)
  - Each step has hidden state - h(t), and cell state - c(t)
  - LSTM can read/erase/write information from cell
    - forget gate (what's forgotten from previous state), input gate (what's written from new cell content), output gate (what parts are output to hidden state)
    - selection of information erased/written/read based on gates
    - gates are vectors of length n, each element in gate can be open(1), closed(0)
    - gates are dynamic based on current context

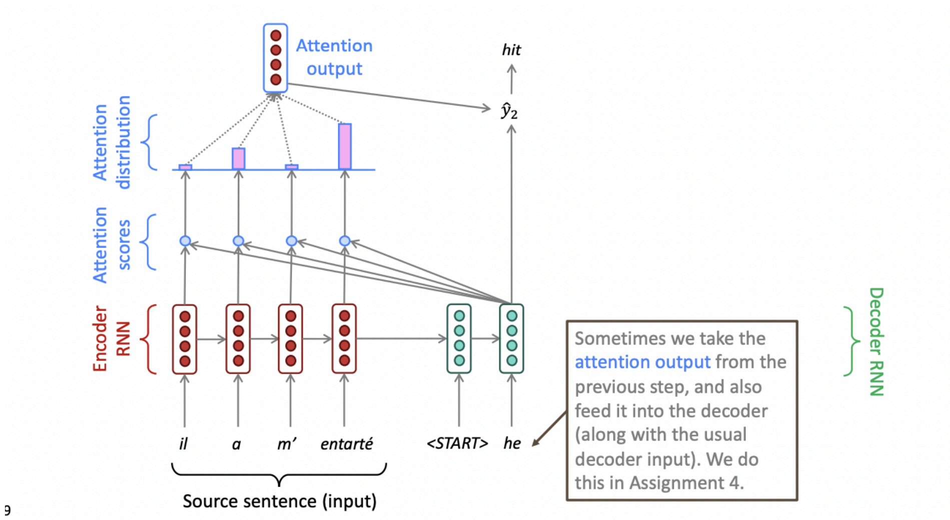


-

- Residual Learning (ResNet)/skip-connections are used to solve vanishing gradient problem for deep/convolutional networks
    - Connect/pass identity function from one layer to the next
- Other methods:
    - Dense connections: directly connect each layer to all future layers
    - Highway connections: inspired by LSTM cell state weights/gates + applied to feed-forward/convolutional networks
- Chaining LSTM/RNN to a classification model by taking element-wise max/mean of all hidden states
    - e.g. sentiment classification
- **Hidden state of a representation of a word in the sentence is a contextual representation of the sentence**
    - Bidirectional RNN - create forward and backward RNN/LSTM (context on both left and right side)
- Bidirectional is only applicable if you have access to the entire input sequence <- protein sequence prediction/masking?
    - Not applicable to LM (only have left context), useful for creating sentence representations
- Multi-layer RNN: stack RNNs (RNNS are already deep in one dimension because they unroll over many timesteps)
    - lower RNN compute lower-level features, higher compute higher-level
- Machine Translation
    - Powers google translate
    - Statistical Machine Translation (Bayes theorem conditioned on translating from one language to another; required tons of feature engineering) => Neural Machine Translation (sequence-to-sequence model)
- Neural Machine Translation
    - Encoder-Decoder RNN
        - Encoder builds up sentence meaning/representation, conditional generation using decoder RNN with sentence representation as start and output of time step t_(n-1) is input for t_n
- NLP tasks that can be phrased as seq2seq
    - Summarization (long-> short text), Dialogue, Parsing, Code generation
- NMT is less interpretable and difficult to control generation for

Lecture 7: NMT, Attention

- NMT decoding (conditional generation)
    - Greedy decoding has no way to undo decisions since it just takes argmax at each step
    - Exhaustive search decoding (try compute all possible sequences y) - too expensive
    - Intermediate solution: Beam search decoding - keep the top k sequences at each step (prune after each step to prevent it from exploding)
        - Usually beam search until we reach timestep or completed n hypothesis

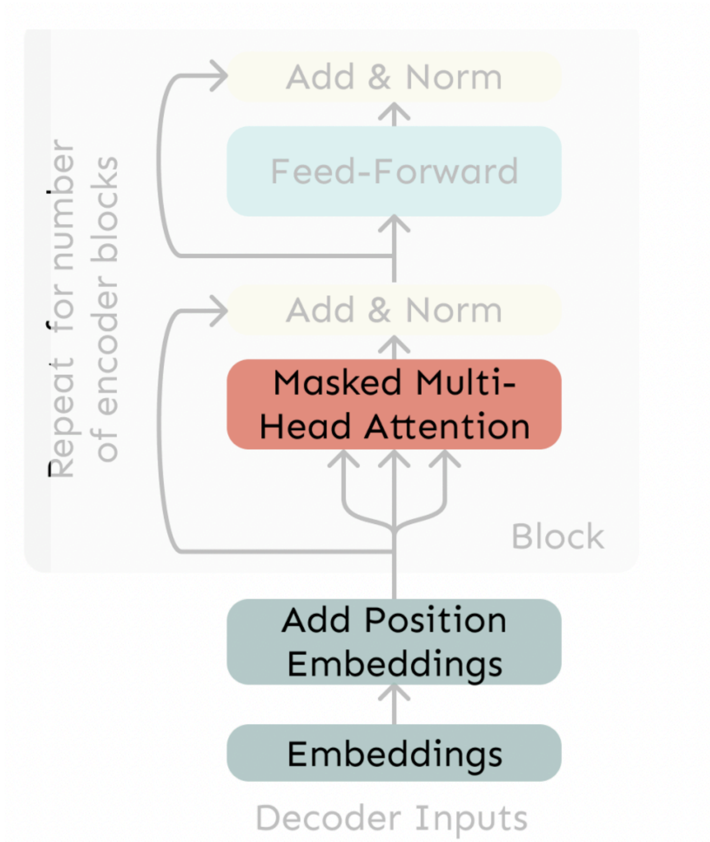
        - Multiple Hypotheses don't always have END token at the same time
        - Another problem: longer beam hypotheses have lower scores, so normalize by length
- Evaluating Machine Translation
    - BLEU - Bilingual Evaluation Understudy: compares machine-written translation to one/many human-written translations and computes similarity score based on geometric mean of n-gram precision
        - Plus penalty for too-short system translations
- Attention for NMT
    - Encoding source sentence needs to capture ALL information about the source sentence => Information bottleneck
    - Core idea: At each step of decoder, use a direct connection to encoder to focus on a particular part of the source sequence
    - Steps
        - Dot product (one method) decoder representation with each input vector in encoder to get attention scores
            - Other methods: multiplicative attention (e = s^T * W * h), reduced-rank attention (e = s^T (U^T * V) * h)
        - Use softmax to turn scores -> distribution, take weighted average of encoded hidden states to create attention output
        - Concatenate attention output vector with decoder hidden state to compute first output
        - 

    - Provides some interpretability by inspecting attention distribution
- More general definition of attention
    - Given a set of vector values and a vector query, attention can compute a weighted sum of values dependent on the query

Lecture 8: Transformers

- Issues with Recurrent Models

- Linear interaction distance: nearby words affect each other's meaning => O(sequence length steps) for distant word pairs to interact (hard to learn distant meanings)
    - Lack of parallelizability: Forward/backward passes have O(sequence length) non parallelizable operations (future RNN states can't be computed before past ones)
- Attention as a soft, averaging lookup table
    - each key returns a value between 0 and 1 that is weighted and summed
- Barriers for self-attention as a "building block"
    - No notion of sequence order, solution: encode each sequence index as a position vector and add it to the attention-learned vector to create final vector embedding
        - Learned absolute position representations between indices 1 and N > sinusoidal representation vectors
    - Currently just weighted averages, no non-linearities. Solution: Add a feed-forward network to post-process each output attention vector
    - Need to ensure we don't "look at the future" when predicting a sequence. Solution: To enable parallelization, mask out attention to future words by setting attention scores to -infinity
- Transformer Decoder Overview



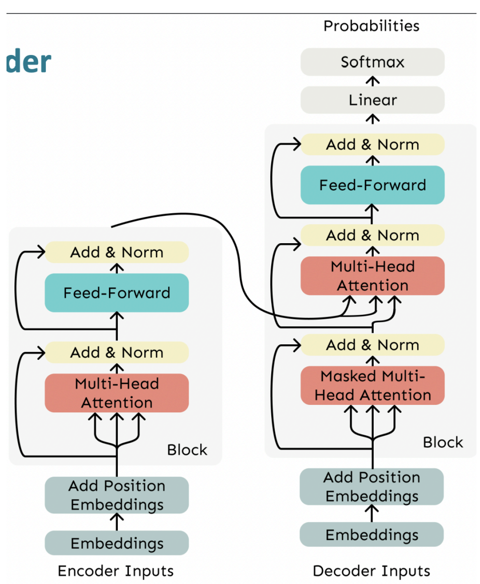
    -
- Multi-headed attention/attention heads

- Let, $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and ℓ ranges from 1 to h.
- Each attention head performs attention independently:
  - $\text{output}_\ell = \text{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell$, where $\text{output}_\ell \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
  - $\text{output} = [\text{output}_1; \dots; \text{output}_h]Y$, where $Y \in \mathbb{R}^{d \times d}$

    - Query Matrix, Key Matrix, Value Matrix
- Scaled Dot Product attention
    - When dimensionality d becomes large, dot products tend to be large => inputs to softmax is large => gradients small

$$\text{output}_\ell = \text{softmax}\left(\frac{XQ_\ell K_\ell^\top X^\top}{\sqrt{d/h}}\right) * XV_\ell$$

- Optimization Tricks
    - Residual Connections
        - X^i = X^(i-1) + Layer(X^(i-1)) to bias towards identity function for gradient
    - Layer Normalization - make model train faster
        - Cut down on uninformative variation in hidden vector values by normalizing to unit mean/std within each layer
        - Thought to normalize gradients
- Transformer Encoder
    - Same thing as Decoder, but remove masking (because we want it to be bi-directional)
- Transformer Encoder-Decoder



- Cross-attention
    - Output vectors from Transformer encoder and input vectors from Transformer decoder
    - For attention, keys/values drawn from encoder (like memory), but queries are from decoder
- Transformer Bottlenecks
    - Computing all pairs of interactions grows quadratically O(n^2 * d) where n is sequence length, d is dimensionality

Lecture 9: Pretraining, Word Structure

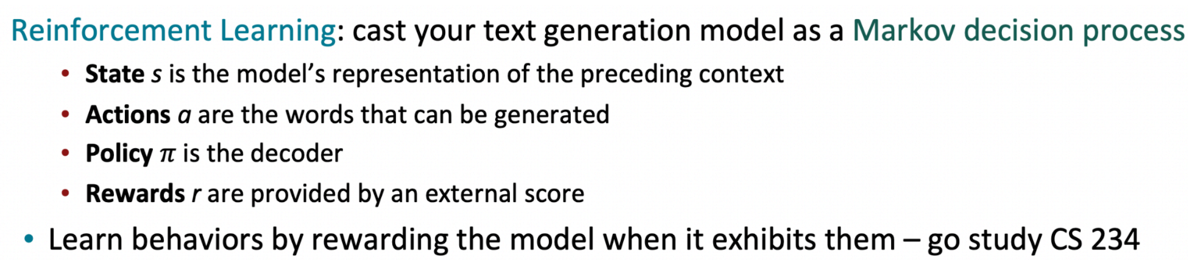
- Word structure
    - Up until now, we have assumed there exists only a finite vocabulary (which is built from the training set) => Novel words at test time would be mapped to an "UNK" token
    - Most languages have morphology (multiple words derived from some word structure => more word types that occur only a few times)
- Subword Models
    - Dominant paradigm is to learn a vocabulary of parts of words/subword tokens
    - Byte-word encoding for defining subword vocabulary
        - 1. Start with a vocabulary containing only characters and an “end-of-word” symbol.
          2. Using a corpus of text, find the most common adjacent characters “a,b”; add “ab” as a subword.
          3. Replace instances of the character pair with the new subword; repeat until desired vocab size.
- Pre-training + Fine Tuning through language modeling
    - Train NN to performer language modeling on a large amount of text (learn general things)
    - save model parameters => finetune on your task (not many labels, adapt to the task)
- Pre-training whole models
    - Helps us build strong representations of language, gives us good parameter initializations, etc.
- Pre-training encoders
    - Pre-training objective can't be language modeling (because they get bidirectional context)
        - Masked LM: Instead it uses masking (Given x-tilda, learn p(x | x-tilda))
- BERT: Bidirectional Encoder Representations from Transformers
    - Masked LM (either replace input with [MASK] 10% of time, replace it with the wrong word 10% of time, keep input 80% of time)
    - Segment embedding - have two chunks (A + B) for each input sentence, can the model predict whether a random chunk B actually came after a given chunk A
- BERT Evaluation Datasets
    - Quora QUestion Pairs, Natural Language Inference (what can you infer from language), Sentiment Analysis, etc.

- Limits of Pretrained encoders: Aren't nice for autoregressive (1 word-at-a time) generation
- Parameter-Efficient Finetuning
  - Prefix-Tuning - adds prefix of parameters
  - Low-Rank Adaptation: learns row-rank "difference" between pretrained and fine tuned weight matrices (most parameters are frozen)
- Pre-training Encoder-Decoder networks
  - Objective: Span corruption: corrupt words/tokens and predict the sentence
  - T5 - can be fine tuned to answer a wide range of questions, retrieving knowledge from parameters
- Pre-training Decoders
  - Fine-tune by adding a hidden layer on the outputs of the transformer/LSTM decoder to classify/predict outputs
- GPT-3: In-context learning (in-context examples seem to specify the task to be performed)
  - LLMs seem to perform learning without gradient steps simply from examples you provide within their contexts
- Chain-Of-Thought Prompting

Lecture 10: Natural Language Generation

- NLP = NLU + NLG
- NLG tasks
  - Less Open-Ended: Machine translation, summarization
  - More Open-Ended Task-driven dialog, chitchat dialog, story generation
  - Formalized by entropy of task
- These different classes require different pre-training + architecture schemes
- For non open-ended tasks, encoder-decoder system is better; for open-ended task, input => autoregressive decoder => output text
- Repetition in Language Generation
  - Happens bc LM keeps predicting the same most likely token (self-amplification) when predicting the next most likely string (MLE)
  - NOT SOLVED BY ARCHITECTURE (transformer or LSTMs suffer from this)
- Reducing repetition
  - Simple option: heuristic is don't repeat n-grams
  - Different training objective: unlikelihood objective (penalizes generation of already-seen tokens), coverage loss (prevents attention mechanism from attending to same words multiple times), contrastive decoding (maximize logprob difference between large LM and small LM)
- Most likely string isn't reasonable for open-ended generation
- Ideas to Improving Decoding - focused on sampling instead
  - Top-K sampling
    - Weighted sampling of only the top k tokens in the probability distribution

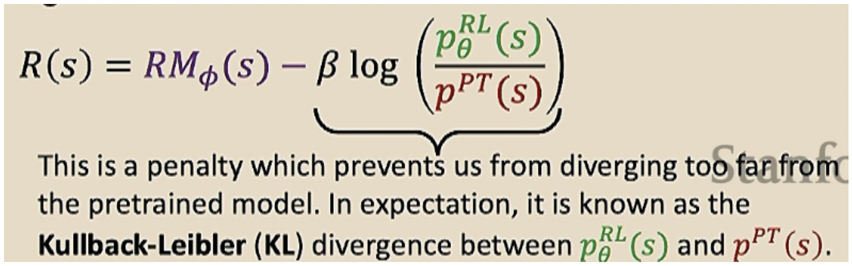
    - increase k = diverse, but risky outputs. decrease k = safe, but generic outputs
    - Main issue, top-k can cut off too quickly or too slowly
    - Top-p (nucleus) sampling
        - Sample from tokens in the the top p percentile from probability distribution
    - Typical Sampling
        - Reweights token sampling by entropy (flatter distribution has higher entropy)
    - Epsilon sampling
        - Set threshold for lower bounding valid probabilities
- Temperature hyperparameter (tau) - raising temp (tau > 1): next token distribution becomes more uniform a.k.a more diverse output, lower temp: next token distribution becomes more spiky
- To prevent decoding a bad sequence from the model
    - Decode a bunch of sequences (10 is normal number)
    - Define scoring function to re-rerank sequences at the end (perplexity is one)
        - Repetitive utterances generally get low perplexity
- MLE => Exposure bias during generation time
    - Training time, MLE input is gold context token from real, human-generated text
    - Generation time, MLE input is previously generated (predicted) tokens
- Exposure Bias Solutions
    - DAgger (dataset aggregation) - add fully generated sequences
    - Scheduled sampling - decode token and feed that as next input rather than gold standard token, increase P of replacement over course of training



- Evaluating NLG systems
    - Content overlap metrics
        - Fast/efficient, based on n-gram overlap
        - Bad for open-ended tasks
    - Model-based metrics
        - Calculates on learned embeddings/representations
        - Semantic similarity: Vector Similarity, Word Mover's Distance (distance between sequences via embedding similarity matching of individual words), BERTSCORE
        - Open-ended Text Generation evaluation: MAUVE - information divergence (distributional module) between sentences/sequences
    - Human evaluation

Lecture 11: Prompting, RLHF

- GPT-3 paper: LM's are few-shot learners
    - Specify a task by simply prepending example of the task before your example (in-context learning)
    - Few-shot learning is an emergent property of model scale
- Zero-shot learning
    - Ability to do many tasks with no examples, and no gradient updates
    - Examples
        - QA
        - Comparing probabilities of sequences based on LM
- Prompting: Right way to define a task so that a model picks up/knows what task to run
    - Frozen LM no gradient updates
- Chain-of-thought prompting
    - Also emergent property w/ scale
    - Zero-shot strategy: Append a statement like "Lets think about this step by step" to the LM.
- "Prompt Engineering"
    - Jailbreaking LMs, Asking a model for reasoning
- Limits of Zero-Shot and Few-Shot Learning
    - Limits of transformer contexts (~1000 tokens), complex tasks need gradient updates
- Instruction fine-tuning
    - Scaling fine-tuning to many tasks
    - Bigger model = bigger delta between pre-training + fine-tuning
    - Fine-tuning small model on a bunch of tasks > pre-training large model
- Limits of instruction fine tuning
    - Expensive to collect ground-truth data for tasks
    - Open-ended/creative generation tasks have no right answer
    - LM penalizes all token-level mistakes equally (some are worse than others)
    - LM's can't pick up human preferences
- RL to the rescue for LM
- Policy gradient methods allow us to perform gradient ascent/maximize a reward function
    - For arbitrary, non-differentiable reward function R(s)
- Modeling human preferences
    - Human-in-the-loop is expensive => model their preferences as an NLP problem
    - Don't ask for direct ratings => ask for pairwise comparisons (more accurate)
- RLHF
    - Take pretrained (or fine-tuned) LM p and optimize its parameters with a reward function

        - 

- Whats next?
    - Reward Hacking is a common issue (chatbots are rewarded to produce responses that seem helpful/authoritative regardless of truth)
    - Results in making up facts + hallucinations
    - Models of human preference are unreliable
    - RL from AI feedback
    - Finetuning LMs on their outputs

Lecture 12: Question Answering
- QA 2015-2019: Text passage + question from it (structured knowledge graph)
- Unstructured text -> answering a question
- SQuAD - Stanford question answering dataset
    - 100k annotated (passage, question, answer) triples
    - Crowd-sourced, each answer is a short segment of text
    - Evaluation: Exact Match (0/1) and F1 (partial credit) <- harmonic mean of precision + recall
- TriviaQA, Natural Questions (Google search questions), HotpotQA (need 2 pages at least/multistep query)
- QA Reading Comprehension Problem Formulation
    - Input: C (c1 to cN), Q (q1 to qM),
    - Output: 1 <= start <= end <= N
- Stanford Attentive Reader
    - Question represented as query vector
        - Pass input through bidirectional LSTM
    - Passage - embed it as a bunch of tokens
        - Use attention to identify where in the passage the answer to the question should start + end (a_i = softmax over i (q^T * W * passage_i))
- BERT-based systems for reading comprehension
    - All representation learning stuff is done by BERT (attention done over both question + reference)
- Open-domain question answering
    - Document Retriever -> Document Reader Framework
    - Generating answers is better than extracting answers from the documents

Lecture 13: ConvNets, Tree Recursive NNs

- ConvNet Idea: What if we can use word subsequences to model language?
  - 1-D convolutions over chars/words to learn features
- Max pooling/average pooling: Average over the filters that are applied
  - k-max pooling
- Stride, dilations
  - Dilations = skipping words/characters to expand context window
- Convolution size = length of subsequence that is being looked at
- Issues with fine-tuning word vectors
  - Only words seen in the fine-tuning dataset will have their parameters updated/errors backpropagated
- LayerNorm (Transformers) - calculate statistics across all feature dimensions for each instance independently, BatchNorm (CNNs) - normalizes all elements and items in a batch
- TreeRNN's
  - Recursion in human language (noun phrase containing a noun phrase, etc)
  - Parse sentence to create a recursive tree => models can jointly learn parsed tree and compositional vector representations
- Recursive Neural Tensor Networks
  - Modeling representation of phrase like "not very good" as negative sentiment, multiplicative and additive interactions between words (i.e. very + good = more positive, negate very good with not = very negative)
  - Weight matrices learned over these interactions
- Stanford Sentiment TreeBank
  - How do people judge the individual sentiment of sentences?

Lecture 14: Language relationship with LLM's

- Language has underlying structure
- grammatical - follows rules in structure, ungrammatical is opposite
- Initial self-supervised learning
  - Input => Syntax => Semantics => Discourse
- Syntactic structure allows new words to be integrated
  - COGS benchmark: new word-structure combinations
- Testing Jabberwocky sentences to assess/examine model's latent space
- Structurally, anything can be an object BUT many languages mark objects differentially (differential object marking) with specific words/phrases before object
- LLM's acceptability judgements for new/weirder constructions is similar to humans
- Meaning is sensitive to context
  - i.e. break X can have multiple meanings
- Big NLP question: striking balance between surface-level memorization and deeper level context-specific abstractions
- Multi-linguality: helpers share parameters between high-resource and low-resource languages

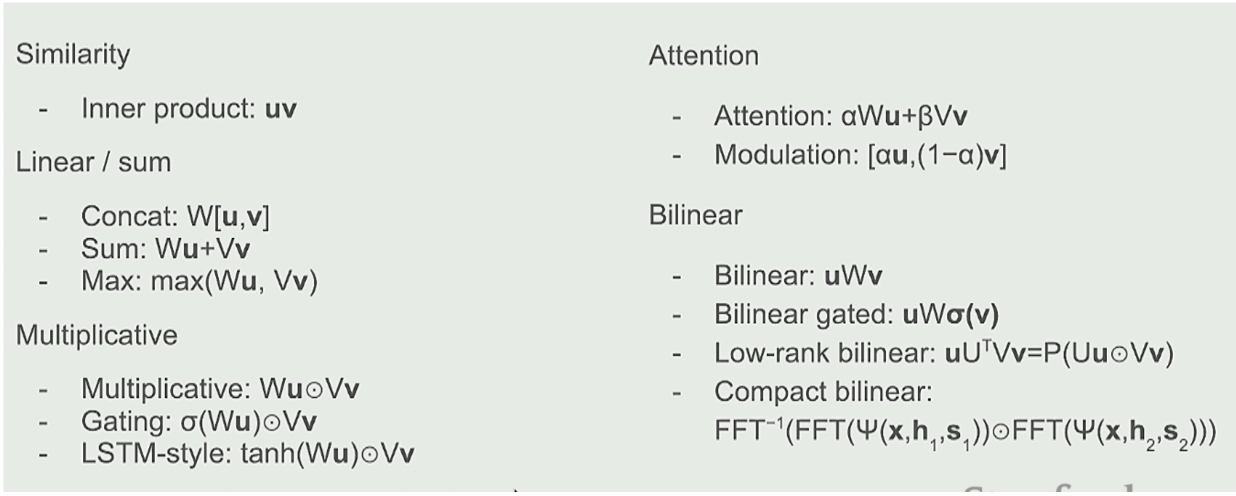
- Language typology vs language universals

Lecture 15: Code Generation

- Program synthesizer: program takes specification and outputs a program that satisfies it
  - ex. Logical formula, input/output examples, natural language description
- Problem with synthesis from examples: ambiguity
- Pragmatic Reasoning helps humans overcome ambiguity
  - Rational speech act
- Program synthesis with LLMs
  - Training Idea: P (code | docstring)
  - Training paradigm: given docstrings with input/output examples, produce program that implements functionality
  - Sampling more programs is important to getting one correct program => Ranking => re-rank + show only top k
  - Sampling P (docstring | code) is worse since its less frequent
- AlphaCode: pre-training (standard cross-entropy loss training on github + encoder trained with MLM loss), fine-tuning (human solutions to competitive programming problems)
  - RL fine-tuning: only one correct solution needs to be generated
  - Value-conditioning: use incorrect submissions to augment training with comment saying correct or incorrect
  - Sampling => Testing => Filtering scheme
- Compositionality in LM's doesn't necessarily exist yet (given trivial piece X and Y, chaining X and Y is easy for humans but not LMs)
- Automation bias in generated code

Lecture 16 (Guest Lecture - Douwe Kiela (FAIR + Hugging Face)): Multimodal Deep Learning

- McGurk Effect - speech sounds are miscategorized when auditory and visual cues are in conflict
- Early Multimodal Models
  - Visual-Semantic Embeddings
  - Visual Bag of words (identify keypoints + get descriptors, cluster these + map to counts => concatenate with textual vector => apply SVD to fuse information)
  - Sentence-level representations
  - Image to text captioning with Seq2Seq models
- Multimodality problems - noise, hard to cover all modalities, one modality can dominate others (text >> audio/vision sometimes)
- YOLO for region features (i.e. bounding boxes)
- Vision Transformers
  - Patch + Position embedding as input into encoder => attach MLP head at end for classification
- Multimodal fusion

- 

- "Late fusion" - contrastive models
  - CLIP (both text + image encoder) + more robust than ResNet
- LAION-5B: huge open source datasets of image-text pairs exist (PMD, Common objects in context)
- Multimodal foundation models
  - VisualBert - early fusion of BERT + convnet Image features near input layer
  - ViLT - word embeddings concatenated with linear projection of flattened patches into transformer
  - FLAVA - model for vision + language, computer vision, and NLP
- CoCa Constrastive Captioner - best of contrastive + generative worlds
- Evaluation
  - Visual Question Answering (dominant task in vision + language)
  - Clever - compositionality of text + vision in questions asked
- Word order in VQA isn't very good - Winoground example
- Text + Audio - multilingual, multitask models
- Text to 3D - point cloud diffusion

Lecture 17: Co-reference Resolution
- Problem: Want to identify all mentions that refer to a specific entity
  - Hard problem because of ambiguity
  - Uses: Information extraction, question answering, summarization
- Mention Detection: marking all pronouns/named entities/NPs (noun phrase) as mentions over-generates mentions
  - Traditionally use pipeline of NLP systems
  - Train a classifier specifically for mention detection instead of POS/NER tagger
- Co-reference: two mentions refer to the same entity in the world (Barack Obama + Obama refer to same person)
- Anaphoric relations (he => Barack Obama)
  - Not all are coreferential (No dancer twisted her knee)

- Important contextual elements of language - word-sense disambiguation, co-reference, discourse model
- Hobbs naive algorithm - traditional pronominal anaphora (co-reference) resolution
- Winograd Schema
    - Proposed by some as an alternative to the Turing test. If you've solved coreference, you've solved AI!
- Mention Pair Model
    - Train binary classifier that assigns every pair of mentions (m_i, m_j) a probability of being co-referent
    - Want positive examples to be near 1 if m_i and m_j are co-referent
    - $$J = -\sum_{i=2}^{N} \sum_{j=1}^{i} y_{ij} \log p(m_j, m_i)$$

      Iterate through mentions

      Iterate through candidate antecedents (previously occurring mentions)

      Coreferent mentions pairs should get high probability, others should get low probability
    - To go from pairs of mentions to mention clusters, pick mention threshold and add co-reference links between mention pairs above threshold
        - Transitive closure to get clustering (has to be circular at some point)
    - Bad on long documents; many mentions only have one clear antecedent but we are asking the model to predict ALL
- Mention Ranking (better alternative)
    - Only add one co-reference link to an antecedent by applying softmax over probabilities
    - current mention to be linked to any one of its candidate antecedents its co-referent with
- Neural Coref Model
    - Input: Candidate antecedent embeddings + features, mention embeddings + features + additional features
- End-to-End Neural Coref Model's
    - Consider spans of text as candidate mentions
    - Span embedding = [hidden states for span's start and end (left + right context), attention-based representation (represents span itself), additional features]
    - Score spans: s(i,j) = s(i) <- is i a mention + s(j) <- is j a mention + s_a(i,j) <- are they co-referent
    - O(T^4) runtime for naive approach; requires pruning
- Evaluation metrics: MUC, CEAF, B-CUBED
- NN improves common noun co-references (different nouns for same object)

Lecture 18: Modal Analysis and Explanation

- Motivation: black box, understanding model bias, understanding model architecture to move towards to the models of tomorrow
- Model analysis at different levels of abstraction
  - Neural model as probability distribution/decision function
  - Neural model as sequence of vector representations with depth and time as axes
  - Parameter weights, attention, dropout
- OOD test set w/ carefully construction manual evaluation methods
- Checklisting: careful test sets as unit test suites
- Saliency maps for prediction explanations (what about input led to output)
  - Gradient method - calculate gradient norm
  - high gradient norm means changing local word would affect score a lot
- Breaking models: adversarial inputs
- Interpretable architecture components
  - Attention heads are correlated with linguistic properties (one even related to co-referent properties!)
- Probing (supervised analysis of NN)
  - BERT representations contain enough meaning to add layer on BERT output that can "mask" embedding for a specific downstream task (i.e. POS tagging or NER)
  - Dependency parsing trees can be generated from BERT representations
-

Lecture 19 (Guest Lecture - Been Kim, Google Brain) : Model Editing and Interpretability

- Human and Machine's representational spaces are like overlapping sets
  - Expand what we know by analyzing machine spaces
- Reasons why we can't understand machines
  - Assumptions, Expectations (misalignment), Beyond us (humans can't understand them)
- Attribution from explainability tools (like SHAP) isn't necessarily accurate
  - 0 attribution doesn't necessarily mean model isn't using feature
- Downstream tasks related to explainability: Recourse + spurious features
- Improving explainability can happen by sampling more (better approximation of function shape)
- Model Editing
  - Research has found low correlation between factual knowledge storage and a given localization layer
  - Causal Tracing algorithm (ROME)
- Does editing success correlate with tracing effect/fraction restored (localization)?
  - No, to edit a model, pick a good layer. Don't worry about localization (for NLP word/subword tokens, for images, specific pixels)
- Emergent behaviors in multi-agent systems
- General Principles

- Observational study: given data, discover behavior
- Controlled study: intervene, observe, discover