Tuesday, August 11, 2020

Stanford CS224N - NLP

Lecture 01~02 - Word Vectors

Word Representing

- **As discrete symbols**
  - one-hot encoding
  - problems: words are infinite, no natural notion of similarity (one-hot vectors are orthogonal)
- **As distributional semantics** (by their contexts)
  - A word's meaning is given by the words that frequently appear close-by (One of the most successful ideas of modern statistical NLP!)
  - When a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
  - Use the many contexts of *w* to build up a representation of *w*
  - How: count-based vs. shallow window-based
    - Count-based
      - Rely on matrix factorization, e.g. LSA, HAL.
      - Effectively leverage global statistical information and primarily used to capture word similarities
      - Do poorly on tasks such as word analogy
    - Shallow window-based
      - Learn word embeddings by making predictions in local context windows, e.g. word2vec
      - Able to capture complex linguistic patterns beyond word similarity
      - Fail to make use of the global co-occurrence statistics

---

Word2vec

Word vectors

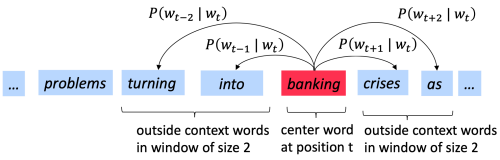
- We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts
- Word vectors are sometimes called word embeddings or word representations. They are a distributed representation

Idea

- We have a large corpus of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context ("outside") word o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability
- Every word has two vectors. Why? (it's easier for optimization. You can average both of them at the end)
- Intuition: If two different words have very similar "contexts" (that is, what words are likely to appear around them), then our model needs to output very similar results for these two words (similar probabilities for same context words). And one way for the network to output similar context predictions for these two words is if the word vectors are similar. So, if two words have similar contexts, then our network is motivated to learn similar word vectors for these two words!

Algorithm

- **Objective function**
  - For each position $t=1, \ldots, T$, predict context words within a window of fixed size m, given center word w_j.
  - Likelihood $= L(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m\ (j \ne 0)} P(w_{t+j} \mid w_t; \theta)$, θ is all variables to be optimized
  - The objective function $J(\theta)$ is the average negative log likelihood



✦

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

  - Why negative? Change maximizing to minimizing. Why log? Change product to sum to avoid float underflow.

- **Probability**

- How to calculate $P(w_{t+j} \mid w_t; \theta)$?

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

- Why dot product for $u_o^T v_c$? Dot product compares similarity of o and c. $u_T v = \sum_{i=1}^{n} u_i v_i$, larger dot product = larger probability.
- Why exponentiation for $\exp(u_o^T v_c)$? Exponentiation makes anything positive.
- Why normalize $\sum_{w \in V}$? Normalize over entire vocabulary to give probability distribution.
- It's an example of softmax function $\text{softmax}(x_i) = \frac{exp(x_i)}{\sum_{j=1}^{n} exp(x_j)}$. "max" means it amplifies probability of largest x_i. "soft" mean it still assigns some probability to smaller x_i (contrast to hardmax which assign 0 to smaller x_i).

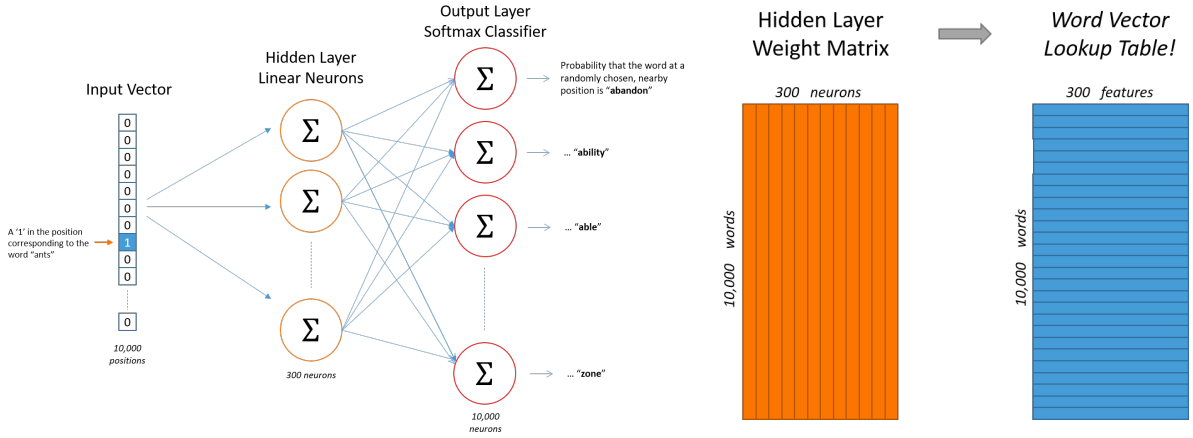
- **Derivations of gradient**

$$\frac{\partial \log P(o \mid c)}{\partial v_c} = \frac{\partial \log \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}}{\partial v_c} = \frac{\partial \log \exp(u_o^T v_c)}{\partial v_c} - \frac{\partial \log \sum_{w \in V} \exp(u_w^T v_c)}{\partial v_c} = u_o - \sum_{x \in V} P(x \mid c) \cdot u_x$$

- $\frac{\partial \log \exp(u_o^T v_c)}{\partial v_c} = \frac{\partial u_o^T v_c}{\partial v_c} = u_o$, since log and exp are inverse of each other
- $\frac{\partial \log \sum_{w \in V} \exp(u_w^T v_c)}{\partial v_c} = \frac{\sum_{x \in V} \exp(u_x^T v_c) \cdot u_x}{\sum_{w \in V} \exp(u_w^T v_c)} = \sum_{x \in V} \frac{\exp(u_x^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \cdot u_x = \sum_{x \in V} P(x \mid c) \cdot u_x$
- The gradients equal the observed context word subtracting from what out model thinks the context should look like.

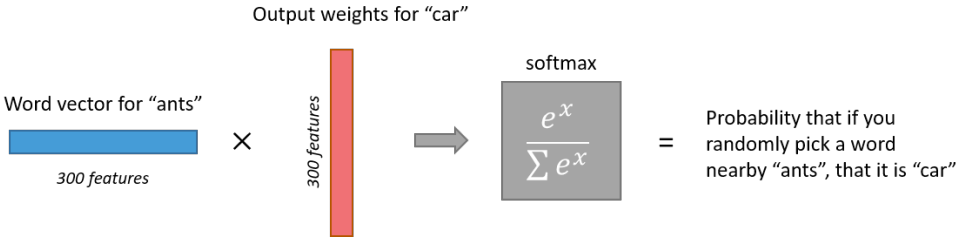
- **Architecture**
  - single hidden layer, no activation function on the hidden layer neurons, but the output neurons use softmax.
  - Why two vectors (the hidden layer and the output layer)? Easier optimization. Average both at the end.
  - Two model variants
    1. Skip-grams (SG): Predict context ("outside") words (position independent) given center word.
       - Works well with small amounts of training data and represents even words that are considered rare
    2. Continuous Bag of Words (CBOW): Predict center word from (bag of) context words

- Trains several times faster and has slightly better accuracy for frequent words



Model architecture

Hidden layer



Output layer

Problems

- It's a huge neural network! A lot of training data and a lot of weights to update. For a network with 300 neurons in hidden layer and a vocabulary of 10000 words, we have a weight matrix with 300 x 10000 = 3 million
- **Solution**
  - Hierarchical softmax to reduce computation cost and subsampling frequent words to decrease the number of training examples.
  - Modifying the optimization objective with a technique they called "*Negative Sampling*", which causes each training sample to update only a small percentage of the model's weights.

Solution 1.1: hierarchical softmax

- Use Binary Huffman Tree in the output layer instead of flatten layer, as it assigns short codes to the frequent words which results in fast training
- Computation cost of $P(w_{t+j} \quad w_t; \theta)$ is proportional to log(vocab_size)
- unlike the standard softmax formulation of the Skip-gram which assigns two representations v_w and $v_{w'}$ to each word w, the hierarchical softmax formulation has one

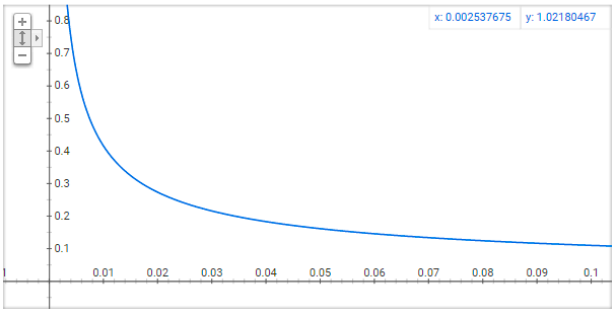
representation v_w for each word w and one representation $v_{n'}$ for every inner node n of the binary tree.

Solution 1.2: subsampling frequent words

- Frequent words like “the”, “a” doesn’t tell much about the context word, and they’re far more than needed
- For each word we encounter in our training text, there is a chance that we will effectively delete it from the text. The probability (1-sampling rate) that we cut the word is related to the word’s frequency.
- Sampling rate (probability to keep a given word):

$$P(w_i) = (\sqrt{\frac{z(w_i)}{0.001}} + 1) \cdot \frac{0.001}{z(w_i)}$$

Graph for (sqrt(x/0.001)+1)*0.001/x



Solution 2: negative sampling

- the size of our word vocabulary means that our skip-gram neural network has a tremendous number of weights, all of which would be updated slightly by every one of our billions of training samples!
- With negative sampling, we are instead going to randomly select just a small number of “negative” words (let’s say 5) to update the weights for. (In this context, a “negative” word is one for which we want the network to output a 0 for). We will also still update the weights for our “positive” word (which is the word “quick” in our current example).
- Recall that the output layer of our model has a weight matrix that’s 300 x 10,000. So we will just be updating the weights for our positive word (“quick”), plus the weights for 5 other words that we want to output 0. That’s a total of 6 output neurons, and 1,800 weight values total. That’s only 0.06% of the 3M weights in the output layer!
- In the hidden layer, only the weights for the input word are updated (this is true whether you’re using Negative Sampling or not).
- Selecting negative samples (probability to be selected as negative sample):

$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n}(f(w_i)^{3/4})}$, $f(w_i)$ is the number of times word w_i appears in the corpus

- Interesting implementation: They have a large array with 100M elements (which they refer to as the unigram table). They fill this table with the index of each word in the vocabulary multiple times, and the number of times a word’s index appears in the table is given by $P(w_i)$ * table_size. Then, to actually select a negative sample, you just generate a random integer between 0 and 100M, and use the word at that index in the table. Since the higher probability words occur more times in the table, you’re more likely to pick those.

Why not capture co-occurrence counts directly?

You shall know a word by the company it keeps. *(Firth, J. R. 1957:11)*

With a co-occurrence matrix X

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

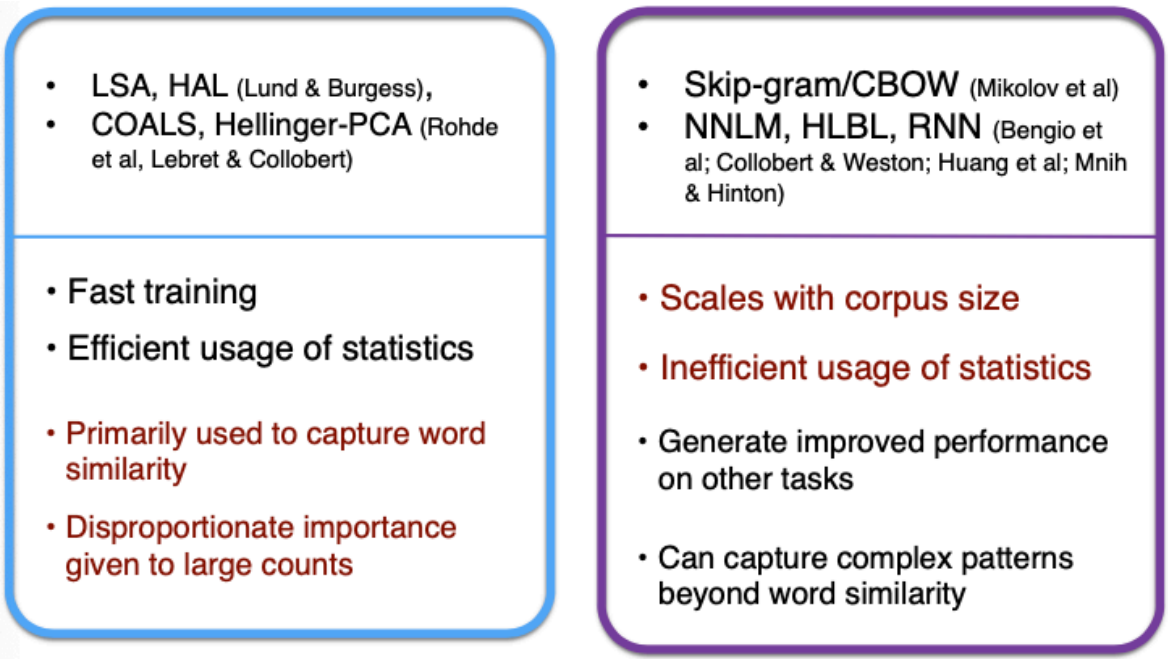
- 2 options: windows vs. full document
- Window: Similar to *word2vec*, use window around each word -> captures both syntactic (POS) and semantic information (example see right)
- Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to "Latent Semantic Analysis"

Problems with simple co-occurrence vectors

- Increase in size with vocabulary
- Very high dimensional: requires a lot of storage
- Subsequent classification models have sparsity issues (causes model less robust)

Solution: low dimensional vectors

- Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
- Usually 25-1000 dimensions, similar to word2vec
- How to reduce the dimensionality?
  - Singular Value Decomposition of co-occurrence matrix X (SVD)
  - Tricks: scaling the counts in the cells can help **a lot**
    - function words (the, he, has) are too frequent
    - ramped windows that count closer words more



- use Pearson correlations instead of counts, then set negative values to 0)

Towards ***GloVe***: Count based vs. direct prediction

GloVe

- GloVe (Global Vectors for Word Representation)
- Crucial insight: Ratios of co-occurrence probabilities can encode meaning components

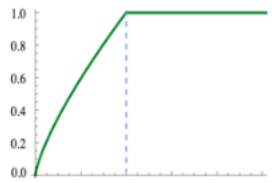
	x = solid	x = gas	x = water	x = fashion
$P(x\|\text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(x\|\text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	8.9	8.5×10^{-2}	1.36	0.96

- Encoding meaning in vector differences
- Combining the best of both worlds (count based vs. direct prediction), GloVe consists of a weighted least squares model that trains on global word-word co-occurrence counts and thus make efficient use of statistics

Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?

A: Log-bilinear model: $w_i \cdot w_j = \log P(i \;\; j)$, with vector differences: $w_x \cdot (w_a - w_b) = \log \frac{P(x \;\; a)}{p(x \;\; b)}$

Cost function: $J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$

$f \sim$ 

Algorithm

- **Idea**
  - Using global statistics to predict the probability of word j appearing in the context of word i with a least squares objective

- **Co-occurrence Matrix**
  - X: word-word co-occurrence matrix
  - X_{ij}: the number of times word j occur in the context of word i
  - $X_i = \sum_k X_{ik}$: the number of times any word k appears in the context of word i
  - $P_{ij} = P(w_j \mid w_i) = \frac{X_{ij}}{X_i}$: the probability of j appearing in the context of word i
- **Cost function**
  i. Define probability of word j appears in the context of word i:
  $Q_{ij} = P(j \mid i) = \frac{\exp(\overrightarrow{u_j}^T \overrightarrow{v_i})}{\sum_{w \in V}^{W} \exp(\overrightarrow{u_w}^T \overrightarrow{v_i})}$ (same as skip-gram)
  ii. Cost function: $J = -\sum_{i \in \text{corpus}} \sum_{j \in \text{context(i)}} log Q_{ij}$ (Why log? Change product to sum to avoid float underflow.)
  iii. As the same words i and j can appear multiple times in the corpus, it is more efficient to first group together the same values for i and j:
  $J = -\sum_{i=1}^{W} \sum_{j=1}^{W} X_{ij} log Q_{ij}$
  iv. One significant drawback of the cross-entropy loss is that it requires the distribution Q to be properly normalized, which involves the expensive summation over the entire vocabulary. Instead, we use a least square objective in which the normalization factors in P and Q are discarded :
  $\hat{J} = \sum_{i=1}^{W} \sum_{j=1}^{W} X_i (\hat{P}_{ij} - \hat{Q}_{ij})^2$, where $\hat{P}_{ij} = X_{ij}$ and $\hat{Q}_{ij} = \exp(\overrightarrow{u_j}^T \overrightarrow{v_i})$ are the unnormalized distributions.
  v. X_{ij} can takes on very large values and makes the optimization difficult. An effective change is to minimize the squared error of the logarithms of $\hat{P}$ and $\hat{Q}$:
  $\hat{J} = \sum_{i=1}^{W} \sum_{j=1}^{W} X_i (\log(\hat{P})_{ij} - \log(\hat{Q}_{ij}))^2 = = \sum_{i=1}^{W} \sum_{j=1}^{W} X_i (\overrightarrow{u_j}^T \overrightarrow{v_i} - \log(X_{ij}))^2$
  vi. Another observation is that the weighting factor Xi is not guaranteed to be optimal. Instead, we introduce a more general weighting function, which we are free to take to depend on the context word as well:
  $\hat{J} = \sum_{i=1}^{W} \sum_{j=1}^{W} f(X_{ij}) (\overrightarrow{u_j}^T \overrightarrow{v_i} - \log(X_{ij}))^2$

Summary

- Fast training

- Scalable to huge corpora
- Good performance even with small corpus and small vectors
- In conclusion, the GloVe model efficiently leverages global statistical information by training only on the nonzero elements in a word-word co-occurrence matrix, and produces a vector space with meaningful sub-structure. It consistently outperforms *word2vec* on the word analogy task, given the same corpus, vocabulary, window size, and training time. It achieves better results faster, and also obtains the best results irrespective of speed.

How to evaluate word vectors?

- Related to general evaluation in NLP: intrinsic vs. extrinsic
- Intrinsic
  - Evaluation on a specific/intermediate subtask
    - Word vector analogies $\vec{a} : \vec{b} :: \vec{c} :?$
      - Find the word vector that maximizes the cosine similarity
        $d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{\| x_b - x_a + x_c \|}$
    - Word vector distances and their correlation with human judgements
  - Fast to compute
  - Helps to understand that system
  - Not clear if really helpful unless correlation to real task is established
- Extrinsic
  - Evaluation on a real task (e.g. NER)
  - Can take a long time to compute accuracy
  - Unclear if the subsystem is the problem or its interaction or other subsystems
  - If replacing exactly one subsystem with another improves accuracy -> Winning!

Word senses and word sense ambiguity

- Most words have lots of meanings!
  - Especially common words
  - Especially words that have existed for a long time

- Improving word representations via global context and multiple word prototypes (Huang et al. 2012)
  - Idea: Cluster word windows around words, retrain with each word assigned to multiple different clusters bank1, bank2, bank3, etc
- Linear algebraic structure of word senses, with applications to polysemy
  - Different senses of a word reside in a linear superposition (weighted sum) in standard word embeddings like *word2vec*
  - $v_{bank} = \alpha_1 v_{bank_1} + \alpha_2 v_{bank_2} + \alpha_3 v_{bank_3}$, *where* $\alpha_1 = \frac{f_1}{f_1 + f_2 + f_3}$ for frequency f_i

Lecture 03 ~ 04 - Neural Nets

Classification review/introduction

- Softmax classifier: $p(y \mid x) = \frac{\exp(W_y \cdot x)}{\sum_{c=1}^{C} \exp(W_c \cdot x)}$
- Training with softmax and cross-entropy
  - Training objective: maximize the probability of the correct class y: $p(y \mid x)$
  - Equivalent to: minimizing the negative log probability of that class: $-\log p(y \mid x)$
    - Using log probability converts our objective function to sums, which is easier to work with on paper and in implementation (avoid underflow)
  - Cross-entropy (from information theory)
    - Let the true probability distribution be p
    - Let our computed model probability be q
    - The cross entropy is: $H(p, q) = -\sum_{c=1}^{C} p(c) \log q(c)$
    - Assuming a ground truth (or true or gold or target) probability distribution that is 1 at the right class and 0 everywhere else: p = [0,...,0,1,0,...0] then:
    - Because of one-hot p, the only term left is the negative log probability of the true class
- Classification difference with word vectors
  - We learn both **W** (conventional parameters) and word vector **x** (representations)

Pre-trained word vectors

- Should I use available "pre-trained" word vectors?
  - Almost always, yes!
  - They are trained on a huge amount of data, and so they will know about words not in your training data and will know more about words that are in your training data
  - Have 100s of millions of words of data? Okay to start random
- Should I update ("fine tune") my own word vectors?
  - If you only have a small training data set, don't train the word vectors
  - If you have have a large dataset, it probably will work better to train = update = fine-tune word vectors to the task

Lecture 05 - Dependency Parsing

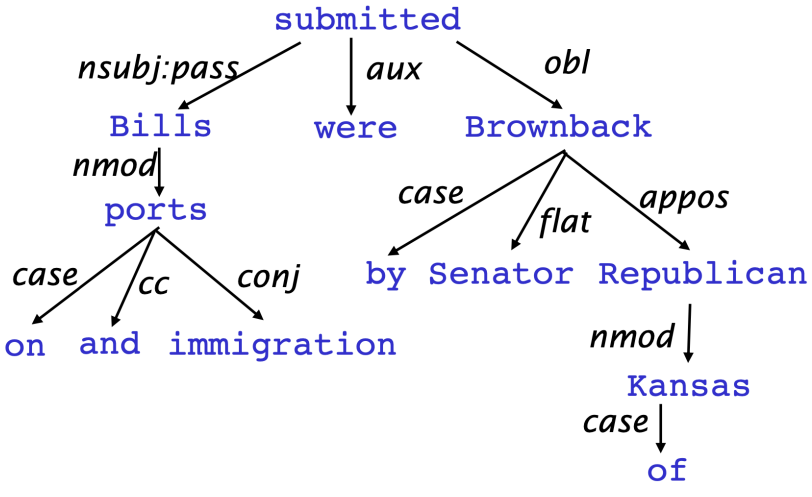
Syntactic Structure: Consistency and Dependency

- Tow views of linguistic structure
  - Constituency: phrase structure grammar = context-free grammars (CFGs)
    - Words -> words combine into phrases -> phrases combine into bigger phrases
    - Phrase structure organizes words into nested constituent
  - Dependency: dependency structure shows which words depend on (modify or are arguments of) which other words.
    - Ambiguity: prepositional phrase attachment ambiguity, coordination scope ambiguity, adjectival modifier ambiguity, verb phrase attachment ambiguity

Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations ("arrows") called dependencies

The arrow connects a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate)

Usually, dependencies form a tree (connected, acyclic, single-head)



Dependency Grammar and Treebanks

- Treebank
  - Reusability of the labor
    - Many parsers, part-of-speech taggers, etc. can be built on it
    - Valuable resource for linguistics
  - Broad coverage, not just a few intuitions
  - Frequencies and distributional information
  - A way to evaluate systems

Dependency Conditioning Preference

- Source of information for dependency parsing
  - Bilexical affinities
  - Dependency distance
  - Intervening material
  - Valency of heads

Dependency Parsing

- A sentence is parsed by choosing for each word what other word (including ROOT) is it a dependent of
- Usually some constraints:
  - Only one word is a dependent of ROOT
  - Don't want cycles A→B, B→A
- This makes the dependencies a tree
- Final issue is whether arrows can cross (non-projective) or not
- Methods of Dependency Parsing
  - Dynamic programming
  - Graph algorithms
  - Constraint satisfaction
  - "Transition-based parsing" or "deterministic dependency parsing"

Lecture 06 - Language Models and RNN

Language Models

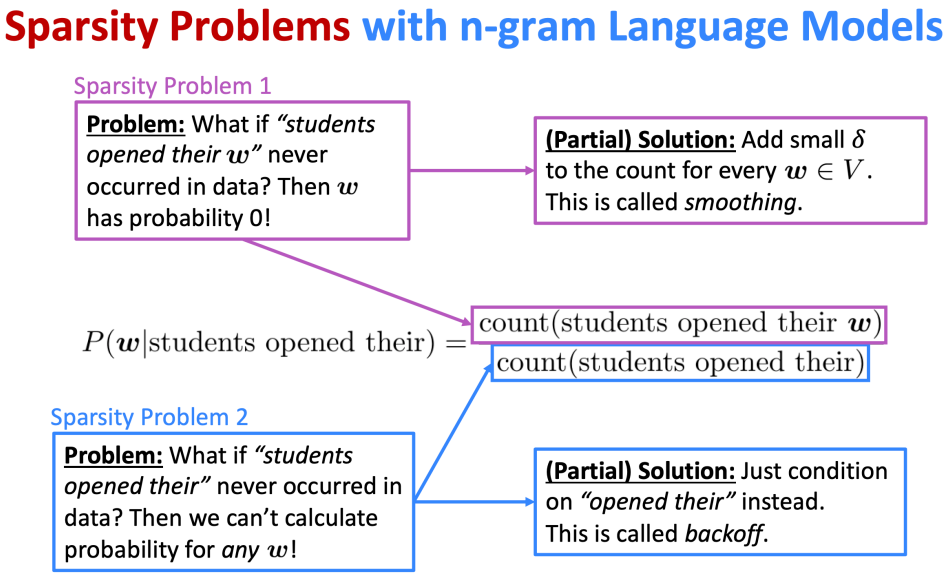
- A system that performs the task of predicting what word comes next.
- More formally: given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$: $P(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(1)})$, where $x^{(t+1)}$ could be any word in Vocabulary
- You can also think of a Language Model as a system that assigns probability to a piece of text.
- Conditional Language Modeling: the task of predicting the next word, given the words so far, **and also some other input x.** $P(y_t \mid y_1, \ldots, y_{t-1}, x)$
  - Examples of conditional language modeling tasks:
    - Machine Translation (x=source sentence, y=target sentence)
    - Summarization (x=input text, y=summarized text)
    - Dialogue (x=dialogue history, y=next utterance)

N-gram Language Models

- Definition: to compute the probabilities mentioned above, the count of each n-gram could be compared against the frequency of each word.
- A n-gram is a chunk of n consecutive words. Unigram, bigram, trigram, 4-grams.
- Assumption: $x^{(t+1)}$ only depends on the preceding n-1 words (Markov Assumption),

$$P(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(1)}) = P(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(t-n+2)}) = \frac{P(x^{(t+1)}, x^{(t)}, \ldots, x^{(t-n+2)})}{P(x^{(t)}, \ldots, x^{(t-n+2)})} = \frac{\text{count(students opened their w)}}{\text{count(students opened their)}}$$

- Sparsity Problem



- Storage Problem: Need to store count for all n-grams you saw in the corpus
- N usually is no bigger than 5 (sparsity and storage problems get worse with bigger N)

Neural Language Models

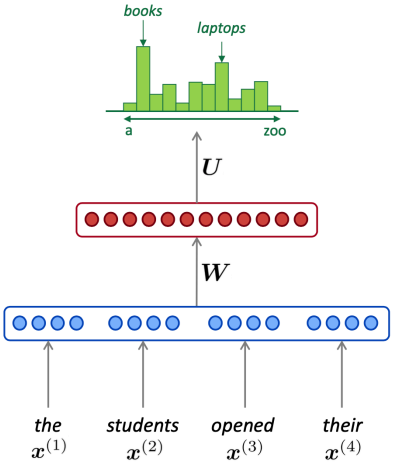
A fixed-window neural Language Model

Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

Improvements over *n*-gram LM:
- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:
- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W. No symmetry in how the inputs are processed.

We need a neural architecture that can process *any length input*



Recurrent Neural Networks (RNN)

- A neural architecture that can process any length input
- **A simple RNN**

A Simple RNN Language Model

$\hat{y}^{(4)} = P(x^{(5)}|\text{the students opened their})$

output distribution

$$\hat{y}^{(t)} = \text{softmax}\left(Uh^{(t)} + b_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_e e^{(t)} + b_1\right)$$

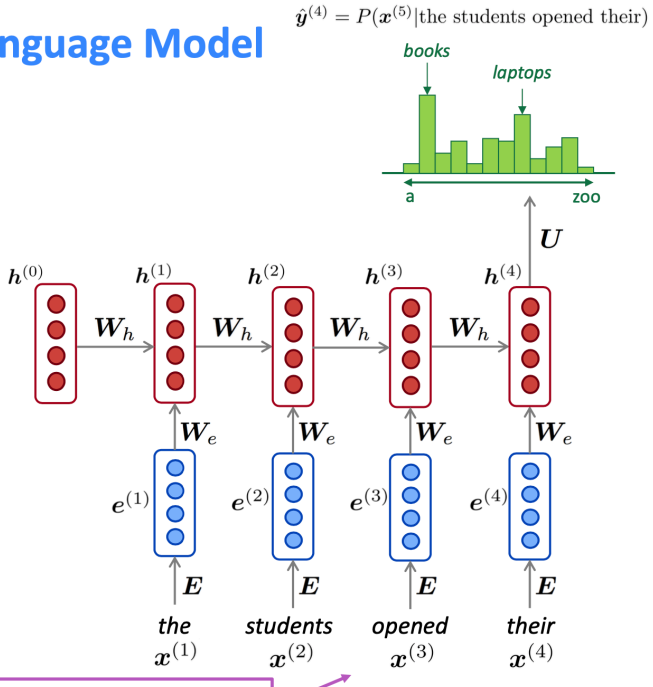
$h^{(0)}$ is the initial hidden state

word embeddings

$$e^{(t)} = Ex^{(t)}$$

words / one-hot vectors

$$x^{(t)} \in \mathbb{R}^{|V|}$$



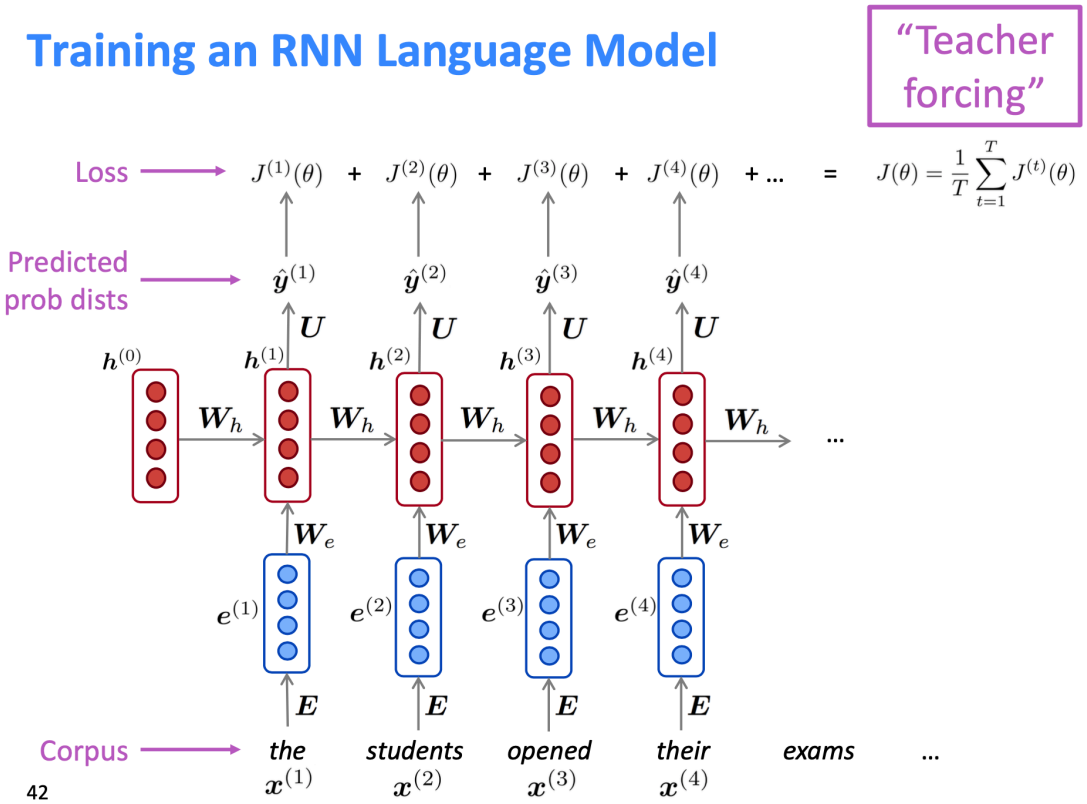
Note: this input sequence could be much longer, but this slide doesn't have space!

- RNN advantages
  - Can process any length input
  - Computation for step t can (in theory) use information from many steps back
  - Model size doesn't increase for longer input
  - Same weights applied on every timestep, so there is symmetry in how inputs are processed.
- RNN disadvantages
  - Recurrent computation is slow - because it is sequential, it cannot be parallelized
  - In practice, it's difficult to access information from many steps back

---

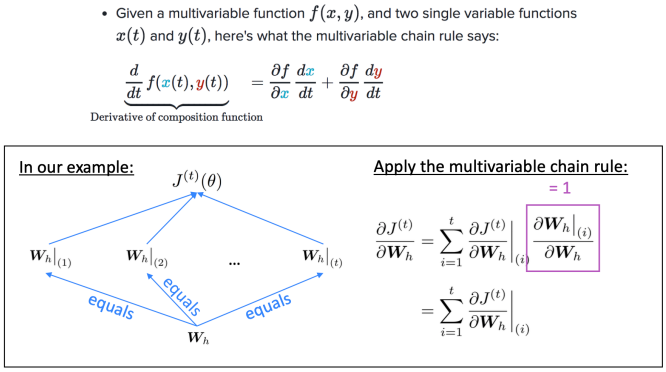
Training a RNN

2. Get a big corpus of text which is a sequence of words $x^{(1)}, \ldots, x^{(T)}$
3. Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ for every step t
   - i.e. predict probability distribution of every word, given words so far
4. Loss function on step t is cross-entropy between predicted probability distribution $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):
   - $J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = -\sum y_w^{(t)} log\, \hat{y}_w^{(t)} = -log\, \hat{y}_{x_{t+1}}^{(t)}$
5. Average this to get overall loss for entire training set:
   - $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T}\sum_{t=1}^{T} -log\, \hat{y}_{x_{t+1}}^{(t)}$



6. Backpropagation for RNN

- What's the derivative of $J^{(t)}(\theta)$ w.r.t the repeated weight matrix W_h?
  - $\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\Big|_{(i)}$
  - The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears
  - Why? See graph.



- How to calculate $\sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\Big|_{(i)}$
  - Backpropagate over timesteps i=t,...,0, summing gradients as you go. This algorithm is called "**backpropagation through time**" *[Werbos, P.G., 1988, Neural Networks 1, and others]*

Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

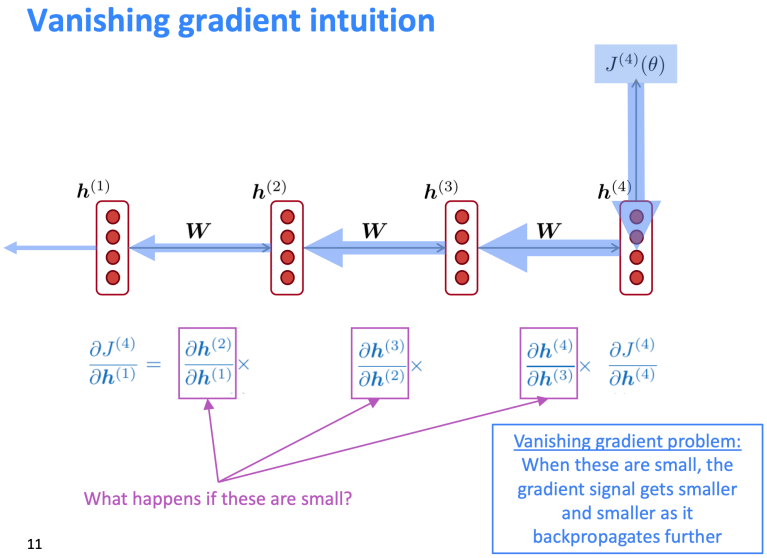
$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

Lower perplexity is better!

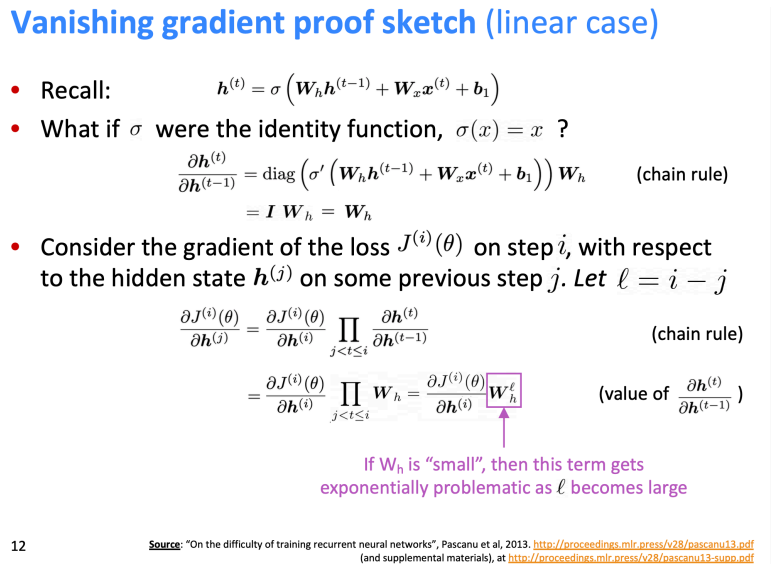
Lecture 07 - Vanishing Gradients and Fancy RNNs

Vanishing Gradients

- Intuition



- Proof sketch



- Why is it a problem
  - Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by. So model weights are updated only with respect to near effects, not long-term effects.
  - If the gradient becomes vanishingly small over longer distances (step t to step t+n), then we can't tell whether:

    - There's no dependency between step t and t+n in the data
    - We have wrong parameters to capture the true dependency between t and t+n

- Solution
  - The main problem is that it's too difficult for the RNN to learn to preserve information over many timesteps. (The hidden state is constantly being rewritten). => A RNN with separate memory (motivation for LSTM)

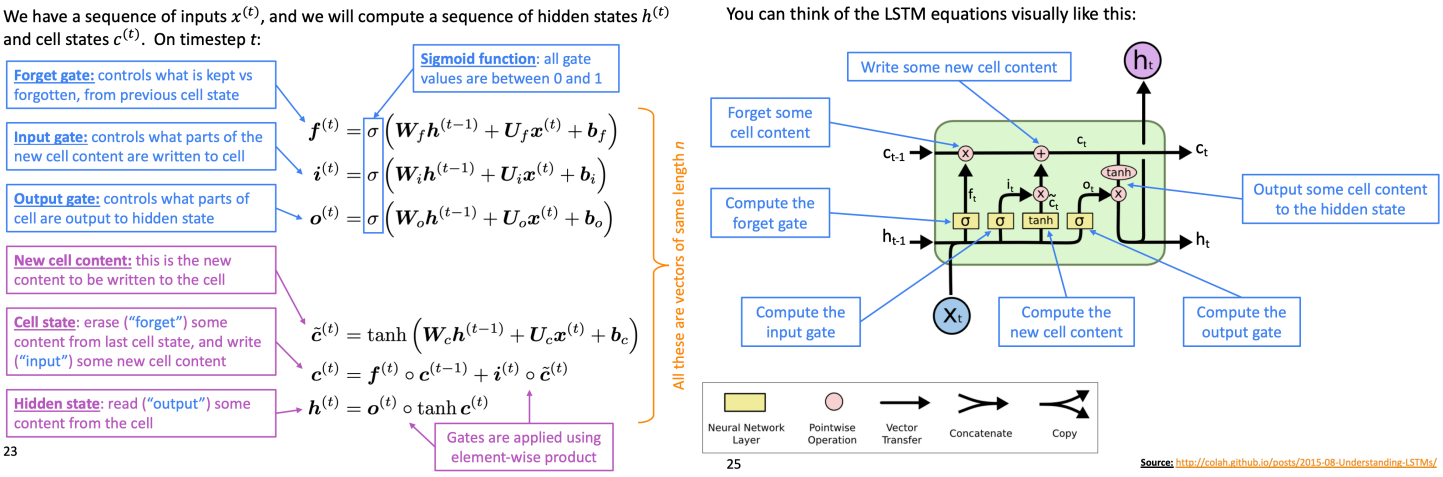
Exploding Gradients

- Why is it a problem
  - This can cause bad updates: we take too large a step and reach a bad parameter configuration (with large loss)
  - In the worst case, this will result in *Inf* or *NaN* in your network (then you have to restart training from an earlier checkpoint)
- Solution
  - Gradient clipping
    - if the norm of the gradient is greater than some threshold, scale it down before applying SGD update
    - Intuition: take a step in the same direction, but a smaller step

---

Long Short-Term Memory (LSTM)

- Overview
  - A type of RNN proposed by *Hochreiter and Schmidhuber* in 1997 as a solution to the vanishing gradients problem.
  - On step t, there is a hidden state $h^{(t)}$ and a cell state $c^{(t)}$
    - Both are vectors length n
    - The cell stores long-term information
    - The LSTM can erase, write and read information from the cell
  - The selection of which information is erased/written/read is controlled by three corresponding gates
    - The gates are also vectors length n
    - On each timestep, each element of the gates can be open(1), closed(0), or somewhere in-between.
    - The gates are dynamic: their value is computed based on the current context
- Architecture

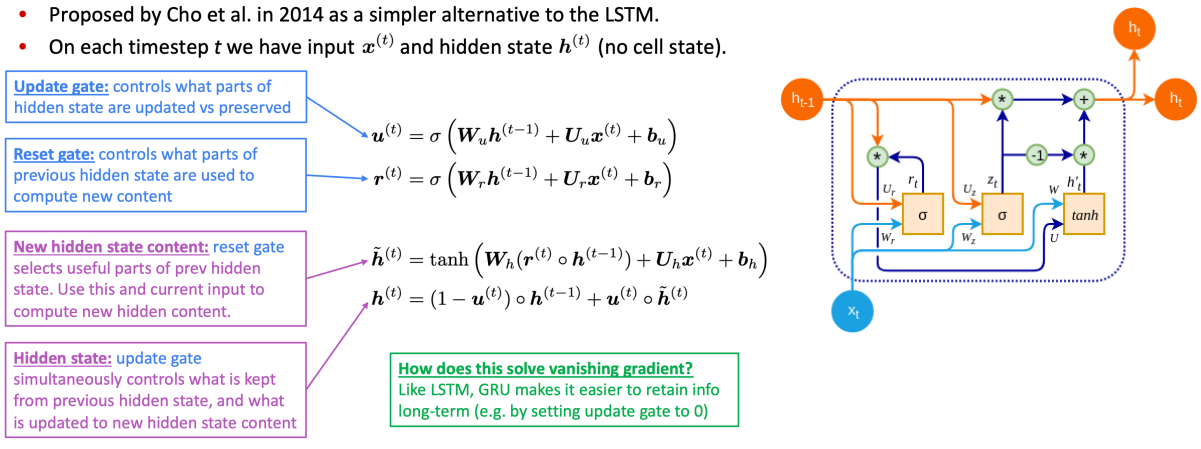


- How does LSTM solve vanishing gradients
  - The LSTM architecture makes it easier for the RNN to preserve information over many timesteps
    - e.g. if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely.
    - By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix Wh that preserves info in hidden state
  - LSTM doesn't guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

---

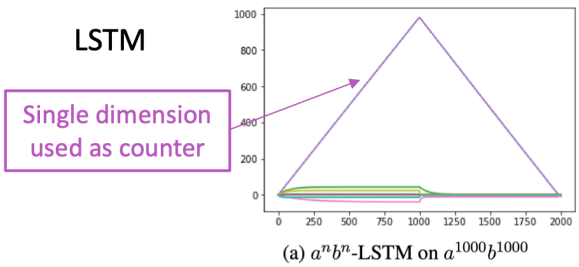
Gated Recurrent Units (GRU)

- Overview

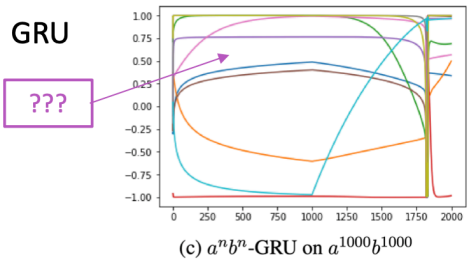


- LSTM vs. GRU

- Researchers have proposed many gated RNN variants, but LSTM and GRU are the most widely-used
- Rule of thumb: LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data); Switch to GRUs for speed and fewer parameters.
- Note: LSTMs can store unboundedly* large values in memory cell dimensions, and relatively easily learn to count. (Unlike GRUs.)



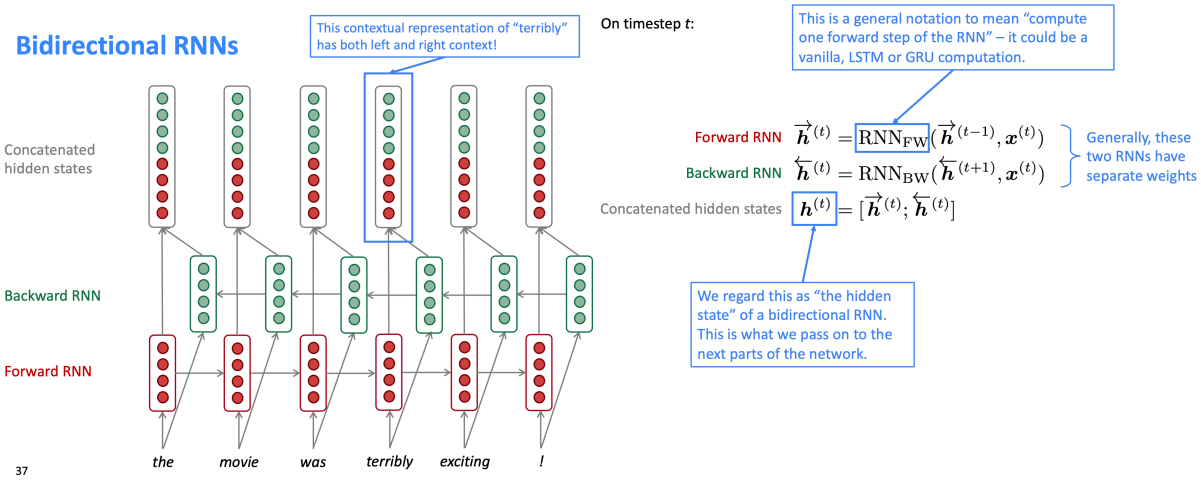
(a) $a^n b^n$-LSTM on $a^{1000} b^{1000}$



(c) $a^n b^n$-GRU on $a^{1000} b^{1000}$

Bidirectional RNNs

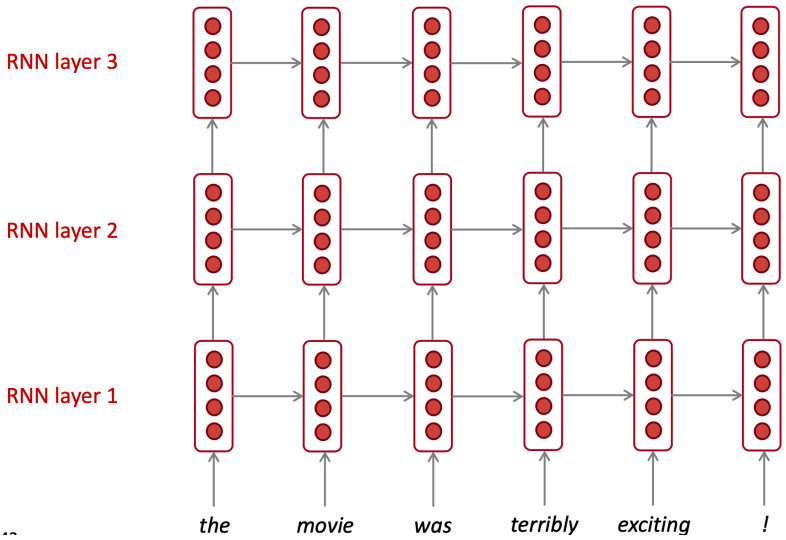
- Architecture



- Notes
  - Bidirectional RNNs are only applicable if you have access to the entire input sequence.
    - They are not applicable to Language Modeling, because in LM you only have left context available.
  - If you do have entire input sequence (e.g. any kind of encoding), bidirectionality is powerful (you should use it by default).

Multi-layer RNNs

- Overview
  - RNNs are already "deep" on one dimension (they unroll over many timesteps)
  - We can also make them "deep" in another dimension by applying multiple RNNs – this is a multi-layer RNN.
  - This allows the network to compute more complex representations
    - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.
  - Multi-layer RNNs are also called stacked RNNs.
- Architecture
  - The hidden states from RNN layer i are the inputs to RNN layer i+1



- In practise
  - High-performing RNNs are often multi-layer (but aren't as deep as convolutional or feed-forward networks)
  - For example: In a 2017 paper, Britz et al find that for Neural Machine Translation, 2 to 4 layers is best for the encoder RNN, and 4 layers is best for the decoder RNN
    - However, skip-connections/dense-connections are needed to train deeper RNNs (e.g. 8 layers)
  - Transformer-based networks (e.g. BERT) are frequently deeper, like 12 or 24 layers.

Lecture 08 - Machine Translation, Seq2Seq and Attention

Machine Translation

- Machine Translation (MT) is the task of translating a sentence x from one language (the source language) to a sentence y in another language (the target language).
- 1950s Early Machine Translation: mostly rule-based
- 1990s-2010s Statistical Machine Translation (SMT)
  - Core idea: Learn a probabilistic model from data
  - Find best English sentence y, given French sentence x such that $\arg\max_y P(y|x)$
  - Applying Bayes Rule to break it down into two components to be learn separately

$$= \text{argmax}_y P(x|y)P(y)$$

Translation Model

Models how words and phrases should be translated (*fidelity*). Learnt from parallel data.

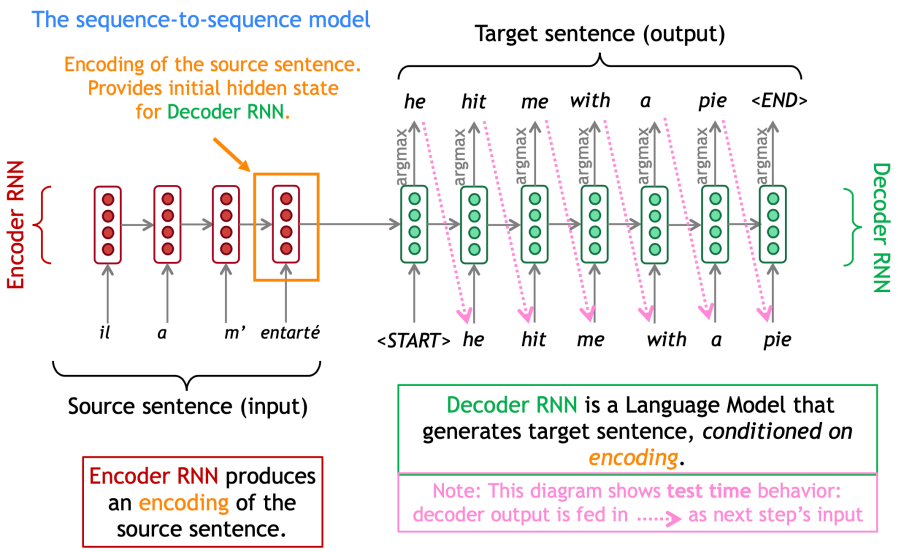
Language Model

Models how to write good English (*fluency*). Learnt from monolingual data.

  - Algorithm
    - Learn alignment for SMT
    - Decoding for SMT (Viterbi)

Neural Machine Translation (seq2seq)

- Neural Machine Translation (NMT) is a way to do Machine Translation with a single neural network
- Since 2014, Neural MT rapidly replaced intricate Statistical MT
- The neural network architecture is called sequence-to-sequence (aka seq2seq) and it involves two RNNs.
- Model Architecture
  - See diagram =>
- Seq2seq Application
  - Machine Translation
  - Summarization



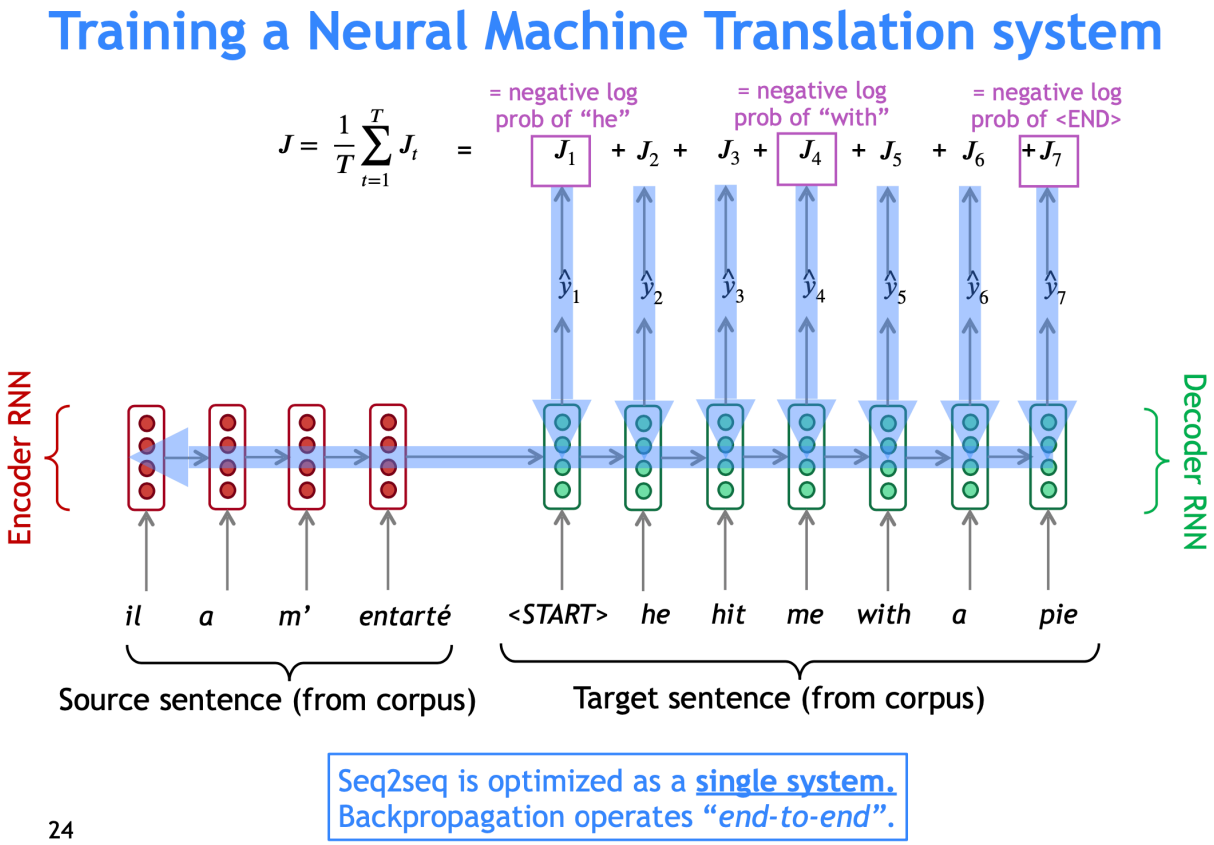
(long text -> short text)

- Dialogue (previous utterance -> next utterance)
- Parsing (input text -> output parse as sequence)
- Code generation (natural language -> python code)

- Seq2seq is a Conditional Language Model
  - Language model because the decoder is predicting the next word of the target sentence y
  - Conditional because its predictions are conditioned on the source sentence x
  - It directly calculates $P(y \ \ x) = P(y_1 \ \ x)P(y_2 \ \ y_1, x)\ldots P(y_T \ \ y1,\cdots, y_{T-1}, x)$

Training a Neural Machine Translation system

- Training



24

- Decoding
  - Greedy decoding: taking argmax on each step of the decoder
    - Problem: no way to undo decision (lack of backtracking), output can be poor (e.g. ungrammatical, unnatural, nonsensical)
  - Exhaustive search: compute all possible sequences y
    - Time complexity is far too expensive

- Beam search: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)
  - k is the beam size (in practice around 5 to 10)
  - A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability:

    $$\text{score}(y_1, \ldots, y_t) = \log P_{LM}(y_1, \ldots, y_t \mid x) = \sum_{i=1}^{t} \log P_{LM}(y_i \mid y_1, \ldots, y_{i-1}, x)$$

  - Stopping criterion: Usually we continue beam search until:
    - We reach timestep T (where T is some pre-defined cutoff), or
    - We have at least n completed hypotheses (where n is pre-defined cutoff)
  - Beam search is not guaranteed to find optimal solution
  - But much more efficient than exhaustive search!
  - What's the effect of changing beam size k?
    - Small k has similar problems to greedy decoding (k=1)
    - Ungrammatical, unnatural, nonsensical, incorrect
  - Larger k means you consider more hypotheses
    - Increasing k reduces some of the problems above
    - Larger k is more computationally expensive
    - But increasing k can introduce other problems:
      - For NMT, increasing k too much decreases BLEU score. This is primarily because large-k beam search produces too-short translations (even with score normalization!)
      - In open-ended tasks like chit-chat dialogue, large k can make output more generic (see picture below)

Evaluation on Machine Translation

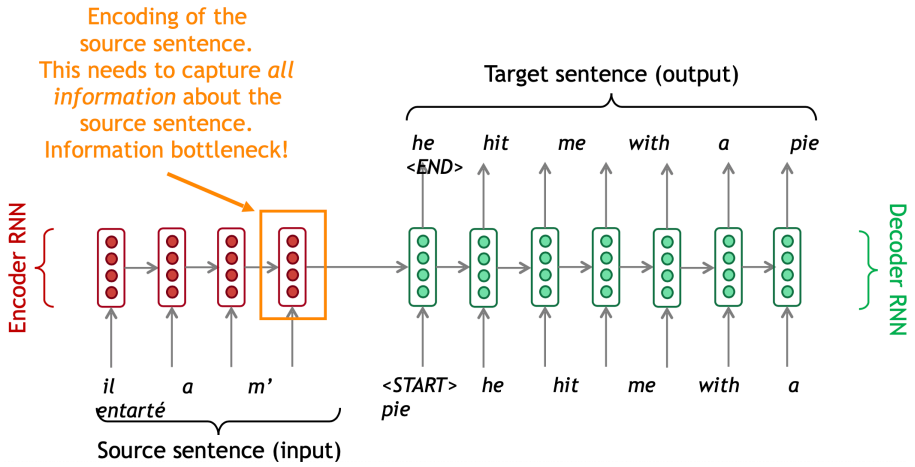
- Advantages of NMT (compared to SMT)
  - Better performance
    - More fluent
    - Better use of context
    - Better use of phrase similarities
  - A single neural network to be optimized end-to-end
    - No subcomponents to be individually optimized

  - Requires much less human engineering effort
    - No feature engineering
    - Same method for all language pairs
- Disadvantages of NMT (compared to SMT)
  - NMT is less interpretable
    - Hard to debug
  - NMT is difficult to control
    - For example, can't easily specify rules or guidelines for translation
    - Safety concerns!
- NMT Problems
  - Out-of-vocabulary words
  - Domain mismatch between train and test data • Maintaining context over longer text
  - Low-resource language pairs
  - Using common sense is still hard
  - Idioms are difficult to translate
  - NMT picks up biases in training data
  - Uninterpretable systems do strange things
- How to evaluate Machine Translation
  - BLEU (Bilingual Evaluation Understudy)
    - BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:
      - n-gram precision (usually for 1, 2, 3 and 4-grams)
      - Plus a penalty for too-short system translations
    - BLEU is useful but imperfect
      - There are many valid ways to translate a sentence
      - So a good translation can get a poor BLEU score because it has low n-gram overlap with the human translation

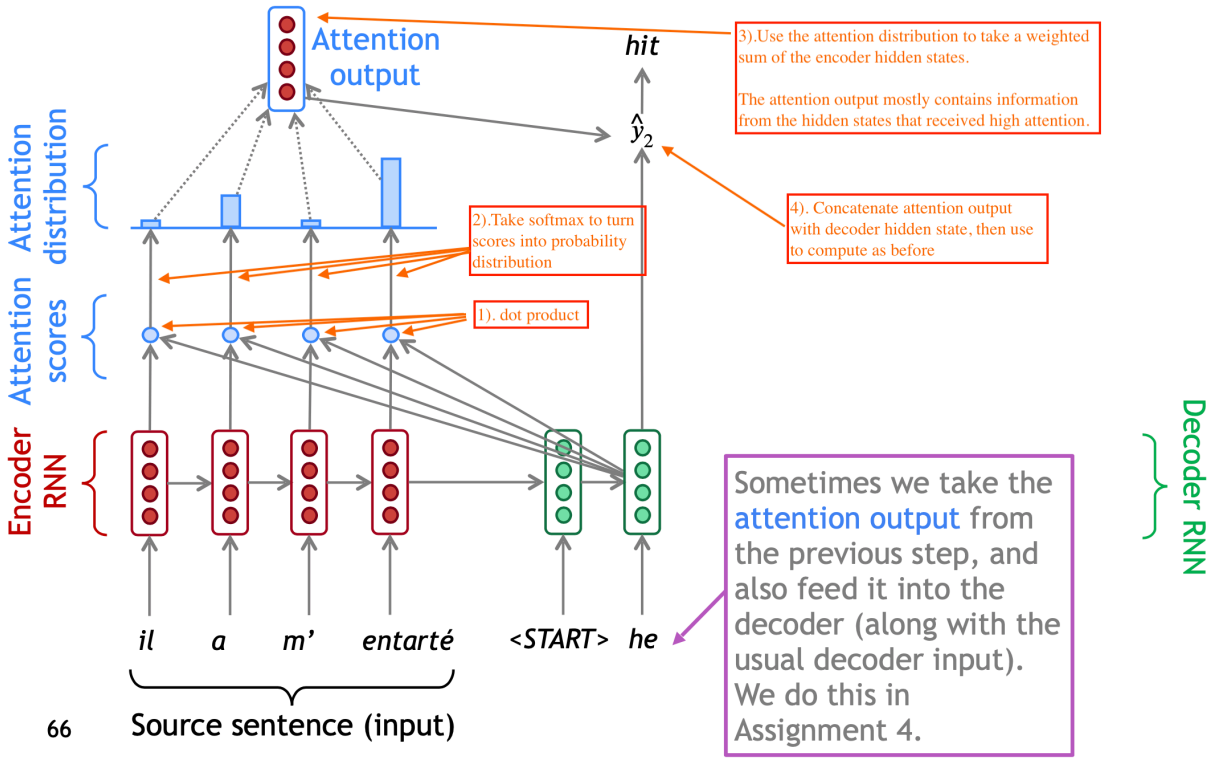
---

Attention

- Motivation: Bottleneck problem of seq2seq model— encoding of the source sentence.



- Attention: provides a solution to the bottleneck problem
- Core idea: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence



- **Attention in equation**
  1. We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
  2. On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$
  3. We get the attention scores e^t for this step: $e_t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$
  4. We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1): $\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$

5. We use α^t to take a weighted sum of the encoder hidden states to get the attention output: $a_t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$
6. Finally we concatenate the attention output a^t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model: $[a_t; s_t] \in \mathbb{R}^{2h}$

- Advantages
  - Attention significantly improves NMT performance
    - It's very useful to allow decoder to focus on certain parts of the source
  - Attention solves the bottleneck problem
    - Attention allows decoder to look directly at source; bypass bottleneck
  - Attention helps with vanishing gradient problem
    - Provides shortcut to faraway states
  - Attention provides some interpretability
    - By inspecting attention distribution, we can see what the decoder was focusing on
    - We get (soft) alignment for free!
    - This is cool because we never explicitly trained an alignment system
    - The network just learned alignment by itself
- General definition of attention
  - Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query.
  - Intuition
    - The weighted sum is a selective summary of the information contained in the values, where the query determines which values to focus on.
    - Attention is a way to obtain a fixed-size representation of an arbitrary set of representations (the values), dependent on some other representation (the query)
- Several attention variants (multiple ways to compute *attention scores*)
  - Basic dot-product attention: $e_i = s^T h_i \in \mathbb{R}$
    - Note: this assumes $d_1 = d_2$
  - Multiplicative attention: $e_i = s^T W h_i \in \mathbb{R}$
    - where $W \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix
  - Additive attention:$e_i = v^T \tanh(W_1 h_i + W_2 s) \in \mathbb{R}$
    - where $W_1 \in \mathbb{R}^{d_3 \times d_1}$, $W_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $v \in \mathbb{R}^{d_3}$ is a weight vector
    - d_3 (the attention dimensionality) is a hyperparameter

Lecture 11 - ConvNets for NLP

From RNNs to CNNs

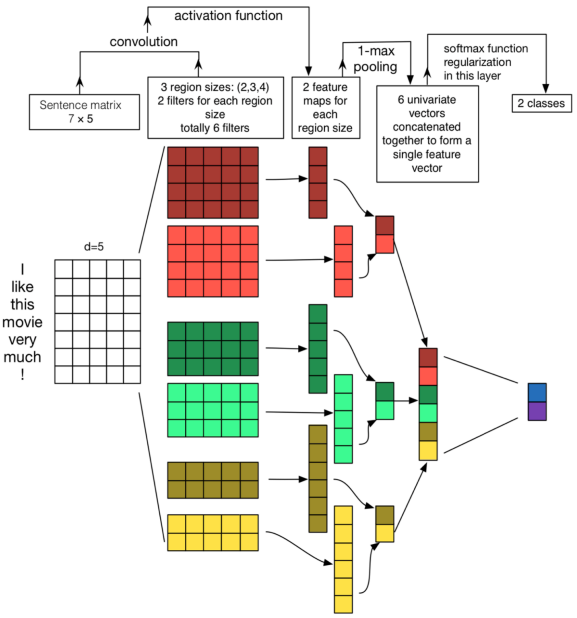
- Motivation
  - Recurrent neural nets cannot capture phrases without prefix context
  - Often capture too much of last words in final vector
  - Eg: softmax is often calculated at the last step
- Main CNN idea:
  - what if we compute vectors for every possible word subsequences of a certain length
  - Regardless of whether phrase is grammatical. Not very linguistically or cognitively plausible.
  - Just group them afterwards

Single Layer CNN for sentence classification

- A simple use of one convolutional layer and pooling
- Terminology:
  - Word vectors: $x_i \in \mathbb{R}^k$
  - Sentence: $x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ (Vectors concatenated)
  - Concatenation of words in range: $x_{i:i+j}$ (Symmetric more common)
  - Convolutional filter: $w \in \mathbb{R}^{hk}$ (over window of h words)
- Filter:
  - Note, filter is a vector!
  - Filter could be size 2, 3 or 4
  - Filter w is applied to all possible windows (concatenated vectors)
  - To compute feature (one channel) for CNN layer: $c_i = f(w^T x_{i:i+h-1} + b)$
  - all possible windows of length h: $\{x_{1:h}, x_{2:h+1}, \ldots, x_{n-h+1:n}\}$
  - Result is a feature map: $c = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
- Pooling
  - Pooling: max-over-time pooling layer
  - Idea: capture most important activation (maximum over time)

- From feature map $c = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$, pooled a single number $\hat{c} = \max\{c\}$

- Channels
  - Use multiple filter weight w (i.e. multiple channels)
  - Useful to have different window sizes h
  - So we could have some filters that look at unigram, bigram, tri-grams, 4-grams, etc.
- Classification after one CNN layer
  - First one convolution, followed by one max-pooling
    - To obtain final feature vector: $z = [\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_m]$ (assuming m filters w)
    - Used 100 feature maps each of sizes 3, 4, 5
  - Simple final softmax layer: $y = \text{softmax}(W^{(s)}z + b)$



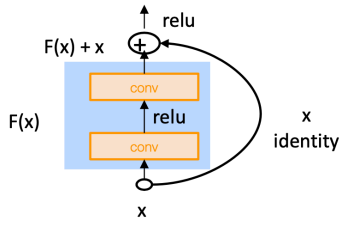
- Regularization
  - Use **Dropout**: Create masking vector r of Bernoulli random variables with probability p (a hyperparameter) of being 1
  - Delete features during training:
    $$y = softmax\left(W^{(S)}(r \circ z) + b\right)$$
  - Reasoning: Prevents co-adaptation (overfitting to seeing specific feature constellations) (Srivastava, Hinton, et al. 2014)
  - At test time, no dropout, scale final vector by probability p
    $$\hat{W}^{(S)} = pW^{(S)}$$
  - **Also:** Constrain l_2 norms of weight vectors of each class (row in softmax weight $W^{(S)}$) to fixed number s (also a hyperparameter)
  - If $\|W_{c\cdot}^{(S)}\| > s$, then rescale it so that: $\|W_{c\cdot}^{(S)}\| = s$
  - Not very common

Model comparison: Our growing toolkit

- **Bag of vectors**: Surprisingly good baseline for simple classification problems. Especially if followed by a few ReLU layers! (See paper: Deep Averaging Networks)
- **Window Model**: Good for single word classification for problems that do not need wide context. E.g., POS, NER
- **CNNs**: good for classification, need zero padding for shorter phrases, somewhat implausible/hard to interpret, easy to parallelize on GPUs. Efficient and versatile
- **Recurrent Neural Networks**: Cognitively plausible (reading from left to right), not best for classification (if just use last state), much slower than CNNs, good for sequence tagging and classification, great for language models, can be amazing with attention mechanisms

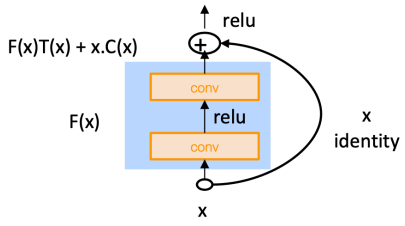
CNN potpourri

- Gated Units used vertically
  - The gating/skipping that we saw in LSTMs and GRUs is a general idea, which is now used in a whole bunch of places
  - Indeed the key idea – summing candidate update with shortcut connection – is needed for very deep networks to work



Residual block
(He et al. ECCV 2016)

Note: pad **x** for conv so same size when add them



Highway block
(Srivistava et al. NeurIPS 2015)

Note: can set C(x) = (1 – T(x)) more like GRU

- Batch Normalization (BatchNorm)
  - Often used in CNNs
  - Transform the convolution output of a batch by scaling the activations to have zero mean and unit variance
  - Use of BatchNorm makes models much less sensitive to parameter initialization, since outputs are automatically rescaled
  - It also tends to make tuning of learning rates simpler
  - Related but different: LayerNorm, standard in Transformers
- Size 1 Convolutions
  - Size 1 convolutions (“1x1”), a.k.a. Network-in-network (NiN) connections, are convolutional kernels with kernel_size=1
  - A size 1 convolution gives you a fully connected linear layer across channels!
  - It can be used to map from many channels to fewer channels
  - Size 1 convolutions add additional neural network layers with very few additional parameters
    - Unlike Fully Connected (FC) layers which add a lot of parameters

Lecture 12 - Subword Models: Information from parts of words

Linguistic

- Human language sounds
  - Phonetics (语音学): Phonetics is the sound stream – uncontroversial “physics” (语音指声音流，毫无争议的物理特征)
  - Phonology (音系学) posits a small set or sets of distinctive, categorical units: phonemes or distinctive features (音系学预设了一系列唯一的，特定的单元：音位或者特定特征)
- Morphology (词法学)
  - Traditionally, we have morphemes (词素) as smallest semantic unit (语义单元)
  - $[[un[[fortun(e)]_{root}ate]_{stem}]_{stem}ly]_{word}$

Models below the word level

- Motivation
  - Need to handle large, open vocabulary
    - Rich morphology (丰富的形态会导致过大的词向量空间和OOV问题)
    - Transliteration (音译问题，比如人名地名的音译，Christopher -> 克里斯托弗)
    - Informal spelling
- Character-Level Models
  - Word embeddings can be composed from character embeddings
    - Generates embeddings for unknown words
    - Similar spellings share similar embeddings
    - Solves OOV problem
  - Connected language can be processed as characters

Sub-word models

- Motivations:

  - 可以处理OOV问题
  - 可以学习词缀之间的关系，old->older->oldest与smart->smarter->smartest
  - Character-level models 作为解决OOV问题的方法粒度太细
  - sub-word粒度介于character和word之间，能较好的平衡OOV问题
- Two trends
  - Same architecture as for word-level model, but use smaller units: “word pieces”
    - BPE
    - WordPiece / SetencePiece
  - Hybrid architectures: main model has words, something else for characters
- Byte Pair Encoding (BPE)
  - Originally a compression algorithm: most frequent byte pair -> new byte (replace bytes with character ngrams)
  - A word segmentation algorithm: though done as bottom up clustering
  - Algorithm
    1. Start with a unigram vocabulary of all (Unicode) characters in data (准备语料，生成字节级词表，确定目标词表大小)
    2. Most frequent ngram pairs -> a new ngram (统计每一个连续字节对的出现频率，选择最高频者合并为新的subword并加入词表)
    3. Have a target vocabulary size and stop when you reach it (重复第二步直到达到目标词表大小或下一个最高频率为1)
  - Do deterministic longest piece segmentation of words
  - Segmentation is only within words identified by some prior tokenizer (commonly Moses tokenizer for MT)
  - Automatically decides vocabulary for system
- WordPiece / SentencePiece Model
  - Google NMT (GNMT) uses a variant of this: V1-WordPiece, V2-SentencePiece
  - Rather than char n-gram count, uses a greedy approximation to maximizing language model log likelihood to choose the pieces
    - Add n-gram that maximally reduces perplexity
  - WordPiece model tokenizes inside words
  - SentencePiece model works from raw text (no tokenization beforehand)
    - Whitespace is retained as special token (_) and grouped normally

    - You can reverse things at end by joining pieces and recoding them to spaces

---

Character-level to build word-level

- Learning Character-level Representations for Part-of-Speech Tagging (*Dos Santos and Zadrozny 2014*)
  - Convolution over characters to generate word embeddings
  - Fixed window of word embeddings used for PoS tagging
- Character-based LSTM to build word representations
- Character-Aware Neural Language Models
  - Motivation
    - Derive a powerful, robust language model effective across a variety of languages.
    - Encode subword relatedness: eventful, eventfully, uneventful...
    - Address rare-word problem of prior models.
    - Obtain comparable expressivity with fewer parameters.
  - Take-aways
    - Paper questioned the necessity of using word embeddings as inputs for neural language modeling.
    - CNNs + Highway Network over characters can extract rich semantic and structural information.
    - Key thinking: you can compose “building blocks” to obtain nuanced and powerful models!

Lecture 13 - Contextual Word Representations and Pre-training

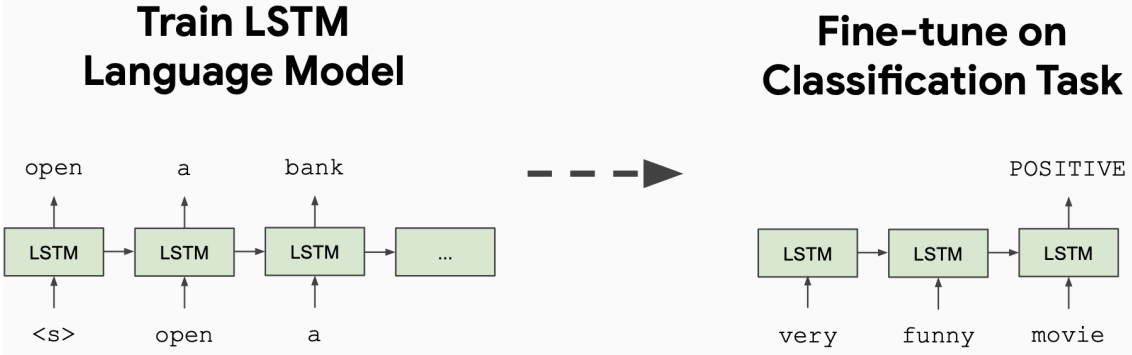
---

Word Representation

- Problems with traditional representations (word2vec, GloVe, fastText)
  - Always the same representation for a word type regardless of the context in which a word token occurs
    - We might want very fine-grained word sense disambiguation
  - We just have one representation for a word, but words have different aspects, including semantics, syntactic behavior, and register/connotations

- Solution
  - In an NLM, we immediately stuck word vectors (perhaps only trained on the corpus) through LSTM layers
  - Those LSTM layers are trained to predict the next word
  - But those language models are producing context-specific word representations at each position!

1. *Semi-Supervised Sequence Learning, Google, 2015*

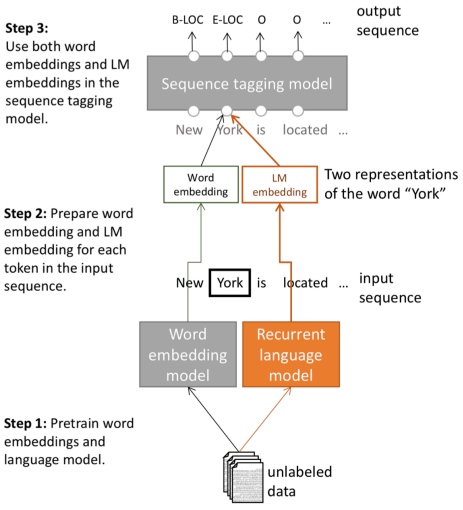


Semi-Supervised 2015

2. *ELMo: Deep Contextual Word Embeddings, AI2 & University of Washington, 2017*
3. *Improving Language Understanding by Generative Pre-Training, OpenAI, 2018*

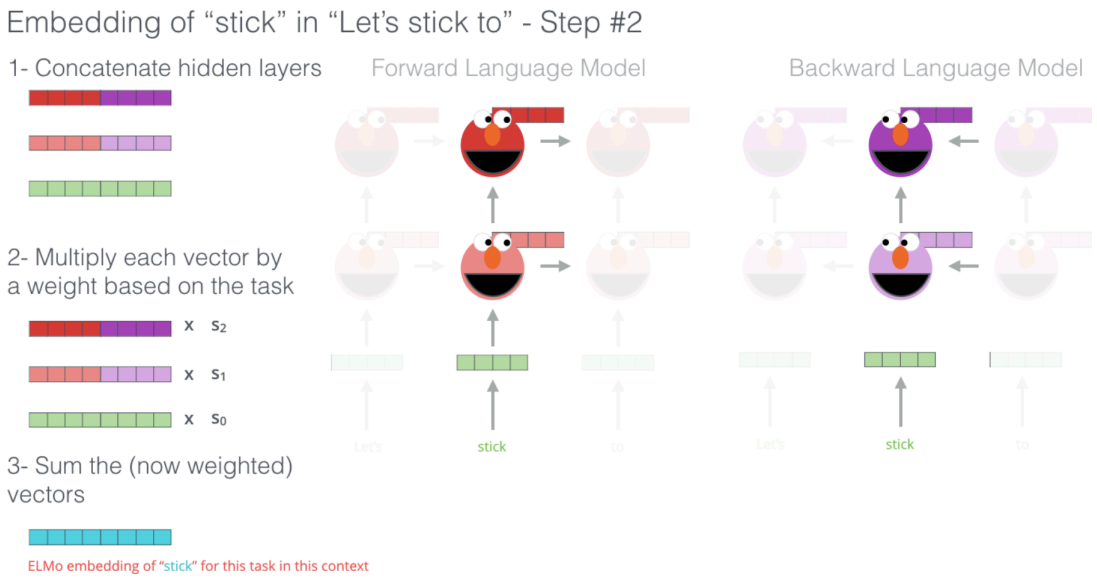
Pre-ELMo and ELMo

- TagLM (*Peters et al. 2017*) - "Pre-ELMo"
  - Why don't we do semi-supervised approach where we train NLM sequence model on large unlabeled corpus, rather than just word vectors and use as pretraining for sequence model
  - Idea: Want meaning of word in context, but standardly learn task RNN only on small task-labeled data (e.g., NER)
  - Language model observations
    - An LM trained on supervised data does not help
    - Having a bidirectional LM helps over only forward, by about 0.2



- Having a huge LM design (ppl 30) helps over a smaller model (ppl 48) by about 0.3

- *Peters et al. (2018)*: Embeddings from Language Models - "ELMo"
  - Initial breakout version of word token vectors or contextual word vectors
  - Learn word token vectors using long contexts not context windows (here, whole sentence, could be longer)
  - Learn a deep Bi-NLM and use all its layers in prediction (innovation that improves on just using top layer of LSTM stack)

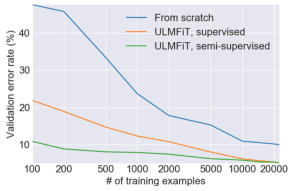


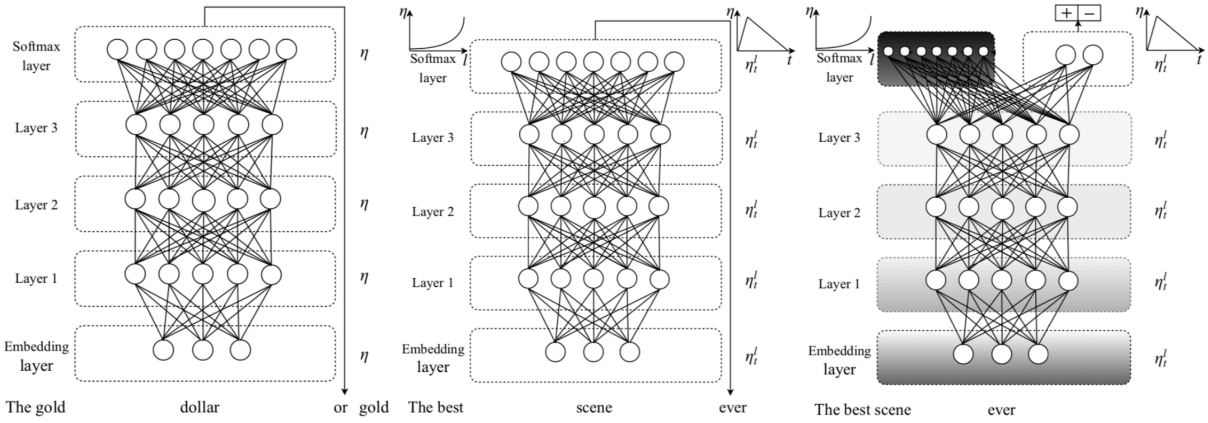
  - Aim at performant but not overly large LM
  - ELMo learns task-specific combo of biLM layer representations:
    - $R_k = \{x_k^{LM}, \vec{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} \;\; j = 1,\ldots,L\} = \{h_{k,j}^{LM} \;\; j = 0,\ldots,L\}$
    - $ELMo_k^{task} = E(R_k, \Theta^{task}) = \gamma_{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}^{LM}$
    - γ^{task} scales overall usefulness of ELMo to task;
    - s^{task} are softmax-normalized mixture model weights

- ELMo: Use with a task
  - First run biLM to get representations for each word
  - Then let (whatever) end-task model use them
    - Freeze weights of ELMo for purposes of supervised model
    - Concatenate ELMo weights into task-specific model
      - Details depend on task
      - Concatenating into intermediate layer as for TagLM is typical

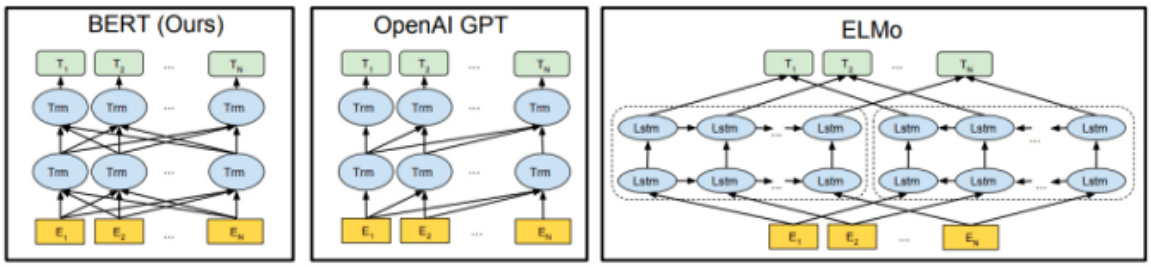
  - Can provide ELMo representations again when producing outputs, as in a question answering system
- The two biLSTM NLM layers have differentiated uses/meanings
  - Lower layer is better for lower-level syntax, etc.
    - Part-of-speech tagging, syntactic dependencies, NER
  - Higher layer is better for higher-level semantics
    - Sentiment, Semantic role labeling, question answering, SNLI

- *Howard and Ruder (2018) Universal Language Model Fine-tuning for Text Classification* - "ULMfit"
  - Train LM on big general domain corpus (use biLM)
  - Tune LM on target task data
  - Fine-tune as classifier on target task





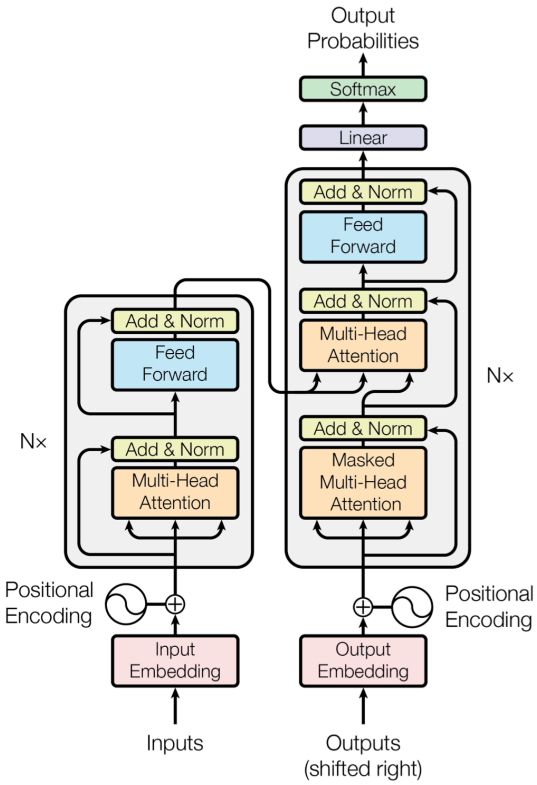
(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning



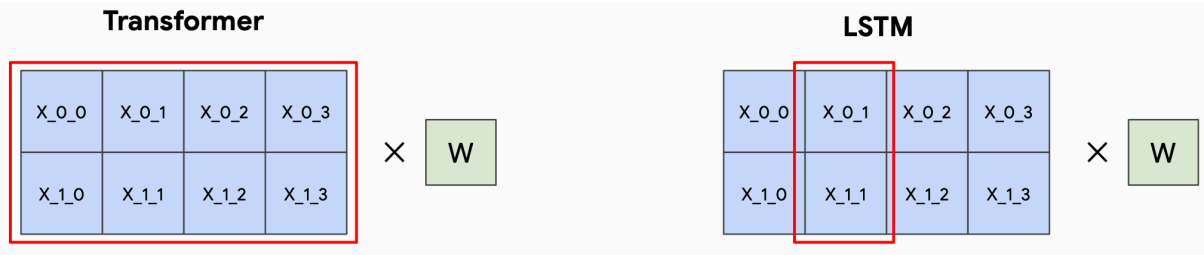
Lecture 14 - Transformer and BERT

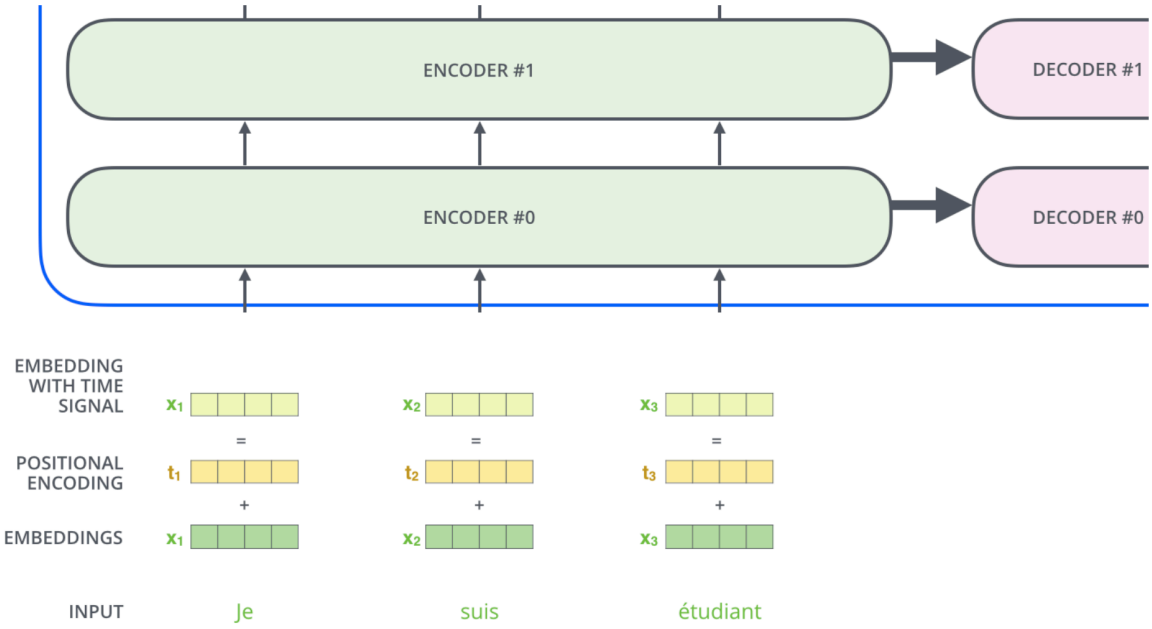
Transformer Overview

- Motivation
  - Learn context representations of variable length data
    - RNN: inhibits parallelization, no explicit modeling of long and short range dependencies, no hierarchy
    - CNN: trivial to parallelize (per layer), long-distance dependencies require many layers
  - Why not use attention for representations?
- *Attention is all you need. 2017*
  - **Non-recurrent** sequence-to-sequence encoder-decoder model
  - Task: machine translation with parallel corpus
  - Predict each translated word
  - Final cost/error function is standard cross-entropy error on top of a softmax classifier



- Transformer Encoder Intuition
  - Multi-headed Self-Attention: Models context
  - Feed-forward layers: Computes non-linear hierarchical features
  - Layer norm and residuals: Makes training deep networks healthy
  - Positional embeddings: Allows model to learn relative positioning
- Empirical advantages of Transformer vs. LSTM
  - Self-Attention == **no locality bias**
    - Long-distance context has “equal opportunity”
  - Single multiplication per layer == **efficiency on TPU**
    - Effective batch size is number of words, not sequences





Positional embedding

Transformer Components

- Embedding
  - The embedding only happens in the bottom-most encoder.
  - The abstraction that is common to all the encoders is that they receive a list of vectors each of the size $d_{embedding}$. In the bottom encoder that would be the word embeddings, but in other encoders, it would be the output of the encoder that's directly below.
  - The size of this list is hyperparameter we can set – basically it would be the length of the longest sentence in our training dataset.
  - Add positional embeddings to each of input embedding.
- Multi-Head Self-Attention
  - Self-Attention:
    - Self-attention is the method the Transformer uses to bake the "understanding" of other relevant words into the one we're currently processing (helps the encoder look at other words in the input sentence when it encodes a specific word)
  - Multi-Headed
    - It expands the model's ability to focus on different positions.
    - It gives the attention layer multiple "representation subspaces".

- Positional Encoding
  - The intuition here is that adding these values to the embeddings provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention.
- Residuals
  - Make deep network training healthy
  - Residuals carry positional information to higher layers, among other information
- Decoder side
  - One more sub-layer in the middle compared to encoder side: Encoder-Decoder Attention sub-layer which create its Query matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack.
  - In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to *-inf*) before the softmax step in the self-attention calculation.
- Default configuration: 6 encoder layers, 512 hidden units, and 8 attention heads
- QA
  - In self-attention sub-layer, why use dimension=64 for Q, K, V whereas dimension for embedding=512?
    - They don't HAVE to be smaller, this is an architecture choice to make the computation of multi-headed attention (mostly) constant.
  - In self-attention sub-layer, why divide attention scores by 8 ($\sqrt{d_{Q,K,V}} = \sqrt{64}$)?
    - This leads to having more stable gradients. Scaling factor to make sure score don't blow up.
  - In self-attention sub-layer, why add positional embeddings to word embeddings?
    - The intuition here is that adding these values to the embeddings provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention.

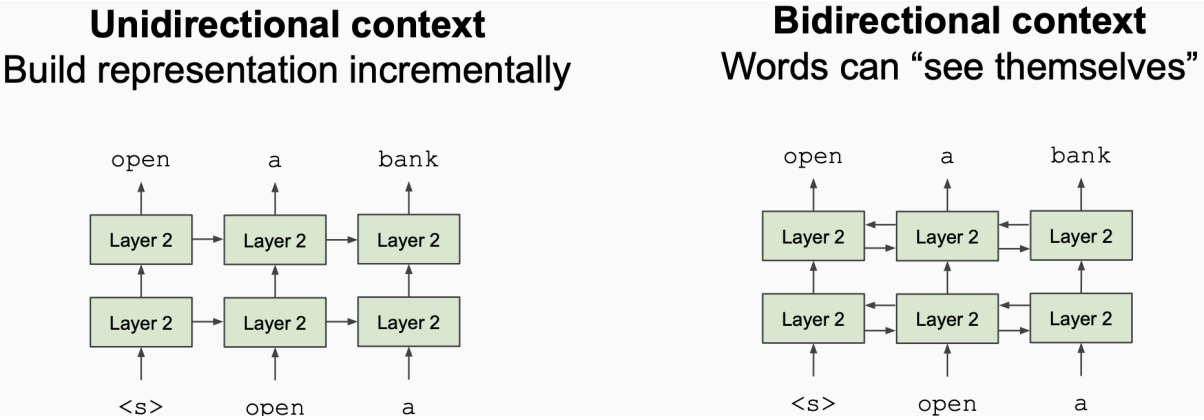
BERT

Problems with previous method

- History: two existing strategies for applying pre-trained language representations to downstream tasks
  - *Feature-based:* ELMo, uses task-specific architectures that include the pre-trained representations as additional features.

- *Fine-tuning*: GPT (Generative Pre-trained Transformer), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning all pre-trained parameters.
- They both share the **same objective function** during pre-training, where they use **unidirectional** language models to learn general language representations

- Problems: Language models only use **left** context or **right** context, but language understanding is **bidirectional**.
- Why are LMs are unidirectional?
  - Directionality is needed to generate a well-formed probability distribution. - (We don't care about this.)
  - Words can "see themselves" in a bidirectional encoder.



- BERT contribution: further generalizing unsupervised pre-training to deep ***bidirectional*** architectures, allowing the same pre-trained model to successfully tackle a broad set of NLP tasks

Pre-training Task #1: Masked LM (MLM, Exploit bidirectional context)

- Mask out k% of the input words/tokens, and then predict the masked words - We always use k=15
  - Too little masking: Too expensive to train
  - Too much masking: Not enough context
- Problem: Mask token never seen at fine-tuning
  - Solution: 15% of the words to predict, but don't replace with [MASK] 100% of the time. Instead:
    - 80% of the time, replace with [MASK]: went to the store → went to the [MASK]
    - 10% of the time, replace random word: went to the store → went to the running
    - 10% of the time, keep same: went to the store → went to the store
    - Then use final hidden vector of it to predict the original token with cross entropy loss

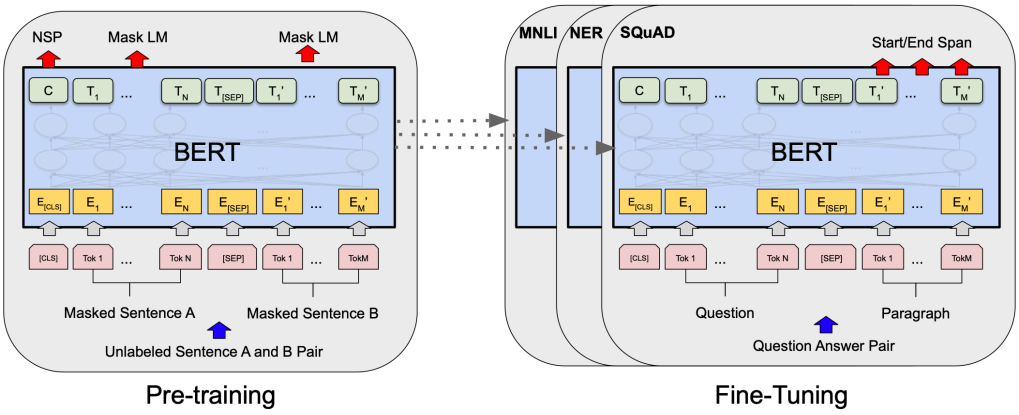


Figure 14.1: Overall pre-training and fine-tuning procedures for BERT.

Pre-training Task #2: Next Sentence Prediction (NSP)

- Motivation: many downstream tasks such as QA and NLI are based on understanding the *relationship* between two sentences, which is not directly captured by language modeling.
- To learn relationships between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence (final hidden state *C* in Figure 14.1)
  - 50% of the time, sentence B is the actual next sentence that follows A (labeled *IsNext*)
  - 50% of the time, sentence B is a random sentence from the corpus (labeled *NotNext*)

Input Representation

- Use 30,000 WordPiece vocabulary on input.
- The first token of every sequence is always a special classification token [CLS]. The final hidden state corresponding to this token (*C* in Figure 14.1) is used as the aggregate sequence representation for classification tasks.
- Each token is sum of three embeddings
- Single sequence is much more efficient.

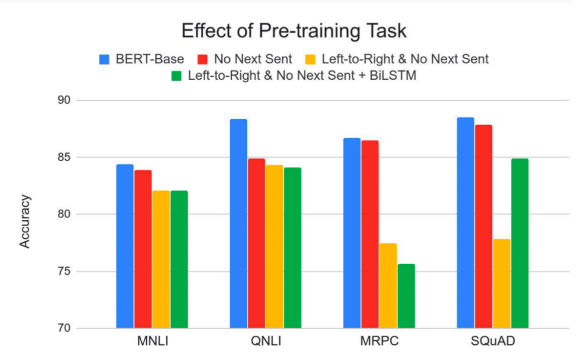
Model Details

- Architecture: BERT is basically a trained Transformer Encoder stack.
- Data: Wikipedia (2.5B words) + BookCorpus (800M words)
- Batch Size: 131,072 words (1024 sequences * 128 length or 256 sequences * 512 length)
- Training Time: 1M steps (~40 epochs)
- Optimizer: AdamW, 1e-4 learning rate, linear decay
- BERT-Base: 12-layer, 768-hidden, 12-head

- BERT-Large: 24-layer, 1024-hidden, 16-head. Trained on 4x4 or 8x8 TPU slice for 4 days

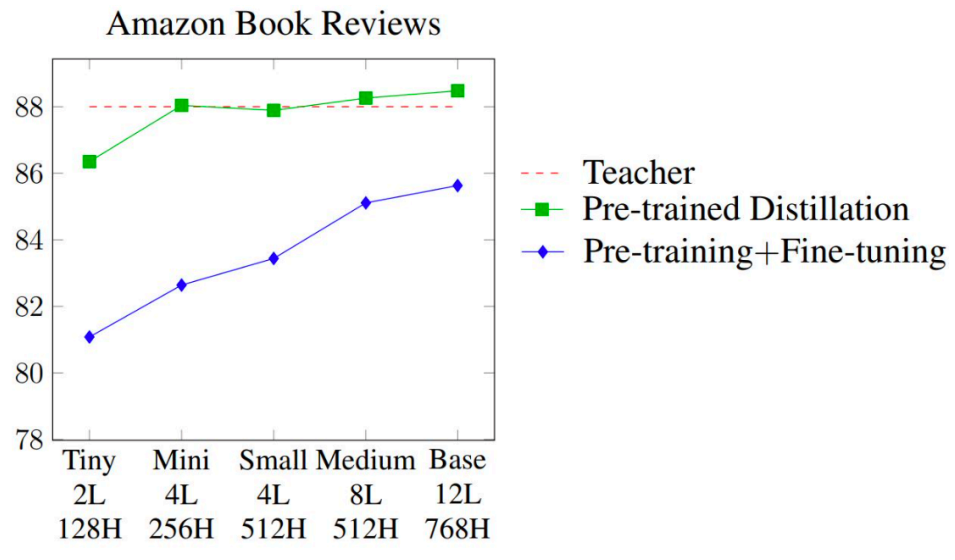
Effect

- Effect of Pre-Training task
  - Masked LM (compared to left-to-right LM) is very important on some tasks, Next Sentence Prediction is important on other tasks.
  - Left-to-right model does very poorly on word-level task (SQuAD), although this is mitigated by BiLSTM
- Effect of Directionality and Training Time
  - Masked LM takes slightly longer to converge because we only predict 15% instead of 100%
  - But absolute results are much better almost immediately
- Effect of Model Size
  - Going from 110M -> 340M params helps even on datasets with 3,600 labeled examples
  - Improvements have not asymptoted



Distillation

- **Problem**: BERT and other pre-trained language models are extremely large and expensive, How are companies applying them to low-latency production services?
- Answer: Distillation (a.k.a., model compression)
- Simple technique:
  - Train “Teacher”: Use SOTA pre-training + fine-tuning technique to train model with maximum accuracy
  - Label a large amount of unlabeled input examples with Teacher
  - Train “Student”: Much smaller model (e.g., 50x smaller) which is trained to mimic Teacher output
  - Student objective is typically Mean Square Error or Cross Entropy
- Distillation works much better than pre-training + fine-tuning with smaller model
- Why does distillation works so well? A hypothesis
  - Language modeling is the “ultimate” NLP task in many ways
    - I.e., a perfect language model is also a perfect question answering/entailment/ sentiment analysis model



- Training a massive language model learns millions of latent features which are useful for these other NLP tasks
- Fine-tuning mostly just picks up and tweaks these existing latent features
- This requires an oversized model, because only a subset of the features are useful for any given task
- Distillation allows the model to only focus on those features
- Supporting evidence: Simple self-distillation (distilling a smaller BERT model) doesn't work

---

Post-BERT Pre-Training Advancements

- RoBERTa
  - A Robustly Optimized BERT Pretraining Approach (*Liu et al, University of Washington and Facebook, 2019*)
  - Trained BERT for more epochs and/or on more data
    - Showed that more epochs alone helps, even on same data
    - More data also helps
- XLNet
  - Generalized Autoregressive Pretraining for Language Understanding (*Yang et al, CMU and Google, 2019*)
  - Innovation #1: Relative position embeddings
    - Sentence: *John ate a hot dog*

    - Absolute attention: "How much should *dog* attend to *hot*(in any position), and how much should *dog* in position 4 attend to the word in position 3? (Or 508 attend to 507, ...)"
    - Relative attention: "How much should *dog* attend to *hot* (in any position) and how much should *dog* attend to the previous word?"
  - Innovation #2: Permutation Language Modeling
    - In a left-to-right language model, every word is predicted based on all of the words to its left
    - Instead: Randomly permute the order for every training sentence
    - Equivalent to masking, but many more predictions per sentence
    - Can be done efficiently with Transformers
  - Also used more data and bigger models, but showed that innovations improved on BERT even with same data and model size
- ALBERT
  - A Lite BERT for Self-supervised Learning of Language Representations (*Lan et al, Google and TTI Chicago, 2019*)
  - Innovation #1: Factorized embedding parameterization
    - Use small embedding size (e.g., 128) and then project it to Transformer hidden size (e.g., 1024) with parameter matrix
  - Innovation #2: Cross-layer parameter sharing
    - Share all parameters between Transformer layers
  - ALBERT is light in terms of parameters, not speed

Lecture 15 - Natural Language Generation

What is NLG

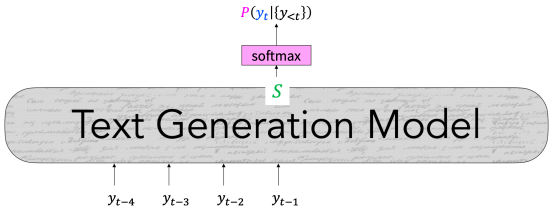
- NLG refers to any setting in which we generate new text
- NLG is a subcomponent of: Machine Translation, (Abstractive) Summarization, Dialogue (chit-chat and task-based), Creative writing (story-telling, poetry-generation), Freeform Question Answering (i.e. answer is generated, not extracted from text or knowledge base), image captioning

Formalizing NLP: a simple model and training algorithm

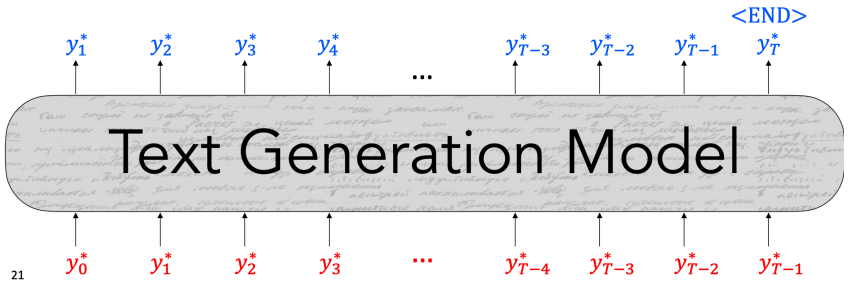
- In autoregressive text generation models, at each time step t, our model takes in a sequence of tokens of text as input $\{y\}_{<t}$ and outputs a new token, $\hat{y}_t$

  1. At each time step t, our model computes a vector of scores for each token in our vocabulary, $S \in \mathbb{R}^V$: $S = f(\{y_{<t}\}, \theta)$
     ($f(\cdot)$ is your model)
  2. Then, we compute a probability distribution P over $w \in V$ using these scores:

$$P(y_t = w \mid \{y_{<t}\}) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$



- At inference time, our decoding algorithm defines a function to select a token from this distribution: $\hat{y}_t = g(P(y_t \mid \{y_{<t}\}))$ ($g(\cdot)$ is your decoding algorithm)
- We train the model to minimize the negative loglikelihood of predicting the next token in the sequence: $L_t = -\log P(y_t^* \mid \{y^*\}_{<t})$ (sum L_t for the entire sequence)
  - Maximum Likelihood Training (i.e. teacher forcing): $L = -\sum_{t=1}^{T} \log P(y_t^* \mid \{y^*\}_{<t})$

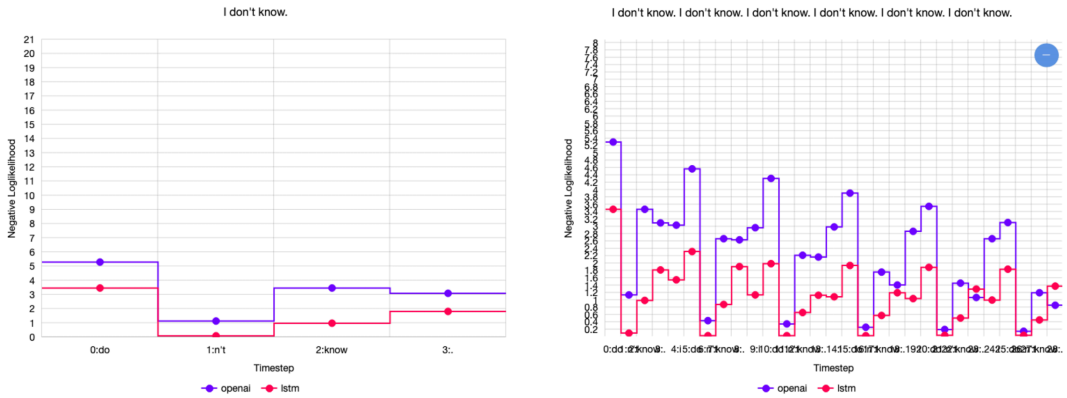


21

Decoding from NLG models

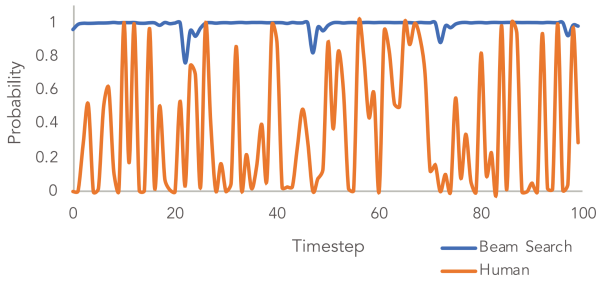
Greedy methods

- Argmax Decoding
  - On each step, take the most probable word, $\hat{y}_t = \arg\max_{w \in V} P(y_t = w \mid y_{<t})$
- Beam search
  - See lecture 8
  - Also a greedy algorithm, but with wider search over candidates
- Greedy methods get repetitive
  - Why:
  - Solution:



- Simple option: Heuristic: Don't repeat n-grams
- More complex:
  - Minimize embedding distance between consecutive sentences (Celikyilmaz et al., 2018): Doesn't help with intra-sentence repetition
  - Coverage loss (See et al., 2017): Prevents attention mechanism from attending to the same words
  - Unlikelihood objective (Welleck et al., 2020): Penalize generation of already-seen tokens

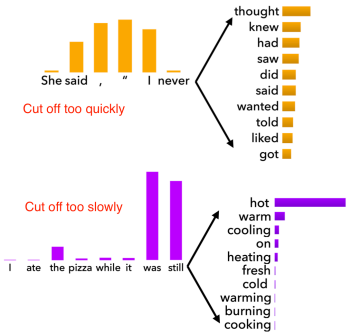
- Greedy methods are not reasonable



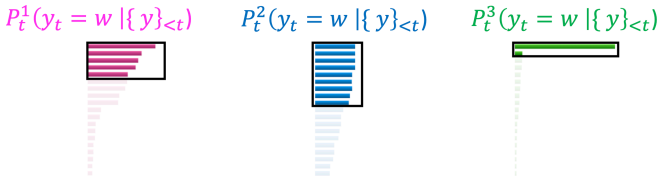
Sampling-based decoding

- Pure sampling
  - On each step t, randomly sample from the probability distribution P_t to obtain your next word.
  - Like greedy decoding, but sample instead of argmax
  - Issue: Many tokens are probably irrelevant in the current context, etc.
- Top-k sampling *
  - Randomly sample from P_t restricted to just the top-k most probable words
  - n=1 is greedy search, n=V is pure sampling, common values are 5, 10, 20

- Increase n to get more diverse/risky output
- Decrease n to get more generic/safe output
- Issue: see example in the right



- Top-p sampling
  - Randomly sample from the tokens in the top-p cumulative probability mass.
  - Varies k depending on the uniformity of P_t
  - This way you get a bigger sample when probability mass is spread
  - Seems like it may be even better



  - Both of these are more efficient than beam search - no multiple hypotheses

- Softmax temperature
  - Apply a temperature hyperparameter τ to the softmax to rebalance P_t:

$$P_t(w) = \frac{\exp(s_w/\tau)}{\sum_{w' \in V} \exp(s_{w'}/\tau)}$$

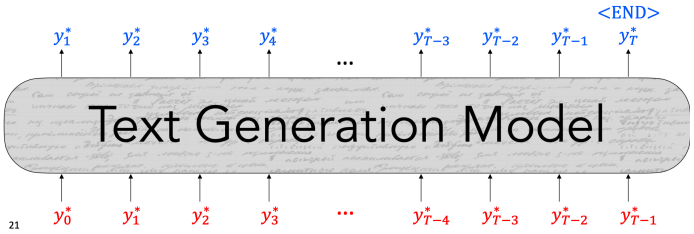
  - Raise the temperature τ: P_t becomes more uniform
    - Thus more diverse output (probability is spread around vocal)
  - Lower the temperature τ: P_t becomes more spiky
    - Thus less diverse output (probability is concentrated on top words)
  - Note: softmax temperature is not a decoding algorithm! It's a technique you can apply at test time, in conjunction with a decoding algorithm (such as beam search or sampling)

- Takeaways
  - **Greedy decoding** is a simple method; gives low quality output
  - **Beam search** (especially with high beam size) searches for high-probability output
    - Delivers better quality than greedy, but if beam size is too high, can return high-probability but unsuitable output (e.g. generic, short)
  - **Sampling methods** are a way to get more diversity and randomness
    - Good for open-ended / creative generation (poetry, stories)

    - Top-n/p sampling allows you to control diversity
  - Softmax temperature is another way to control diversity
    - It's not a decoding algorithm! It's a technique that can be applied alongside any decoding algorithm.
  - Decoding is still a challenging problem in NLG
  - Human language distribution is noisy and doesn't reflect simple properties (i.e. probability maximization)
  - Different decoding algorithms can allow us to inject biases that encourage different properties of coherent NLP
  - Some of the most impactful advances in NLG of the last few years have come from simple, but effective, modifications to decoding algorithms
  - A lot more work to be done!
- Improving decoding: re-balancing distributions

Training NLG models

- Maximum Likelihood Training (i.e. teacher forcing): Trained to generate the minimize the negative loglikelihood of the next token y_t^* given the preceding tokens in the sequence $\{y^*\}_{<t}$: $L = -\sum_{t=1}^{T} \log P(y_t^* | \{y^*\}_{<t})$



- Diversity Issues
  - Maximum Likelihood Estimation discourages diverse text generation
- Unlikelihood Training
  - Given a set of undesired tokens ϵ, lower their likelihood in context: $L_{UL}^t = -\sum_{y_{neg} \in \epsilon} \log(1 - P(y_{neg} | \{y^*\}_{<t})$
  - Keep teacher forcing objective and combine them for final loss function: $L_{ULE}^t = L_{MLE}^t + \alpha L_{UL}^t$
  - Set $\epsilon = \{y^*\}_{<t}$ and you'll train the model to lower the likelihood of previously-seen tokens!

- Limits repetition!
- Increases the diversity of the text you learn to generate

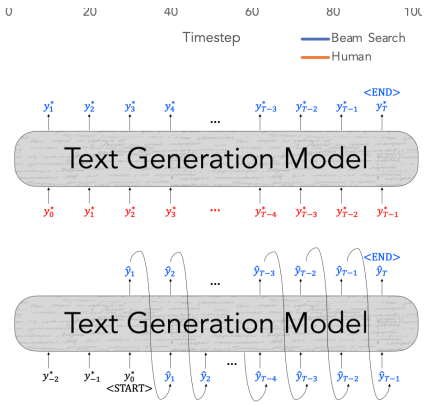
- Exposure bias: training with teacher forcing leads to exposure bias at generation time

  - During training, our model's inputs are gold context tokens from real, human-generated texts

$$\mathcal{L}_{MLE} = -\log P(y_t^*|\{y^*\}_{<t})$$

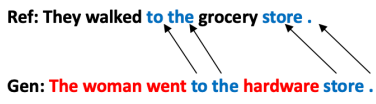
  - At generation time, our model's inputs are previously–decoded tokens

$$\mathcal{L}_{dec} = -\log P(\hat{y}_t|\{\hat{y}\}_{<t})$$



- Solutions
  - Scheduled sampling (Bengio et al., 2015)
  - Dataset Aggregation (DAgger; Ross et al., 2011)
  - Sequence re-writing (Guu*, Hashimoto* et al., 2018)
  - Reinforcement Learning: cast your text generation model as a Markov decision process

- Takeaways
  - Teacher forcing is still the premier algorithm for training text generation models
  - Diversity is an issue with sequences generated from teacher forced models
    - New approaches focus on mitigating the effects of common words
  - Exposure bias causes text generation models to lose coherence easily
    - Models must learn to recover from their own bad samples (e.g., scheduled sampling, DAgger)
    - Or not be allowed to generate bad text to begin with (e.g., retrieval + generation)
  - Training with RL can allow models to learn behaviors that are challenging to formalize
    - Learning can be very unstable!



Content Overlap Metrics | Model-based Metrics | Human Evaluations

Evaluating NLG Systems

- Content overlap metrics
  - Compute a score that indicates the similarity between generated and gold-standard (human-written) text
  - Fast and efficient and widely used
  - Two broad categories:
    - N-gram overlap metrics (e.g., BLEU, ROUGE, METEOR, CIDEr, etc.)
      - Not ideal for machine translation
      - They get progressively much worse for tasks that are more open-ended than machine translation
        - Worse for summarization, as longer output texts are harder to measure
        - Much worse for dialogue, which is more open-ended that summarization
        - Much, much worse for story generation, which is also open-ended, but whose sequence length can make it seem you're getting decent scores
    - Semantic overlap metrics (e.g., PYRAMID, SPICE, SPIDEr, etc.)
- Model-based metrics
  - Use learned representations of words and sentences to compute semantic similarity between generated and reference texts
  - No more n-gram bottleneck because text units are represented as embeddings!
  - Even though embeddings are pretrained, distance metrics used to measure the similarity can be fixed
    - Word distance functions: vector similarity, word mover's distance, BERTSCORE
    - Beyond word matching: sentence movers similarity, BLEURT
- Human evaluations
  - Ask humans to evaluate the quality of generated text
  - Overall or along some specific dimension:
    - fluency
    - coherence / consistency
    - factuality and correctness • commonsense
    - style / formality
    - grammaticality
    - typicality
    - redundancy

  - Note: Don't compare human evaluation scores across differently-conducted studies
  - Automatic metrics fall short of matching human decisions
  - Most important form of evaluation for text generation systems
    - >75% generation papers at ACL 2019 include human evaluations
  - Gold standard in developing new automatic metrics
    - New automated metrics must correlate well with human evaluation!
  - Problems: slow, expensive, inconsistent, can be illogical, lose concentration, misinterpret your question, can't always explain why they feel the way they do
- Takeaways
  - Content overlap metrics provide a good starting point for evaluating the quality of generated text, but they're not good enough on their own.
  - Model-based metrics are can be more correlated with human judgment, but behavior is not interpretable
  - Human judgments are critical.
    - Only ones that can directly evaluate factuality – is the model saying correct things?
    - But humans are inconsistent!
  - In many cases, the best judge of output quality is YOU!
    - Look at your model generations. Don't just rely on numbers!

Ethical Consideration

- Ethics: Bias in text generation models
- Hidden Biases: Universal adversarial triggers
- Hidden Biases: Triggered innocuously
- Ethics: Think about what you're building

Concluding thoughts

- Interacting with natural language generation systems quickly shows their limitations
- Even in tasks with more progress, there are still many improvements ahead
- Evaluation remains a huge challenge.
  - We need better ways of automatically evaluating performance of NLG systems
- With the advent of large-scale language models, deep NLG research has been reset

  - it’s never been easier to jump in the space!
- One of the most exciting areas of NLP to work in!

Lecture 16 - Coreference Resolution

What is Coreference Resolution

- Definition: Identity all mentions that refer to the same entity in the world.
- Applications
  - Full text understanding
    - Information extraction, question answering, summarization, …
  - Machine translation
    - Languages have different features for gender, number, dropped pronouns, etc.
  - Dialogue systems
- Coreference Resolution in Two Steps
  1. Detect the mentions (easy)
  2. Cluster the mentions (hard)

Mention Detection

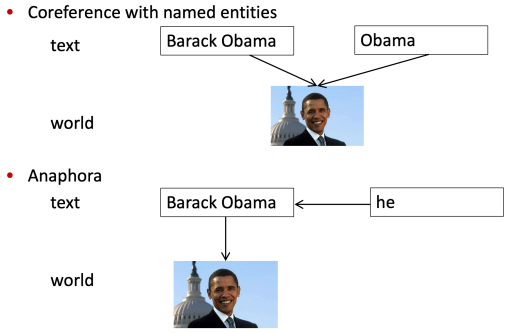
- Mention: A span of text referring to some entity
- Three kinds of mentions:
  3. Pronouns: I, you, it, she, him, etc.
     - Use a part-of-speech tagger
  4. Named entities: people, places, etc.
     - Use a Named Entity Recognition system
  5. Noun phrases: “a dog”, “the big fluffy cat stuck in the tree”
     - Use a parser (especially a constituency parser)
- Bad mentions
  - Examples:
    - It is sunny
    - Every student
    - No student
    - The best donut in the world
    - 100 miles

- Solution
  - Could train a classifier to filter out spurious mentions
  - Much more common: keep all mentions as "candidate mentions" (After your coreference system is done running, discard all singleton mentions)
- Avoiding a traditional pipeline system
  - We could instead train a classifier specifically for mention detection instead of using a POS tagger, NER system, and parser
  - Or we can not even try to do mention detection explicitly: build an end-to-end model

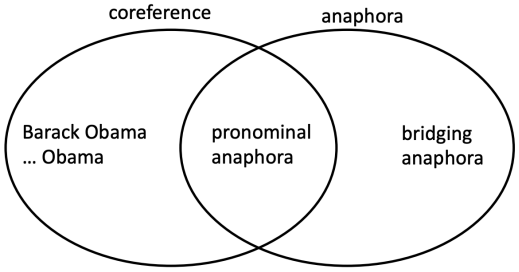
---

Coreference

- Linguistics background
  - Coreference is when two mentions refer to the same entity in the world
    - Barack Obama traveled to … Obama …
  - Anaphora (different but related) is when a term (anaphor) refers to another term (antecedent)
    - The interpretation of the anaphor is in some way determined by the interpretation of the antecedent
    - Barack Obama (antecedent) said he (anaphor) would sign the bill.



  - Not all anaphoric relations are coreferential
    - Not all noun phrases have reference
    - Every dancer twisted her knee.
    - No dancer twister her knee.
    - Because the first one is non-referential, the other isn't either.
  - Bridging anaphora
    - We went to see a concert last night. The tickets were really expensive.
  - Anaphora vs. Coreference
    - Usually the antecedent comes before the anaphor (e.g., a pronoun), but not always (Cataphora)



Four Kinds of Coreference Models

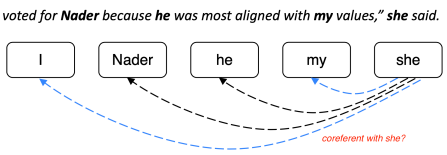
- Rule-based (pronominal anaphora resolution)
- Mention Pair
- Mention Ranking
- Clustering

1. Rule-based (pronominal anaphora resolution)

- Traditional pronominal anaphora resolution: Hobbs' naive algorithm
  1. Begin at the NP immediately dominating the pronoun
  2. Go up tree to first NP or S. Call this X, and the path p.
  3. Traverse all branches below X to the left of p, left-to-right, breadth-first. Propose as antecedent any NP that has a NP or S between it and X.
  4. If X is the highest S in the sentence, traverse the parse trees of the previous sentences in the order of recency. Traverse each tree left-to-right, breadth first. When an NP is encountered, propose as antecedent. If X not the highest node, go to step 5.
  5. From node X, go up the tree to the first NP or S. Call it X, and the path p.
  6. If X is an NP and the path p to X came from an on-head phrase of X (a specifier or adjunct, such as a possessive, PP, apposition, or relative clause), propose X as antecedent (The original said "did not pass through the N' that X immediately dominates", but the Penn Treebank grammar lacks N' nodes....)
  7. Traverse all branches below X to the left of the path, in a left- to-right, breadth first manner. Propose any NP encountered as the antecedent
  8. If X is an S node, traverse all branches of X to the right of the path but do not go below any NP or S encountered. Propose any NP as the antecedent.
  9. Go to step 4
- Until deep learning still often used as a feature in ML systems
- Bad case
  - She poured water from the pitcher into the cup until it (cup) was full
  - She poured water from the pitcher into the cup until it (pitcher) was empty"

2. Mention Pair (Coreference Models)

- Train a binary classifier that assigns every pair of mentions a probability of being coreferent: $p(m_i, m_j)$
- Training



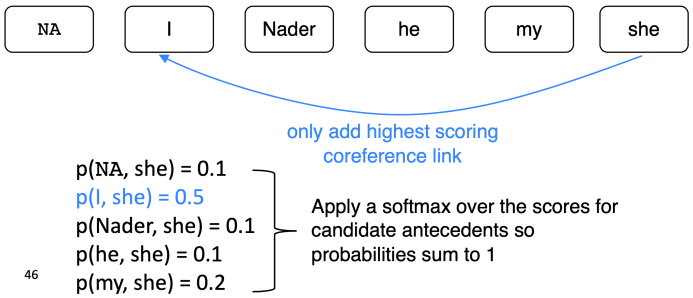
- N mentions in a document
- $y_{ij} = 1$ if mentions m_i and m_j are coreferent, -1 if otherwise
- Just train with regular cross-entropy loss (looks a bit different because it is binary classification)
- $$J = -\sum_{i=2}^{N}\sum_{j=1}^{i} y_{ij} \log p(m_j, m_i)$$

- Test time
  - Pick some threshold (e.g. 0.5) and add **coreference links** between mention pairs where $p(m_i, m_j)$ is above the threshold
  - Take the transitive closure to get the clustering
- Disadvantage
  - Suppose we have a long document with the following mentions
  - Many mentions only have one clear antecedent
    - But we are asking the model to predict all of them
  - Solution: instead train the model to predict only one antecedent for each mention
    - More linguistically plausible

---

3. Mention Ranking (Coreference Models)

- Assign each mention its highest scoring candidate antecedent according to the model
- Dummy NA mention allows model to decline linking the current mention to anything ("singleton" or "first" mention)



- Training
  - We want the current mention m_j to be linked to *any* one of the candidate antecedents it's coreferent with.
  - Mathematically, we want to maximize this probability
  - The model could produce 0.9

$$\sum_{j=1}^{i-1} \mathbb{1}(y_{ij} = 1) p(m_j, m_i)$$

Iterate through candidate antecedents (previously occurring mentions)

For ones that are coreferent to m_j

...we want the model to assign a high probability

probability for one of the correct antecedents and low probability for everything else, and the sum will still be large

- How do we compute the probabilities

A. Non-neural statistical classifier: Features

- Person/Number/Gender agreement
  - Jack gave Mary a gift. She was excited.
- Semantic compatibility
  - … the mining conglomerate … the company …
- Certain syntactic constraints
  - John bought him a new car. [him can not be John]
- More recently mentioned entities preferred for referenced
  - John went to a movie. Jack went as well. He was not busy.
- Grammatical Role: Prefer entities in the subject position
  - John went to a movie with Jack. He was not busy.
- Parallelism:
  - John went with Jack to a movie. Joe went with him to a bar.
- …

B. Simple neural network

- Standard feed-forward neural network
- Input layer: word embeddings and a few categorical features
- Embeddings: previous two words, first word, last word, head word, … of each mention
  - The head word is the "most important" word in the mention - you can find it using a parsers. e.g., *The fluffy **cat** stuck in the tree*
- Still need some other features to get a strongly performing model
  - Distance, document genre, speaker information

C. End-to-end Neural Core Model

- Current state-of-the-art model for coreference resolution *(Kenton Lee et al. from UW, EMNLP 2017)*
- Mention ranking model
- Improvements over simple feed-forward NN
  - Use an LSTM
  - Use attention
  - Do mention detection and coreference end-to-end
    - No mention detection step!

- Instead consider every span of text (up to a certain length) as a candidate mention (a pan is just a contiguous sequence of words)

- Model architecture

  1. First embed the words in the document using a word embedding matrix and a character-level CNN
  2. Then run a bidirectional LSTM over the document
  3. Next, represent each span of text i going from START(i) to END(i) as a vector

Span representation: $\boldsymbol{g}_i = [\boldsymbol{x}^*_{\text{START}(i)}, \boldsymbol{x}^*_{\text{END}(i)}, \hat{\boldsymbol{x}}_i, \phi(i)]$

BILSTM hidden states for span's start and end
Represents the context to the left and right of the span

Attention-based representation (details next slide) of the words in the span
Represents the span itself

Additional features
Represents other information not in the text

  4. Lastly, score every pair of spans to decide if they are coreferent mentions
     - Scoring functions take the span representations as input

$$s(i,j) = s_{\text{m}}(i) + s_{\text{m}}(j) + s_{\text{a}}(i,j)$$

Are spans *i* and *j* coreferent mentions?

Is *i* a mention?

Is *j* a mention?

Do they look coreferent?

$$s_{\text{m}}(i) = \boldsymbol{w}_{\text{m}} \cdot \text{FFNN}_{\text{m}}(\boldsymbol{g}_i)$$
$$s_{\text{a}}(i,j) = \boldsymbol{w}_{\text{a}} \cdot \text{FFNN}_{\text{a}}([\boldsymbol{g}_i, \boldsymbol{g}_j, \boldsymbol{g}_i \circ \boldsymbol{g}_j, \phi(i,j)])$$

include multiplicative interactions between the representations

again, we have some extra features

     - Intractable to score every pair of spans
     - Attention learns which words are important in a mention (a bit like head words)

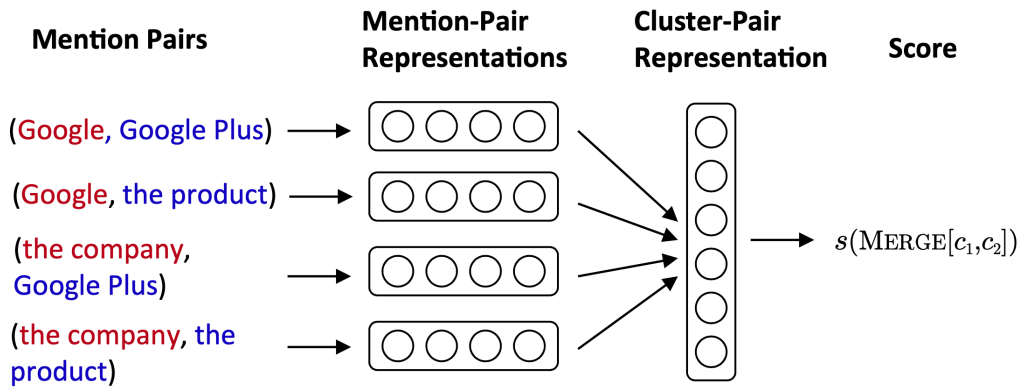
- BERT-based coref: Now has the best results!
  - Idea 1, SpanBERT: Pretrains BERT models to be better at span-based prediction tasks like coref and QA
  - Idea 2, BERT-QA for coref: Treat Coreference like a deep QA task

---

4. Clustering

- Coreference is a clustering task, let's use a clustering algorithm!
  - In particular we will use agglomerative clustering
- Start with each mention in it's own singleton cluster
- Merge a pair of clusters at each step
  - Use a model to score which cluster merges are good
- Clustering Model Architecture (*Clark & Manning, 2016*)
  1. First produce a vector for each pair of mentions
     - e.g., the output of the hidden layer in the feedforward neural network model

2. Then apply a pooling operation over the matrix of mention-pair representations to get a cluster-pair representation
3. Score the candidate cluster merge by taking the dot product of the representation with a weight vector



- Training
  - Current candidate cluster merges depend on previous ones it already made
    - So can't use regular supervised learning
    - Instead use something like Reinforcement Learning to train the model
      - Reward for each merge: the change in a coreference evaluation metric

---

Coreference Evaluation

- Many different metrics: MUC, CEAF, LEA, B-CUBED, BLANC
  - Often report the average over a few different metrics
- Essentially the metrics think of coreference as a clustering task and evaluate the quality of the clustering

Lecture 17 - Multitask Learning

Multitask

- Pretraining and sharing knowledge is great
  - Computer Vision
    - ImageNet+CNN (huge success)
    - Classification was the blocking task in vision
  - NLP
    - word2vec, GloVe, CoVe, ELMo, BERT (begin success)

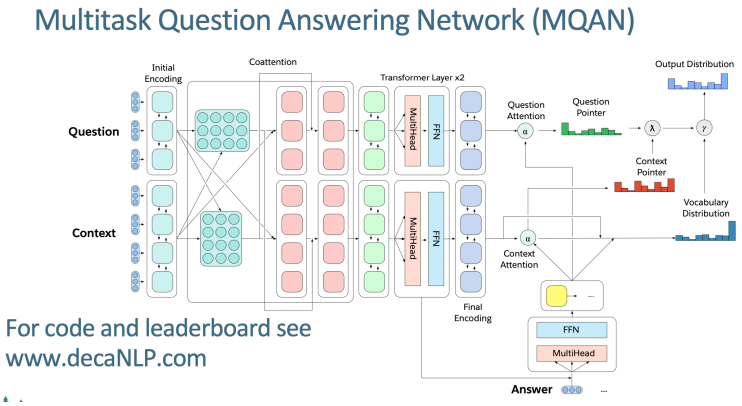
    - No single blocking tasking in natural language

- Why has weight & model sharing not happened as much in NLP
  - Requires many types of reasoning
  - Requires short and long term memory
  - NLP had been divided into intermediate and separate tasks to make progress
  - Can a single unsupervised task solve it all? No
  - Language clearly requires supervision in nature
- How to express many NLP tasks in the same framework?
  - Sequence tagging: NER
  - Text classification: sentiment classification
  - Seq2seq: machine translation, summarization, question answering

---

The Natural Language Decathlon (decaNLP)

- Training data format (question answering)
- Specification
  - No task-specific modules or parameters because we assume the task ID is not available
  - Must be able to adjust internally to perform disparate tasks
  - Should leave open the possibility of zero-shot inference for unseen tasks
- Architecture
  - Start with a context
  - Ask a question



  - Generate the answer one word at a time by

    - Pointing to context
    - Pointing to question
    - Or choosing a word from an external vocabulary
  - Pointer switch is choosing between those three options for each output word
- Training strategy:
  - fully joint
  - Anti-curriculum Pre-training (decreasing order of difficulty)

Lecture 18 - Tree Recursive Neural Networks

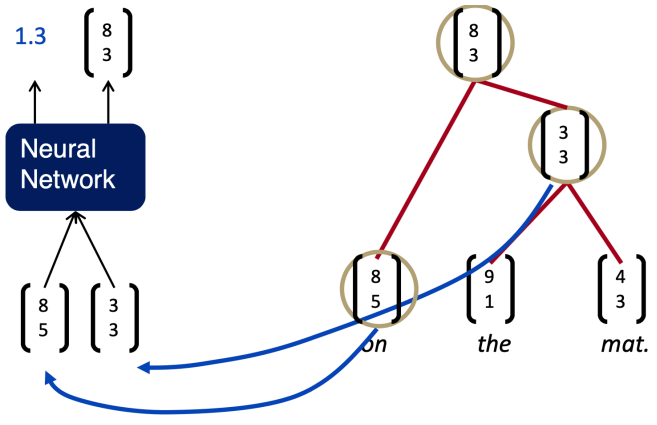
Motivation: Compositionality and Recursion

- Semantic Interpretation of language
  - People interpret the meaning of larger text units – entities, descriptive terms, facts, arguments, stories – by semantic composition of smaller elements
  - *snowborder = person on a snowboard*
- Are languages recursive?
  - recursion is natural for describing language
  - *[The person standing next to [the man from [the company that purchased [the firm that you used to work at]]]]*
  - noun phrase containing a noun phrase containing a noun phrase

---

Structure prediction with simple Tree RNN: Parsing

- Recursive neural networks for structure prediction
  - Inputs: two candidate children's representations
  - Output:
    - The semantic representation if the two nodes are merged.
    - Score of how plausible the new node would be.
  - Max-Margin framework: The score of a tree is computed by the sum of the parsing decision scores at each node



Backpropagation Through Structure

- Calculations resulting from the recursion and tree structure:
  1. Sum derivatives of W from all nodes (like RNN)
  2. Split derivatives at each node (for tree)
  3. Add error messages from parent + node itself

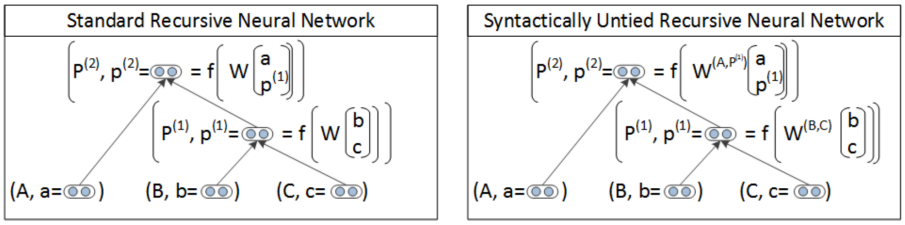
More Complex TreeRNNs

- Syntactically-United RNN
- Compositionality Through Recursive Matrix-Vector Spaces
- Recursive Neural Tensor Network
- Improving Deep Learning Semantic Representations using a TreeLSTM

4. Version 2: Syntactically-Untied RNN

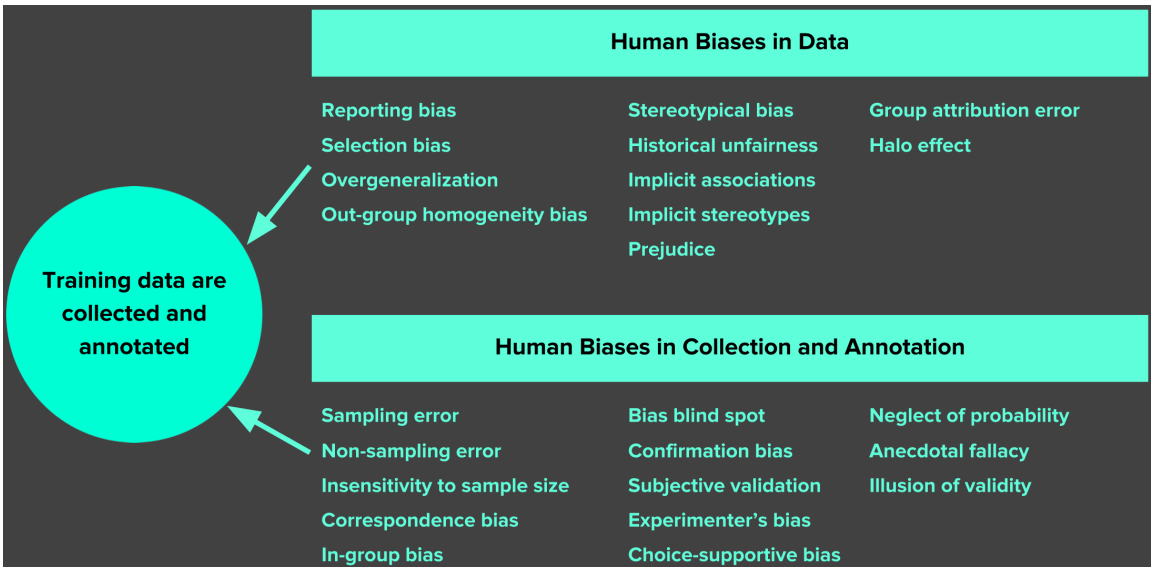
[Socher, Bauer, Manning, Ng 2013]

- A symbolic Context-Free Grammar (CFG) backbone is adequate for basic syntactic structure
- We use the discrete syntactic categories of the children to choose the composition matrix
- A TreeRNN can do better with different composition matrix for different syntactic environments
- The result gives us a better semantics



Lecture 19 - Bias and Fairness

- Bias in Data
- Evaluate for Fairness & Inclusion
  - "Equality of Opportunity" fairness criterion: Recall is equal across subgroups
- ML Techniques for Bias Mitigation and Inclusion



Lecture 20 - Future of Deep Learning and NLP

Harnessing Unlabeled Data

- Back-translation and unsupervised machine translation
- Scaling up pre-training and GPT-2

What's next?

- Risks and social impact of NLP technology
- Future directions of research
  - Harder Natural Language Understanding (QuAC, HotPotQA)
  - Multi-Task Learning (GLUE and DecaNLP)
  - Low-Resource Settings
  - Interpreting/Understanding Models (Diagnostic/Probing classifiers)
- NLP in Industry
  - Dialogue: chatbots, customer service
  - Health Care: understanding health records / biomedical literature