coreference

NLU

final exam notes

Intro

- "natural language understanding" used to contrast w/ natural language generation

Distributional semantics

- vector space models
  - co-occurence matrix has word vs context word:

determined using context window

target words	c_1	c_2	c_3	c_4	c_5
w_1	1	0	0	1	0
w_2	1	0	1	1	0
w_3	0	1	0	0	1

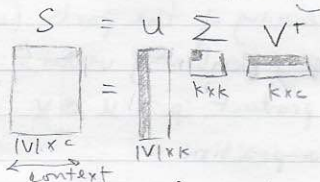
→ more similar

  - cosine similarity: $\cos(x,y) = \frac{x \cdot y}{\|x\| \cdot \|y\|}$
  - document-term co-occurence matrix:

just words in the document

	c_1	c_2	c_3	c_4	c_5
d_1					
d_2					
d_3					

  - pointwise mutual information b/t target word w and context word c: $PMI(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)}$
    - measures how often two words x and y occur compared with what we'd expect if they were independent
  - tf-idf another way of representing word vector: $w_{t,d} = \left(1 + \log\left(tf_{t,d}\right)\right) \log\left(\frac{N}{df_t}\right)$ (term freq. in document; documents in collection; # docs where t occurs)
    - higher if term occurs a lot and if term is rare
- latent semantic analysis (LSA)
  - reduces dimensionality of co-occurence matrix
  - SVD (singular value decomposition):
    - $S = U \Sigma V^T$ where Σ contains the square roots of eigenvalues from U or V in descending order



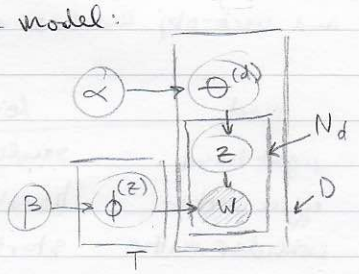
  - best known vector space model
  - performs as well as non-native English speakers in TOEFL (although modern methods have nearly 100% accuracy)

Topic models

- can do things like (1) document retrieval; (2) feature extraction; (3) analyze document collections; (4) other stuff
- Latent Dirichlet Allocation (LDA)
  - generative model: generate word w_i in document by
    1. randomly choosing a topic from the distribution over topics
    2. randomly choose a word from the corresponding topic

```
for d in {1,2,...,D}:
    for w_i in {w_i}_{i=1}^n:
        z_i ~ random (topic of w_i)
for d in {1,2,...,D}:
    for w_i in d:
        compute p(t|d)
for t in {1,2,...,T}:
    compute p(w|t)
choose t w/ p(t|d)p(w|t)
```

  - learning process:
    1. go through each document and randomly assign each word in the document to one of T topics
    2. for each document d, go through each word w in d and for each topic t compute:
       i. $p(\text{topic } t \mid \text{document } d)$
       ii. $p(\text{word } w \mid \text{topic } t)$
    3. reassign w a new topic, where we choose topic t with probability $p(t|d)\,p(w|t)$ (Gibb's sampling)
    4. repeat step 3 until "convergence"
  - graphical model:



α = proportions parameter
$\theta^{(d)}$ = per-document topic proportions
z = per-word topic assignment
w = word
β = topics parameter
$\phi^{(z)}$ = per-topic word proportions

▨ = observed

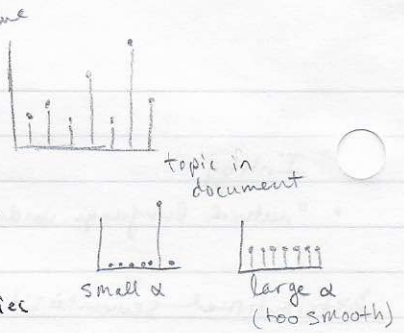
parameters:
- # topics
- α
- β

  - dirichlet distribution:

$$\text{Dir}(\alpha_1, \cdots, \alpha_T) \frac{\Gamma(\sum_j \alpha_j)}{\prod_j \Gamma(\alpha_j)} \prod_{j=1}^{T} p_j^{\alpha_j - 1}$$

topic models (cont'd)

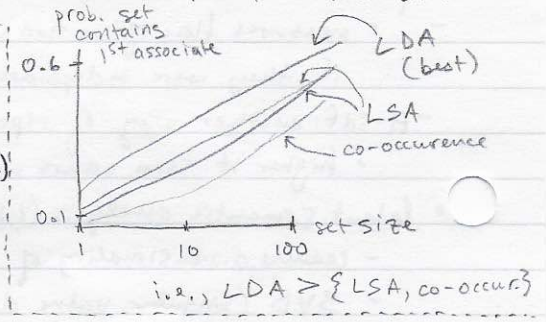
- LDA (cont'd)
  - parameter α smooths the topic distribution in the document →
  - parameter β smooths the word distribution in every topic



- can use this to compare topics with each vector being the word probabilities
- evaluation:
  - use synonym tests, free-association
  - LDA: can compare words with $p(w_2|w_1) = \sum_z P(w_2|z)P(z|w_1)$ for LDA
  - LSA: for comparison, compare words for LSA using cosine similarity

} these compute "associates" of the word of interest

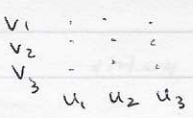
  - then can take top n associates and ask: what's the prob the set contains the true first associate?



Models of semantic composition

- composition is the idea that the meaning of a whole is more than the meaning of the parts (although there are different definitions)
- ways of simply combining vectors:

  tensor product $\underline{p} = \underline{u} \otimes \underline{v}$ ⇒



- Semantic composition
  - framework: $\underline{p} = f(\underline{u}, \underline{v}, OBJ)$

    assumptions:
    1. eliminate background knowledge K
    2. vary syntactic relationship R
    3. $\underline{p}$ is in same space as $\underline{u}$ and $\underline{v}$
    4. $f(\cdot)$ is a linear function of Cartesian product (additive)
    5. $f(\cdot)$ is a linear function of tensor product (multiplicative)

  - additive models: $\underline{p} = A\underline{v} + B\underline{v}$:

    $\underline{p} = \underline{u} + \underline{v}$

    $\underline{p} = \underline{u} + \underline{v} + \sum_i \underline{n}_i$ ← neighbors: - choose by cosine sim (or something else) - fewer is better

    $\underline{p} = \alpha\underline{u} + \beta\underline{v}$ → weighted means; only one that takes word order into account

    $\underline{p} = \underline{v}$

  - dilation models

    $\underline{p} = (\lambda - 1)(\underline{u} \cdot \underline{v})\underline{u} + (\underline{u} \cdot \underline{u})\underline{v}$

  - multiplicative models: $\underline{p} = C\underline{u}\underline{v}$:

    $\underline{p} = \underline{u} \odot \underline{v}$ $(p_i = u_i v_i)$ ← sparser than addition; if $\underline{u}$ or $\underline{v}$ = 0 then $\underline{p} = 0$

    $\underline{p} = \underline{u} \otimes \underline{v}$ $(p_{ij} = u_i \cdot v_j)$ tensor product

    $\underline{p} = \underline{u} \circledast \underline{v}$ $(p_i = \sum_j u_j \cdot v_{i-j})$ circular convolution

- evaluation:
  - phrase similarity task
    - similarity judgments for adj-noun, noun-noun, and verb-obj combos and correlate with human ratings:

	high	med	low
old person	elderly lady	right hand	small house
kitchen door	bedroom window	office worker	housing dept.
produce effect	achieve result	consider matter	start work

    - summary:
      - multiplicative and dilation work best for simple space
      - dilation and additive best for LDA
      - circular convolution worst-performing model
  - paraphrase detection: simple multiplicative model almost as good as neural net

NLU
final exam notes

neural networks

- can model logical functions like AND and OR
- perceptron learning rule: $w_i \leftarrow w_i + \Delta w_i$ where $\Delta w_i = \eta(t-o)x_i$
- gradient descent learning rule: where $\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$. (for differentiable E)
  - Δw_{ij} tells us which direction and how much we should change each weight to roll down
- generalized delta rule (e.g., squared error): $\Delta w_{ij} = \eta \delta^p_i x_{ij}$ where δ^p is the error signal (slope)

neural networks for representing word meaning

- logistic functions:

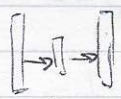
logistic function: $f(x) = \frac{L}{1+\exp(-k(x-x_0))}$

sigmoid: $f(x) = \frac{1}{1+e^{-x}}$ ← special case; generalization →

softmax: $P(y=j|\underline{x}) = \frac{\exp(\underline{x}^T \underline{w}_j)}{\sum_k \exp(\underline{x}^T \underline{w}_k)}$

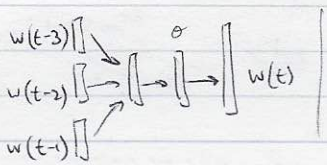
for bigram: $P(W) = \prod_i (w_i | w_{i-1})$

- not too many hidden layers needed for NLU tasks
- n-gram defining prob of sentence W: $P(W) = \prod_i (w_i | w_1, \ldots, w_{i-1})$
- represent words as one-hot vectors
- basic neural language model (NNLM):



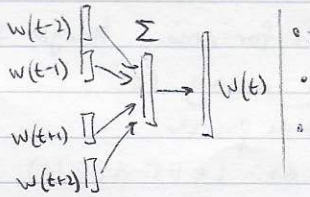
prev word → current word

- another one (that no one uses):



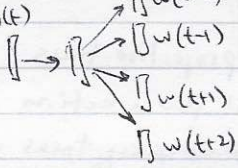
  - very computationally expensive
  - one hidden layer non linear

- continuous bag-of-words model (cBOW)



  - more computationally efficient
  - hidden layer linear
  - don't use this, but was inspiration for NNLM

- skip-gram NNLM: (supervised in terms of learning objective but doesn't rely on manual annotations)



  - like cBOW, but reformulated: predict surrounding words
  - only care about word embeddings (hidden layer)
  - input and outputs are one-hot: $[0,0,\ldots,0,1,0,\ldots,0]^T$

- more on skip-gram (which is the "best"):
  - defines $p(c|w;\theta)$ using the softmax function: $p(c|w;\theta) = \frac{e^{\underline{v}_c \cdot \underline{v}_w}}{\sum_{c' \in C} e^{\underline{v}_{c'} \cdot \underline{v}_w}}$ C is all available contexts
    - similar to cosine similarity in that this is a dot product; but this is just softmax
  - training
    - usual gradient descent and backprop
    - negative sampling solves problem of very large output layer size ⇒ only propagate signal to output neuron and a few randomly sampled negative neurons

likelihood objective: $\arg\max_\theta \prod_{(w,c)\in D} p(D=1|w,c;\theta) \prod_{(w,c)\in D'} p(D=0|w,c;\theta)$

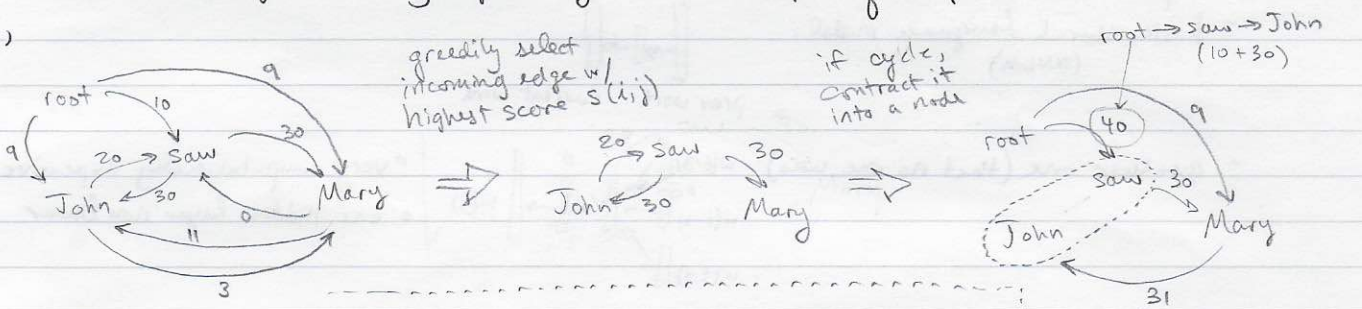
D is dataset of observed (w,c) pairs
D' are negative examples

Neural networks for representing word meaning (cont'd)

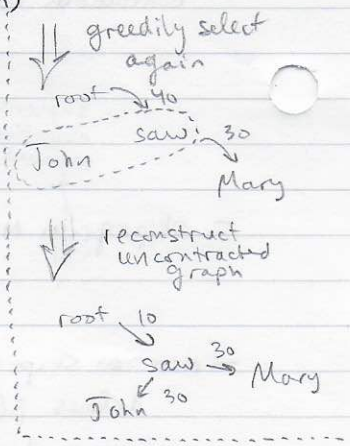
- evaluation:
  - word relationship pairs: e.g., Athens to Greece as Oslo to ____.
    - Skip-gram more than doubles accuracy of other models for semantic accuracy
    - CBOW a bit better with syntactic accuracy
  - can do word math and see how close answers are to expected, e.g., Paris - France + Italy = Rome

intro to dependency parsing

- constituency modeled as tree structure; has constituent parts that are broken down further
- dependency modeled as dependency relations: head (H), dependent (D), and label identifying relationship b/t the two
  - projective: lines don't cross
    - efficient algos (CKY, shift-reduce) for context-free grammars
    - a non-projective dependency grammar is not context-free, but efficient non-projective parsers exist, e.g., the MST parser
  - context-free algorithms base their decisions on adjacency
- two types of dependency parsers:
  - graph-based, based on maximum spanning trees (MST)
  - transition-based, an extension of shift-reduce (e.g., MALT)
- graph-based dependency parsing:
  - goal is to find the highest-scoring dependency tree in the space of all possible trees for a sentence
  - e.g.,



  - scores s(i,j) can be learned using Margin Infused Relaxed Algorithm (MIRA)
- transition-based dependency parsing
  - these will be transitions in NN model (next lecture)
  - MALT is a shift-reduce parser
  - things algorithm can do: LEFT-ARC(l), RIGHT-ARC(l), SHIFT
- comparing MST and MALT:
  - MST can naturally handle projective and non-projective trees
  - MALT makes sequence of local decisions about best parse action
    - can be extended to "pseudo-projective" dependency trees with transformation techniques
  - accuracies similar but MALT is faster
  - both need manually engineered features
  - neural networks can learn these (both features and feature weights)



dependency parsing with neural networks

- problems w/ traditional dependency parsing:
  - feature templates need to be hand crafted
  - millions of features
  - features are sparse and slow to extract
- solutions:
  - keep simple shift-reduce parser but use NN for transition classifier w/ dense features (embeddings)
- results: efficient (2x speed of MALT) w/ good performance (2% higher precision)

NLU

final exam notes

dependency parsing with neural networks (cont'd)

- network architecture (e.g., nsubj, dobj)
  - need words and POS tags (e.g., has/VBZ), head of words w/ dependency label, position of words on stack and buffer

softmax layer:

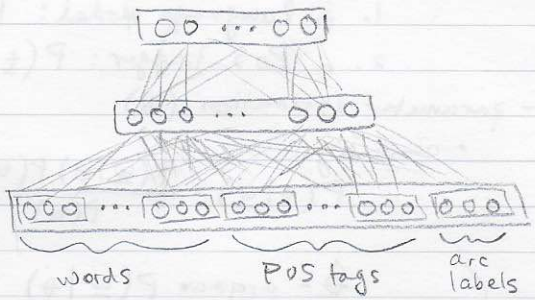
$$p = \text{softmax}(W_2 h)$$

hidden layer: (note cubic function)

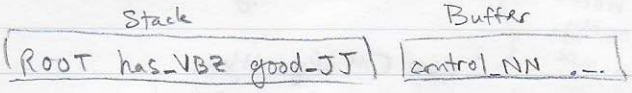
$$h = \left(W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1\right)^3$$

input layer:

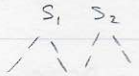
$$[x^w, x^t, x^l]$$



configuration:



- following word embeddings S^w:
  1. top 3 words on stack and buffer: $s_1, s_2, s_3, b_1, b_2, b_3$
  2. some of the children of top 2 words on stack
  3. children of children...



- training:
  - generate examples $\{(c_i, t_i)\}_{i=1}^m$ from sentences with gold parse (i.e., human-labeled) trees
  - objective: minimize cross-entropy loss w/ l_2-regularization:

$$L(\theta) = -\sum_i \log p_{t_i} + \frac{\lambda}{2}\|\theta\|^2 \qquad \theta = \{W_1^w, W_1^t, \ldots, E^l\}$$

(p_{t_i}: prob of transition t_i (from softmax layer))

  - use pre-trained word embeddings to initialize E^w, random initialization for E^t and E^l
- decoding (greedy, which doesn't guarantee globally optimum tree; more sophisticated ways of doing this):
  - for each parsing step, extract all word, POS and label embeddings from configuration c
  - compute hidden layer $h(c)$
  - pick transition w/ highest score: $t = \text{argmax}_t W_2(t, \cdot) h(c)$
  - execute transition: $c \to t(c)$ (this is the greedy step)
- results:
  - faster, more accurate
  - can do fairly well w/ word+POS (leaving off label)

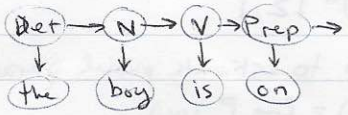
intro to unsupervised POS tagging

- unsupervised! traditional taggers require manually labeled training data → not available in many languages
- HMM:

tags $\underline{t} = t_1, \ldots, t_n$

words $\underline{w} = w_1, \ldots, w_n$

(n = length of sentence)



$$P(\underline{t}, \underline{w}) = \prod_{i=1}^n P(t_i \mid t_{i-1}) P(w_i \mid t_i)$$

  - parameters are $\theta = (\tau, \omega)$
    - τ: prob dist over tag-tag transitions
    - ω: prob dist over word-tag outputs

(matrices)

$$\tau = \underset{T \times T}{\square} \qquad \omega = \underset{T \times W}{\square}$$

- inference/decoding, where we've already estimated θ:

$$P(\underline{t}, \underline{w}) = \prod_{i=1}^n P(t_i \mid t_{i-1}) P(w_i \mid t_i) = \prod_{i=1}^n \tau_{t_i}^{(t_{i-1})} \omega_{w_i}^{(t_i)}$$

intro to unsupervised PoS tagging (cont'd)

- HMM (cont'd)
  - inference (cont'd)
    - knowing $P(\underline{t}, \underline{w})$, we can have:
      1. a language model: $P(\underline{w}) = \sum_{\underline{t}} P(\underline{t}, \underline{w})$
      2. a PoS tagger: $P(\underline{t} \mid \underline{w}) = \frac{P(\underline{t}, \underline{w})}{P(\underline{w})}$ (i.e., the topic of this lecture)
  - parameter estimation (θ)
    - Bayes!

$$P(\theta \mid \underline{w}) = \frac{P(\underline{w} \mid \theta) P(\theta)}{P(\underline{w})} \propto P(\underline{w} \mid \theta) P(\theta)$$

(can omit this, which has the effect of assuming a uniform prior)

Works poorly, so use Bayesian integration →

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, P(\underline{w} \mid \theta) \quad \text{(MLE)}$$

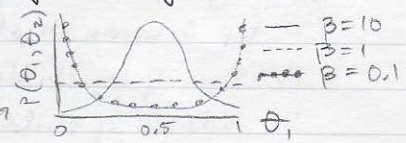
- use EM algorithm
- but we want $P(\underline{t} \mid \underline{w})$, not $P(\underline{t} \mid \theta, \underline{w})$. So...

$$P(\underline{t} \mid \underline{w}) = \int_{\Delta} P(\underline{t} \mid \underline{w}, \theta) P(\theta \mid \underline{w}) \, d\theta \quad \text{("Bayesian integration")}$$

- for prior, use Dirichlet: $P(\theta) = \frac{1}{Z} \prod_{j=1}^{k} \theta_j^{\alpha_j - 1}$ (so integrates to 1)

$$\underline{t}^{(t)} \mid \alpha \sim \text{Dirichlet}(\alpha)$$

$$\underline{w}^{(t)} \mid \beta \sim \text{Dirichlet}(\beta)$$



2-D Dirichlet w/ $\theta = (\theta_1, \theta_2)$

$\beta > 1$: prefer uniform distributions

$\beta = 1$: no preference

* $\beta < 1$: prefer sparse (skewed) dists
  * use this for language modeling since some words have most of the mass and most have almost none

- evaluation:
  - use dict that lists possible tags for each word
  - train and test on unlabeled corpus (WSJ)
- results:
  - integrating over priors is useful even if they're uninformative ($\alpha = \beta = 1$)
  - better priors help, but don't reach state-of-the-art

unsupervised PoS tagging with neural networks

- want to replace output distributions ω with something more sophisticated
- one idea: use local features to define $\omega^{(t)}$:

$$\omega_w^{(t)} = \frac{\exp(\underline{\lambda} \cdot \underline{f}(t, w))}{\sum_{w'} \exp(\underline{\lambda} \cdot \underline{f}(t, w'))}$$

where $f(t, w)$ can indicate capitalization, "+ing", etc.

→ use neural network to get these

- another idea: use word embeddings to define $\omega^{(t)}$ (multinomials become d-dim Gaussians)

$$\omega_w^{(t)} = \frac{\exp\left(-\frac{1}{2}(\underline{v}_w - \underline{\mu}_t)^T \Sigma_t^{-1} (\underline{v}_w - \underline{\mu}_t)\right)}{\sqrt{(2\pi)^d |\Sigma_t|}}$$

- for both ideas: use EM algorithm to estimate model parameters
  - standard EM: optimize $L(\theta) = \log P_\theta(\underline{w})$
    - E: compute expected counts for emissions: $e_{(t,w)} \leftarrow \mathbb{E}_\omega \left[\sum_i \mathbb{I}(t, w_i) \mid \underline{w} \right]$

$\mathbb{I}(t, w_i) = \begin{cases} 1 \text{ if } (t, w_i) \text{ is in data} \\ 0 \text{ otherwise} \end{cases}$

    - M: normalize expected counts to re-estimate θ:

$$\omega_w^{(t)} \leftarrow \frac{e_{(t,w)}}{\sum_{w'} e_{(t,w')}}$$

NLU
final exam notes

unsupervised POS tagging with neural networks (cont'd)

- EM algorithms (cont'd)
  - EM for HMMs with features
    - E: compute $\omega_w^{(t)}$ given $\underline{\lambda}$ (see earlier)
    - M: optimize regularized expected log-likelihood over all word-tag pairs:

$$\ell(\underline{\lambda}, \underline{e}) = \sum_{(t,w)} e_{(t,w)} \log \omega_w^{(t)}(\underline{\lambda}) - \kappa \|\underline{\lambda}\|_2^2$$ (use gradient-based search algorithm)

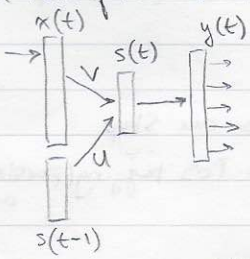
    - in this framework, can add arbitrary features, e.g., whether token contains digit, contains hyphen, is capitalized, n-gram
    - rich features boosts accuracy from 43. to 56 (+13)
  - EM for multivariate Gaussians: use pretrained $\underline{v}_w \in \mathbb{R}^d$
    - so: $\omega_w^{(t)} = P(\underline{v}_w; \underline{\mu}_t, \underline{\Sigma}_t)$ ⇒ embeddings of words for tag t are concentrated at $\underline{\mu}_t$

$$P(\underline{t}, \underline{w}) = \prod_{i=1}^{n} P(t_i | t_{i-1}) \overbrace{P(\underline{v}_w; \underline{\mu}_t, \underline{\Sigma}_t)}^{\text{replaces } P(w_i|t_i)}$$

    - in each iteration, update $\underline{\mu}_t$ and $\underline{\Sigma}_t$
- model comparison
  - setup:
    - 8 languages, gold standard tagged data
    - for word embeddings: skip-gram w/ window size 1 and d=100
    - estimate $\underline{\mu}_t$ as above; assume $\underline{\Sigma}_t$ is fixed, diagonal covariance matrix (estimation didn't help)
    - HMM params initialized randomly
  - summary:
    - word embeddings improve unsupervised POS tagging
    - Gaussian models perform best, but use a lot more data
    - structured skip-gram slightly outperforms skip-gram
    - d=20 performed better than d>20
    - window size 1 optimal

recurrent neural networks

- n-gram language model: $P(w_1 \dots w_k) = \prod_{i=1}^{k} P(w_i | w_{i-n+1} \dots w_{i-1})$
  - to choose next word: $\hat{w}_{k+1} = \underset{w_{k+1}}{\operatorname{argmax}} P(w_{k+1} | w_1 \dots w_k)$
  - estimating n-gram probabilities is difficult: need lots of data and fancy backoff/smoothing
  - can be limited in context (fixed n)
- RNNs: solve some of the issues just mentioned
  - simple RNN architecture: →



y(t) is prob dist over next word given current word w(t) and context s(t-1)

    - look back just one timestep, but in theory you keep information from all past timesteps
    - use standard backprop with SGD
    - error signal: error(t) = desired(t) − y(t)
- evaluation (RNNs):
  - interpolate w/ standard 5-gram model w/ smoothing
  - replace rare words w/ rare words token
  - measure perplexity and word rate error (WER)
  - test on WSJ and other data

smaller is better: higher prob when making predictions

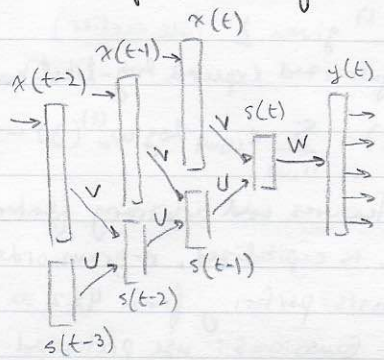
perplexity is the inverse of the probability of the test corpus, normalized by # words:

$$ppl(w_1 \dots w_N) = \sqrt[N]{\frac{1}{P(w_1 \dots w_N)}}$$

$$= \sqrt[N]{\prod_{i=1}^{N} P(w_i | w_{i-n+1} \dots w_{i-1})^{-1}}$$

recurrent neural networks (cont'd)

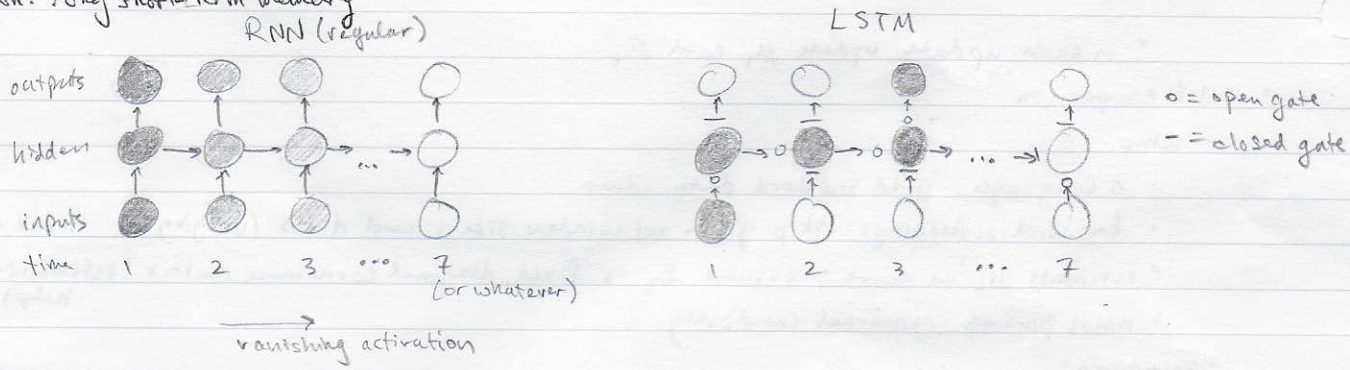
- results:
  - combining smoothing model w/ RNN increased performance of smoothing model (output probs combined as weighted avg)
  - RNN also good on its own
- full RNN ("unfolded over time")
  - can do this for arbitrary # time steps τ
  - inc $\tau \Rightarrow$ better performance
  - but vanishing gradients problem...



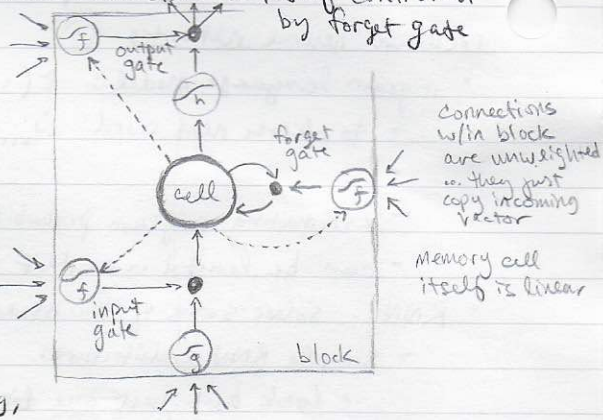
trained w/ back propagation through time (BPTT)

long short-term memory

- "vanishing gradients": since every timestep, weights are multiplied w/ another gradient; gradients are < 1, so the deltas become smaller and smaller.
  - in practice, this prevents RNN from learning long-range dependencies (rapidly "forgets" previous inputs)
- solution: long short-term memory



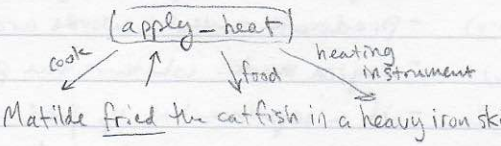
- hidden layer of memory blocks; each block consists of four units:
  - input gate: controls whether input is passed to memory cell or is ignored
  - output gate: controls whether current activation vector of memory cell is passed to output layer or not
  - forget gate: controls whether activation vector of memory cell is reset to zero or maintained
  - memory cell: stores current activation vector; with recurrent connection to itself controlled by forget gate
- gates are trainable regular hidden units
- why it solves vanishing gradient problem:
  - memory cell is linear, so gradients don't vanish
  - LSTM block can retain info indefinitely (if forget gate is open [close to 1] and input gate is closed [close to 0])
- training:
  - BPTT truncated to one step
- LSTMs can be used for POS tagging/parsing, semantic role labeling, and opinion mining



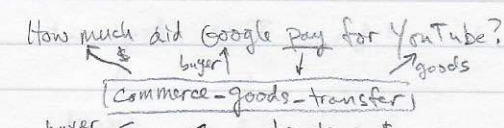
NLU
final exam notes

Semantic role labeling

- parsing is breaking up sentences into meaningful parts, which can then be fed to downstream semantic tasks
  - one such task is semantic role labeling
- frame semantics:
  - a frame describes a prototypical situation
  - evoked by a frame evoking element (predicate)
  - can have several frame elements (arguments; sem. roles)



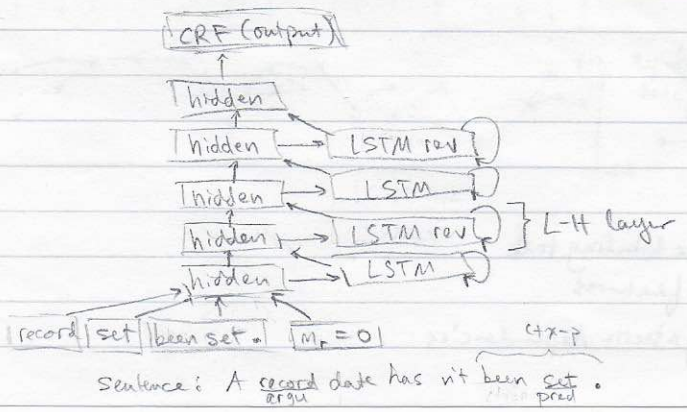
- properties
  - granularity between "universal" and "verb specific" roles
  - generalizes well across languages
- PropBank is a version of the Penn Treebank annotated with semantic roles; more coarse-grained than Frame Semantics:



Arg 0: proto-agent	→	causer of increase
Arg 1: proto-patient	→	thing increasing
Arg 2: benefactive, instrument, attribute, and state	→	amt. increased
Arg 3: start point, benefactive, instrument, or attribute	→	start point
Arg 4: end point	→	end point
Arg M: modifier	→	NA

e.g.: for increase.01

- SRL usually broken down into sub-tasks:
  1. parse training corpus
  2. match frame elements to constituents
  3. extract features from parse tree
  4. train a probabilistic model on the features
- parsed sentences can yield the following features: phrase type (e.g., NP, VP, S), governing category, parse tree path, position (before/after predicate), voice (passive/active), head word
- since SRL is a sequence labeling task, we should be able to use neural networks
- this lecture will discuss end-to-end SRL system using deep bi-directional LSTM (DB-LSTM)
  - uses no explicit syntactic information
  - requires no separate frame element matching step
  - needs no expert-designed, language specific features
  - outperforms previous approaches using feedforward nets
- architecture:
  - bidirectional LSTM usually has two hidden layers, both connected to the same input and output layer, processing the same sequence in opposite directions
  - for our DB-LSTM, a standard LSTM layer processes the input in forward direction; the output of this layer is input to another LSTM layer but in reverse direction; these LSTM pairs are stacked to obtain a deep model



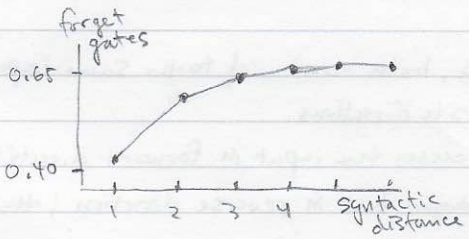
Semantic role labeling (cont'd)

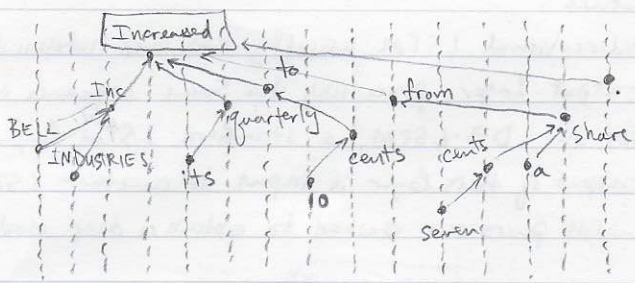
- features: processed word by word
  - argument and predicate: argument is word being processed, predicate is word it depends on
  - (ctx-p) predicate context: words around the predicate
  - (m_r) region marks: whether the predicate is in the predicate context region or not
  - (n_p) if a sequence has n_p predicates it's processed n_p times
- output: Semantic role label for the predicate/argument pair using IOB tags (inside, outside, beginning)

beginning-Arg1

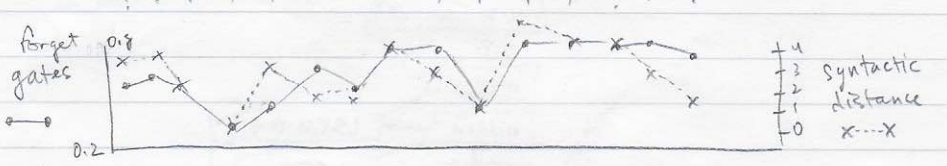
e.g.;	Time	Argument	Predicate	ctx-p	m_r	Label	
	1	A	set	been set .	0	B-A1	at test time you need
	2	record	set	been set .	0	I-A1	both the words and
	3	date	set	been set .	0	I-A1	the predicate.
	4	has	set	been set .	0	O	
	5	n't	set	been set .	0	B-AM-NEG	
	6	been	set	been set .	1	O	
	7	set	set	been set .	1	B-V	
	8	.	set	been set .	1	O	
		↑	↑	word = 1		is word in ctx-p (=1)	
		word being processed	word the argument depends on (same for whole sentence)				

- training:
  - word embeddings used as input
  - embeddings for arguments, predicate, ctx-p, m_r are concatenated and used as input
  - trained w/ standard backprop using SGD
  - Viterbi decoding used to compute best output sequence
- experimental set-up
  - train/test on CoNLL-2005 (essentially a dependency parsed version of PropBank) — although the parsing aspect of data isn't needed
  - word embeddings either randomly initialized or pretrained
- results
  - model learns "syntax": associates argument and predicate words using forget-gate





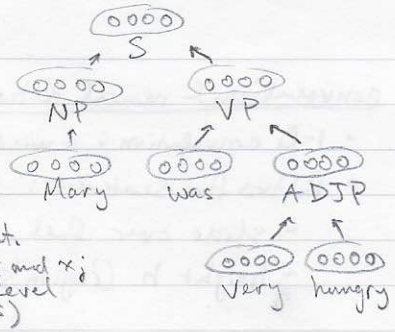
- summary:
  - SRL means identifying arguments (frame elements) that participate in a prototypical situation (frame) and labeling them w/ their roles
  - traditionally consists of parsing, frame element matching, feature extraction, and classification
  - but BD-LSTM views it as a sequence labeling task
  - no parsing needed; no handcrafted features
  - forget gate helps the net learn syntactic dependencies



NLU
final exam notes

recursive autoencoders

- idea of compositionality: whole > sum of parts
- can learn representation of tree nodes in binary tree: ⇒ not just terminals
- autoencoders: you know what they are
- in NLU formulation:



vertical concat. of vectors x_i and x_j (two lower-level nodes)

$$y_k = f\left(W\left[x_i; x_j\right] + b\right)$$

$$\left[z_i; z_j\right] = U y_k + c$$

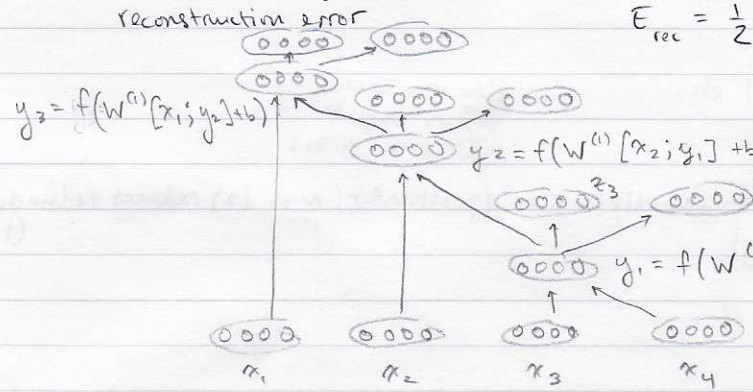
$$E_{rec} = \frac{1}{2}\left\|\left[x_i; x_j\right] - \left[z_i; z_j\right]\right\|^2$$

meaning at node k

reconstructions of inputs x_i and x_j

reconstruction error

z_i and z_j are approx. reconstructions of x_i and x_j



hidden representations y_i same dimensions as x_i

- error of tree is sum of errors at all non-leaf nodes: $\sum_{k \in T} E_{rec}(k)$
- enforce all meaning vectors to have unit length (to avoid all meanings = 0)
- make loss for node k weighted:

$$E_{rec}(k) = \frac{n_i}{n_i + n_j}\left\|x_i - z_i\right\|^2 + \frac{n_j}{n_i + n_j}\left\|x_j - z_j\right\|^2$$

n_i and n_j are # words covered by nodes i and j

- selecting tree structure
  - use greedy algorithm to find good but not necessarily optimal tree

$x_1\ x_2\ x_3\ x_4$ — say this is smallest error

$x_1\ x_2\ p(3,4)$ — search again

$p(1,2)\ p(3,4)$

$p((1,2),(3,4))$

- prediction
  - apply linear model to meaning vectors to predict labels
  - objective function:

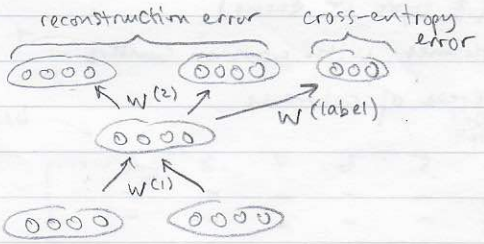
$$J = \frac{1}{M}\sum_{\langle s,t\rangle \in \$} E(s,t,\theta) + \underbrace{\frac{\lambda}{2}\left\|\theta\right\|^2}_{\text{regularization}}$$

$\theta = \langle W, b, U, c, V \rangle$

where $E(s,t,\theta)$ is total error for one sentence with label t:

$$E(s,t,\theta) = \underbrace{\sum_{k \in T(s)}}_{\text{set of non-leaf nodes}} \alpha \underbrace{E_{rec}(k)}_{\text{reconstruction error}} + \underbrace{(1-\alpha)}_{\text{usually larger than } \alpha} \underbrace{E_z(k)}_{\text{cross-entropy error}}$$

⇒ weighted average of reconstruction error and label prediction error



- results:
  - tested on short stories that were voted into 1 of 5 categories
  - had 50% accuracy (best)

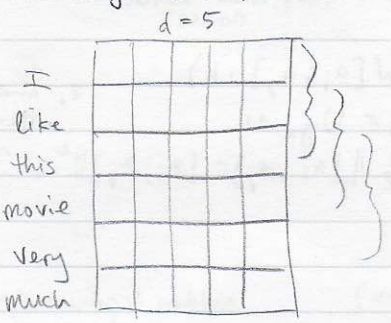
However, LSTMs still seem to be the most powerful at understanding meaning

Convolutional neural networks (CNN)

- 1-D convolution: a windowed average of the input vector

10 50 60 10 20 40 30
1/3 1/3 1/3 ← kernel (g), learned
= 40
1/3 1/3 1/3
= 40
→ keep sliding

- filters for sentences:
  - slide over full rows of the matrix (want the whole embedding)
  - height h ("region size") is number of adjacent rows (usually window is 2-5 rows)



since sentences and filters can be of different sizes

- use pooling for two reasons: (1) standardizes size (practical); and (2) reduces redundant information (theoretical)
  - most common is 1-max pooling
- hyperparameters for CNN
  - zero-padding preserves size
  - stride length
  - others already mentioned (e.g., region size, pooling size)
- final model
  - each sentence represented as feature vector and fed through softmax function for final classification
  - can use dropout, L2 regularization, ...
  - training objective: min cross-entropy loss
  - word vectors can be learned or inferred
- model comparisons:
  - different flavors of model change how input embeddings are handled: pretrained, randomly initialized, pretrained but fine-tuned during training
  - also vary filter windows (h)
- results
  - competitive (or better) vs. RNN for classification task; however, RNNs used for sentence generation since CNNs can't do this
- discussion
  - simple architectures work well across the board

Entity-based coherence

- coherence: sentences are meaningfully related
  - entities in an utterance are ranked according to salience
  - each entity has one center (≈ topic or focus)
  - coherent discourses have utterances with common centers → e.g., subject of sentence remains the same
  - entity transitions capture degrees of coherence

S = subject
O = object
X = other

(X→)X1
(—)X4

	entity 1	2	3	4	5	6	7	8	9	...
sentence 1	S	X	X	–	–	–	–	–	–	
2	S	–	–	X	–	–	–	–	–	
3	–	–	–	–	S	X	X	O	–	
4	O	–	–	–	–	–	–	–	O	
5	S	–	–	–	–	–	–	–	–	
6	–	–	–	–	O	–	–	–	–	

← basically the unique nouns in the document

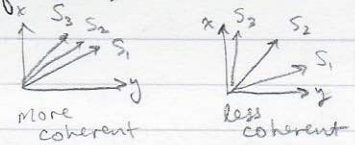
S O X 1 S O X ...
S S S S O O O ...

d_1 0 0 0 .03 0 0 0 ... ← feature vector for document 1
d_2 0 0 0 .02 0 .07 0 ...
d_3 .02 0 0 .03 0 0 0 ...

NLU
final exam notes

entity-based coherence (cont'd)

- learning a ranking function
  - ordered pairs (x_{ij}, x_{ik}), where x_{ij} and x_{ik} represent the same document d_i, and x_{ij} is more coherent than x_{ik} (assume $j > k$)
  - goal is to find parameter vector $\vec{w}$ such that
    $$\vec{w} \cdot (\Phi(x_{ij}) - \Phi(x_{ik})) > 0 \quad \forall j, i, k \text{ such that } j > k$$
- text ordering
  - data: in sourced document, assume original sentence order is coherent
  - given k documents, with n permutations, obtain $k \cdot n$ pairwise rankings for train and test
- comparison with state of the art:
  - vector based model: sentence meaning is mean of vector of words, then take average distance of adjacent sentences (unsupervised, local)
  - HMM-based content models: model states and their order in texts;
    - model is HMM: states correspond to topics ($\approx$ sentences)
    - model selects sentence order with highest probability
    - supervised, global

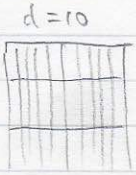


- discussion
  - omission of coreference causes performance drop
  - entity model better than LSA
  - HMM-based content models have high variability
  - models seem to be complementary
- Summarization
  - Summaries naturally exhibit coherence violations
  - compare model against rankings elicited by human judges
  - useful for automatic evaluation of machine generated text
- Summarization results
  - coreference decreases accuracy (unlike before)
  - entity model better than LSA (like before)
- Summary:
  - entity model is a novel framework for representing and measuring coherence
  - suited for learning appropriate ranking function
  - fully automated and robust; useful for system development
  - weaknesses: doesn't contain lexical information or notion of global coherence; can't model multi-paragraph text;

distributed representations for documents 13

(CNN)

- model trained w/ classification task in mind to create embeddings in hidden layer
- hidden layers learn document representation
- let clique C denote a window of sentences, where each has a label $y_c \in 1$ if coherent and 0 otherwise; to create negative cases (0's), randomly swap in an unrelated sentence



convolution over this grid, as with words from CNN classification

- coherence score S_d for document d is prob. that all cliques w/in d are coherent:
  $$S_d = \prod_{c \in d} p(y_c = 1)$$
  ⇒ only aggregates local coherence; if it evolves a lot over the course of the document, this rating wouldn't capture that non-coherence

distributed representations for documents (cont'd)

- conclusions
  - techniques for sentence modeling transfer to documents
  - have different classes of models depending on choice of composition model for sentences / documents
  - not really reasonable to compress the meaning of a document into a single vector...
  - choice is motivated by computational reasons
- NLU models that work best, overall:
  - word2vec
  - LSTM
  - CNN (for classification baseline)