Lecture 1: Regex

Lookaround Assertions: what is inside **()** is not matched.

+ve look ahead: a(?=b): a followed by b.

-ve look ahead: a(?|b): a not followed by a.

+ve look behind: (?<=a): b preceded by a.

-ve look behind: (?<|a): b not preceded by a.

Capturing Groups:

Must specify the regex in parentheses ()

Ex1: `CMP([0-9]+) are \1 years old` will match `CMP 23 are 23 years old`

Ex2: `(?:some|few) (people) like \1` will match some `people like people` and `few people like people`. Where the `\1` will be replaced with `people` not `some` nor `few`, because they are in a **non-capturing group**.

Binary string with four ones: (0*1+0*){4,}

Binary string with zeros multiple of 3: ((1*01*){3})+

The **^** matches the **beginning of a string** or a **new line** if the multiline flag is on. The $ matches the **end of a string** or a **line** if the multiline flag is on.

Preprocessing: Tokenization -> Word Normalization -> Segmenting sentences.

Normalization: **Word Normalization**: standard Form, **Case folding**: lowercase, **Lemmatization**: convert to the root.

BPE:**Pros**(subword-tokenizer, out-of-vocabulary (OOV) words are represented)|**Cons**(greedy, generated tokens are dependent on the number of iterations)

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Lecture 2: Language Models

N-gram(Substitute by N): using Maximum Likelihood Estimation (MLE), combination of **chain rule** and **Markov.**

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$

$$P(w_n|w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}\, w_n)}{C(w_{n-N+1:n-1})}$$

Perplexity(w): the lower the better

$$\sqrt[N]{\square} \rightarrow \prod \frac{1}{\square} \rightarrow P(w_i|w_{n-N+1:n-1})$$

Back-off: if no (n)-gram, use (n-1)-gram and so on.

Interpolation:
$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n) \quad \sum_i \lambda_i = 1$$

Smoothing(k = 1 if Laplace).
$$P^*_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+k}{C(w_{n-1})+kV}$$

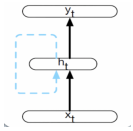
Extrinsic Evaluation: put the end-to-end model directly in the application; **Intrinsic Evaluation:** Use metrics such as perplexity to measure model performance.
Closed Vocabulary: <unk> words don't occur but **open vocabulary** needs to add it.

Lecture 3: RNNs

Markov N-gram **limit the context** to a fixed window. RNN has no window and allows variable input.

function FORWARDRNN(x, $network$)

$h_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH(x) **do**
  $h_i \leftarrow g(U\, h_{i-1} + W\, x_i)$
  $y_i \leftarrow f(V\, h_i)$
return y



W is (dh,din)**U** is (dh,dh) and **V** is (dout,dh)

Training RNNs (Find ∂E/∂U,V,W then Gradient Desc)

$$\frac{\partial Et}{\partial V} = \frac{\partial Et}{\partial y_t}\frac{\partial y_t}{\partial V} \qquad \frac{\partial Et}{\partial W} = \sum_{k=1}^{t} \frac{\partial Et}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W} \qquad \frac{\partial Et}{\partial U} = \sum_{k=1}^{t} \frac{\partial Et}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial U}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \qquad \frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial Et}{\partial W}$$

Et is the loss at each step

Truncated BPTT: Only backbrop after each k tokens(not </s>)

RNN as LM:

$$\mathbf{e}_t = \mathbf{E}\mathbf{x}_t$$
$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t)$$
$$\mathbf{y}_t = \text{softmax}(\mathbf{V}\mathbf{h}_t)$$

RNN Probability of Seq:

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i|w_{1:i-1}) = \prod_{i=1}^{n} \mathbf{y}_i[w_i]$$

CrossEnt Loss:

$$-\sum_{c=1}^{N} y_c log(p_c)$$

Becomes

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

Teacher Forcing: Feed the model the correct history (next word) rather than its own prediction in training (faster training).
Weight Tying: To reduce redundancy, parameters and improve perplexity use E for embeddings and E.T instead of V.
Hidden state in RNN: is form of memory that encodes earlier processing.

Lecture 11: Machine Translation

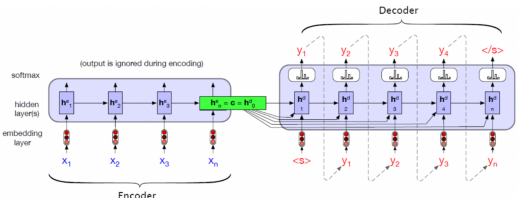
Needed as translation and may sequence to sequence mappings are **not direct mappings on token level**. Encoder-decoder models handle such complex mappings.

Encoder generates **contextualized representation(final hidden state)** of input (e.g. Arabic sentence)
Decoder uses it and generates **task specific output** like language model (e.g. English sentence)

Here **c** is **produced by encoder** then **decoder** uses it as **initial hidden** state with initial input token <s>.
Equations with t should be in for loop.

$$\mathbf{c} = \mathbf{h}^e_n$$
$$\mathbf{h}^d_0 = \mathbf{c}$$
$$\mathbf{h}^d_t = g(\hat{y}_{t-1}, \mathbf{h}^d_{t-1}, \mathbf{c})$$
$$\mathbf{z}_t = f(\mathbf{h}^d_t)$$
$$y_t = \text{softmax}(\mathbf{z}_t)$$

In the equations above c, is available to **decoder at each timestep** as well **otherwise its influence decreases as decoder generates output.**



C is a bottleneck as it has to capture whole input. **Attention** is a way to allow decoder to get info from all encoder hidden states and not just last one by **making c dynamic.** Compute it as follows **for each one decoder timestep i and all encoder timestep j**:

$$\alpha_{ij} = \text{softmax}(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) \;\forall j \in e) = \frac{\exp(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j))}{\sum_k \exp(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_k))}$$

$$score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) = \mathbf{h}^d_{i-1} \cdot \mathbf{h}^e_j \qquad \mathbf{c}_i = \sum_j \alpha_{ij}\, \mathbf{h}^e_j$$

Once we have it compute h as:

$$\mathbf{h}^d_i = g(\hat{y}_{i-1}, \mathbf{h}^d_{i-1}, \mathbf{c}_i)$$

Alpha tells us **the proportional relevance** of each encoder hidden state to decoder.

Beam search: use in decoder instead of taking the highest probability token greedily (local opt) to improve results.

Lecture 4 Naive Bayes

$$\hat{c} = \underset{c \in C}{\text{argmax}} \overbrace{P(f_1, f_2, \ldots, f_n|c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}} \qquad \hat{P}(c) = \frac{N_c}{N_{doc}}$$

$$c_{NB} = \underset{c \in C}{\text{argmax}} \log P(c) + \sum_{i \in positions} \log P(w_i|c)$$

	gold positive	gold negative	
system positive	true positive	false positive	precision = $\frac{tp}{tp+fp}$
system negative	false negative	true negative	
	recall = $\frac{tp}{tp+fn}$		accuracy = $\frac{tp+tn}{tp+fp+tn+fn}$

$$F_1 = \frac{2PR}{P+R}$$

Macro Avg(P):more appropriate when performance on all the classes is equally important.

$$\frac{\Sigma tp}{\Sigma tp + \Sigma fp}$$

Micro Avg(P): dominated by the more frequent class.

$$\frac{\sum_i^N p_i}{N}$$

Lecture 6: Classical Word Embeddings

Distributional Hypothesis: Words in similar contexts have similar meanings. Standard way to represent words in NLP is **vector semantics**.

Term Document Matrix
ROW: Word Vec
Column: Document Vec

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

TF-IDF: TF=cound(t,d)/len(d) and IDF=log10(N/df)
Better than raw counts (words that occur frequently in one document are better than ones that occur frequently over all documents. TF also solves issue of longer documents having more counts)

Term Term Matrix
In W[i,j] put the number of times word i occurred in the context of word j. Window of size 2 means 2 to each side.
ROW & COL: Word Vec
PPMI(Positive Pointwise Mutual Information)

$$\text{PPMI}(w,c) = \max\left(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0\right)$$

p(w)=sum(row)/sum(W)
p(c)=sum(col)/sum(W)
p(w,c)=W[i,j]/sum(W)

PPMI measures how frequent words occur together compared to independence **but is biased towards infrequent ones.** To solve the issue smooth counts or calc p(c) as $P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha}$

Word Similarity: Use **cosine** and not raw dot product as raw dot product can favor vectors with more frequencies regardless of similarity. **Applies to word or document vectors,** higher cosine means higher similarity.

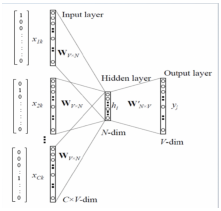
$$\cos(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}||\mathbf{w}|}$$

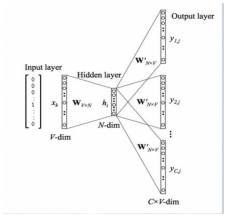
Evaluation: compare word similarity scores to humans, or semantic textual similarity (sentences) or analogy (a to b is c to what=> a-b=c-x then the vector with highest similarity to x)

Your Notes

Lecture 7

Vectors we considered last lecture have many 0s **(are sparse)**, dense vectors are better. **Dense vectors** can be **static** or **dynamic** (change for each word **depending on context**).





CBOW
Average inputs or their look ups
(h computed for each c then avg)

Skipgram
Sum loss for each context
(same y computed for each c as y_c)

$$h_i = \sum_j w_{ji} * x_j \text{ where } i = 1,2,3 \text{ and } j = 1,2,3,4,5 \quad (1)$$
$$u_j = \sum_i w'_{ij} * h_i \text{ where } i = 1,2,3 \text{ and } j = 1,2,3,4,5 \quad (2)$$
$$y_j = Softmax(u_j) \text{ where } j = 1,2,3,4,5 \quad (3)$$

Take contexts give target

Take target give contexts

training sample	context words	target word
1	(word,are)	vectors
2	(vectors,widely)	are
3	(are,used)	widely
4	(widely)	used

training sample	context word	target word
1	(word,are)	vectors
2	(vectors,widely)	are
3	(are,used)	widely
4	widely	used

Given "word vectors are widely used" and C=1 (window looks on the left) we get the two datasets above for the task. For C=2 window looks both sides. // Skipgram proof is skipped.

Fasttext can handle unknown words (e.g., using 3-gram "where"="<where>"+"<wh"+"whe"+"her"+"ere"+"re>") if the original word doesn't exist just use subwords.
GloVe also learns from co-occurrence matrix to capture global statistics besides local statistics.
Sphere is not distributional and unlike others distinguishes antonyms (-ve cosine) from synonyms and unrelated words.

Lecture 8: HMM

Part-of-speech tagging: is the task of taking a sequence of words and assigning each word a part of speech (POS) like NOUN or VERB.

NER: is the task of assigning words or phrases tags like PERSON, LOCATION, or ORGANIZATION.

HMM is a generative approach.

Transition probabilities estimation:
$$P(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})}$$

Emission probabilities estimation:
$$P(w_i|t_i) = \frac{C(t_i,w_i)}{C(t_i)}$$

HMM Tagging as Decoding:

$$\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} P(t_1 \ldots t_n|w_1 \ldots w_n) \approx \underset{t_1 \ldots t_n}{\operatorname{argmax}} \prod_{i=1}^{n} \overbrace{P(w_i|t_i)}^{\text{emission}} \overbrace{P(t_i|t_{i-1})}^{\text{transition}}$$

Viterbi Algorithm:
$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

This algorithm is a dynamic programming algorithm.

If you have n words and N tags so you have (N ^ n) possibilities, but Viterbi gives you the result in O(n * N^2)

Lecture 9

Linear chain CRF: discriminative sequence model based on log-linear.

$$\begin{aligned}\hat{Y} &= \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} P(Y|X) \\ &= \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} \frac{1}{Z(X)} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y)\right) \\ &= \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} \exp\left(\sum_{k=1}^{K} \left(w_k \sum_{i=1}^{n} f_k(y_{i-1},y_i,X,i)\right)\right) \\ &= \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} \sum_{k=1}^{K} \left(w_k \sum_{i=1}^{n} f_k(y_{i-1},y_i,X,i)\right) \\ &= \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} \sum_{i=1}^{n} \sum_{k=1}^{K} w_k f_k(y_{i-1},y_i,X,i)\end{aligned}$$

Using Viterbi Algorithm:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) \sum_{k=1}^{K} w_k f_k(y_{t-1},y_t,X,t) \quad 1 \le j \le N, 1 < t \le T$$

known-word templates:

$$\langle y_i, x_i \rangle, \langle y_i, y_{i-1} \rangle, \langle y_i, x_{i-1}, x_{i+2} \rangle$$

- We use **stochastic gradient descent** to train the weights to maximize the log-likelihood of the training corpus.

- Part-of-speech taggers are evaluated by the standard metric of **accuracy**.

- (DC10-30): Word shape(XXdd-dd), short word shape:(Xd-d).

- **Gazetteer**: list of place names, providing millions of entries for locations with detailed geographical and political information.

- Named entity recognizers are evaluated by **recall, precision, and F1 measure** (because of segmentation).

Lecture 10

Syntactic Parsing : It is the task of assigning a syntactic structure to a sentence

1- Constituency parsing:

CFG: A context-free grammar G is defined by four parameters: N:nonterminals, Σ:terminals, R:rules, S:start

The following conventions are followed:
Capitals: non-terminals, Lowers: Terminals, Greek: both

We usually distinguish two kinds of grammar equivalence:

- **strong equivalence**:(only renaming non-terminal).
- **weak equivalence**: two grammars generate the same set of strings but don't assign the same phrase structure to each sentence.

Any context-free grammar can be converted into a weakly equivalent **Chomsky normal form** and this needs CFG to be:

- ε-free (no empty rules).
- each production is either of the form A⟶ B C or A⟶ a.

we need **four steps** to convert CFG to CNF:

1) Eliminate ε Productions.

2) Eliminate Variable Unit Productions: Consider production A ⟶ B. Then for every production B ⟶ α, add the production A ⟶ α.

3)Replace Long Productions by Shorter Ones: if have production A ⟶ BCD, then replace it with A ⟶ BE and E ⟶ CD.

4) Move Terminals to Unit Productions: replace production A ⟶ bC with productions A ⟶ BC and B ⟶ b.

CKY (Cocke-Kasami-Younger) Parsing Algorithm skipped.

Dependency Parsing:

- **Binary grammatical relations** between the words
- This is called **typed dependency structure**
- **cross-linguistically applicable.**
- Dependency parsing include: transition-based parsing and graph-based parsers

---

Lecture 5

Logistic Regression is discriminative.

Output: Func(W.x + b), x: feature vector, W: Weight Vec.

if binary LR -> Func: $\sigma(z) = \frac{1}{1+e^{-z}}$

if multinomial LR -> Func: $\text{softmax}(\mathbf{z}_i) = \frac{\exp(\mathbf{z}_i)}{\sum_{j=1}^{K} \exp(\mathbf{z}_j)} \quad 1 \le i \le K$

Zi = Wi.X+bi for each class.
Logistic Regression vs Naive Bayes: LR is better if the features are correlated, NB is easier to implement and train

---

Lecture 12

Term Weighting and Document Scoring:

Two term weighting schemes are common: the tf-idf weighting and BM25 a slightly more powerful variant.

$\text{score}(q,d) = \cos(\mathbf{q},\mathbf{d}) = \frac{\mathbf{q}\cdot\mathbf{d}}{|\mathbf{q}||\mathbf{d}|}$ Non Default: TF=log10(count(t,d)+1)

Document Scoring:

$$\text{score}(q,d) = \sum_{t \in q} \frac{\text{tf-idf}(t,q)}{\sqrt{\sum_{q_i \in q} \text{tf-idf}^2(q_i,q)}} \cdot \frac{\text{tf-idf}(t,d)}{\sqrt{\sum_{d_i \in d} \text{tf-idf}^2(d_i,d)}}$$

Queries are usually very short, so each query word is likely to have a count of 1. And the normalization for the query will be the same for all documents.

So we use this score: $\text{score}(q,d) = \sum_{t \in q} \frac{\text{tf-idf}(t,d)}{|d|}$

The answer extraction task is commonly modeled by **span labeling**

Algorithm for reading comprehension is to pass the question and passage to any encoder like BERT.

Evaluation of Factoid Answers:

Mean Reciprocal Rank (MRR):
$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

Your Notes