A Neural Probabilistic Language Model

Pengfei Tan

Background

- Statistical Language modeling:

  - learn the joint probability function of sequences of words in a language
  - Given a word sequence w_1^T, the probability is

$$P(w_1^T) = \prod_{t=1}^{T} P(w_t \mid w_1^{t-1})$$

- Curse of Dimensionality
  - a word sequence in test data is likely to be different from training data

Background

- Traditional Solution
  - N-grams with smooth or back-off
  - Only 1 or 2 contexts words taken into account (N in N-gram is hardly over 3)
  - Don’t consider Similarity between words:

If we knew:

dog and cat played similar roles

Similarity of (the,a), (bedroom,room), (is,was)

Generalize: *The cat is walking in the bedroom*

to

A dog was running in a room

Key Idea

- Associated each word to a distributed ***word feature vector***
- Using feature vector of words in the sequence to express joint probability function
- Learn *simultaneously* the word feature vector and parameters of probability function word feature vector

Neural Probabilistic Language Model

- Training Set:
  - Sequence $w_1, \ldots, w_T$ of words $w_t \in V$, V is the vocabulary
- Model to be learnt:

$$f(w_t, \ldots, w_{t-n+1}) = P(w_t \mid w_1^{t-1})$$

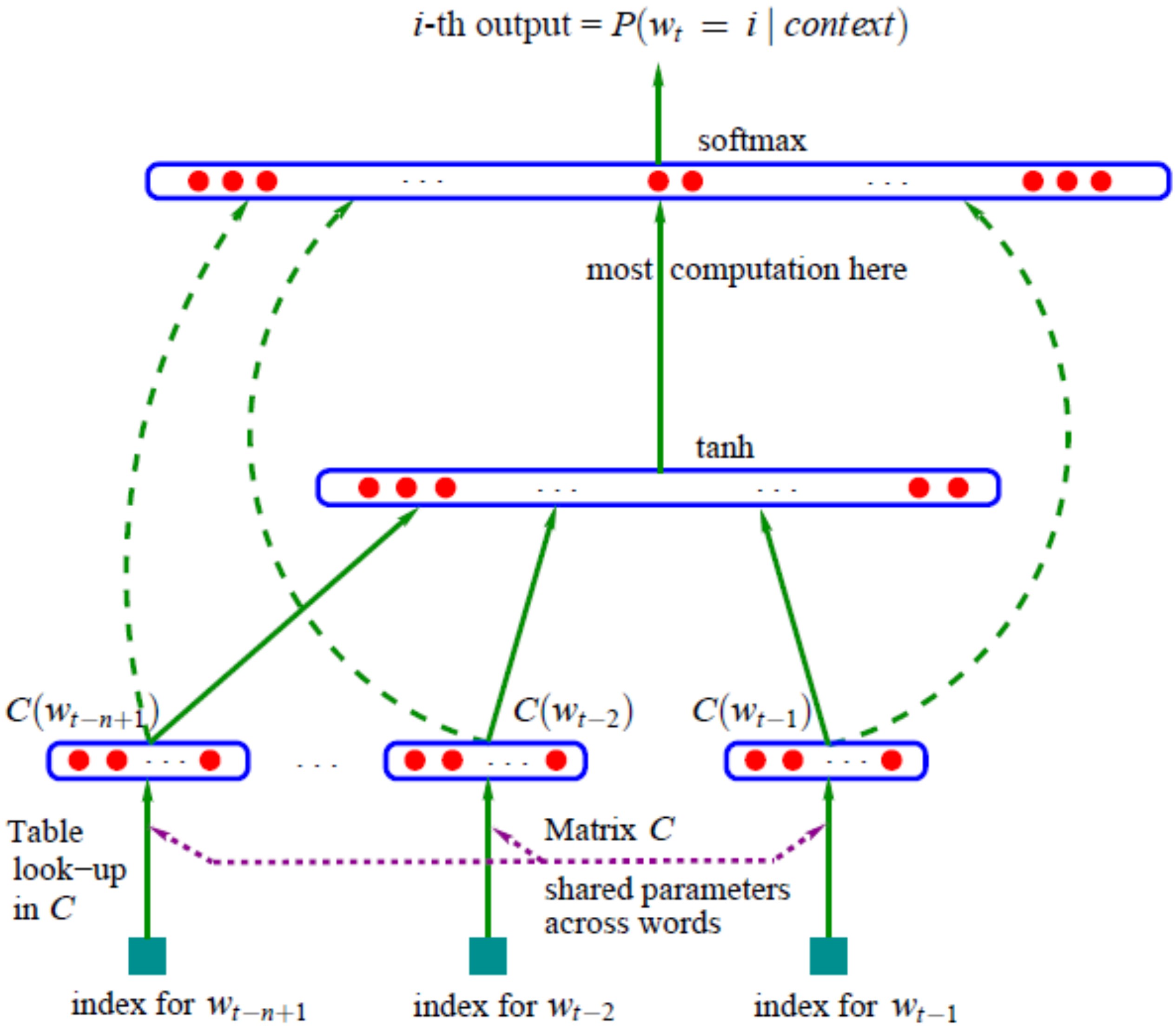
Decompose

- 2 parts:
  - A map C
    - Mappinng from each i in V to a real vector(distributed feature vector) $C(i) \in \mathbb{R}^m$
    - A |V|×m matrix of free parameters in practice
    - Shared across all the words in the context
  - Function g

$$f(i, w_{t-1}, \ldots, w_{t-n+1}) = g(i, C_{t-n+1}, \ldots, C_{t-1})$$

    - Feed-forward or recurrent neural network

Structure of Model



Neural Network

- 1 hidden layer
  - h hidden Unit
  - Use tanh as activation function
- Output Softmax Layer

$$P(w_t \mid w_1^{t-1}) = e^{y_{w_i}} / \sum_i e^{y_i}$$

  - ith input of output layer y_i:

$$y = b + Wx + U\tanh(d + Hx)$$

  - b: bias in output layer (|V| elements);
  - W connect feature vectors(a $|V| \times (n-1) \times m$) matrix))
    - Could be 0 (non-connected)
  - d: bias in hidden layer(h elements)
  - H is hidden layer weights ($h \times (n-1)$ matrix)

Parameters

- $\theta=(b,d,W,U,H,C)$
- $|V|(1+nm+h)+h(1+(n-1)m)$ parameters.
- Maximizes :training corpus penalized log-likelihood

$$L=\frac{1}{T}\sum_{t}\log f(w_t,\ldots,w_{t-n+1}:\theta)+R(\theta)$$

- Stochastic gradient ascent

  For t-th word of the training corpus

$$\theta\leftarrow\theta+\varepsilon\frac{\partial\log(P(w_t|w_1^{t-1}))}{\partial\theta}$$

  ε is learning rate

Implementation: Parallel

- A lot of Computation
- Data-Parallel Processing
  - Shared-memory processor, each works on a different subset of data
  - Asynchronous
  - Expensive
- Parameter-Parallel Processing
  - Each CPU is responsible for the computation of the unnormalized probability for a subset of the outputs and performs the updates
  - low communication overhead

Experiments

- 2 Corpora:
- Brown corpus:
  - 800,000 words stream for training
  - 200,000 words stream for validation
  - 181,041 words stream for testing
  - |V| = 16383
- AP News
  - 14 million words stream for training
  - 1 million words stream for validation
  - 1 million words stream for testing
  - |V| = 17964

Experiments

- Comparison
- Smoothed trigram model
  - (Jelinek and Mercer, 1980)
- Back-off n-gram models with the Modified Kneser-Ney algorithm
  - (Kneser and Ney, 1995, Chen and Goodman., 1999)
- Class-based n-gram models
  - (Brown et al., 1992, Ney and Kneser, 1993, Niesler et al., 1998)

Results:

	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**
Del. Int.	3						31	352	336
Kneser-Ney back-off	3							334	323
Kneser-Ney back-off	4							332	321
Kneser-Ney back-off	5							332	321
class-based back-off	3	150						348	334
class-based back-off	3	200						354	340
class-based back-off	3	500						326	**312**
class-based back-off	3	1000						335	319
class-based back-off	3	2000						343	326
class-based back-off	4	500						327	312
class-based back-off	5	500						327	312

Results:

- Neural network obtains significantly better results than the best n-grams model
- Neural network was able to take advantage of more context
- Hidden units are useful
- Mixing is good
  - neural network and the trigram make errors in different places
- Direct connections from input to output

Future Work

- Decomposing the network in sub-networks
- Propagating gradients only from a subset of the output words