Knowledge Representation in *pAninI* Framework Using Neural Network Model

Smita Selot[1], Neeta Trpathi[2] and **A.S Zadgaonkar[3]**

Submitted in October 2012; Accepted in February 2013

Abstract - Knowledge representation is base for expressing semantic content of input in intelligent information retrieval systems. Identification of semantic requires processing of input language at various levels. To make system understand text or speech is a challenging task as it involves extracting semantics of the language which itself is a complex problem. At the same time languages posses with multiple ambiguities and uncertainty which needs to be resolved at various phases of language processing. Level of understandability depends upon the grammar, syntactic and semantic representation of the language and methods employed for these analysis. Processing depends on the type of language, grammar of the language, ambiguities present and size of corpus available. Order free language posses different features as compared to rigid order language. Most of the Indian languages are order free; hence mechanism for such language needs to be formulated. One of the ancient Indian Sanskrit grammarians, pAninI has defined grammar of Sanskrit language in such a way that it is suitable for computational analysis. Six main semantic class identified under this theory is a baseline model for knowledge representation. This paper exploits the features of the language, applicability of rules and resolving ambiguities using neural network model. A hybrid model incorporating the features of rules based and neural network the is designed and implemented for pAninI based semantic analysis, generating case frames as output.

Index Terms - pAninI Grammar framework, Knowledge Representation, Case Frame, Natural Language Processing, Semantic.

1. INTRODUCTION

Knowledge representation is a technique to represent the meaningful and logical content embedded in the language; in a structured form. Development of such tool requires an exhaustive analysis of input language at syntactic and semantic level with capacity to handle ambiguities at each level. Natural languages are not so natural for computer processing; hence a KR tool acts as bridge between the natural language and understanding of language by machine. Development of such tool is heavily guided by language processing techniques and type of language. Order free language posses different characteristics than rigid order language. As most of the Indian languages are order free, they require different mechanism to handle their processing. KR, Natural Language Processing (NLP) and Information Retrieval (IR) are close module of such applications as depicted in Figure 1.

[1]SSCET, Bhlai, [2]SSITM, Bhilai, India
[3]C V Raman University, Bilaspur, *E-mail:* [1]*sselot@sify.com*

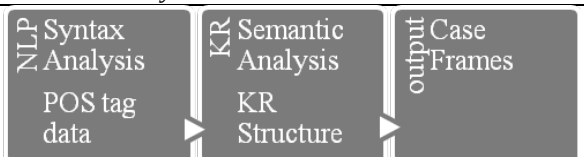

Figure 1: Inter relation between NLP and KR

Statistical methods are applied for syntactic analysis of Indian language with Hidden Markov Model(HMM) [12], support Vector machine (SVM) being popular statistical Technique [4] [19]. Application of Neural Network for classification task is less observed as both are complex domain. This paper presents a method for generation of Case Frames (CF) as KR structure for Sanskrit Language under *pAninI* framework. Method identifies semantic role of each word with respect to action or verb present in the sentence, there by presenting a verb-argument relation. Six main semantic classes are defined under *pAninI* framework. Identification and classification of word into one of the class is achieved by analyzing suffix attached to word. Identified class along with word is stored in KR structure called CF. However while performing the classification one suffix may map into multiple domain resulting into conflicting output. Such conflict is resolved by training neural network for ambiguous cases. Non conflicting cases are handled by one-to-one *vibhakti_kArka* mapping resulting into a hybrid model for case frame generation. This paper describes the concept of *pAninI* grammar for semantic analysis, database of suffix, algorithm and solutions for conflict cases. KR based system are widely used in applications like translation system, learning algorithm and question answer based system

2. *pAninI* GRAMMAR

One of the ancient languages of the world, Sanskrit, has well defined grammatical and morphological structure which precisely defines the relation of suffix-affix of the word with the syntactic and semantic classification of the sentence [2][3] [11]). Such analysis leads to development of KR structure. For order free language like Sanskrit, processing is quite interesting as suffix based analysis reveals syntacto-semantic features of the sentence. Sanskrit is analyzed from computational perspective on vedic text [7] as well as capability of *pAninI* grammar is equivalent to finite state machine [8]. Development of automatic segmentiser is an effort in this field [13]. Hindi and Arabic clauses are also analysed from *pAninian* aspect [14]. Parallelism of *pAninI* in field of computer science is well explained [15]. Rule based POS tagger developed at JNU,Delhi uses lexicon and displays all possible outcome for conflicting cases [9]. This paper explains processing of Sanskrit for classifying words in one of six semantic roles defined by *pAnini* under *kAraka* theory implementing a novel approach –Neural Network.

Generally, dictionary of words is maintained and each word is mapped to find its respective syntactic category. As *pAninI* has identified the syntacto-semantic information of the word by the suffix attached to the word, instead of maintaining dictionary of words, lexicon of suffix is sufficient for extracting features. *kAraka* roles are similar to case based semantics required for event-driven situations, where entities like agent , object, location are identified with respect to each event [6] [10].

pAninI, an ancient Sanskrit grammarian has given nearly 4000 rules called *sutra* to describe behavior of the language in the book called *asthadhyAyi;* meaning eight chapters [10]. Ancient old *kAraka* theory rules are in parallel with finite state machine [8] and concept is being extended for English language [20]. It describes transformational grammar which applies sequence of rules to transform root word to number of dictionary words. From small set of root words, millions of words are generated by firing set of rules. For highly inflectional language like Sanskrit, sequence of declension tables are memorized in such a way that similar ending words follow the same declension. Hence, if one table is memorized, number of words can be generated if their base word falls under same group. This structural representation in optimum form is used to identify the semantic class of the word. In Sanskrit language, fundamental six roles, given by *pAninI* as *kAraka* values, are key semantic component of a sentence as described in Table 1.

SN	Case	*kAraka*	*Vibhakti*	Meaning
1	Nominative	*Karta*	*prathamA*	Agent
2	Accusative	*Karma*	*dviteeyA*	Object
3	Instrumental	*karNa*	*triteeyA*	Instrument
4	Dative	*sampradAn*	*Chaturthi*	Recipient
5	Ablative	*apAdAn*	*Panchai*	Departure
6	Locative	*adhikaraN*	*Saptami*	Place

Table 1: Six *kAraka* in *pAnanian* model.

Suffix driven analysis is performed by mapping the suffix to database which contains suffix and a key number. Key is designed in such a way that it contains all the syntactic information as per grammar of the language

3. KEY NUMBER DESIGN FOR SUFFIX

All the nouns in the language follow nominal declension tables for each category of word. For example all '*a*' ending word follow the declension given in Table 2 with word as '*rAma'* and Table 3 shows the suffix attached to the word.

Vibhakti	***Ekvachan***	***Dwivacahn***	***Bahuvachan***
1	*rAmH*	*rAmau*	*rAmAH*
2	*rAmam*	*rAmau*	*rAmAn*
3	*rAmen*	*rAmAbhyAm*	*rAmaiH*
4	*rAmAya*	*rAmAbhyAm*	*rAmebhyH*
5	*rAmAt*	*rAmAbhyAm*	*rAmebhyH*
6	*rAmasya*	*rAmayoH*	*rAmAnAm*
7	*rAmen*	*rAmayoH*	*rAmeShu*

Table 2: Declension set for *rAma*

Vibhakti	***Ekvachan***	***Dwivacahn***	***Bahuvachan***
1	*H*	*Au*	*AH*
2	*Am*	*Au*	*An*
3	*En*	*AbhyAm*	*aiH*
4	*Aya*	*AbhyAm*	*ebhyH*
5	*At*	*AbhyAm*	*ebhyH*
6	*Asya*	*yoH*	*AnAm*
7	*En*	*yoH*	*eShu*

Table 3: Suffix for all *a* ending word

Each row corresponds to a *vibhakti* value and column represents the number or *vachan*. Unlike English language, which contains only singular and plural, Sanskrit has singular as *ekvachan* , plural as *bahuvachan* and two in number is labeled as *dwivachan*. *Vibhakti* is related to *kArka* values, Suffixes present in first row or *vibhakti* is *kartA kAraka* (agent). Likewise each row represents a *kArka* role as given in the Table 3.1.Sixth *vibhakti* is not included in Table 3.1 as it is *sambandh kAraka* which has relation with its immediate argument and not related to verb directly, hence not considered as *kAraka* by *pAninI*.

Four digit key number schemes for noun suffix is designed as given in Fig 2

x ending	**Gender**	**Vibhakti**	**Number**

Figure 2: Four digit number scheme for noun

Type of ending is in first column where 9 different type of ending is considered with values in range 1-9.Gender are masculine, feminine and neuter with values 1,2 and 3. Seven *vibhakti* from 1 to 7 and three number from 1to 3 are considered. For example suffix *'am'* is present in second row, first column as given in Table 2.2; it is assigned the value 1121 where description of each digit is as follows:

1: *a* ending
1: Masculine gender
2: *dvitIyA vibhakti*
1: *ekvachan*

On similar guideline, five digit number schemes is designed for storing verb suffix [16]. In Sanskrit grammar, verbs are classified into ten groups called *gan* represented by most significant place in the number scheme. When a root word joins with the suffix (*pratya*), some changes takes place at the junction. With respect to these changes verbs are classified into nine different *gan.* Second digit from left denotes *pad* which occur in three different forms- *Atmnepad*, *parsmaipad* and

ubhaypad. Verbs whose outcome is for another person, they fall under
Parasmaipad and verbs whose outcome is for one self come under *Atmnepad.* Verbal words whose outcome is for both, other person and one self, they come under *ubhaypad*. Time in which action takes place is given in various tenses. There are 10 different tenses in Sanskrit [5]. Giving the context of the person is *purush* and number is *vachan.* A five digit number scheme for verb is presented in Fig 3.

Gan	*Pad*	Tense and mood	Person	Number

Figure 3: Five Digit Number Scheme for Verb Suffix

Verbs in Sanskrit decline with respect to *gan*, *pad*, tense, person and number. These are influential parameters as they govern the behavior of the nouns in the sentence. Range and example values associated with each field are described in Table 4.

Digit	*Gan*	*Pad*	Tense & mood	Person	Number
Range	0-9	1-3	0-9	1-3	1-3
Example	*bhavAdigan,*	*Parasmaipad*	*Latlakar*	*pratham puruSh*	*Ekvachan*

Table 4: Range and Example Values for Each Digit Position of Verb Number Scheme

For example *paThati* (to read), has suffix *ti* which extracts the number 11011 from database there by giving the following information:

1-*bhavAdigan*;
1-*parasmaipad*;0-*latlakAr*(Present tense);
1-*pratham puruSh* (Third Person);
1-*ekvachan* (singular) .

Pronouns decline in manner similar to noun and total number of pronoun is less, hence set of most commonly used pronoun are stored in a separate database with their key values[17]. Number scheme for pronouns is given in Fig 4.

P_number	Gender	Vibhakti	Number

Figure 4: Four Digit Number Scheme for Pronoun

P_number identifies a particular pronoun like *serva, sH* etc. For example, 1 is given to *tad*, meaning 'that' in English. Rests of digit have same values as for noun. All the pronouns stated in *rachanAnuvAdakaumudI* are considered with nearly 400 entries in database [5]. Some words which do not change their form under any condition, they are termed as *avyay.* List of these words are maintained separately.

4. ALGORITHM FOR CASE FRAME

Objective is to generate the CF by identifying the semantic role (*kAraka* value) of each word with respect to action in a given sentence. Every word within the sentence is searched in *avyay* list. On unsuccessful search in the list, word is mapped in pronoun, verb suffix and then noun suffix database. This order has been followed as the quantity of words in each category follows an ascending order.

After check in *avyay* list; word w_i, is mapped in pronoun database; If found, its semantic role is identified by extracting the 4th digit from its key and performing *vibhakti kAraka* mapping on it. Otherwise next look up is performed on verb database (VDB) followed by noun database (NDB).

String is processed in reverse order from right to left. Last character of the word w_i is identified (x) and all the suffix which has x as their last character is extracted from the VDB and stored it in a set. If any one of the value from this set matches as suffix in word; word w_i, it is added to list of mapped suffix. If this list contains one element then a unique mapping has been found, else multiple matches are discovered. In case of multiple matches, splitter algorithm is activated to check for the category of word. Splitter breaks the word in base word and suffix. If base word is present in lexicon of verbal base, current word is tagged as action entity. If no match is found in verbal base then check for noun is performed.

A similar mapping process is also performed for nouns, but this mapping process face a problem as occurrence of suffix in database is not unique; at time multiple matches are obtained. It is due to intergroup and intragroup redundancy of suffix. Occurrence of same suffix within one declension table is intragroup redundancy and occurrence of same suffixes across tables is intergroup redundancy. Depending upon frequency of occurrence and redundancies, suffixes are divided into three classes. Class I identifies unique occurrence of suffix, class II identifies intergroup redundancy and class III identifies intragroup redundancy.

Class 1: Unique suffix
Suffix with frequency of occurrence =1.
Format of the data is
(<key number>, <suffix>, <frequency of occurrence>)
Example: (1111, 'H', 1)

Class 2: Intragroup redundancy
Suffix with frequency of occurrence greater than 1 and same *kAraka* value
Example: (1131, 'en', 2)(1331, 'en', 2), suffix 'en' have same *kAraka* value 3.

Class 3: Intergroup redundancy
Suffix with frequency of occurrence greater than 1 and different *kAraka* value.
Example: (1112, 'au', 4) (1122, 'au',4) (2171, 'au', 4)(3171, 'au', 4), suffix 'au' has *kAraka* values 1, 2, 7, 7.

kAraka value is the 3rd digit in the key from left. Categorization of the suffixes is presented in Fig 5

All x-ending suffixes s_i, are extracted from NDB and stored in a set. If suffix belong to class I, it is easily given a *kAraka* role. If more than one suffix is present, then for every suffix s_i, word w_i is split in base word and suffix using splitter algorithm. This

base word is searched in lexicon of base words. If a match is found, then s_i and its key number are stored as final suffix and final number in a list. Class II type Intergroup redundancy of the suffixes are handled by splitter routine. Ambiguity related to class III may still exist. To resolve such cases conflict resolution using neural network is applied. *Vibakti kAraka* mapping is applied to 4th digit of this number and a semantic tag is assigned to the word. All the words with their semantic tag are stored in the case frame.

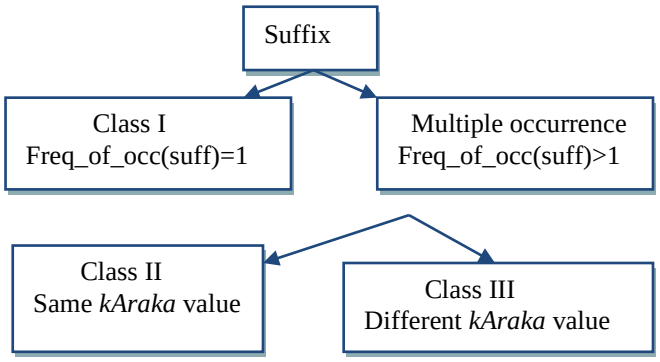

Figure 5: Categorization of Suffix with Respect to their *kAraka* Value

Out of the total 252 suffixes for noun, number of suffix belonging to each class is given in Table 5.

Quantitative parameter for noun suffix	Quantity
Number of suffix	252
Number Unique suffix	120
Class 1	55
Class 2	22
Class 3	43

Table 5: Class wise Quantification of Suffix Data

Algorithm for hybrid model

vlist = lexicon of verb bases
nlist= lexicon on nominal bases
w_i = word under process
s_i = suffix
x= last character of word under process.
verb_xmatch: set of all suffix ending with *x*
verb_suffix: set of all suffix in verb_xmatch which map as suffix in word w_i
noun_xmatch: set of all suffix ending with *x*
noun_suffix: set of all suffix in noun_xmatch which map as suffix in word w_i
frame: is structure for storing elements like action, agent, object etc.

Pseducode

Input sentence S.
for each word w_i € S
match w_i in pronoun database
if match found ***then***
w_i.cat=pronoun
get key number from database of w_i
vibhakti kAraka mapping
Frame.<case>= w_i
Break
else
last char of w_i = x
verb_xmatch = s_i from VDB such that last_char(s_i) = x
verb_suffix= suffix in verb_xmatch that appear in word w_i as suffix
if verb_suffix != NULL ***then***
for each s_i € verb_suffix ***do***
(base, s_i)=splitter(w_i s_i)
If base € vlist ***then*** frame.action = w_i
end for
endif
if match not found in VDB, ***then*** check in NDB
noun_xmatch = all suffixes s_i from NDB such that last_char(s_i)=x
noun_suffix = all suffix in noun_xmatch that appear in word w_i as suffix
num_set = respective number of matched suffix
if noun_suffix != NULL ***then***
for each s_i € noun_suffix and num_i € num_set
(base, suffix)=splitter(w_i $s_{i,}$, $num_{i,}$)
if base € nlist ***then***
Identify the vibhakti of the word
If class =1 ***then*** one value of *vibhakti* is obtained ***else***
If class=2 ***then*** one value of *vibhakti* is obtained ***else***
If class=3 ***then*** more than one value of *vibhakti* is obtained
Call for conflict resolution using NN
Perform *vibhakti-kAraka* mapping
Store the *kAraka* value as semantic role of the word
frame.<case> = w_i
end for
endif
Return frame

Results of the algorithm is discussed in last section, here conflict cases under class II are resolved using neural network. , discussed in next section.

5. CONFLICT RESOLUTION USING NN

For nouns in Sanskrit, intragroup redundancy of type III can be resolved using either statistical methods or NN based method. Statistical techniques require large corpus of data, due to lack of large size data with good vocabulary coverage, NN is implemented for conflict resolution. Back propagation algorithm with three layers is used to train the system for conflict cases. A set of pre annotated text is prepared which contain the suffix, category and *vibhakti* for each word of sentence. Sample of the annotated text is presented in Fig 6.

tvam/am.p.1 bhojanam/am.n.2 pachasi/si.v.x.pach|
devH/H.n.1 vanam/am.n.2 gachchhati/ti.v.x.gachch|
hariH/H.n.1 putrAya/Aya.n.4 bhojanahm/am.n.2
pachati/ti.v.x.pac|
rathavAhakH/H.n.1 aShvebhyH/ebhyH.n.4 ghAsam/am.n.2
anayati/ti.v.x.anaya|
idam/am.p.1 chAtrasya/asya.n.6 pustakam/am.n.2 asti/ti.v.x.as|
devH/H shakten/en.n.7 gRAmam/am.n.2

Figure 6: Sample annotated data

Most common ambiguous cases for *vibhakti* or *kAraka* value fall under four main domain (1,2),(1,2,7),(3,4,5) and(4,5). NN takes features of corpus as input and final *vibhakti* or *kArka* value as output. Features selected for training network is given in Table 6

Parameter	**Feature type**
Candidate suffix	Morphological feature
Candidate word category	Syntactic feature
Verb suffix	Morphological feature
Verb prefix	Morphological feature
Verb root	Lexical feature
Successive word	Context based feature
Previous word	Context based feature
Probability vector for suffix	Corpus based feature

Table 6: Feature selected for NN training

A NN based system takes the input in numerical form; hence the word features are converted into suitable numerical value. Mapping of features into numerical values is shown in Table 7.
Input coding algorithm reads the pre annotated text and generates the data for training the neural network.
BPN is feed forward multi layer network consisting of mainly three layers. Algorithm uses two passes - forward and backward pass. In the forward pass, inputs are multiplied by respective weight and a bias added to it. Weighted sum of input along with bias is fed as input to hidden layer. Hidden layer uses a squashing function to limit the output value in desired range. Output from hidden layer is multiplied by respective weight and fed to output layer. Sigmoid function is used to limit the range of output. Output obtained is compared with actual value and error is calculated as difference of the two. Error is a measure of difference between actual and the desired output. This calculated error is propagated back in the backward pass. To improve the performance of the network, weights are modified as a function of propagated error. In the forward pass, weights of the directed links remain unchanged at each processing unit of the hidden layer. For n input values weighted sum is obtained and sigmod function is applied to this weighted sum. Time taken to train the network is directly proportional to size of data. If number of neurons is increased, training time increases. Classification of *pAninI kAraka* with NN require large size corpus for training. Hybrid model overcomes the problem of large training time by classifying the word with their *vibhakti* value in non-conflicting situations and applying NN under conflicting situations only. This requires a small set of data and network is trained for conflicting classes only, thereby reducing the time. All the cases under same conflicting domain require same network. Major conflicting domains are (1, 2) (3, 4, 5) (6, 7).As data set for each conflicting case focuses on limited set of suffix, small data size is sufficient. For example *am* ending suffix fall in two class 1 and 2; NN was trained on these cases and result of the cases is presented next section.

Category	
1	Pronoun
2	Noun
3	Verb
4	Avyaya
5	Adverb

Prefix(verb)	
1	up
2	anu
3	adhi
4	a
5	abhini

Verb root			
1	gachch	11	dhuh
2	pach	12	yAch
3	kar	13	danD
4	bhu	14	rudh
5	anay	15	pricchch
6	As	16	Chi
7	pashy	17	bru
8	vas	18	Ji
9	dly	19	math
10	yachch	20	suSh

Vibhakti-karka	
1	Agent
2	Object
3	Inst
4	Cause
5	From
6	Relation
7	Location

Suffix(verb)	
1	ti
2	si
3	at
4	Ami
5	anti
6	AmH
7	te
8	tH

Suffix(noun)	
0	No suffix
1	Am
2	H
3	Aya
4	ebhyH
5	Asya
6	en
7	e

Table 7: Sample set of encoded values

6. CONFLICT RESOLUTION USING NN

For nouns in Sanskrit, intragroup redundancy of type III can be resolved using either statistical methods or NN based method. Statistical techniques require large corpus of data, due to lack of large size data with good vocabulary coverage, NN is implemented for conflict resolution. Back propagation algorithm with three layers is used to train the system for conflict cases. A set of pre annotated text is prepared which contain the suffix, category and *vibhakti* for each word of sentence. Sample of the annotated text is presented in Fig 7.

tvam/am.p.1 bhojanam/am.n.2 pachasi/si.v.x.pach|
devH/H.n.1 vanam/am.n.2 gachchhati/ti.v.x.gachch|
hariH/H.n.1 putrAya/Aya.n.4 bhojanahm/am.n.2 pachati/ti.v.x.pac|
rathavAhakH/H.n.1 aShvebhyH/ebhyH.n.4 ghAsam/am.n.2
anayati/ti.v.x.anaya|
idam/am.p.1 chAtrasya/asya.n.6 pustakam/am.n.2 asti/ti.v.x.as|
devH/H shakten/en.n.7 gRAmam/am.n.2 gachchhati/ti.v.x.gachch|
sH/H.p.1 mitre/e.n.7 vishvAsam/am.n.2

Figure 7: Sample annotated data

Most common ambiguous cases for *vibhakti* or *kAraka* value fall under four main domain (1,2),(1,2,7),(3,4,5) and(4,5). NN takes features of corpus as input and final *vibhakti* or *kArka* value as output. Features selected for training network is given in Table 8
A NN based system takes the input in numerical form; hence the word features are converted into suitable numerical value. Mapping of features into numerical values is shown in Table 9.

Parameter	Feature type
Candidate suffix	Morphological feature
Candidate word category	Syntactic feature
Verb suffix	Morphological feature
Verb prefix	Morphological feature
Verb root	Lexical feature
Successive word	Context based feature
Previous word	Context based feature
Probability vector for suffix	Corpus based feature

Table 8: Feature selected for NN training

Category	
1	Pronoun
2	Noun
3	Verb
4	Avyaya
5	Adverb

Prefix(verb)	
1	up
2	anu
3	adhi
4	a
5	abhini

Verb root			
1	gachch	11	dhuh
2	pach	12	yAch
3	kar	13	danD
4	bhu	14	rudh
5	anay	15	prichch
6	As	16	Chi
7	pashy	17	bru
8	vas	18	Ji
9	dIy	19	math
10	yachch	20	suSh

Vibhakti-karka	
1	Agent
2	Object
3	Inst
4	Cause
5	From
6	Relation
7	Location

Suffix(verb)	
1	ti
2	si
3	at
4	Ami
5	anti
6	AmH
7	te
8	tH

Suffix(noun)	
0	No suffix
1	Am
2	H
3	Aya
4	ebhyH
5	Asya
6	en
7	e

Table 9: Sample set of encoded values

Input coding algorithm reads the pre annotated text and generates the data for training the neural network.

BPN is feed forward multi layer network consisting of mainly three layers. Algorithm uses two passes - forward and backward pass. In the forward pass, inputs are multiplied by respective weight and a bias added to it. Weighted sum of input along with bias is fed as input to hidden layer. Hidden layer uses a squashing function to limit the output value in desired range. Output from hidden layer is multiplied by respective weight and fed to output layer. Sigmoid function is used to limit the range of output. Output obtained is compared with actual value and error is calculated as difference of the two. Error is a measure of difference between actual and the desired output. This calculated error is propagated back in the backward pass. To improve the performance of the network, weights are modified as a function of propagated error. In the forward pass, weights of the directed links remain unchanged at each processing unit of the hidden layer. For n input values weighted sum is obtained and sigmod function is applied to this weighted sum.

Time taken to train the network is directly proportional to size of data. If number of neurons is increased, training time increases. Classification of *pAninI kAraka* with NN require large size corpus for training. Hybrid model overcomes the problem of large training time by classifying the word with their *vibhakti* value in non-conflicting situations and applying NN under conflicting situations only. This requires a small set of data and network is trained for conflicting classes only, thereby reducing the time. All the cases under same conflicting domain require same network. Major conflicting domains are (1, 2) (3, 4, 5) (6, 7).As data set for each conflicting case focuses on limited set of suffix, small data size is sufficient. For example *am* ending suffix fall in two class 1 and 2; NN was trained on these cases and result of the cases is presented next section.

7. RESULT AND DISCUSSION

Training time for each conflicting domain is reported in Table 10.

***pAninI* class**	**F**	**n**	**C**	**P**	**R**	**k**
Karta	122	110	108	0.981	0.885	0.930
Karma	64	61	58	0.951	0.906	0.931
Karan	52	50	49	0.980	0.942	0.957
sampradAn	35	30	29	0.966	0.828	0.891
apAdAn	34	30	32	0.93	0.823	0.873
Adhikaran	38	35	33	0.943	0.868	0.903

Table 10: Performance of NN for Various Data Set Training Size=50 Test Data Size=10

Fifty sentences used in training phase and ten sentences in testing phase. As depicted in the Table 6.1; 90% accuracy is achieved in *am* and *ebhyH* domain. Training of network for *am abhyam* conflicting case is given in Fig 8 and Fig 9.

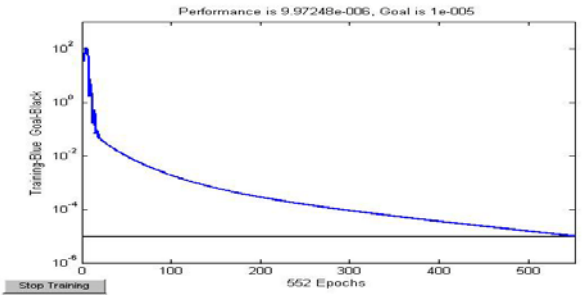

Figure 8: Training graph for *am* data set

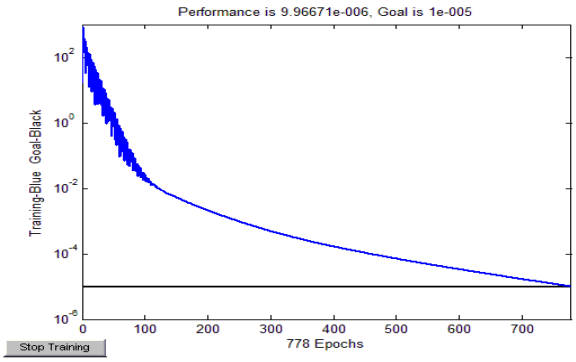

Figure 9: Training data for *abhyam* data set

After training the network for conflicting cases; algorithm is tested on 100 sentences and accuracy of the output obtained is calculated by finding the F-score as given in Eq 6.1

$$F\_score = 2(p \times r)/(p+r) \quad \text{---------- (6.1)}$$

It uses precision (p) and recall (r) to compute the score.

Precision (p) = Number of correct result / Number of returned result

Recall (r) = Number of correct result / Number of results that should have been returned.
F_score is understood as weighted average of precision and recall and is calculated as given in Eq (6.1). F-Score of each class under NN is given in Table 11.
f= frequency of occurrence
n= number present in the data
c=number correctly identified

Suffix	Conflicting	Training time	Epochs	Correctly identified
am	1, 2	09.89	125	9
Abhyam	3, 4, 5	11.27	210	8
ebhyH	4, 6	08.54	116	9
e	1, 2, 7	10.32	198	7

Table 11: Result for Hybrid Classification

Hybrid model is better approach for semantic classification as compared to pure rule based system or NN based system. Performance of NN is dependent on the size of annotated corpus available for training with good coverage of the vocabulary and suffix. As it is suffix driven analysis, annotated corpus must include the suffix attached to the words. Due to lack of availability of corpus, with good coverage, results of pure NN based system lags behind hybrid system. Hybrid model exploits the potential of rules of the grammar and handles conflicting situation by implementing NN model. Requirement of large size corpus is reduced as corpus is designed for conflicting cases only. Pure rule base system require in depth knowledge and understanding of complex set of recursive and meta rules for transformation and exceptional cases. A person with good computational skill along with complete *pAninian* knowledge is difficult to achieve.
Case frames for 100 sentences are generated by three algorithms and accuracy is checked at word level and sentence level. Word level accuracy is correctness of semantic tag assigned to each word and sentence level accuracy is correctness of generated case frame. Word level accuracy is discussed in Table 5.3, 5.6 and 5.8. For sentence level accuracy, accuracy of case frame is calculated. Accuracy of case frame depends upon two parameters:-

- Number of significant words from a sentence appearing in case frame ---(x)
- number of words tagged correctly ----(y)

x \ y	100%	50-100%	50%	Total
100%	56	11	5	72
50 -100%	12	5	2	19
< 50%	8	1	--	9

For hybrid model, sentence level accuracy is calculated and presented in Table 12.

Table 12.: Case Frame Accuracy for 100 Sentences under Hybrid Model

Out of 72 CF with all significant word of sentences; 56 are correctly tagged giving the accuracy of 77 % which is so far the best as none of the NLP processor for Sanskrit language has worked on KR tool generation for Sanskrit language.
Use of NN in NLP is less frequent due to complexity prevailing in both domains [1]. Sanskrit language has rich inflectional morphological structure suitable for computational processing. Tabular declension of words with syntactic-semantic significant suffix occupying predefined cell position drives the path for well structure knowledge representation mechanism. Identifying the semantic class of the word with suffix driven analysis under *pAninI* concept was the main theme of the work. Use of NN for resolving conflicting *kAraka* role under *pAninI* framework appears to be a better mechanism for semantic labeling of words. Initial identification is a baseline model upon which further extensions can be developed. Enhanced corpus with better coverage can further improve the results.

REFERENCES

[1]. Babu, A. Suresh, K. & Pavan, P.N.V.S. 2010. *Comparing Neural Network Approach With N-Gram Approach For Text Categorization.* International Journal on Computer Science and Engineering. 2(1): 80-83.

[2]. Bharati, A. & Kulkarni, A. 2007. *Sanskrit and Computational Linguistic*. First International Sanskrit Computational Symposium. Department of Sanskrit Studies, University of Hyderabad.

[3]. Bharati, A., Kulkarni, A. & Sivaja S. N. 2008. *Use of Amarako´sa and Hindi WordNet in Building a Network of Sanskrit Words.* 6th International Conference on Natural Language Processing.-ICON-08.Macmillan Publishers. India.

[4]. Brants, T. 2000. *TnT - A Statistical Part Of Speech Tagger*. Proceedings of the 6th Conference on Applied Natural Language Processing 2000.

[5]. Briggs, R. 1995. *Knowledge Representation in Sanskrit and Artificial Intelligence*. AI Magazine, 6 (1): 32-39.

[6]. Dwivedi, K. (padamshri) 2002. *Prarambhik RachanAnuvAdakumaudi*. VishavavidyAlaya PrakAshan. Varanasi. 19th ed. ISBN:81-7124-86-0

[7]. Hellwig, O. 2007. *SanskritTagger, a stochastic lexical and POS tagger for Sanskrit.* First International Sanskrit Computational Linguistics Symposium. LNCS Springer. 5402:266-277.

[8]. Huet, G. 2003. *Towards Computational Processing of Sanskrit.* Recent advances in Natural Language Processing. Proceedings in International conference ICON.:1-10

[9]. Hyman D. M. 2009 *From pAninian sandhi to finite state calculus.* Sanskrit Computational Linguistics Springer-Verlag.

[10]. Jha, G. & Chandrashekar, R. 2010. Annotation Guidelines for tagging Sanskrit using MSRI-JNU Sanskrit tagset http://sanskrit.jnu.ac.in/corpora/ (Browsing Date: 25th Dec 2011).

[11]. Kak, S. C. 1987. *The panian approach to natural language processing.* International Journal of Approximate Reasoning. Elsvier Publishing. 1(1):117-130.

[12]. Kiparsky, P. 2002. *On the architecture of P¯anini's grammar*. International Conference on the Architecture of Grammar.

[13]. Kumar, D. & Josan, G. 2010. *Part of Speech Taggers for Morphologically Rich Indian Languages: A Survey*. International Journal of Computer Applications. 6(5): 32-41.

[14]. Mittal, V. 2010. *Automatic Sanskrit Segmentizer Using Finite State Transducers.* LRTC, Research Centre. IIIT-Hyderabad. Proceedings of the ACL 2010:85-90.

[15]. Pedersen, M., Eades, D., Amin, S. K. & Prakash, L. 2004. *Relative Clauses in Hindi and Arabic: A Paninian Dependency Grammar Analysis* in COLING.

[16]. Rao, B. N. 2005. *Panini and Compuer Science – into the future with knowledge from past.* A Sourcebook:9- 13.

[17]. Selot, S. & Singh J. 2007. *Knowledge representation and Information Retrieval in PAninI Grammar Framework.* International Conference ICSCIS- 2007. 2: 45-51.

[18]. Selot, S., Tripathi, N. & Zadgaonkar, A. S. 2009. *Transition network for processing of Sanskrit text for identification of case endings*" icfai Journal of Computer Science 3(4):32-38.

[19]. Shan, H. & Gildea, D. 2004. *Semantic labelling by Maximum entropy model.* University of Rochester Technical Reort 847.

[20]. Timothy, J. D., Hauser M. & Tecumseh, W. 2005. *Using mathematical models of language experimentally* TRENDS in Cognitive Sciences 9(6):284-289.

[21]. Vaidya, A., Husain, S., Mannem, P. & Misra, D. S. 2009. *A Karaka Based Annotation Scheme for English* Language Technologies Research Centre. IIIT Hyderabad. Springer-Verlag: LNCS 5449:41–52.