Machine Translation from Kannada to Telugu

Greeshma Karanth
Computer Science
PES University
Bangalore, India
greeshmakaranth.13@gmail.com

Shivangi Gupta
Computer Science
PES University
Bangalore, India
shivangig078@gmail.com

K Suhaas
Computer Science
PES University
Bangalore, India
suhaask.1999@gmail.com

Vishal Kanteppa
Computer Science
PES University
Bangalore, India
vishalkanteppa@gmail.com

Abstract*— *This project aims to translate Kannada text to Telugu text by using structural probe machine translation technique.

Keywords—machine translation, regional languages, structural probe, parse tree

I. INTRODUCTION

Machine Translation pertains to translation of one natural language to other by using automated computing. The main objective is to fill the language gap between two different languages speaking people, communities or countries. In India, we have multiple and hugely diverse languages and scripts, hence scope and need of language translation is immense.

Machine Translation (MT) is a sub-field of computational linguistics that investigates the use of computer software to translate text or speech from one natural language to another. At its basic level, MT performs simple substitution of words in one natural language for words in another. Machine Translation system are needed to translate literary works which from any language into native languages. The literary work is fed to the MT system and translation is done. Such MT systems can break the language barriers by making rich sources of literature available to people across the world. MT also overcomes technological barriers. Most of the information available is in English. This has led to digital divide in which only small section of society can understand the content presented in digital format. MT can help in this regard to overcome the digital divide.

In recent years, machine translation software has increasingly been integrated into our daily lives. People routinely use machine translation for various applications. One of the key advantages of machine translation is speed. A computer program can translate tremendous amounts of content quickly. Human translators, on the other hand, are more accurate but take much longer to complete the same amount of work. Machine translation is also less expensive than using a human translator. The key here is to save that expertise for when it's actually needed. If all you need is a rough translation, machine translation can get you the information you need much more cheaply than a human translator.

MT tools are often used to translate vast amounts of information involving millions of words that could not possibly be translated the traditional way. The quality of MT output can vary considerably; MT systems require "training" in the desired domain and language pair to increase quality.

II. IMPLEMENTATION

The first stage of this project was to build the structural probe (or obtain the transformation matrix B) from the BERT (Bidirectional Encoder Representations from Transformers) embeddings of an English sentence. This transformation matrix converts the Euclidean distances between words to syntax tree distances. When the structural probe is applied to any English sentence, it can generate the syntax tree for that sentence. A different transformation matrix can be developed for different languages, thus giving us the ability to generate parse trees for a sentence in any language. Using the parse trees, a translation of a particular sentence from one word to another can be found. This is the basic implementation of the MT project.

A word embedding indicates the representation of the given word in an n-dimensional vector space. The words are all converted to points in this vector space and then, based on the similarity between points – calculated as the distance between them, word translations between languages can be found. There

are many modules available to convert words to vectors, popular of which are Word2Vec, GloVe etc. The aforementioned two convert an entire corpus to embedding based on the literal meaning of the words. In this project, BERT was chosen because BERT does the embedding sentence wise and not one entire corpus at a time. It also takes into account the context of the sentence before generating the embedding, thus providing a more accurate translation.

Gradient Descent

In order to arrive at a structural probe for a language, we need to apply gradient descent on a rough approximation of the matrix.

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient (or approximate gradient) of the function at the current point. If, instead, one takes steps proportional to the *positive* of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.

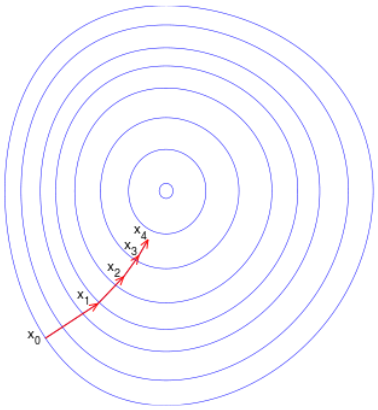

Fig 1. Illustration of gradient descent on a series of level sets

The Structural Probe

In this section we provide a description of our proposed structural probe, first discussing the distance formulation. Let M be a model that takes in a sequence of n words $w^l_{1:n}$ and produces a sequence of vector representations $h^l_{1:n}$, where ` identifies the sentence. Starting with the dot product, recall that we can define a family of inner products, $h^T Ah$, parameterized by any positive semidefinite, symmetric matrix $A \in S^{m\times m}$. Equivalently, we can view this as specifying a linear transformation $B \in R^{k\times m}$, such that $A = B^T B$. The inner product is then $(Bh)^T (Bh)$, the norm of h once transformed by B. Every inner product corresponds to a distance metric. Thus, our family of squared distances is defined as:

$$d_B(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)^2 = \left(B(\mathbf{h}_i^\ell - \mathbf{h}_j^\ell)\right)^T \left(B(\mathbf{h}_i^\ell - \mathbf{h}_j^\ell)\right)$$

where i, j index the word in the sentence. The parameters of our probe are exactly the matrix B, which we train to recreate the tree distance between all pairs of words (w^l_i , w^l_j) in all sentences T^l in the training set of a parsed corpus. Specifically, we approximate through gradient descent:

$$\min_B \sum_\ell \frac{1}{|s^\ell|^2} \sum_{i,j} \left| d_{T^\ell}(w_i^\ell, w_j^\ell) - d_B(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)^2 \right|$$

where $|s^l|$ is the length of the sentence; we normalize by the square since each sentence has $|s^l|^2$ word pairs.

In human languages, the meaning of a sentence is constructed by composing small chunks of words together with each other, obtaining successively larger chunks with more complex meanings until the sentence is formed in its entirety.

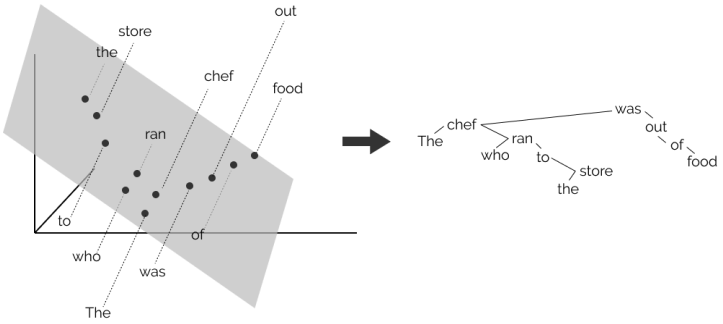

Fig 2. Finding Syntax with Structural Probes

The order in which these chunks are combined creates a tree-structured hierarchy like the one in the picture above, which corresponds to the sentence *the chef who ran to the store was out of food.* Note in this sentence that *the store* is combined eventually with *chef*, which then is combined with *was*, since it is the chef who was out of food, not the store. We refer to each sentence's tree-structured hierarchy as a *parse tree*, and the phenomenon broadly as *syntax*.

In recent years, however, neural networks used in NLP have represented each word in the sentence as a real-valued vector, with no explicit representation of the parse tree. To these networks, our example sentence looks like the image below (though instead of three-dimensional vectors, they're more like one thousand dimensions.) By finding the right linear transformation of the points, it can be found that the tree constructed by connecting each word to the word closest to it

approximates the human parse tree that can be drawn [1].

Approach

Using BERT, embeddings for a sentence are obtained. Utilizing these embeddings and pre-trained syntax trees, the transformation to convert the Euclidean distances to syntax or parse tree distances can be applied.

Methodology

We first initialize an m*k matrix to represent the B matrix with random values between 0 and 1. Then we use the cost function to calculate the gradient with respect to each of the elements in the B matrix to perform gradient descent on the matrix.

$$\frac{\partial(cost)}{\partial B_{ab}} = \sum_{i}^{sl}\sum_{j}^{sl} 2\,(\sum_{x}^{k} B_{ax}(h_{ix} - h_{jx}))\,(h_{ib} - h_{jb})$$

III. PROGRESS

```
def vect_sub(x1,x2,n):
        subs=[]
        for i in range(n):
                subs.append(x1[i]-x2[i])
        return subs

def transpose(matrix,m,n):
        newmat=[]
        for i in range(n):
                newmat.append([])
        for i in range(n):
                for j in range(m):

        newmat[i].append(matrix[j][i])
        return newmat

def tree_dist(x1,x2,syntax_tree,sentence_length):
        subvector=[]
        dist=0
        for i in range(sentence_length-1):

        subvector.append(syntax_tree[x1][i]-
syntax_tree[x2][i])
        for i in subvector:
                if(i):
                        dist=dist+1
        return dist

def transform_dist(x1,x2,B_mat,m,n):
        subt=[]
        subt.append(vect_sub(x1,x2,n))
        subtT=transpose(subt,1,n)
        dist=np.dot(transpose(np.dot(B_mat,subtT)
,m,1),np.dot(B_mat,subtT))
        return dist
```

```
def gradient(a,b,B_mat,m,n,sl,word_vectors):
        summations=[]
        for i in range(sl):
                for j in range(sl):
                        summation=0
                        for x in range(n):

        summation+=abs(B_mat[a][x]*(word_vectors[
i][x]-word_vectors[j][x]))

        summations.append(2*summation*(word_vecto
rs[i][b]-word_vectors[j][b]))
        return sum(summations)

def
gradient_descent(word_vectors,syntax_tree,B_mat,al
pha,iterations,m,n):
        sl=len(word_vectors)
        while(True):
                e=0
                for i in range(sl):
                        for j in range(sl):

        e+=abs(tree_dist(i,j,syntax_tree,sentence
_length)-
transform_dist(word_vectors[i],word_vectors[j],B_m
at,m,n))
                e=e/sl**2
                print(e)
                for i in range(m):
                        for j in range(n):

        #print(B_mat[i][j])

        B_mat[i][j]=B_mat[i][j] -
alpha*(gradient(i,j,B_mat,m,n,sentence_length,word
_vectors))
        pass
```

IV. FUTURE SCOPE

Currently, the gradient descent algorithm that has been implemented does not converge to give us the local minima. In fact, the system seems to reach a local minima and then proceed to move away from it. After fixing this overshooting of the local minimum, we can get a fully functional structural probe. From the transformation trained, other syntax trees can be generated for different languages. This can be used to translate text between two languages.

ACKNOWLEDGEMENT

To our dedicated guide – Dr. Dinkar Sitaram; for mentoring us through the course of this project.
To our teacher – Dr. KV Subramaniam; for providing us with sufficient knowledge to undertake this project confidently.

REFERENCES

[1] John Hewitt, Christopher D. Manning, “A Structural Probe for Finding Syntax in Word Representations.”