**Introduction to Romanic Bangla (Banglish) Natural Language Processing**

Motivation

Consider one old school traditional software engineering problem where you might have lots of lots of names and lots of lots of numbers and those names are hopefully sorted alphabetically(in ascending order) from A through Z in a book like this, even though most of us don’t really reach for this technology anymore but consider that it’s really the same as your iPhone or Android phone or other device which has all of your contacts top to bottom and you can scroll through them from A to Z or you can search for them by typing into the little autocomplete box. How is even your phone solving this problem? Well let’s consider a simple approach, suppose we want to find the name “mobassir” from our phone book, now “we can iterate through whole list and check where our desired person name appears”. This approach is correct but slow and the name of this algorithm is **linear search**. Or for making this search process faster we can take other algorithm like instead of traversing whole list from top to bottom we can directly jump into the middle of the list (as the list is sorted in ascending order) and then check if the first character of our present mid index contains M or not, for example say at midpoint we find the name “jack” where first character is ‘j’.so we understand that mobassir is ahead of jack(as m comes after j) and we don’t have to look back anymore. Now we can make our present mid index as starting index and exclude the first half of our list as we know now that mobassir name is after jack so we don’t need first half of the array/list anymore for this search. Again we find the mid position of our new list and check if ‘M’ is ahead or behind, by following **Divide and Conquer** strategy like this, within few iteration we get the name “mobassir” from our large list that contains all the phone records.this algorithm is faster compared to linear search and the name of this algorithm is **Binary Search.** These are the “Classical Stack” of **software 1.0** that we all **Software Engineers (SWE)** familiar with. It consists of explicit instructions to the computer written by a programmer. By writing each line of code, the programmer identifies a specific point in program space with some desirable behavior.

Now Consider a different problem where we have some sentences, we need to find sentiment for each of them for understanding if a particular sentence contains positive sentiment or negative sentiment. Now you can’t use any heuristic (rule-based) system for approaching this problem and you can’t provide explicit instructions to the computer for tacking this issue. There are a lot of positive, negative words available, each word is different in different languages, how many **if else** or explicit instructions you can provide for tackling problem like this? This is when we use [**Software 2.0**](https://medium.com/@karpathy/software-2-0-a64152b37c35) **which** can be written in much more abstract, human unfriendly language, such as the weights of a neural network. No human is involved in writing this code because there are a lot of weights (typical networks might have millions)

As[Andrej Karpathy](http://karpathy.github.io/) said, **“Software 1.0 is code we write. Software 2.0 is code written by the optimization based on an evaluation criterion (such as “classify this training data correctly”). It is likely that any setting where the program is not obvious but one can repeatedly evaluate the performance of it (e.g. — did you classify some images correctly? do you win games of Go?) will be subject to this transition, because the optimization can find much better code than what a human can write.”**



Machine Translation (Multilingual NLP):

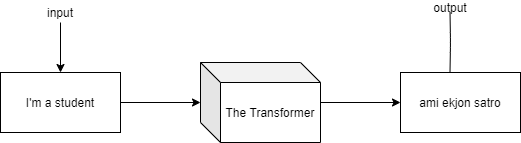
**Machine Translation** has usually been approaches with phrase-based statistical techniques, but neural networks are quickly becoming dominant. Some famous [Cross Lingual Models](https://arxiv.org/abs/1901.07291) were trained for solving Multilingual Machine Translation Problems, where a single model translates from any source language to any target language, and in weakly supervised (or entirely [unsupervised](https://arxiv.org/abs/1710.11041)) settings.

**How Can We Approach Banglish (Romanic Bangla) Machine Translation and Document Classification problems? :**

**According to Andrew Ng, one of the pioneers of today’s world in promoting Artificial Intelligence,** **“**[**Transfer Learning**](https://machinelearningmastery.com/transfer-learning-for-deep-learning/#:~:text=Transfer%20learning%20is%20a%20machine,model%20on%20a%20second%20task.) **will be the next driver of ML success”.**

For approaching various pure Bengali natural language processing task we have something like [BNLP toolkit](https://sagor-sarker.medium.com/bengali-natural-language-processing-toolkit-e1a4d2d2f182) but for problems like Bengali Romanized, misspelled (phonetically incorrect) Bengali Romanized, misspelled (phonetically incorrect) Bengali we are lacking resources and support.

Cross lingual models are based on several key concepts, transformers is one of them. The Transformer architecture is at the core of almost all the recent major developments in NLP.It introduced an attention mechanism that processes the entire text input simultaneously to learn contextual relations between words (or sub-words). A Transformer includes two parts — an encoder that reads the text input and generates a lateral representation of it (e.g. a vector for each word), and a decoder that produces the translated text from that representation. Let’s begin by looking at the model as a single black box. In a machine translation application, it would take a sentence in one language (for example: in English), and output its translation in another (for example: in Romanic Bangla). For better understanding please check the figure attached:



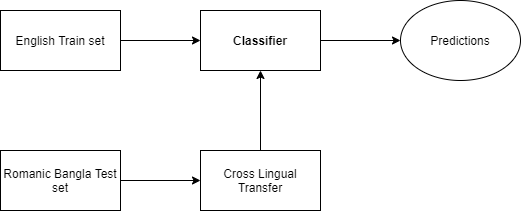
The paper [**Cross-lingual Language Model Pretraining**](https://arxiv.org/abs/1901.07291) presents two innovative ideas — a new training technique of BERT for multilingual classification tasks and the use of BERT as initialization of machine translation models.

[XLM-R: State-of-the-art cross-lingual understanding through self-supervision model](https://ai.facebook.com/blog/-xlm-r-state-of-the-art-cross-lingual-understanding-through-self-supervision/)

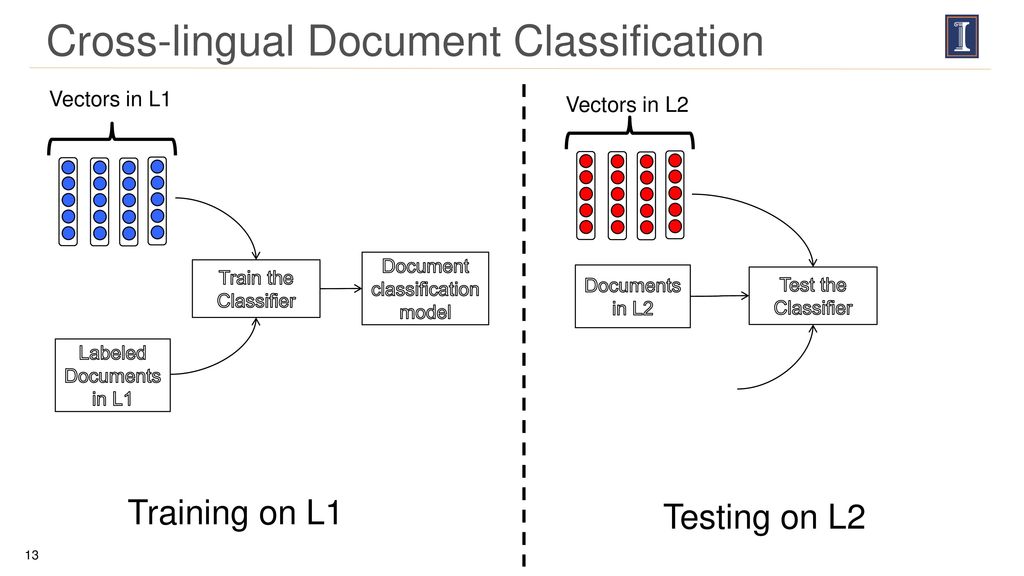
handles the following 100 languages: Afrikaans, Albanian, Amharic, Arabic, Armenian, Assamese, Azerbaijani, Basque, Belarusian, **Bengali, Bengali Romanized**, Bosnian, Breton, Bulgarian, Burmese, Burmese, Catalan, Chinese (Simplified), Chinese (Traditional), Croatian, Czech, Danish, Dutch, English, Esperanto, Estonian, Filipino, Finnish, French, Galician, Georgian, German, Greek, Gujarati, Hausa, Hebrew, Hindi, Hindi Romanized, Hungarian, Icelandic, Indonesian, Irish, Italian, Japanese, Javanese, Kannada, Kazakh, Khmer, Korean, Kurdish (Kurmanji), Kyrgyz, Lao, Latin, Latvian, Lithuanian, Macedonian, Malagasy, Malay, Malayalam, Marathi, Mongolian, Nepali, Norwegian, Oriya, Oromo, Pashto, Persian, Polish, Portuguese, Punjabi, Romanian, Russian, Sanskri, Scottish, Gaelic, Serbian, Sindhi, Sinhala, Slovak, Slovenian, Somali, Spanish, Sundanese, Swahili, Swedish, Tamil, Tamil Romanized, Telugu, Telugu Romanized, Thai, Turkish, Ukrainian, Urdu, Urdu Romanized, Uyghur, Uzbek, Vietnamese, Welsh, Western, Frisian, Xhosa, Yiddish.

As we can see that the model already knows **Bengali and Bengali Romanized** so we can use cross lingual model like this to solve some Romanic Bangla document classification problem

In The process of cross-lingual document classification, we assume that the opinion units have already been determined. The English train set is used to train a classifier. The Banglish/Romanic Bangla test set is mapped accordingly and the classifier is tested on this cross-lingual test set. Check the pictures attached for better understanding:



**In picture below, L1 means language 1 and L2 means language 2.**



For simple [multilingual binary document classification problem](https://www.kaggle.com/mobassir/understanding-cross-lingual-models) using tf 2.x we can use the code snippet below to use model like [xlm Roberta large from popular huggingface library](https://huggingface.co/transformers/model_doc/xlmroberta.html)

maxlen = 192

MODEL = 'jplu/tf-xlm-roberta-large'

def build\_model(transformer, loss='binary\_crossentropy', max\_len=512):

input\_word\_ids = Input(shape=(max\_len,), dtype=tf.int32, name="input\_word\_ids")

sequence\_output = transformer(input\_word\_ids)[0]

cls\_token = sequence\_output[:, 0, :]

x = tf.keras.layers.Dropout(0.3)(cls\_token)

out = Dense(1, activation='sigmoid')(x)

model = Model(inputs=input\_word\_ids, outputs=out)

model.compile(Adam(lr=3e-5), loss=loss, metrics=[tf.keras.metrics.AUC()])

return model

with strategy.scope():

transformer\_layer = transformers.TFXLMRobertaModel.from\_pretrained(MODEL)

model = build\_model(transformer\_layer,loss='binary\_crossentropy', max\_len=maxlen)

model.summary()

Now if we want to deploy such custom tf 2.x trained models using aws sagemaker then we can follow this step by step process tutorial [**Deploy trained TensorFlow 2.x models using Amazon SageMaker**](https://github.com/mobassir94/Deploy-trained-TensorFlow-2.x-models-using-Amazon-SageMaker)

References:

1. [**https://medium.com/@karpathy/software-2-0-a64152b37c35**](https://medium.com/@karpathy/software-2-0-a64152b37c35)
2. [**https://www.kaggle.com/mobassir/understanding-cross-lingual-models**](https://www.kaggle.com/mobassir/understanding-cross-lingual-models)
3. [**https://ai.facebook.com/blog/-xlm-r-state-of-the-art-cross-lingual-understanding-through-self-supervision/**](https://ai.facebook.com/blog/-xlm-r-state-of-the-art-cross-lingual-understanding-through-self-supervision/)
4. [**https://arxiv.org/abs/1911.02116**](https://arxiv.org/abs/1911.02116)
5. [**https://sagor-sarker.medium.com/bengali-natural-language-processing-toolkit-e1a4d2d2f182**](https://sagor-sarker.medium.com/bengali-natural-language-processing-toolkit-e1a4d2d2f182)