# Speech to Speech Translation

By Bryant McArthur and Wade McMillan

Linguistics 581

Natural Language Processing

**Abstract:**

The main goal of this project was to find a way to implement a speech to speech translation method and become involved to understand each level of the process as we could. In this paper we will discuss related research and other similar projects, frameworks and tools used, things we built, as well as results, and opportunities for further research.

**Related Work:**

2014 Microsoft Segmentation and Disfluency Removal for Conversational Speech Translation:

This paper is about how Microsoft was able to overcome disfluencies in speech to improve real-time conversational speech to speech translation systems. They proposed a new way to remove disfluencies in two steps as opposed to the conventional one-step method. This method proved to increase the quality of the speech translation by as much as 3 Bleu points with only 6 second latency to look ahead.

2019 Google Leveraging Weakly Supervised Data to Improve End-to-End Speech-to-Text MT:

This paper compared and contrasted the pros and cons of end-to-end speech translation verse the cascade of Automatic Speech Recognition, text Machine Translation and text to speech synthesis. One difficulty to speech to speech translation is the lack of data to train a model, but this paper demonstrates how a quality system can be trained using only weakly supervised datasets and synthetic data sourced from unlabeled monolingual speech.

This paper described how without careful methods and a long process the direct end-to-end model will likely not outperform the traditional cascade approach from ASR to text Machine Translation especially with a low resource language.

Because of the complex nature that arises in both disfluencies in natural language and a lack of training speech data to train a model, these articles persuaded us to use the more traditional cascade or waterfall approach to flow from speech to text to translated text and back to speech.

**Question:**

The need for effective speech to speech translations continues to grow as the world becomes increasingly global. Business exists on a worldwide scale but language is a constant barrier. While several systems exist there exists a clear need for improvement in terms of speed, quality, and language variety.

Implementing a Speech to speech translation system can take one of two approaches, a genuine speech to speech in which audio waves are given as the input read directly through a neural network and then output as audio waves. Or there is a waterfall approach in which audio waves are first turned into text, then translated using a standard text to text translation system and finally being turned into audio files at the end. For the scope of this project we decided to use the waterfall approach and explain each step of the process, tools used, and any customizations we made.

**Method:**

Speech to Text:

Converting Speech to Text is the first step in the process and will carry through all the continuing steps of the process. This makes quality on this step essential if our output is to have any meaningful value. Thankfully there are a variety of systems that exist that enable this operation. We selected to use Microsoft Azure's existing version for our project.

This made this step relatively simple as they have powerful built-in commands that make it easy to use. It is subscription based so a subscription key needs to be provided as well as the language it should be listening for as you build one of their Speech Configuration Objects that will then listen for that language.

Two great qualities of Microsoft Azure's ASR system is first, the basic removal of disfluencies and second, the addition of punctuation. Both of these tasks are improved by the use of an N-gram language model built into the system. Microsoft Azure does not implement the same amount of disfluency removal as given in their 2014 paper referenced above, however it does a fine job at removing simple disfluencies from the text to output only what was said. Moreover, with the addition of proper punctuation it improves the translation quality after pushing it through the MT model, especially when multiple sentences are spoken into the system. It does have a major drawback in that it stops listening after any major pauses and then converts it to text and has to be reassessed if you want to process any more speech data.

As part of our project and our commitment to quality we built in a feature that enables users to view the text as the machine heard it and manually edit it if they so desire. This is one way that we decided to incorporate Human Assisted Machine Translation or HAMT into our project.

HAMT is a common practice for ensuring high-quality results usually after an MT system has output results, this is slightly different because we are putting it in the middle of what would be considered our translation process but we found it to be a good fit in usability because of the quality assurance that it gives users and potential users.

Machine Translation:

Originally we just used a variety of existing Machine Translation systems as the middle, but to increase involvement we decided to build our own model for the language pair of Polish and Spanish, selected because of their morphological differences, different language families, and general unrelatedness. Additionally we each speak one of these languages which allows for better human evaluation which is key in determining actual quality of a machine translator.

The first step in creating a machine translation tool is data. We decided to get our data from Opus Corpus, a free bilingual corpora repository. We selected to use their data from TED Talks 2020,

wikipedia translated sentences and the corpus from “MultiPara Crawl” which gave us roughly 800 thousand translation memory pairs before cleaning. This data was relatively clean to begin with, but to be safe we decided to use a cleaning pipeline using Olifant and Rainbow by Okapi. This made it easy to remove duplicates, extra line breaks, uneven quotes, inline tags, unwanted non-word characters, and other common noise in data.

The next step was selecting a development environment and framework for training our model. We decided to do it in Python as it is a leading language for all forms of data science and machine learning with lots of powerful packages. To train a model a GPU is necessary to perform a meaningful number of training steps. In order to achieve this we elected to use Google Colab Pro.

The most important part of this was selecting an appropriate training network. We elected to use OpenNMT. Other common NMT frameworks are Marian, Sockeye, FairSeq, Tensor2Tensor, Nematus and NeuralMonkey.

The two best candidates would be Marian and Sockeye. Marian is a very high quality customizable framework based in C++ used more for large scale research and commercial purposes. Like OpenNMT, Sockeye is also written in python, but attempts to include Self-Attention Transformer, and Convolutional models as opposed to just RNN models.

By using either one of these other candidates we may see a slight increase in translation quality, but that would come at a cost. Both these other systems are more complex to implement than using Py-Torch and OpenNMT’s easy configurations. Also because of their many encoding and decoding layers and various models it would require a large amount of space and time to train these models. For this reason we decided to go with OpenNMT-py.

OpenNMT has lots of different configuration options that we experimented with until we found what we felt was our best fit.

The main basic configurations involve where to load training and validation data and where to save the source and target vocabulary.

We created a model with 6 encoding layers and 6 decoding layers of transformers. Because we were training the data on tokenized sentences rather than normal text we added the parameters, “batch_type: tokens” and “normalization: ‘tokens’” with a batch_size of 4096. This tells the model how many tokens to train on during each step of the training phase. Likewise, we set the valid_batch_size to 4096 as well to keep things consistent.

For the model we also added a dropout rate and attention_dropout rate both at [0.1]. This is standard practice to reduce the risk of substantial overfitting. However because we included this dropout rate it does roughly double the number of iterations required for the model to converge.

For our model_dtype we chose to use “fp32”. This represents the Floating Point format for the activations, weights

and input of the model. Most Machine learning models use FP32 or FP16.

The parameter “optiml: ‘adam’” refers to the optimization method. Adam is a “replacement optimization algorithm for stochastic gradient descent” and very adaptable. Our corresponding learning_rate was set to 2 (which really means .002) in order to minimize the loss function.

Of the OpenNMT decay_methods we chose to use “noam” which works well with Adam. Having a constant learning rate is very difficult so noam normalizes the rate in order to make weight-decay per parameter.

We set our label_smoothing to .1. Label smoothing is a “regularization technique” to prevent the model from predicting output tokens too confidently during training and generalizing too poorly.

We opted to include positional encoding. Position encoding helps by describing the position of the word so that each position is assigned a unique representation. Although Polish and Spanish may grammatically change the position of words, they tend to follow a similar general pattern and are also fairly consistent within their respective domain. Positional encoding is not always a positive inclusion for all language pairs.

There are also several more configurations we included that we did not describe, and many more optional configurations provided by OpenNMT.

To split the data into train, test, and validation sets, we decided to save the bulk of the data for training and set aside 3000 lines for test and validation sets using Sci-kitlearns built in packages on our already cleaned data.

One important part of building our model was deciding on a tokenization model to use. While OpenNMT has several available for use we decided to build our own. The purpose of tokenization is to reduce the number of unknown translations for our model. This is down by breaking words into smaller pieces. We did this by breaking it down to the lemma and then having the endings separated to go into the machine as a separate token. This helps match languages that have varying morphological complexity. Tied with this is also determining how large of a vocab size to use when training the model. Because Polish is much more morphologically complex than Spanish it requires a higher vocabulary size. After testing different vocab sizes, the optimal sizes that worked for our models were 8000 for Spanish and 12,000 for Polish.

In order to evaluate and get a representation of the performance of our model we chose to use BLEU scores. While OpenNMT has the option to stop automatically once the change in BLEU score isn’t significant in the positive direction we elected to not use this option because we found that we had occasional rises and falls in our scores over time. We found that at around 250,000 steps we reached what seemed to be a fairly stable plateau and while it might be a local optimization due to our usage of a base learning rate we were satisfied with the results achieved after this many iterations.

For the other language pairs we elected to continue to use Microsoft's

existing models allowing us to provide high quality output for a variety of languages with one pair being designed and built by us. Microsoft being a multi-billion dollar commercial company heavily invested in machine translation does produce better outputs in general for this language pair. Additionally, our model does have a smaller domain than Microsoft standard models but it still does provide good results and relatively high BLEU scores. Throughout the process we continued to add additional data to try and increase the vocabulary and diversity of our model.

## Text To Speech:

To go from Text to Speech or Speech Synthesis was the last step of our project in terms of going all the way from speech to speech. This task is commonly referred to as Speech Synthesis and to do this task we used Microsoft Azure as a resource and coded it in python, in Pycharm, because unlike our previous task of MT we no longer had need for a GPU and having a stable environment that doesn't time out.

Speech Synthesis is a complex task composed of many parts. First the system has to tokenize the input. This is important because depending on where a word falls in comparison with other words, for example two words might be spelled the same way but because it is being used in a different sense be a different part of speech the pronunciation can vary significantly. This is why it is important that the first step is performed at a high level.

Then after the preprocessing the tokens are converted into phonemes, or a textual approximation of the auditory sound. To finish the process these phonemes are then synthesized into audio by using models that have been trained to approximate features such as pitch, intonation, and other sound qualities. This is the basics of the process of synthesizing text to speech.

Microsoft's Azure resource has several trained models for each language. In order to add valuable customization to our project we added the general male and female voice for each language so that users are able to select a speech output that would be more representative of their speech.

An advanced service is offered wherein those with applications to Microsoft's speech services are able to train their own language onto the model and an approximation of their voice in the other language would be output. This is a very interesting opportunity for further research and advancement of our project that we will discuss in further detail later in the paper.

The final step in turning our work into practical application was to combine these steps in a meaningful and effective way. This was done through building an interactive GUI. This actually adds more value than we initially thought. It first allows for a user to select language pairs and voice options easily making customization, which can be tricky when using services. Then it also allows for people to perform manual checks as mentioned to ensure that they are satisfied with the intermediary results of the process. We even provide the option for manual editing to ensure the highest user satisfaction possible.

**Results:**

Primarily the part of the system that we were most interested in evaluating was the translation portion as most of the services have high quality as they are Microsoft's commercial grade systems.

Evaluation of machine translation systems and establishing good quality is a difficult process. There are several different systems in place such as COMET, and BLEU. However, essential to remember regarding any evaluation metric is that they don't paint a picture and that human evaluation is essential. That is why the bilingual model we chose to train was Spanish and Polish based as we have relative fluency combined in both these languages allowing for competent human evaluation. Having a single model makes establishing a formal metric for comparison difficult but we were able to see that for the most part the translations are of high but definitely imperfect quality with unknown translations being one of our larger issues.

The non-human evaluation metric we chose to use was BLEU due primarily to its high acceptance rate. Several versions of BLEU exist as people have tweaked the hyper parameters of the program. We used SacreBLEU to be as standardized as possible.

We trained for 250000 iterations and tested BLEU scores every 8000 iterations so that we could select the high for operation. We stopped at this point because we found that the results had plateaued at 30.7 BLEU points.

Originally, we had trained on a smaller vocabulary and diversity of topics and if you watched the video associated with this paper we had a BLEU score of around 35 which is higher than our current BLEU score. This is because we now handle a wider variety of topics at a slightly lower level. This is a good illustration of how BLEU scores are not representative of the whole story because while the model had a better score for its training set before we added in more data, the overall usefulness of the original model is less than our final model.

Our final BLEU score of 30 represents a lower than average BLEU score for a commercial model however BLEU scores are best used looking for significant differences in results between models for the same language pair. However it can still be considered a decent score as BLEU points are better for comparison than a blanket quality statement.

Our human evaluation of the model found that for the most part our model produced human readable and understandable content. We found it to perform quite well in terms of adequacy. This means that the translation tended to match the intended meaning of the other sentence at high rates. In terms of fluency or how well the model sounds like a native speaker. This is more about proper grammar and style than about substance. Our model had medium but acceptable fluency for the purposes and scope of this class. Overall the translation model performed well and had satisfying results.

The rest of our speech to speech platform including a variety of other bilingual translation options are microsoft services and so their quality is at a professional and commercial level. The

combination of tools, services and our Polish to Spanish model evaluated as a whole would be describable as a really effective and interesting result as it is easy to use and consistently gives the desired output when run through the system as a whole.

**Further Work:**

There is a lot of potential work that could be done to further this project. Things that we will discuss are speech-to-speech direct, refining the language model, and additional features.

For our speech translation we used the common waterfall approach that is effective and fast, but going direct is theoretically better. This is where instead of going first to text and then translating and then synthesizing back into audio, you take in audio files and train a neural machine network with audio as the output as well. This has shown really good results for others but the lack of available data made this impractical for our project. If we were able to get enough bilingually paired audio data this would be an effective and incredibly interesting progression of this project to be able to explore differences in both final quality and speed.

Our language model while effective there is also plenty of room for improvement. We could have trained using a different framework or using an LSTM rather than an RNN network. Different training methods or learning rates provide a lot of room for changing hyper parameters and exploring potential combinations to optimize our output. Due to training time requirements though we stayed relatively close to the default parameters.

One additional feature that would be meaningful and extremely cool would be to experiment with training our own voices on the output to reach a higher level of customization. Microsoft Azure offers a service wherein voices can be recorded and depending on the quantity and quality of the recordings a voice option which would allow users to then use our voices or any voices that we trained with legal permission. This is a really cool feature to explore in future research and development.

Another option to explore would be experimenting with having our GUI interact with the model in a way that records when users make changes to any part of the output and is able to then record and update that. Difficulties with this are that even though a flaw in output has been identified, fixing it without paying the high costs of retraining a model is almost impossible. This is a cutting edge topic currently in translation and for any interested in furthering our research this piece of the field comes highly recommended.

**Conclusions:**

Speech to speech translation is a rapidly growing field with high demand and important use cases. In our research we explored powerful frameworks and tools that can be combined to make meaningful user experiences. We were able to use Microsoft Azure services to deal with speech to text and text to speech, and some language pairs. We also were able to use openNMT to build a relatively high quality machine translator for polish to spanish.