Automatic generation of short narrative texts in Central Quechua

1st Flavio Huamán
Department of Engineering
Pontifical Catholic University of Peru
Lima, Peru

2nd Fabian Prado
Department of Engineering
Pontifical Catholic University of Peru
Lima, Peru

3rd Jeremy Lopez
Department of Engineering
Pontifical Catholic University of Peru
Lima, Peru

4th Gustavo Torres
Department of Engineering
Pontifical Catholic University of Peru
Lima, Peru

5th Jorge Alvarado
Department of Engineering
Pontifical Catholic University of Peru
Lima, Peru

***Abstract*—In Peru, there are a large number of native Andean languages that are currently used, such as Quechua, Jaqaru or Aymara. However, unlike languages such as Spanish, there are considerably fewer texts and language users that allow the maintenance of these languages. For this reason, using the variation of the Quechua language, Central Quechua, we have worked to create a model capable of generating short texts based on initial words and a number of words to continue the given words, with the aim of facilitating the creation of texts in Quechua that facilitate and strengthen the learning and use of Quechua. A neural network model has been chosen to perform this task, using as a basis for the vocabulary collections of stories in Quechua.**

***Index Terms*—quechua, corpus, natural language processing, tokenize, model, neural, n-grams**

I. Introduction

This project aims to develop a model capable of generating original short texts in Quechua (central region variant), using selected narrative texts as a foundation. By employing advanced natural language processing techniques, the study addresses the pressing issue of limited textual resources in Quechua. The creation of new, contextually appropriate content not only contributes to the preservation and revitalization of the language but also supports efforts to expand its literary corpus and accessibility.

II. State of the Art

Text generation using neural network models has experienced remarkable advances in recent years, especially in major languages such as English and Spanish. However, the application of these technologies to minority languages, such as Quechua, has been limited due to the scarcity of textual resources and the smaller number of users of these languages.

A. Text Generation Models

Among the most advanced text generation models are those based on recurrent neural networks (RNN), such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), as well as transformer models, notably Vaswani et al.'s Transformer and its well-known evolution, OpenAI's GPT (Generative Pre-trained Transformer). These models have demonstrated a great capacity to generate coherent and contextually appropriate texts, using large amounts of data for training.

B. Applications in Minority Languages

Most advances in natural language processing (NLP) have focused on languages with abundant textual resources. However, efforts have been made to apply these technologies to minority languages. A notable example is the work of Mager et al. (2018), who adapted NLP techniques for the Nahuatl language. Such studies provide a methodological and technical basis that can be adapted for other languages, such as Quechua.

C. Quechua and its Central Variant

Quechua, being one of the most spoken indigenous languages in the Andes, has several regional variants. The central variant of Quechua, used in this study, presents particular challenges due to dialectal diversity and lack of standardization. Although there are projects for the digitization and preservation of indigenous languages, such as the efforts to collect stories and narrative texts in Quechua, the creation of specific generative models for this language has been scarce.

D. Recent Contributions

Recently, initiatives presented by collaborative projects between academic institutions and indigenous communities have begun to develop textual corpora that allow the training of NLP models in indigenous languages. These projects are essential for the preservation and revitalization of these languages, providing the necessary data for the implementation of advanced text generation technologies.

E. Challenges and Opportunities

The main challenges in text generation in minority languages include the scarcity of data and the need for models that can adequately handle the complex morphology and dialectal variations. However, the development of transfer learning

techniques and the creation of specialized corpora present significant opportunities to advance in this field.

F. Our Project

In this project, we have developed a neural network model to generate short texts in Central Quechua, using a collection of Quechua stories as a foundation. We have implemented an n-gram-based model that can generate coherent text from an initial sequence of words and a specific number of words to continue. This approach not only facilitates the creation of new texts in Quechua but also supports efforts to learn and use the language, thus contributing to its preservation and strengthening.

Our model has been trained with a corpus of narrative texts in Quechua, and its ability to generate textual sequences that maintain the coherence and context of the language has been validated. This work represents an important step towards integrating advanced NLP technologies in the revitalization of indigenous languages like Quechua.

III. Methodology

A. Corpus description

The corpus was extracted from multiple narrative texts offered by the Peruvian government. The source of these stories are the texts ”Literatura - Quechua Central” and ”Historias y Relatos Quechua Central”, both texts obtained from the Ministry of Education of Peru [1], [2], which is the governmental entity in charge of formulating, implementing and supervising the educational policies in Peru and seeks to ensure that all Peruvian citizens (including Central Quechua speakers) have access to education at all levels.

B. Corpus generation method

To obtain the words that conform the vocabulary of the model, we took the texts of the collected stories and placed them in a .csv file, dividing each story into a tuple of ”title” and ”text”, where ”title” indicates the name of the story and ”text” refers to the content of the story itself. One example can be seen in the Figure 1.

Out[4]:

	title	text
0	Atuqwan haka	Huk shumaq shipash, chakwan mamallanwan qichwa...
1	Awilan waatashqa wamra	Huk kutishi, huk aqtupaakuq wamra kanaq. Payqa...
2	Ukushkuna	Unayshi kimsa panituri ukushkuna kawayaanaq. C...
3	Ashnuwan atuq	Unay wichanshi Tiwllu shutishqa runa huk ashnu...
4	Utush Kuru	Unayshi kay shumaq pampachaw papata muruq huk ...

Figure 1. Example of the tuples created in the corpus csv file

Then, the .csv extension file is preprocessed for several purposes, such as the removal of words and letters that do not belong to the Central Quechua language, irrelevant words (stop words), special or punctuation characters and additional blank spaces. To complete this task, we first define an alphabet composed of all the characters and vowels used in Central Quechua, as well as a list of all the punctuation vowels used in Central Quechua, as well as a list of all the punctuation marks that will be taken into account for removal.

C. Solution training

First, a RougeScoreCallback class is defined in order to evaluate the performance of the text generation model that will be created using the ROUGE metric. Using the class of the tf.keras.callbacks.Callback so that it can be easily integrated into the training cycle of a Keras model. into the training cycle of a Keras model. In addition, a method is created that is called at the end of each training epoch, so that the scores of all comparisons are aggregated in order to obtain the best possible model. A text generation method is also created from which the text created from the algorithm will be generated.

To define the model, the text set is passed to a list and divided into 2 part, a training set and a test set. The words are then tokenized so that the model learns the vocabulary from the set of training texts. The training sequences (N-grams) are then generated. The constructed n-grams have a variable length of up to 5 words.

Each sequence of constructed n-grams is added to a list of input sequences and the pad-sequences function is used to match the sequences in terms of length so that they are compatible with the input of the neural network model. The list of sequences is converted to a Numpy array for better manipulation and performance and the input sequences are separated from the target words, and these are converted to one-hot vectores with the to categorical function.

Finally, to compile the model, a Keras sequential model is created where the embeddings layer with a dimension of 200 is added. also a LSTM layer to capture dependencies along the data sequence and a dense layer with the activation funcion softmax to predict the probability of each word as the next word in the sequence.

The model is compiled using the categorical cross-entropy as a loss function, in addition to the Adam optimizer with very low learning rate to allow fine adjustments to the training.

The model is then saved to the file model.keras and, with the parameter save-vest-only=True the model is saved only if the validation loss is the best observed.

The model is evaluated using the ROUGE metric in the training epochs and the number of words to generate is specified. the number of words to be generated is specified.

To continue, the training of the model is performed with the model fit function, using a number of 50 epochs and using 0.05 of the validation set. Finally, the model is loaded and the model is tested using an initial text and a number of words to be generated. number of words to be generated.

In addition, the tokenizer is saved in the file tokenizer.pkl in order to reuse the tokenizer without the need to re-adjust it in the training data.

D. Solution Deployment

To deploy the solution, it is necessary to set up a Flask-based API that will allow the generation of text based on input parameters. Here are the steps for deploying the solution:

1) Setup Flask API:

- **Install Flask**: Ensure Flask is installed in the Python environment by running the following command:

```
pip install Flask
```

- **Create Flask Application**: Create a new Python file, for example, `api_service.py`, and set up the Flask application. This involves:
  - Importing necessary libraries.
  - Loading the trained model and tokenizer.
  - Defining an endpoint for generating text that accepts input parameters and returns generated text.

  Example of setting up the Flask application:

```
from flask import Flask, request, jsonify
from tensorflow.keras.models
import load_model
import pickle

app = Flask(__name__)

# Load the model and tokenizer
model = load_model('model.keras')
with open('tokenizer.pkl', 'rb') as handle:
    tokenizer = pickle.load(handle)

@app.route('/generate', methods=['POST'])
def generate():
    data = request.get_json()
    seed_text = data['seed_text']
    next_words = data['next_words']
    # Function to generate text
    generated_text = generate_text(
    seed_text, next_words)
    return jsonify({"generated_text":
    generated_text})

def generate_text(seed_text, next_words):
    for _ in range(next_words):
        # Implementation of text generation
        pass

if __name__ == '__main__':
    app.run(debug=True)
```

- **Run the Flask Application**: Execute the Flask application by running the following command in the terminal:

```
python api_service.py
```

  The application will be running on `http://127.0.0.1:5000`.

2) Expose Localhost to Internet Using ngrok: To make the Flask application accessible over the internet, use ngrok.

- **Install ngrok**: Download and install ngrok from https://ngrok.com/.
- **Expose Localhost**: Run ngrok to expose the local Flask application to the internet:

```
ngrok http 5000
```

  ngrok will provide a URL like `https://<ngrok-id>.ngrok-free.app` which can be used to access the API over the internet.

3) Integration with React: To integrate the Flask API with a React frontend, follow these steps:

- **Install Axios**: In the React project, install Axios for making HTTP requests:

```
npm install axios
```

- **Create a React Component**: Create a new React component, for example, `TextGenerator.js`, to interact with the Flask API. This component will include input fields for the seed text and the number of words to generate, a button to trigger the text generation, and a display area for the generated text.
  Example of setting up the React component:

```
import React, { useState } from 'react';
import axios from 'axios';

    const generateText = async () => {
        try {
            const response =
            await axios.post(
            'https://<ngrok-id>.ngrok-free'
            '.app/generate', {
                seed_text: seedText,
                next_words: nextWords
            });
            setGeneratedText(
            response.data.generated_text);
        } catch (error) {
            console.error(
            "Error generating text!", error);
        }
    };
```

- **Include Component in App**: Include the `TextGenerator` component in the main `App.js` file to ensure it is rendered within the application.
  Example of including the component:

```
import React from 'react';
import TextGenerator from './TextGenerator';

const App = () => (
    <div>
        <TextGenerator />
    </div>
);

export default App;
```

- **Run React Application**: Start the React application with the following command:

```
npm start
```

  The text generation functionality should now be accessible from the React frontend, utilizing the Flask API.

IV. Conclusions

Through the development of a text generation model based on neural networks, the creation of texts in Central Quechua has been facilitated, contributing to their preservation and revitalization. This initiative is vital to keep the Peruvian language alive in the face of the shortage of textual resources and the small number of speakers compared to Spanish speakers.

Although the project still faces challenges such as managing a complex morphology, and the lack of texts in this variant of Quechua to further train the model, it gives room for innovation in the creation and implementation of transfer learning techniques, which could work well for small datasets [3]. This project and its methodology can be used as a basis for future research, not only on text generation in Central Quechua, but for other resource-limited languages.

In addition, with the use of deployment tools such as the one shown in this project, such text generators can be made available to allow the use, not only for a group of people, but for the entire population of Peru interested in speaking and/or learning about Central Quechua.

REFERENCES

[1] Ministerio de Educación, "Literatura - Quechua central", 1st ed. Lima, Peru, 2021. [Online]. Available: https://nips2018creativity.github.io/doc/Transfer%20Learning%20for%20Style-Specific%20Text%20Generation.pdf

[2] Ministerio de Educación, "Historia y Relatos Quechua Central", 1st ed. Lima, Peru, 2021. [Online]. Available: https://nips2018creativity.github.io/doc/Transfer%20Learning%20for%20Style-Specific%20Text%20Generation.pdf

[3] K. I. Gero, G. Karamanolakis, and L. Chilton. (2018). "Transfer Learning for Style-Specific Text Generation". Presented at the 32nd Conference on NIPS, Montréal, Canada. [Online]. Available: https://nips2018creativity.github.io/doc/Transfer%20Learning%20for%20Style-Specific%20Text%20Generation.pdf

[4] Pontifical Catholic University of Peru. (2024). Temas Avanzados en Ciencias de la Computación.