Cairo University

Faculty of Engineering Credit Hours System

CMPS454

Natural Language Processing

Project Report

Prepared by:

- Ali Mohamed Hashish 1190223
- Amr Ahmed Abdelzaher 1190074
- John Maurice 1190403
- Joseph Ameer Aziz 1190052

Chosen Model to Submit

We chose the conv_1000 model, as it trains on the longest sentence length, giving the highest accuracy.

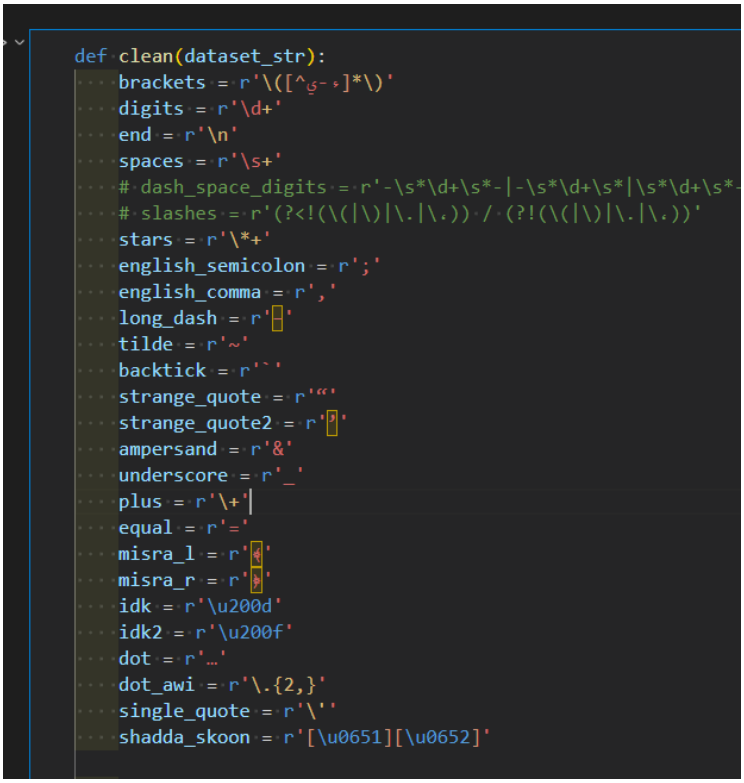
Project Flow

Cleaning and preprocessing the data, then extracting features for our model (characters in case of the conv_1000 model). We then train and predict, calculating the accuracy and DER.

Data Preprocessing

Cleaning

For the data preprocessing and cleaning we remove characters like brackets, digits, english letters, spaces symbols like (&~ – _ ( )), shadda skoon. And other symbols like (\u200d,\u200f). Any character that does not help in understanding the context of the sentence is removed.



```
def clean(dataset_str):
    brackets = r'\([^ؠ-ۦ]*\)'
    digits = r'\d+'
    end = r'\n'
    spaces = r'\s+'
    # dash_space_digits = r'-\s*\d+\s*-|-\s*\d+\s*|\s*\d+\s*-
    # slashes = r'(?<!(\(|\)|\.|\،)) / (?!(\(|\)|\.|\،))'
    stars = r'\*+'
    english_semicolon = r';'
    english_comma = r','
    long_dash = r'—'
    tilde = r'~'
    backtick = r'`'
    strange_quote = r'“'
    strange_quote2 = r'”'
    ampersand = r'&'
    underscore = r'_'
    plus = r'\+'
    equal = r'='
    misra_l = r'«'
    misra_r = r'»'
    idk = r'\u200d'
    idk2 = r'\u200f'
    dot = r'…'
    dot_awi = r'\.{2,}'
    single_quote = r'\''
    shadda_skoon = r'[\u0651][\u0652]'
```

There are also some characters that are replaced with more meaningful ones, that would make it easier for the model to train and learn. Such as replacing the English semicolon with an Arabic one, or unifying how commas are placed.

```
dataset_str = re.sub(english_semicolon, ' ؛ ', dataset_str)
dataset_str = re.sub(long_dash, ' - ', dataset_str)
dataset_str = re.sub(misra_l, ' " ', dataset_str)
dataset_str = re.sub(misra_r, ' " ', dataset_str)
dataset_str = re.sub(idk, '', dataset_str)
dataset_str = re.sub(idk2, '', dataset_str)
dataset_str = re.sub(shadda_skoon, r'\u0651', dataset_str)
dataset_str = re.sub(english_comma, ' ، ', dataset_str)
```

Feature Extraction:

1- Sklearn LabelEncoder and torch.Embeddings:

The Label encoder takes each character and converts it into a number from 0 to number of unique characters-1. The embedding layer by torch converts them into vectors and it acts as a lookup table. These representations are updated during training.

```
sentence_encoder = LabelEncoder().fit(X_train.flatten())
X_train = sentence_encoder.transform(X_train.flatten()).reshape(X_train.shape).astype(np.int16)
X_val = sentence_encoder.transform(X_val.flatten()).reshape(X_val.shape).astype(np.int16)
X_test = sentence_encoder.transform(X_test.flatten()).reshape(X_test.shape).astype(np.int16)
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.embedding = nn.Embedding(num_embeddings=np.unique(X_train).shape[0], embedding_dim=25)
```

2- CBOW:

After cleaning the data, we use tokenizer while accounting for OOV tokens, we train the neural network on the fake task and then take embeddings from the network. This model is then concatenated with an encoder/decoder model. Such that the output of the encoder is passed to the CBOW model.

```
# Define the corpus
# corpus = ['The cat sat on the mat',
#           'The dog ran in the park',
#           'The bird sang in the tree']
train_corpus = open("dataset/train_clean.txt", "r",encoding='utf8').read().split(".")
train_noDiactric, train_labels =extract_data(train_corpus)
# Convert the corpus to a sequence of integers
tokenizer = Tokenizer(oov_token=1)
tokenizer.fit_on_texts(train_noDiactric)
train_sequences = tokenizer.texts_to_sequences(train_noDiactric)
word2idx = tokenizer.word_index
idx2word = {v:k for k, v in word2idx.items()}
word2idx['UKN']= len(word2idx) +1
```

[2]

```
# Define the CBOW model
model = Sequential()
model.add(Embedding(input_dim=vocab_size,
                    output_dim=embedding_size,
                    input_length=2*window_size))
model.add(Lambda(lambda x: tf.reduce_mean(x, axis=1)))
model.add(Dense(units=vocab_size, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam')
model.summary()
```

[9]

3- SG:

Experimented with this feature a bit, but its results were greatly underwhelming.

Model Training:

1- Model Obtained from the Shakkala paper

Shakkala paper [https://aclanthology.org/D19-5229.pdf]

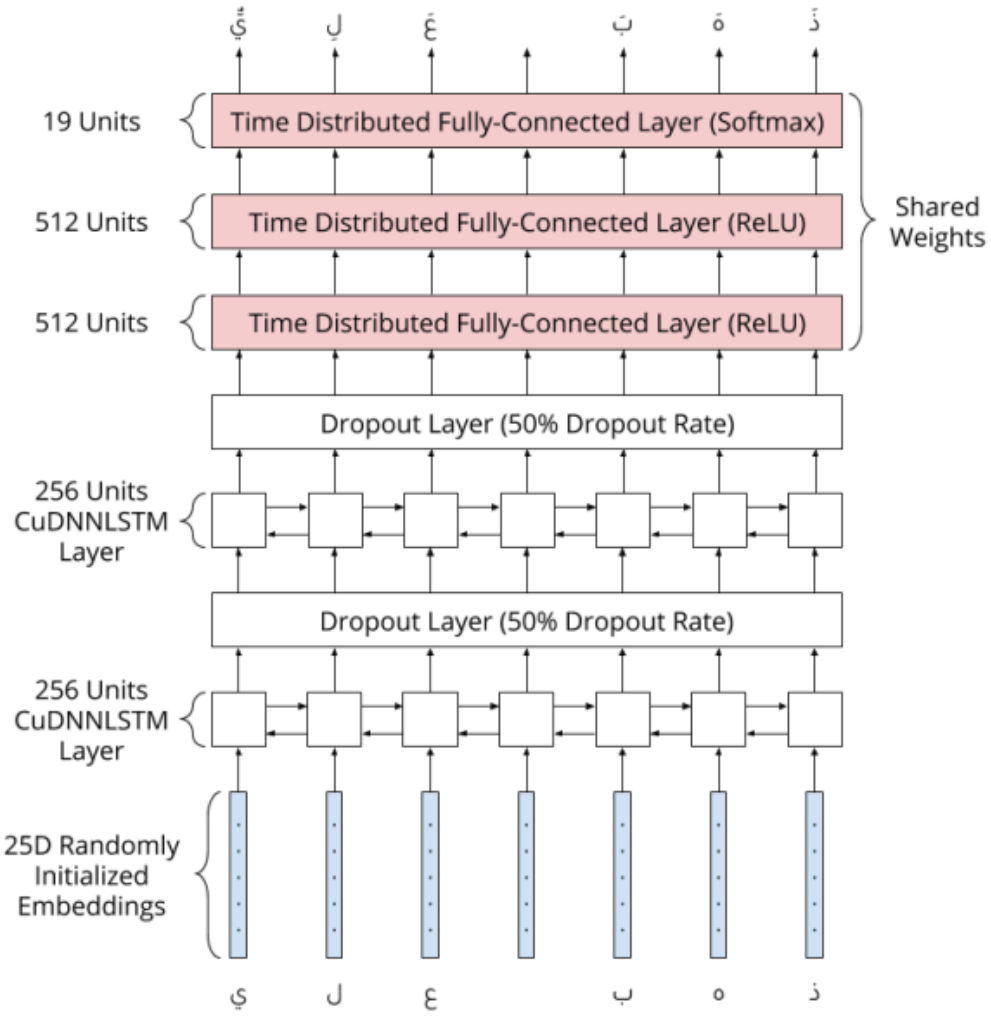


Figure 3: RNN basic model structure.

```
model = Sequential()

model.add(Embedding(input_dim=np.unique(X_train).shape[0], output_dim=25, input_length=max_len))

model.add(Bidirectional(LSTM(256, return_sequences=True)))
model.add(BatchNormalization())
model.add(Dropout(rate=0.5))

model.add(Bidirectional(LSTM(256, return_sequences=True)))
model.add(BatchNormalization())
model.add(Dropout(rate=0.5))

model.add(TimeDistributed(Dense(512, activation='relu')))
model.add(BatchNormalization())

model.add(TimeDistributed(Dense(512, activation='relu')))
model.add(BatchNormalization())

model.add(TimeDistributed(Dense(np.unique(y_train).shape[0], activation='softmax')))

model.summary()
```

The DER was 2.6% on this implementation.

2- First Modification to Shakkala

```
model = Sequential()

model.add(Embedding(input_dim=np.unique(X_train).shape[0], output_dim=25, input_length=max_len))

model.add(Conv1D(filters=256, kernel_size=10, activation='relu', padding='same'))
model.add(MaxPooling1D(pool_size=2, padding='same'))
model.add(BatchNormalization())
model.add(Dropout(rate=0.5))

model.add(Bidirectional(LSTM(256, return_sequences=True)))
model.add(BatchNormalization())
model.add(Dropout(rate=0.5))

model.add(Bidirectional(LSTM(256, return_sequences=True)))
model.add(BatchNormalization())
model.add(Dropout(rate=0.5))

model.add(TimeDistributed(Dense(512, activation='relu')))
model.add(BatchNormalization())

model.add(TimeDistributed(Dense(512, activation='relu')))
model.add(BatchNormalization())

model.add(TimeDistributed(Dense(np.unique(y_train).shape[0], activation='softmax')))

model.summary()
```

A Convolutional 1D layer was added to the model improving and it seemed to better improve the training so this led to the following modification.

3- Second Modification to Shakkala

```python
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.embedding = nn.Embedding(num_embeddings=np.unique(X_train).shape[0], embedding_dim=25)

        # Add Conv1d layer
        self.conv1 = nn.Conv1d(in_channels=25, out_channels=256, kernel_size=11, padding=5)
        self.bn_conv1 = nn.BatchNorm1d(max_len)
        self.dropout_conv1 = nn.Dropout(p=0.5)

        self.lstm1 = nn.LSTM(input_size=256, hidden_size=256, batch_first=True, bidirectional=True)
        self.bn1 = nn.BatchNorm1d(max_len)
        self.dropout1 = nn.Dropout(p=0.5)

        self.conv2 = nn.Conv1d(in_channels=512, out_channels=512, kernel_size=5, padding=2)
        self.bn_conv2 = nn.BatchNorm1d(max_len)
        self.dropout_conv2 = nn.Dropout(p=0.5)

        self.lstm2 = nn.LSTM(input_size=512, hidden_size=256, batch_first=True, bidirectional=True)
        self.bn2 = nn.BatchNorm1d(max_len)
        self.dropout2 = nn.Dropout(p=0.5)

        self.conv3 = nn.Conv1d(in_channels=512, out_channels=512, kernel_size=3, padding=1)
        self.bn_conv3 = nn.BatchNorm1d(max_len)
        self.dropout_conv3 = nn.Dropout(p=0.5)

        self.dense1 = nn.Linear(in_features=512, out_features=512)
        self.bn3 = nn.BatchNorm1d(max_len)

        self.dense2 = nn.Linear(in_features=512, out_features=512)
        self.bn4 = nn.BatchNorm1d(max_len)

        self.dense3 = nn.Linear(in_features=512, out_features=np.unique(y_train).shape[0])
```

Two more convolutional layers were added to the model to improve the final DER to 2.24%. This is the conv_1000 model we submitted

4- Third modification to Shakkala:

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.embedding = nn.Embedding(num_embeddings=np.unique(X_train).shape[0], embedding_dim=25)

        # Add Conv1d layer
        self.conv1 = nn.Conv1d(in_channels=25, out_channels=256, kernel_size=11, padding=5)
        self.bn_conv1 = nn.BatchNorm1d(max_len)
        self.dropout_conv1 = nn.Dropout(p=0.5)

        self.lstm1 = nn.GRU(input_size=256, hidden_size=256, batch_first=True, bidirectional=True)
        self.bn1 = nn.BatchNorm1d(max_len)
        self.dropout1 = nn.Dropout(p=0.5)

        self.conv2 = nn.Conv1d(in_channels=512, out_channels=512, kernel_size=5, padding=2)
        self.bn_conv2 = nn.BatchNorm1d(max_len)
        self.dropout_conv2 = nn.Dropout(p=0.5)

        self.lstm2 = nn.GRU(input_size=512, hidden_size=256, batch_first=True, bidirectional=True)
        self.bn2 = nn.BatchNorm1d(max_len)
        self.dropout2 = nn.Dropout(p=0.5)

        self.conv3 = nn.Conv1d(in_channels=512, out_channels=512, kernel_size=3, padding=1)
        self.bn_conv3 = nn.BatchNorm1d(max_len)
        self.dropout_conv3 = nn.Dropout(p=0.5)

        self.dense1 = nn.Linear(in_features=512, out_features=512)
        self.bn3 = nn.BatchNorm1d(max_len)

        self.dense2 = nn.Linear(in_features=512, out_features=512)
        self.bn4 = nn.BatchNorm1d(max_len)

        self.dense3 = nn.Linear(in_features=512, out_features=np.unique(y_train).shape[0])

    def forward(self, x):
```

We replaced the LSTMs with GRU's. However this did not lead to the desired results.

5- CBHG:

We also tried the Model given by https://ieeexplore.ieee.org/document/9274427 with nearly the same 15m parameters highlighted and it ended up overfitting and it didn't produce good results as compared with the best model, the DER was 2.9%.

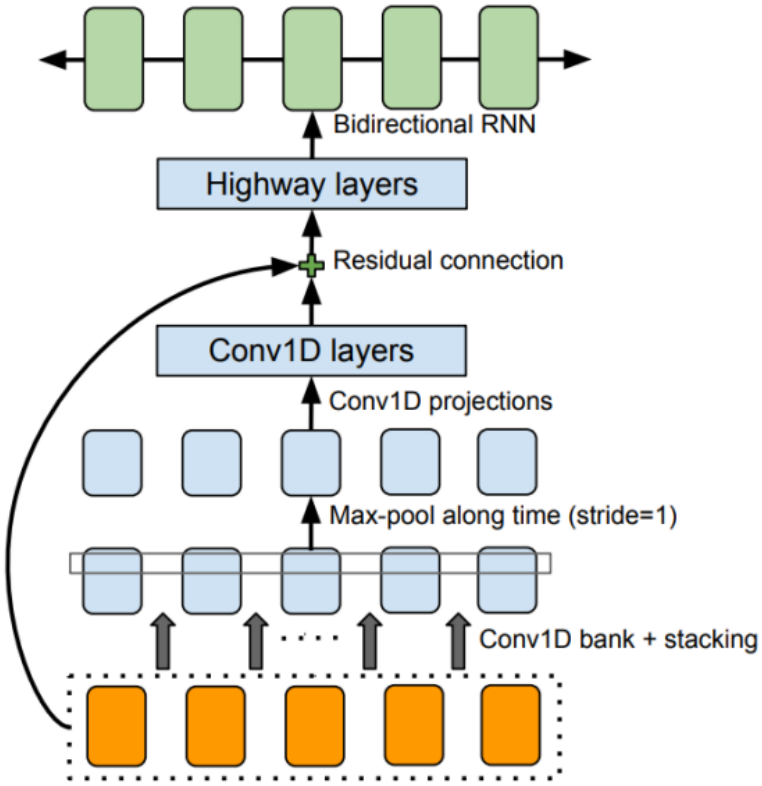


FIGURE 14. The CBHG module architecture, taken from [3]. It consists of a 1-D convolution bank, a multi-layer highway network, and a bidirectional GRU layer.

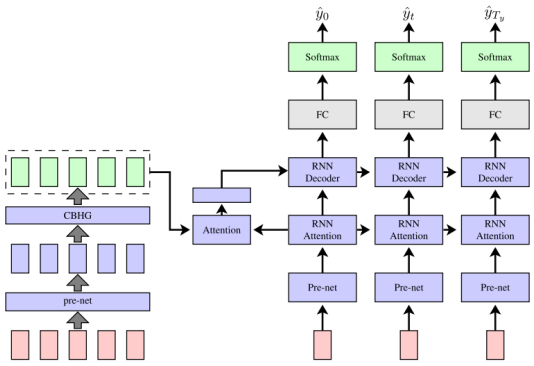


FIGURE 16. The structure of the encoder-decoder model with attention, which is adapted from Tacotron [3]. The encoder part is the same as Tacotron, but the decoder and the attention are different.

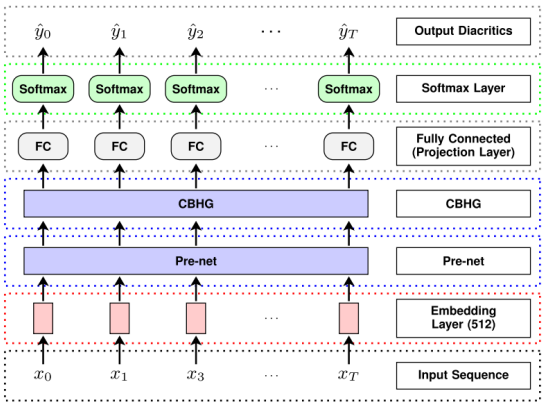


FIGURE 17. The CBHG model architecture. It is implemented using only the encoder of the encoder-decoder model with more robust parameters. We added a fully-connected layer and a softmax layer on top of the encoder to output the probability distribution for each character.

to an RNN attention layer, which uses the location-sensitive