1.Design a neural network to translate text from one language to another, aiming for fluency and accuracy in translating between languages like English and French.

Coding:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

english_sentences = [
   "hello", "how are you", "what is your name", "where are you going"
]
french_sentences = [
   "bonjour", "comment ça va", "quel est ton nom", "où vas-tu"
]

tokenizer_eng = Tokenizer()
tokenizer_eng.fit_on_texts(english_sentences)
english_sequences = tokenizer_eng.texts_to_sequences(english_sentences)

tokenizer_fr = Tokenizer()
tokenizer_fr.fit_on_texts(french_sentences)
french_sequences = tokenizer_fr.texts_to_sequences(french_sentences)

eng_vocab_size = len(tokenizer_eng.word_index) + 1
fr_vocab_size = len(tokenizer_fr.word_index) + 1
```

```
max_len_eng = max([len(seq) for seq in english_sequences])
max_len_fr = max([len(seq) for seq in french_sequences])

english_sequences = pad_sequences(english_sequences, maxlen=max_len_eng, padding='post')
french_sequences = pad_sequences(french_sequences, maxlen=max_len_fr, padding='post')

encoder_inputs = Input(shape=(max_len_eng,))
encoder_embedding = tf.keras.layers.Embedding(eng_vocab_size, 256)(encoder_inputs)
encoder_lstm = LSTM(256, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(encoder_embedding)

decoder_inputs = Input(shape=(max_len_fr,))
decoder_embedding = tf.keras.layers.Embedding(fr_vocab_size, 256)(decoder_inputs)
decoder_lstm = LSTM(256, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_embedding, initial_state=[state_h, state_c])
decoder_dense = Dense(fr_vocab_size, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

model.compile(optimizer='adam', loss='categorical_crossentropy')
model.summary()
```

```
decoder_target_data = np.zeros((len(french_sequences), max_len_fr,
fr_vocab_size), dtype='float32')

for i, seq in enumerate(french_sequences):
    for t, word_index in enumerate(seq):
        if t > 0:
            decoder_target_data[i, t - 1, word_index] = 1.0

decoder_input_data = french_sequences

model.fit([english_sequences, decoder_input_data], decoder_target_data,
          batch_size=16, epochs=500, validation_split=0.2)

def translate_sentence(input_sentence):
    input_seq = tokenizer_eng.texts_to_sequences([input_sentence])
    input_seq = pad_sequences(input_seq, maxlen=max_len_eng,
padding='post')

    states = encoder_lstm.predict(input_seq)

    target_seq = np.zeros((1, max_len_fr))
    target_seq[0, 0] = tokenizer_fr.word_index['start']

    translated_sentence = []
    for _ in range(max_len_fr):
        output_tokens, h, c = decoder_lstm.predict([target_seq] + states)

        sampled_token_index = np.argmax(output_tokens[0, -1, :])
```

```
        sampled_word = tokenizer_fr.index_word[sampled_token_index]
        if sampled_word == 'end':
            break

        translated_sentence.append(sampled_word)

        target_seq = np.zeros((1, max_len_fr))
        target_seq[0, 0] = sampled_token_index
        states = [h, c]

    return ' '.join(translated_sentence)

translated = translate_sentence("hello")
print(translated)
```

Output:

Translated sentence: bonjour

2. Create an autoencoder to clean noisy speech signals. The model should aim to reconstruct clear speech from noisy audio recordings.

Coding:

```
import numpy as np
import os
from os import walk
import soundfile
import librosa
import librosa.display
```

```
import matplotlib.pyplot as plt
import random
from tensorflow import keras
from keras import layers
from sklearn.model_selection import train_test_split
import IPython.display as ipd
def get_filepaths(directory = "audio-mnist/data"):
    filepaths = []
        for root, dirs, files in os.walk("audio-mnist/data", topdown=False):
        for name in files:
            filepaths.append(os.path.join(root, name))

    if directory == "audio-mnist/data":
        filepaths = filepaths[:-1]

    return filepaths
S_noisy = np.abs(librosa.stft(data_noisy, n_fft=1024))
def get_spectrogram_features(filepaths, noise_gain = 1, sample_rate = 22050,
duration_s = 1):
    X_clean = []
    X_noisy = []

    duration = int(sample_rate * duration_s)

    for filepath in filepaths:

        data, _ = librosa.load(filepath, sr = sample_rate)
```

```
        if len(data) < sample_rate:
            max_offset = np.abs(len(data) - duration)
            offset = np.random.randint(max_offset)
            data = np.pad(data, (offset, duration-len(data)-offset), "constant")

        elif len(data) > sample_rate:
            max_offset = np.abs(len(data) - duration)
            offset = np.random.randint(max_offset)
            data = data[offset:len(data)-max_offset+offset]

        else:
            offset = 0

        S = np.abs(librosa.stft(data, n_fft=2048))[:-1,:]
        X_clean.append(S)

        RMS=np.sqrt(np.mean(np.abs(data**2)))
        noise=np.random.normal(0, RMS, data.shape[0])
        data_noisy = data+noise
        S_noisy = np.abs(librosa.stft(data_noisy, n_fft=2048))[:-1,:]
        X_noisy.append(S_noisy)
    X_clean = np.array(X_clean)
    X_clean = np.expand_dims(X_clean, -1)
    X_noisy = np.array(X_noisy)
    X_noisy = np.expand_dims(X_noisy, -1)

    return X_clean, X_noisy
class ZeroOneNorm():
```

```
    def __init__(self):
        self.data_to_fit = []
        self.data_to_transform = []
    def fit(self, data_to_fit):
        self.fitting_constant = np.max(np.abs(data_to_fit))
    def normalize(self, data_to_transform):
        normalized_data = data_to_transform / self.fitting_constant
        return normalized_data
    def denormalize(self, data_to_transform):
        denormalized_data = data_to_transform * self.fitting_constant
        return denormalized_data
N = ZeroOneNorm()
N.fit(X_noisy)
X_clean_n = N.normalize(X_clean)
X_noisy_n = N.normalize(X_noisy)

x_train, x_test, y_train, y_test = train_test_split(X_noisy_n, X_clean_n,
test_size = 0.2)

my_callbacks = [keras.callbacks.EarlyStopping(patience=5),
            keras.callbacks.ReduceLROnPlateau(monitor="loss", patience=3),
           ]
opt = keras.optimizers.Adam(learning_rate=0.001)
autoencoder.compile(optimizer=opt, loss='binary_crossentropy')
history = autoencoder.fit(x_train, y_train,
            epochs=25,
            shuffle=True,
            validation_data=(x_test, y_test),
```

```
        verbose = 1,
        callbacks = my_callbacks
        )

spec = N.denormalize(clean.reshape(spec_shape))
spec_inv = librosa.griffinlim(spec)
soundfile.write('clean.wav', spec_inv, sample_rate)

spec = N.denormalize(noisy.reshape(spec_shape))
spec_inv = librosa.griffinlim(spec)
soundfile.write('noisy.wav', spec_inv, sample_rate)

spec = N.denormalize(denoised.reshape(spec_shape))
spec_inv = librosa.griffinlim(spec)
soundfile.write('denoised.wav', spec_inv, sample_rate)
```

Output: