Υλοποίηση κωδικοποιητή GSM 06.10 Συστήματα Πολυμέσων 2024-2025

Χρήστος Μάριος Περδίκης 10075 cperdikis@ece.auth.gr
Γιώργος Βακασίρης 9661 vakasiris@ece.auth.gr

Περίληψη

Το παρόν έγγραφο είναι η αναφορά της εργασίας του μαθήματος Συστήματα Πολυμέσων του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του ΑΠΘ τη χρονιά 2024-2025. Υλοποιήθηκε κωδικοποίηση φωνής σύμφωνα με το πρότυπο GSM 06.10. Από τα τρία επίπεδα υλοποιήθηκαν το 1ο και το 2ο. Αρχικά δίνεται ένα filename ενός wav αρχείου σε ένα audio wrapper, το οποίο μετατρέπει τα samples του ηχητικού αρχείου σε μορφή η οποία μπορεί να υποστεί επεξεργασία. Στα samples εφαρμόζεται η διαδικασία pre-processing. Το σήμα φωνής εισέρχεται πρώτα σε έναν κωδικοποιητή, και γίνεται short-term analysis. Η έξοδος εισέρχεται σε έναν αποκωδικοποιητή και τα samples που προκύπτουν από τον αποκωδικοποιητή αποθηκεύονται σε ένα νέο wav αρχείο μέσω του audio wrapper. Συγκρίνοντας το αρχικό αρχείο φωνής και του αρχείο φωνής που δημιούργησε το πρόγραμμα, είναι εύκολο να παρατηρηθούν τα αποτελέσματα της κωδικοποίησης.

1 Προετοιμασία - main.py

Το πρόγραμμά μας τρέχει με την εντολή "**`python main.py`**". Το αρχείο **`main.py`** περιέχει τη λογική δομή του προγράμματος, από εκεί καλούνται όλες οι συναρτήσεις που υλοποιούν την κωδικοποίηση. Με τη βοήθεια της βιβλιοθήκης scipy αποθηκεύουμε όλα τα samples του αρχείου φωνής στο numpy array **`audio_data_o`**. Μπορεί ο χρήστης του προγράμματος να αλλάξει το όνομα του αρχείου εισόδου σε αυτό το σημείο. Ακολουθεί το pre-processing σε μήκος όλου του αρχείου. Οι λειτουργίες του κωδικοποιητή και αποκωδικοποιητή βρίσκονται σε μια λούπα η οποία κάνει μια επανάληψη για κάθε frame (160 samples). Κάθε επανάληψη φορτώνεται το επόμενο frame στο numpy array **`s`**, με σταθερό μέγεθος 160 στοιχείων. Πάνω στα στοιχεία του **`s`** γίνεται η κωδικοποίηση short-term analysis. Η έξοδος της κωδικοποίησης είναι η είσοδος του αποκωδικοποιητή short-term analysis. Στο τέλος της κάθε επανάληψης προσθέτονται τα δεδομένα ήχου/έξοδο του αποκωδικοποιητή στο τέλος του **`audio_array`**. Όταν ολοκληρωθούν αυτές οι επαναλήψεις, κατασκευάζεται ένα νέο αρχείο ήχου "reconstructed_audio.wav" το οποίο θα πρέπει να ακούγεται σαν το αρχικό μαζί με την παραμόρφωση που προέκυψε από την κωδικοποίηση, αν αυτή ήταν επιτυχής. Ο χρήστης του προγράμματος μπορεί να αλλάξει το όνομα του αρχείου εξόδου σε αυτό το σημείο.

Αξίζει να σημειωθεί ότι αν περισσέψουν samples του αρχείου φωνής εισόδου (ο αριθμός samples του δεν διαιρείται ακριβώς με το 160), τότε αυτά τα samples δεν συμμετέχουν στη διαδικασία της κωδικοποίησης και αγνοούνται. Θεωρούμε ότι είναι αποδεκτό να κόψουμε στη χειρότερη περίπτωση 159 samples$/8kHz = 19.875ms$ ήχου παρά να αυξήσουμε την περιπλοκότητα του κώδικά μας. Η απόρριψη αυτή γίνεται μετά τη διαδικασία του pre-processing, οπότε ακόμα και τα samples που θα απορριφτούν υπόκεινται σε pre-processing.

Οι λειτουργίες του preprocessing υλοποιούνται στο αρχείο **`preprocessing.py`**. Πρώτα καλείται η συνάρτηση offset_compensation η οποία υλοποιεί την διαδικασία του offset compensation όπως περιγράφεται στην παράγραφο 3.1.1 του προτύπου. Η είσοδός της είναι τα samples φωνής όλου του αρχείου φωνής εισόδου (ακόμα και τα samples που απορρίπτονται). Έπειτα ακολουθεί η συνάρτηση pre_emphasis, η οποία υλοποιεί τη διαδικασία pre-emphasis όπως περιγράφεται στην παράγραφο 3.1.2 του προτύπου.

2 Short Term Analysis

2.1 Κωδικοποιητής

Η υλοποίηση του κωδικοποιητή βρίσκεται στο αρχείο **`encoder.py`**. Η συνάρτηση αυτοσυχέτισης υπολογίζεται σύμφωνα με την παράγραφο 3.1.4 του προτύπου ως:

$$r_s = \sum_{i=k}^{160} s[i] * s[i-k] \tag{1}$$

όπου **`s`** ένα array που περιέχει τα 160 samples του frame εισόδου και **`rs`** array με τις 9 τιμές της αυτοσυσχέτισης. Σχηματίζουμε τους πίνακες:

$$r = \begin{bmatrix} r_s[1] \\ r_s[2] \\ r_s[3] \\ r_s[4] \\ r_s[5] \\ r_s[6] \\ r_s[7] \\ r_s[8] \end{bmatrix}, \; R = \begin{bmatrix} r_s[0] & r_s[1] & r_s[2] & r_s[3] & r_s[4] & r_s[5] & r_s[6] & r_s[7] \\ r_s[1] & r_s[0] & r_s[1] & r_s[2] & r_s[3] & r_s[4] & r_s[5] & r_s[6] \\ r_s[2] & r_s[1] & r_s[0] & r_s[1] & r_s[2] & r_s[3] & r_s[4] & r_s[5] \\ r_s[3] & r_s[2] & r_s[1] & r_s[0] & r_s[1] & r_s[2] & r_s[3] & r_s[4] \\ r_s[4] & r_s[3] & r_s[2] & r_s[1] & r_s[0] & r_s[1] & r_s[2] & r_s[3] \\ r_s[5] & r_s[4] & r_s[3] & r_s[2] & r_s[1] & r_s[0] & r_s[1] & r_s[2] \\ r_s[6] & r_s[5] & r_s[4] & r_s[3] & r_s[2] & r_s[1] & r_s[0] & r_s[1] \\ r_s[7] & r_s[6] & r_s[5] & r_s[4] & r_s[3] & r_s[2] & r_s[1] & r_s[0] \end{bmatrix} \tag{2}$$

Λύνουμε τις κανονικές εξισώσεις $Rw = r$ ως προς w με τη συνάρτηση numpy.linalg.solve. Ο πίνακας w περιέχει τους 8 βέλτιστους συντελεστές $[a_1, a_2 \ldots]$ του short term προβλέπτη. Πρέπει όμως να χρησιμοποιήσουμε τους συντελεστές που θα γνωρίζει ο αποκωδικοποιητής. Σχηματίζουμε ως είσοδο της βοηθητικής συνάρτησης hw_utils.polynomial_coeff_to_reflection_coeff το array **`a_mod`**:

$$a\_mod = [1, -w[0], -w[1], -w[2], -w[3], -w[4], -w[5], -w[6], -w[7]] \tag{3}$$

Με τους συντελεστές ανάκλασης k_r υπολογίζουμε τα Log-Area Ratios (παράγραφος 3.1.6 εξίσωση 3.4 του προτύπου), τα κβαντίζουμε (παράγραφος 3.1.7) και κατόπιν τα αποκβαντίζουμε (παράγραφος 3.1.8). Για τον κβαντισμό και τον αποκβαντισμό των Log-Area Ratios υλοποιήθηκαν και οι βοηθητικές συναρτήσεις **`Nint`**, **`A`** και **`B`**, ο ορισμός τους βρίσκεται στο τέλος του αρχείου **`encoder.py`**. Μετατρέπουμε τα αποκωδικοποιημένα Log-Area Ratios σε αποκωδικοποιημένους συντελεστές ανάκλασης. Καλούμε την συνάρτηση hw_utils.reflection_coeff_to_polynomial_coeff με είσοδο το array **`krd`** των αποκωδικοποιημένων συντελεστών ανάκλασης, για έξοδο παίρνουμε τους αποκωδικοποιημένους συντελεστές του προβλέπτη **`akd`** σε μορφή παρόμοια με αυτή στην Εξίσωση 3, δηλαδή:

$$akd = [1, -a_d[0], -a_d[1], -a_d[2], -a_d[3], -a_d[4], -a_d[5], -a_d[6], -a_d[7]] \tag{4}$$

Τέλος λαμβάνουμε το residual **`curr_frame_st_residual`** εκτελώντας τη συνέλιξη μεταξύ των αρχικών samples **`s`** και της εξόδου του array **`akd`**, δηλαδή $d = s * akd$. Αυτό προκύπτει από το βήμα (β) 1 της παραγράφου 2.1.2 της εκφώνησης της εργασίας αλλά και από την παρακάτω απόδειξη:

$$\begin{aligned} \hat{s}(n) &= \sum_{k=1}^{8} a_k s(n-k) && (5) \\ &= a_1 s(n-1) + a_2 s(n-2) + \ldots && (6) \\ d(n) &= s(n) - \hat{s}(n) && (7) \\ &= s(n) - a_1 s(n-1) + a_2 s(n-2) + \ldots && (8) \\ &= \sum_{k=0}^{8} a'_k s(n-k) && (9) \end{aligned}$$

όπου $a'_k = akd$, το οποίο ισχύει. (Πωωωω τρέλανέ μας δικέ μουυυ!!!)
Δεν υλοποιήθηκε interpolation των τιμών των Log-Area Ratios στον κωδικοποιητή μας.

Κανονικά σαν είσοδο του κωδικοποιητή θέλουμε και το prev_frame_st_resd, παρόλο που δεν το χρησιμοποιούμε, τι κάνουμε με αυτό;

2.2 Αποκωδικοποιητής