Index

Functions

Module **neural_network**

▶ EXPAND SOURCE CODE

Functions

```
def change_tempo(note_sequence, new_tempo)
```

Change the tempo of a NoteSequence to a new tempo.

Args:

`note_sequence` : The NoteSequence to change the tempo of.

`new_tempo` : The new tempo in beats per minute (BPM).

Returns:

The NoteSequence with the tempo changed.

▶ EXPAND SOURCE CODE

```
def chord_encoding(chord)
```

Encode a chord as a one-hot vector.

Args:

`chord` : The chord to be encoded

Returns:

The one-hot encoding of the chord.

▶ EXPAND SOURCE CODE

```
def create_song(va_mood: str, liked: bool)
```

Create a complete song based on the specified mood and user preference.

Args:

`va_mood` : The mood category choosen by the user

`liked` : Indicates whether the user liked the previous generated sequence.

Returns:

The generated MIDI file representing the complete song.

▶ EXPAND SOURCE CODE

```
def create_wav(midi: pretty_midi.pretty_midi.PrettyMIDI)
```

Create a WAV audio file from a MIDI file using the specified SoundFont.

Args:

`midi` : The MIDI file to convert

Returns:

The audio waveform as a floating-point numpy array.

▶ EXPAND SOURCE CODE

```
def fix_instruments_for_concatenation(note_sequences)
```

Fixes instrument assignments for concatenation of note sequences.

Args:

`note_sequences` : List of note sequences to fix instrument assignments.

▶ EXPAND SOURCE CODE

```
def generate_sequence(va_mood: str, liked: bool, num_bars: int)
```

Generate a music sequence based on the specified mood, user preference, and number of bars

Args:

`va_mood` : The mood category.

`liked` : Indicates whether the user liked the previous generated sequence.

`num_bars` : The number of bars to generate.

Returns:

The generated music sequence.

▶ EXPAND SOURCE CODE

```
def get_bpm_and_num_bars(mood: str, max_duration: float)
```

Get the BPM (beats per minute) and the number of bars based on the mood.

Args:

`mood` : The mood category.

`max_duration` : The maximum duration in seconds.

Returns:

A tuple containing the BPM and the number of bars.

▶ EXPAND SOURCE CODE

```
def initialize_model(main_path: str)
```

Initialize the models used for generation and interpolation importing the configuration of Google Magenta API.

Args:

`main_path` : Path of the src folder passed down from the main script.

▶ EXPAND SOURCE CODE

```
def interpolate_songs(va_value: str, liked: bool, num_bars: int, bpm)
```

Interpolate between songs based on the specified mood, user preference, number of bars, and BPM.

Args:

`va_value` : The mood used for the generation of new song.

`liked` : Indicates whether the user liked the previous generated sequence.

`num_bars` : The number of bars to generate.

`bpm` : The tempo in beats per minute (BPM) for the resulting interpolation.

Returns:

The interpolated music sequence.

▶ EXPAND SOURCE CODE

```
def slerp(p0, p1, t)
```

Spherical linear interpolation.

Args:

`p0` : The starting point of the interpolation.

`p1` : The ending point of the interpolation.

`t` : Evenly spaced samples, calculated over the interval [0, 1].

Returns:

The interpolated value between p0 and p1 at the given parameter value t.

▶ EXPAND SOURCE CODE

```
def to_midi(note_sequence, bpm)
```

Converts a NoteSequence to a MIDI file.

Args:

`note_sequence` : The NoteSequence to convert to MIDI.

`bpm` : The tempo in beats per minute (BPM) for the resulting MIDI file.

Returns:

The MIDI file representation of the NoteSequence.

▶ EXPAND SOURCE CODE

```
def trim_sequences(seqs, num_seconds=2.0)
```

Trim the sequences to a specified duration.

Args:

`seqs` : List of music sequences to be trimmed.

`num_seconds` : Duration in seconds to trim the sequences to. Default is BAR_SECONDS.

▶ EXPAND SOURCE CODE