General guidelines

- This assignment is tentatively due on **5pm Tuesday of week 10 (June 3),** with peer grades due during week 11. Final deadlines are **as posted on gradescope.** Note the Tuesday (rather than Sunday) deadline since (a) everyone will need adequate time to perform peer grading during week 11; and (b) I hope some people will perform their compositions in class!
- **Given that assignments must be distributed to peer graders, late submissions will not be accepted (or accepted for at most partial credit); there will be a late window though this is for students with an applicable OSD accommodation and not generally available to others.**
- Peer assignments will be available once the above late window closes (plus 12-24 hours for me to compile submissions and run various peer assignment scripts)
- Assignments can be completed in groups of up to 4, though may also be completed individually
- The assignment will be peer-graded. You are welcome to "blind" your submission (to the extent possible) though you are not required to. *As such, there will be no difference in grading based on the group size.*
- You are required to upload:
    1. A **jupyter notebook**, exported as html. It should be "clean", documented, and readable as-is; peer graders should not be expected to execute your code
    2. A **~20-minute presentation**, uploaded as a video file via a google drive link
    3. Generated music (.mid or .mp3 files)
    4. A peer grading report (due ~1 week later)
- **The above components are worth 20% of your grade.** The notebook and presentation aren't graded separately, rather, *peer graders should be expected to watch your presentation while following along with your code.* The exact format is up to you, though I recommend a combination of slides and code walkthrough. The specific graded components are described below.

Overall description

Task: make some beautiful music!

Choose any two of the following tasks:

1. **Symbolic, unconditioned generation:** I.e., train a model that learns a music distribution p(x) (presumably based on some training set) and samples from that distribution. A Markov chain (Module 3) is an example of such a model (though you

should generally avoid models already implemented in homeworks unless you can significantly extend them).[1]

2. **Symbolic, conditioned generation:** Similar to Option 1, except that generation is done conditionally, i.e., given some input. Example tasks include
    - harmonization (generate notes or chords following a melody)
    - or *vice-versa* (generate a melody which follows some chords)
    - music transcription (i.e., estimate the sequence of notes played in a waveform or spectrogram)
    - (others are also fine as long as the output is *symbolic*)
3. **Continuous, unconditioned generation:** Similar to Option 1, but with continuous output (i.e., generate a waveform). Note that if using a model that generates spectrograms you must also render the spectrogram as a waveform.
4. **Continuous, conditioned generation:** Example tasks include:
    - prompt-based generation (e.g. text-to-spectrogram)
    - "inpainting" (replace part of an existing waveform/spectrogram) or "outpainting" (extend an existing waveform/spectrogram)
    - continuous control (e.g. generate music that follows a given volume or pitch curve)
    - synthesis of a symbolic input (i.e., midi-to-audio using a learned model)

For examples of Options 1 and 2, see Module 3; for examples of Options 3 and 4, see Module 4. Note that you are not required to implement models "from scratch": you can use code from research papers (with acknowledgement), though you are expected to produce a complete pipeline, e.g. you should fit your own model weights using a training set, rather than running a binary or loading somebody else's weight files.

Presentation

Your presentation should focus on the following four sections (duplicated for each of your two tasks):

- (start the presentation by introducing your task)

1. Exploratory analysis, data collection, pre-processing, and discussion:
    - **Context:** Where does your dataset come from? What is it for, how was it collected, etc.?
    - **Discussion:** Report how you processed the data (or how it was already processed);
    - **Code:** Support your analysis with tables, plots, statistics, etc.
2. Modeling:

[1] Technically, you don't have to implement a model that estimates a probability distribution, though your solution should be based on ML.

    - **Context:** How do you formulate your task as an ML problem, e.g. what are the inputs, outputs, and what is being optimized? What models are appropriate for the task?
    - **Discussion:** Discuss the advantages and disadvantages of different modeling approaches (complexity, efficiency, challenges in implementation, etc.)
    - **Code:** Walk through your code, explaining architectural choices and any implementation details.
3. Evaluation:
    - **Context:** How should your task be evaluated? What should be the properties of a "good" output? What is the relationship between the objective being optimized by your model (e.g. perplexity) and musical properties (e.g. does it follow harmonic "rules"?) versus subjective properties?
    - **Discussion:** What are some baselines (trivial or otherwise) for your task? How do you demonstrate that your method is better than these trivial methods?
    - **Code:** Walk through the implementation of your evaluation protocol, and support your evaluation with tables, plots, statistics, etc.
4. Discussion of related work:
    - How has this dataset (or similar datasets) been used before?
    - How has prior work approached the same (or similar) tasks?
    - How do your results match or differ from what has been reported in related work?
    - (you can put this section at the beginning if you'd prefer)

- I would suggest playing your music at the end of the presentation just in case graders have trouble with your files; this shouldn't count against the 20 minute requirement.

Note that you should repeat the above format for each of the two tasks, though if your tasks are closely related you can potentially combine some sections. You can also reorder sections ***as long as your organization is very obvious to graders.*** Try not to go overlength by more than ~10% (not including the time to play your generated music): I would suggest that graders allocate 20-30 minutes per assignment for grading, so if you go too long you risk having graders watch your presentation at 1.5x speed or missing out on the end.

Rubric and grading

The assignment is worth 20 marks. The grade breakdown is as follows:

- Two tasks (above): **8 marks each**
- Participate in peer grading (grade four assignments): **4 marks**

Graders should grade each of the four parts (for each of the two tasks) *roughly* according to the following rubric:

0: This component was not covered

0.5: This component was covered, but appears to contain errors, or cannot be easily understood, or the presentation does not seem to match the code, or the presentation seems to be automatically generated
1: This component was covered, but only superficially, i.e., it has some of the right components, and appears correct, but is missing key elements or comparisons that would be expected from a complete analysis
1.5: This submission is more-or-less minimally acceptable: e.g. a minimal set of essential elements are covered, though maybe not discussed in detail
1.75: The code and presentation seems feature-complete: all the results "make sense", or any negative results are adequately explained; the presentation explains the tradeoffs between different metrics and goals, is clear and easy to follow, etc. ***This should probably be the most common grade.***
2: The code and presentation go above-and-beyond in some way, e.g. it might be particularly thorough, try an interesting or new approach, or shows excellence in some other way

Graders will also be asked to assess musical quality (or really "interestingness"): **this component is not graded,** I just want to be able to find good examples to share with everyone.

Submission instructions

Unfortunately, (a) gradescope isn't set up for peer grading; (b) gradescope doesn't accept large files; and (c) google drive files can't be shared anonymously. So submission is a little complicated... It will work as follows:

Step 1: Submit the following files

- **workbook.html:** this should be a jupyter notebook, exported as html. *You should be able to open it in a browser.*
- **video_url.txt:** should contain a single line, which is a *path to a shared mp4 file on google drive.* It should probably look like: https://drive.google.com/file/d/FILE_ID/view?usp=drive_link
- Two of (depending on which tasks you attempt):
    - **symbolic_unconditioned.mid**
    - **symbolic_conditioned.mid**
    - **continuous_unconditioned.mp3**
    - **continuous_conditioned.mp3**

The autograder will check the file names though you *can* use a different extension if your task doesn't produce (e.g.) midi. E.g. if you symbolically generate music in ABC notation using a language model, you might submit a .txt file. You probably *cannot* submit (e.g.) uncompressed .wav as you'll overrun gradescope file size limits, so please compress your files. If submitting in a format other than the above, maybe ask on gradescope first. You're welcome to submit files other than the ones above, though the autograder will not check them.

Probably, your video url should be generated on google drive by enabling “anyone with the link” sharing and copying the link. If you use some other format, please convert to mp4 (which can be generated by recording a zoom session or similar). Please do your best to make sure your video file is watchable on “normal” hardware (e.g. in the past some people have used a video codec that is not generally available).

The autograder will check that (a) your file is downloadable; (b) greater than 1mb; (c) is of type video/mp4; (d) that your workbook.html file is a html file; and (e) that you have submitted files for two tasks.

Note that your google drive link *won't be shared:* it is not possible to fully anonymize shared files, so I will download your file and put it in a shared folder. Beyond that, it is up to you to what extent you'd like to anonymize your submission.

Step 2: 1-2 days later, I will compile all submissions, make peer assignments, and distribute peer assignments via another gradescope assignment (details to follow). Please be patient as it is quite a process!

Musical performances

I've tentatively planned to reserve week 10 (June 3 and 5) for musical performances. That is, you'll have the opportunity to perform your work for the class! This is not worth any marks (since it would not be fair for people who are not musically trained or extroverted), it is just **for fun.**

If you'd like to give a performance, I'd note the following:

- Your performance should be based on your assignment, i.e., playing your piece live, improvising / singing over it, etc.
- Your A/V requirements need to be pretty minimal / foolproof. The room has a headphone jack + lavalier mic. You can use my (usb 2) midi keyboard if you like but I don't have much else! I'll probably broadcast to twitch just using a mobile phone. **(if anyone has better suggestions for an A/V setup, welcome to share!)**
- Keep performances fairly short (e.g. 5 minutes + a couple of minutes set up time)

I'll circulate a form to gather interest in week 8 or 9. If nobody is interested in giving a performance you'll probably have to listen to me sight-reading various submissions, or worse, playing some stuffy classical repertoire.

How your submission is verified

In case you'd like to check at home, your submission is verified using code similar to the following:

```
import unittest
```

```
import os
import magic
import requests

def test_workbook(path):
    f = open("workbook.html", 'r')
    t = f.read()
    if not t.startswith('<!DOCTYPE html>'):
        print("Your workbook submission is not a html file (does not start with
'<!DOCTYPE html>'")
        print("Please save your work as html (e.g. do not just change the extension)")
        raise Exception
    f.close()

def test_video(url):
    FILE_ID = None
    if "id=" in url:
        FILE_ID = url.split("id=")[1].split('/')[0]
    elif '/d/' in url:
        FILE_ID = url.split("/d/")[1].split('/')[0]
    if FILE_ID == None:
        print("Couldn't determine file ID; expected a url of the form
https://drive.google.com/file/d/FILE_ID/view?usp=drive_link or
https://drive.google.com/uc?export=download&id=FILE_ID")
        print('(got """' + url + '""")')
        raise Exception
    print("Looks like your google drive file ID is " + FILE_ID)
    url_fixed = "https://drive.google.com/uc?export=download&id=" + FILE_ID
    r = requests.get(url_fixed, stream=True)
    print("Trying to download from " + url_fixed)
    chunk = next(r.iter_content(chunk_size=1024*1024))
    if len(chunk) < 1024*1024:
        chunk = chunk.decode('utf8')
        if "Virus scan warning" in chunk:
            print("Looks like your file is too large for Google to scan for viruses")
            uuid = chunk.split('uuid" value="')[1].split('"')[0]
            url_fixed = "https://drive.usercontent.google.com/download?id=" + FILE_ID
+ "&export=download&confirm=t&uuid=" + uuid
            print("Trying to download from " + url_fixed)
            r = requests.get(url_fixed, stream=True)
            chunk = next(r.iter_content(chunk_size=1024*1024))
        else:
            print("Looks like your video file is less than 1mb; it is probably not a
video")
            raise Exception
    print("Confirmed that your video file is greater than 1mb; checking file type")
    mime = magic.Magic(mime=True)
    typename = mime.from_buffer(chunk)
    if not "video/mp4" in typename:
        print("Your file is not an mp4 video file (type=" + typename + ")")
        raise Exception
    print("Everything looks okay! On your own, please verify that you can download a
working video using the assignment script in the spec")
    #print("wget -O output.mp4 '" + url_fixed + "'")
```

```
class TestFiles(unittest.TestCase):
    @weight(0)
    def test_submitted_files(self):
        """Check submitted files"""
        missing_files = check_submitted_files(['workbook.html', 'video_url.txt'])
        for path in missing_files:
            print('Missing {0}'.format(path))
        self.assertEqual(len(missing_files), 0, 'Missing some required files!')
        taskfiles = [f for f in os.listdir('.') if f.split('.')[0] in
["symbolic_conditioned", "symbolic_unconditioned", "continous_conditioned",
"continuous_unconditioned"]]
        print("detected task files: " + str(taskfiles))
        self.assertTrue(len(taskfiles) >= 2, 'Need at least two music files')
        print('All required files submitted!')
        test_workbook("workbook.html")
        f = open("video_url.txt", 'r')
        url = f.read().strip()
        test_video(url)
        f.close()
```