

Homework 1 Part 2

An Introduction to Neural Networks

11-785: INTRODUCTION TO DEEP LEARNING (FALL 2026)

Schedule

Release

Friday, August 28, 2026, 6:00 PM ET

- This homework is scored out of **100 points**, earned entirely through your final submission. The checkpoint is **mandatory**: missing it reduces your HW1P2 score by **3%**.
- ☐ Find the **HW1P2 post on Piazza**. It contains the invitation link you need to join the Kaggle competition.

Checkpoint Deadline (Mandatory, 3% deduction if missed)

Friday, September 4, 2026, 11:59 PM ET

- ☐ Complete HW1P2 **Starter** notebook and submit to **Kaggle**. Achieve the **checkpoint cutoff of 60%**
- ☐ Complete the HW1P2 **Checkpoint** notebook, achieve **FULL** score on MCQ, submit to **Gradescope** and obtain **FULL** score.
- **ALL** above requirements must be met. Otherwise, your HW1P2 score will be reduced by **3%**.
([Click here for submission instructions](#))

On-Time Deadline (100 pts)

Kaggle: Friday, September 18, 2026, 11:59 PM ET

Code submission: Sunday, September 20, 2026, 11:59 PM ET

- Cutoff scores will be released on Gradescope after the checkpoint deadline.
- ☐ [Click here to see P2 Scoring Details](#)
- ☐ Submit final scores to the [Kaggle competition](#) by **Friday, September 18, 2026, 11:59 PM ET**
- ☐ Clean up your code notebook, complete the final submission section, and submit the **output zip file to Gradescope by Sunday, September 20, 2026, 11:59 PM ET** (**REQUIRED**, otherwise 0/100 will be assigned for this homework)
([Click here for submission instructions](#))

Slack Submission

Slack Kaggle: Friday, September 25, 2026, 11:59 PM ET

Code submission: Friday, September 25, 2026, 11:59 PM ET

- ☐ Join and submit your final predictions to the **SLACK** Kaggle competition. Its link is released on Piazza **after the on-time Kaggle deadline closes**. Do not look for it before then.
- ☐ Submit to Gradescope with the same On-Time Deadline submission instructions.
- You are in charge of keeping track of the slack days used, and you will be ineligible for bonus points for reaching high-cutoff for this homework.

Policies

- You are expected to comply with the [University Policy on Academic Integrity and Plagiarism](#).
- You are allowed to talk with and work with other students on homework assignments.
- You can share ideas but not code, you must submit your own code. All submitted code will be compared against all code submitted this semester and in previous semesters using [MOSS](#).
- You are allowed to help your friends debug
- You are allowed to look at your friends code
- You are allowed to copy math equations from any source that are not in code form
- You are not allowed to type code for your friend
- You are not allowed to look at your friends code while typing your solution
- You are not allowed to copy and paste solutions off the internet
- You are not allowed to import pre-built or pre-trained models
- You are not allowed to use any external data or datasets at any point in this assignment.
- Meeting regularly with your study group to work together is highly encouraged. You may discuss ideas and help debug each other's code. You can even see from each other's solution what is effective, and what is ineffective. You can even "divide and conquer" to explore different strategies together before piecing together the most effective strategies. However, the actual code used to obtain the final submission must be entirely your own.

Note:

- The maximum number of parameters allowed for the models for this HW is 20 million (including ensembles).
- Do **NOT** use any other architecture or trick: no CNNs, no convolutional layers, no RNNs (GRUs/LSTMs), no attention or transformer modules, no graph networks, no pretrained encoder/decoder models, no pretrained weights (you need to train from scratch), etc.

Homework Objectives

After this homework, you would ideally have learned:

- To solve a medium-scaled classification problem using an MLP
 - How to set up the MLP
 - How to handle the data
 - How to train the model
 - How to optimize the model
- To explore architectures and hyperparameters for the optimal solution
 - To identify and tabulate all the various design/architecture choices, parameters and hyperparameters that affect your solution
 - To devise strategies to search through this space of options to find the best solution.
- The process of staging the exploration
 - To initially set up a simple solution that is easily implemented and optimized
 - To stage your data (e.g. by initially working on a subsample of the training data) to efficiently search through the space of solutions.
 - To track losses and performance on validation data to ensure the code is working properly and the model is being trained properly
 - To subset promising configurations/settings and then evaluate those on the larger (complete) dataset
- To engineer the solution using your tools
 - To use objects from the PyTorch framework to build an MLP.
 - To deal with issues of data loading, memory usage, arithmetic precision etc. to maximize the time efficiency of your training and inference.

Checklist

Here is a checklist page that you can use to keep track of your progress as you go through the writeup and implement the corresponding sections in your starter notebook. As you complete each function in the notebook, you can check the corresponding boxes aligned with each section. It is recommended that you go through this writeup and starter notebook simultaneously step by step.

1. Getting Started

- ☐ Complete the Homework 1 quiz in the Checkpoint notebook
- ☐ Download the starter code notebook and set up a virtual environment (optional)
- ☐ Install the Kaggle API and create a directory
- ☐ Download the dataset (`.npy` log-mel files and `.txt` transcripts) from Kaggle

2. Complete the *AudioDataset* class

- ☐ Revise recitations for Dataloaders and OOP if necessary, refer Appendix
- ☐ Define label index mapping in `__init__()`
- ☐ Assign data index mapping, length, context, and offset to self
- ☐ Zero pad data as needed for context size
- ☐ Calculate starting and ending timesteps using offset and context in `__getitem__()`
- ☐ Return reshaped frames at index with context and corresponding phoneme label

3. Complete the training loop *train()*

- ☐ Create dataloader for the training dataset
- ☐ Set model in “Training Mode”
- ☐ Clear the gradients
- ☐ Compute the model output and the loss
- ☐ Complete the backward pass
- ☐ Update model weights

4. Complete the inference loop *eval()*

- ☐ Set model to “Evaluation Mode” and create the validation set dataloader
- ☐ Compute model output in “no grad” mode
- ☐ Calculate validation loss using model output
- ☐ Get most likely phoneme as prediction from the model output
- ☐ Calculate validation accuracy

5. Hyperparameter Tuning

- ☐ Make your initial submission before the checkpoint deadline
- ☐ Use Weights & Biases to log metrics for each epoch (required for the final submission)
- ☐ Make sure the model is saved after every few epochs
- ☐ Try changing the parameters listed in Section 11 to improve results
- ☐ Attend office hours if you need help :)

Objectives

The main goal of this homework is to explore neural networks for speech recognition, mainly focusing on phoneme labelling.

Speech recognition is an important field in deep learning with multiple applications. Speech is a fundamental form of human communication. Speech recognition involves converting spoken language into written text or data that could be understood by machines. Speech data refers to audio recordings of human speech, while phonemes represent the smallest units of sound that can convey a different meaning (bake, take: b,t). Spectrograms are visual representations of the acoustic properties of speech signals, i.e. captures the changes in the frequency over time.

The dataset provided has speech data in the form of log-mel spectrograms (more about it ahead in the writeup). The training data has the corresponding phonemes for this data and we need to train an MLP for predicting the same on the test dataset.

In this homework, you would be building a multilayer perceptron (MLP) that can effectively recognize and label the phonemes in the training data. An MLP is a type of neural network that comprises multiple layers of perceptrons, that help it capture the features and patterns of the data. Here, we aim to deepen your knowledge of neural networks, and in addition to that you would work on parameter and hyperparameter tuning. This would help it generalize and predict more accurately and hence enhance its ability to recognize and identify the phonemes.

Introduction : Frame Level Classification of Speech

This part of the homework is a live competition on [Kaggle](#).

In this challenge you will take your knowledge of feedforward neural networks and apply it to a more useful task than recognizing handwritten digits: speech recognition. You are provided a dataset of audio recordings (utterances) and their phoneme labels. The data comes from the **LibriSpeech** corpus of read English speech: public-domain audiobooks (LibriVox) that are read aloud and labelled using the original text. If you have not encountered speech data before or have not heard of phonemes or spectrograms, we will clarify the problem further.

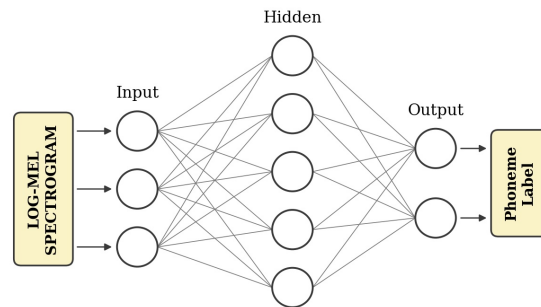


Figure 1: Frame-level speech recognition: a multilayer perceptron maps a log-mel spectrogram frame (together with its surrounding context frames) to one of the 40 phoneme labels.

1 Task

Your task is to generate predictions for the phonemes of the test set. You will be evaluated on the accuracy of the prediction of the phoneme labels for each frame in the test set. Grade cut-offs are released after the checkpoint deadline. For detailed information, please look at the [Kaggle Page](#). If you have not encountered speech data before or have not heard of phonemes or spectrograms, we will clarify the problem further. The training data comprises of:

- Speech recordings (log-mel spectrogram frames)
- Frame-level phoneme labels

The test data comprises of:

- Speech recordings (log-mel spectrogram frames)
- Phoneme labels are not given

Your job is to identify the phoneme label for each frame in the test data set. It is important to note that utterances are of variable length.

2 Getting Started

The starter notebook contains the basic building blocks for training the MLP. Students must follow the flow diagram provided below to get started.

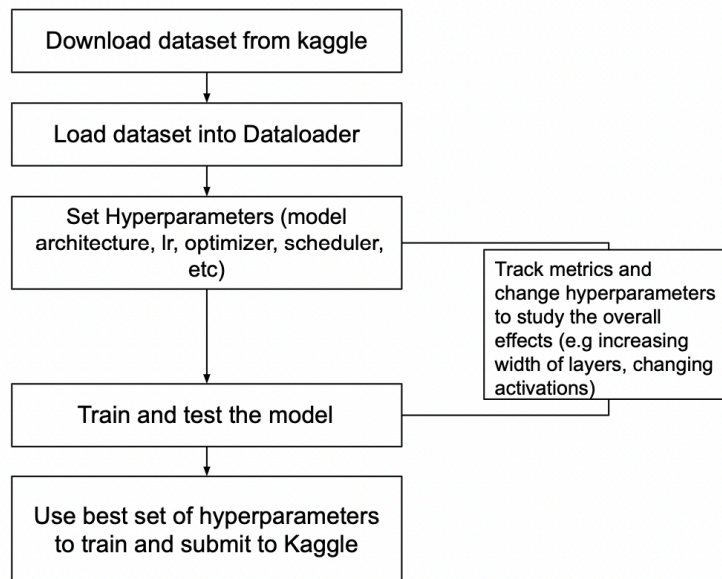


Figure 2: How to use the starter notebooks.

The starter notebook contains a dataset class - AudioDataset. The class reads all utterances from the folders into a list, and concatenates all of them to create one huge utterance along with their corresponding labels. The concatenated array of utterances are then padded with zeros on the beginning and the end as mentioned in Section 5.2. The class returns a frame of interest with context frames around it, and the frame’s corresponding phoneme label at each index. For detailed information on overall deep learning workflow, refer to Recitation 0.

3 Phonemes and Phoneme States

As letters are the atomic elements of written language, phonemes are the atomic elements of speech. It is crucial for us to have a means to distinguish different sounds in speech that may or may not represent the same letter or combinations of letters in the written alphabet.

For this challenge, we will consider a total of 40 phonemes in this language, listed below in label-index order (0–39):

["AA", "AE", "AH", "AO", "AW", "AY", "B", "CH", "D", "DH", "EH", "ER", "EY", "F", "G", "HH", "IH", "IY", "JH", "K", "L", "M", "N", "NG", "OW", "OY", "P", "R", "S", "SH", "T", "TH", "UH", "UW", "V", "W", "Y", "Z", "ZH", "[SIL]"]

A powerful technique in speech recognition is to model speech as a markov process with unobserved states. This model considers observed speech to be dependent on unobserved state transitions. We refer to these unobserved states as phoneme states or subphonemes. For each phoneme, there are 3 respective phoneme states. The transition graph of the phoneme states for a given phoneme is as follows:

Hidden Markov Models (HMMs) estimate the parameters of this unobserved markov process (transition and emission probabilities) that maximize the likelihood of the observed speech data.

Your task is to instead take a model-free approach and classify log-mel spectrogram frames using a neural network that takes a frame (plus optional context) and outputs class probabilities for all 40 phonemes. Performance on the task will be measured by classification accuracy on a held out set of labeled log-mel spectrogram frames. Training/dev labels are provided as phoneme strings (one label per frame), which you should map to integer class indices [0-39] following the order of the phoneme list above.

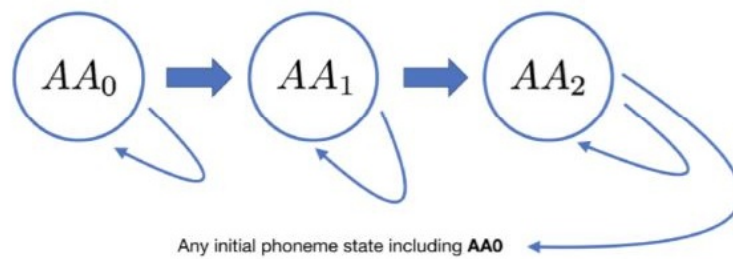


Figure 3: Transition graph of the three phoneme states (subphonemes) of a single phoneme, shown here for AA. Each state may self-loop, and the final state transitions into the first state of any following phoneme.

Thinking Note

Prompt Engineering with AI – Change the label object:

- "How many phonemes is the model predicting?"
- "How many phoneme states is the model predicting?"

The answers imply different output spaces.

4 Speech Representation

Raw speech signal (also known as the *speech waveform*) is stored simply as a sequence of numbers that represent the amplitude of the sound wave at each time step. This signal is typically composed of sound waves of several different frequencies overlaid on top of one another. For human speech, these frequencies represent the frequencies at which the vocal tract vibrates when we speak and produce sound. Since this signal is not very useful for speech recognition if used directly as a waveform, we convert it into a more useful representation called a **"log-mel spectrogram"** in the feature extraction stage.

5 Feature Extraction

The variation with time of the frequencies present in a particular speech sample are very useful in determining the phoneme being spoken. In order to separate out all the individual frequencies present in the signal, we perform a variant of the Fourier Transform, called the **Short-Time Fourier Transform (STFT)** on small, overlapping segments (called frames, each of 25ms) of the waveform.

A single vector is produced as the result of this transform. Since we use a stride of 10ms between each frame, we end up with 100 vectors per second of speech. Finally, we convert each vector into a 32-dimensional vector (refer the links in the optional readings section for exact details of how this is done). For an utterance T seconds long, this leaves us with a matrix of shape (100*T, 32) known as the **log-mel spectrogram**.

Note that in the dataset provided to you, we have already done all of this pre-processing and provided the final **(*, 32) shaped log-mel spectrograms** to you. You can refer to the recitation on Data Pre-processing to understand this better. An illustration of this process is represented in the figure below.

The data provided in this assignment consists of these log-mel spectrograms, and phoneme labels for each 32-dimensional vector in the log-mel spectrogram. The task in this assignment is to predict the label of a particular 32-dimensional vector in an utterance.

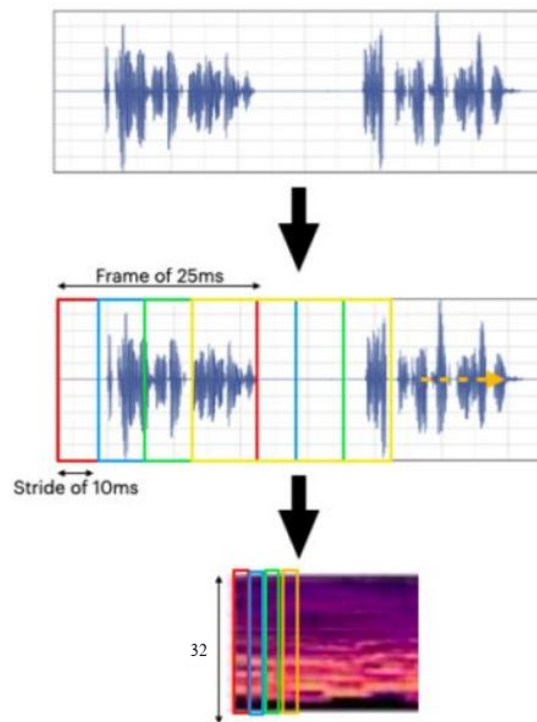


Figure 4: Feature extraction: converting a raw speech waveform into a log-mel spectrogram.

5.1 Data Files

You can find the training and test data under the “Data” [tab](#) on Kaggle.

A diagrammatic representation of the data is as follows:

```

11785-F26-HW1P2
├── train-clean-100
│   ├── log-mel ..... 26624 .numpy files
│   └── transcript ..... 26624 .txt files
├── dev-clean
│   ├── log-mel ..... 2545 .numpy files
│   └── transcript ..... 2545 .txt files
└── test-clean
    └── log-mel ..... 2434 .numpy files, no transcripts
  
```

The folder is named 11785-F26-HW1P2 after extraction, which is the path the starter notebook expects.

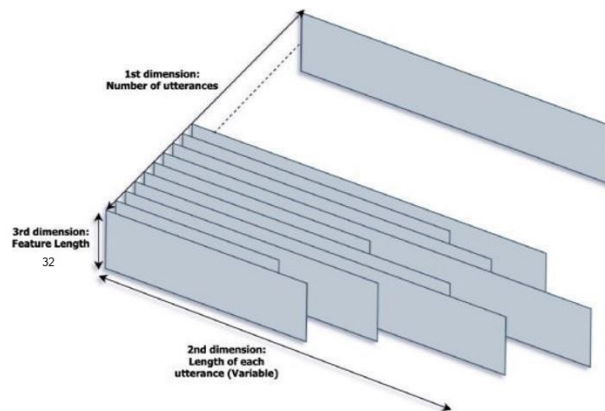


Figure 5: Data representation.

We recommend that you run your code and ablations using a fraction of the train-clean-100 training set, and then train the model again using the full training set once you figure out your best model and configuration.

5.2 Context

Since each vector represents only 25ms of speech it may not be sufficient to feed only a single vector into the network at a time. Instead, it may be useful to provide the network with some “**context**” of size K around each vector in terms of additional vectors from the speech input.

Concretely, a context of size 5 would mean that we provide an input of size $(11, 32)$ to the network - the size 11 may be explained as : the vector to predict the label for, 5 vectors preceding this vector, and 5 vectors following it.

It is worth thinking about how you would handle providing context before one of the first K frames of an utterance or after one of the last K frames.

Thinking Note

Prompt Engineering with AI – Change the unit:

- “What does the model predict for an utterance?”
- “What does the model predict for a single frame?”

The answers differ in how labels, accuracy, and batching are described.

Hint: There are several ways to implement this, but you could try:

1. Concatenating all utterances and padding with K 0-valued vectors before and after the resulting matrix

OR

2. Pad each utterance with K 0-valued vectors, at the extra cost of bookkeeping the beginning index of each utterance’s first vector.

Note: When loading the data and adding context, you might want to add some zero padding to every utterance for better results. The way we add context for the first and last frame of the utterance is to add some form of zero padding. For example, if we consider a single utterance of dimension $(1000, 32)$ and we consider the context to be 5, then we want to implement zero padding before and after this utterance

such that the dimension becomes (1010, 32). Subsequently, the input layer will have $(1+2*\text{context_size})*32$ nodes.

Try to think about why this calculation would make sense with respect to the Dataloader and the speech data given. Context is a hyperparameter and the recommended value of context to be set for this homework is between 0-50. For more information on context, refer to Appendix B.2

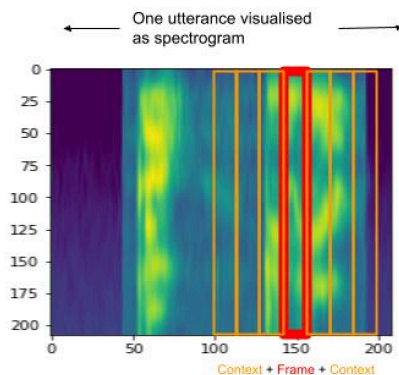


Figure 6: Context: movement of the spectrum from adjacent frames on either side gives cues about what the frame in question could represent.

Thinking Note

Prompt Engineering with AI – Shift what “context” refers to:

- “How does context size affect the model?”
- “How does context size change the input dimension?”

The explanations emphasize different parts of the pipeline.

5.3 Cepstral Normalization

In speaker recognition, we want to remove any channel effects (impulse response of vocal tract, audio path, room, etc.). Cepstral Normalization is a helpful technique you could perform in order to achieve this.

Providing that the input signal is $x[n]$ and channel impulse response is given by $h[n]$, the recorded signal is a linear convolution of both, and by taking the Fourier Transform, due to the convolution-multiplication equivalence property of Fourier Transform, we get $Y[f] = X[f] \cdot H[f]$.

The next step in the calculation of cepstrum is taking the logarithm of the spectrum:

$$Y[q] = \log Y[f] = \log(X[f] \cdot H[f]) = X[q] + H[q]$$

because of the log property. Now we know that in the cepstral domain any convolutional distortions are represented by addition. Let’s assume that all of them are stationary (which is a strong assumption as a vocal tract and channel response are not changing) and the stationary part of speech is negligible. We can observe that for every i -th frame true is:

$$Y_i[q] = H[q] + X_i[q] \quad (1)$$

By taking the average over all frames we get:

$$\frac{1}{N} \sum_i Y_i[q] = H[q] + \frac{1}{N} \sum_i X_i[q] \quad (2)$$

If we define the difference between $Y_i[q]$ and the average:

$$\begin{aligned}
D_i[q] &= Y_i[q] - \frac{1}{N} \sum_j Y_j[q] \\
&= H[q] + X_i[q] - \left(H[q] + \frac{1}{N} \sum_j X_j[q] \right) \\
&= X_i[q] - \frac{1}{N} \sum_j X_j[q]
\end{aligned} \tag{3}$$

We now removed channel effects from our signal. In simple words, to our cepstrum, **we have subtracted the average from each coefficient and divided by standard deviation** to perform Cepstral Normalization.

6 Evaluation

This assignment will be scored out of **100 pts**, all of which are earned through your **Final Submission**.

The **Checkpoint Submission** carries no points of its own, but it is **mandatory**: if you do not meet every checkpoint requirement by Friday, September 4, 2026, 11:59 PM ET, **your HW1P2 score is reduced by 3%**. The deduction is proportional to the score you earn. For example, a score of 80 becomes 77.6, and a score of 100 becomes 97.

The Final Submission component will be graded as the best of the On-Time Submission and Slack Submission (If you only submitted either On-Time or Slack, we will just take the one you submitted).

6.1 Checkpoint Submission:

The checkpoint will consist of two primary tasks:

- Complete the P2 **Starter** notebook to make a submission to the Kaggle competition that passes the checkpoint cutoff of 60%.
- Download and complete the P2 **Checkpoint** notebook which will typically hold a checkpoint quiz (recommended to be completed on Google Colab for best user experience) and some logistical submission verifications (primarily kaggle username information and checkpoint score verification).

Missing either task by Friday, September 4, 2026, 11:59 PM ET results in the 3% deduction described above.

6.2 On-Time Submission:

Performance cut-offs for the Kaggle competition will be released on Gradescope soon after the checkpoint deadline has passed. We will release the following cutoffs with the corresponding point values:

- High (100/100)
- Medium (80/100)
- Low (60/100)
- Very Low (40/100)

Scores will be interpolated linearly between these cutoffs.

6.3 Bonus Points:

Bonus points will be assigned **ONLY** if the submission exceeds the High cutoff **AND** was submitted by the final deadline (Slack submissions are ineligible). Up to 5 bonus points are awarded.

Bonus points are calculated by linearly interpolating between the High cutoff and the highest individual's private score. The highest performance will get 5 bonus points. Bonus points are calculated and assigned at the end of the semester.

6.4 Slack Submissions:

- **Scores submitted to the original Kaggle Competition after the final deadline will NOT be considered, no matter how many slack days a student has left.** A student wishing to submit a P2 score after the final deadline must join and continue by submitting to the **SLACK Kaggle Competition** instead. **The Slack competition is only opened once the on-time Kaggle deadline has passed**, and its link is announced on Piazza at that time, so that students cannot accidentally submit to the wrong competition. The Slack Kaggle deadline is Friday, September 25, 2026, 11:59 PM ET, and the corresponding code submission is due Friday, September 25, 2026, 11:59 PM ET.

- A student is granted up to 10 Slack days, which they may use across all P2 final submissions (and no other type of assignments) at their discretion throughout the semester. Once a student runs out of Slack days, the best viable submission before the student ran out of slack days will be considered.
- Students are in charge of tracking their own usage of Slack days.
- As mentioned, submissions made to slack will be ineligible for the bonus points for achieving high cutoff by the original final deadline.

7 Checkpoint Submission Instructions

The Checkpoint-Kaggle Submission and Checkpoint-Quiz Completion may be completed in any order:

7.1 Checkpoint-Kaggle Submission:

- Download and complete the HW1P2 **Starter** Notebook.
- Submit a score of 60% or better on Kaggle

7.2 Checkpoint-Quiz Completion:

- Download and complete the HW1P2 **Checkpoint** Notebook.
- Answer all checkpoint-quiz questions and achieve full score.
- Run all the cells in the checkpoint notebook, and submit the output .json file to the checkpoint Gradescope assignment as needed until full score on the quiz questions is achieved.
- It is not required to have finished the kaggle submission component yet, but you will still need to input your kaggle username and api key.

7.3 Checkpoint-Submission:

- Once both the Checkpoint-Kaggle submission and the Checkpoint-Quiz Completion components are completed, re-run all the cells of the HW1P2 **Checkpoint** Notebook and submit the resulting .json file to the checkpoint Gradescope assignment.
- Be sure to achieve full score on this submission. If full score is not achieved, that may mean something is wrong with your submission.

8 Final (On-Time/Slack) Submission Instructions

Main Submission Instructions:

You generate this submission from the last section of your HW1P2 notebook, titled “**Gradescope Submission Instructions**”. Work through its steps in order:

- Assign your final trained model to the global variable `MODEL`. The notebook uses it to generate `model_metadata.json`, which records your parameter count and architecture. `MODEL` must be the **exact** model behind your best Kaggle submission. If it does not match, your score may be invalidated.
- Fill in the `README` variable with a brief description of your work.
- Provide the required credentials:
 - `KAGGLE_USERNAME`, `KAGGLE_API_KEY`
 - `WANDB_API_KEY`, `WANDB_USERNAME_OR_TEAMNAME`, `WANDB_PROJECT`

These are used only to fetch your official Kaggle score and to export your top Weights & Biases runs for grading, so **you must log your runs to W&B while working on this homework**.

- Set the file paths for your platform, and attach any optional additional files.
- Run the “Generate the Final Submission Zip” cell (do **not** modify it) and upload the resulting zip to the Gradescope Final assignment.

If you are submitting late, submit to the Slack Kaggle competition first (link announced on Piazza after the on-time deadline), then upload the same Gradescope zip by Friday, September 25, 2026, 11:59 PM ET.

9 Submission Format

Your Kaggle submission must be a CSV file with two columns, **ID** and **TARGET**:

- **ID**: The 1-based index of the frame in the test set, ranging from 1 to 1763971 (data type: int).
- **TARGET**: The predicted phoneme for that frame, given as a string (e.g. [SIL]).

There is exactly one row per test frame, so the file contains 1763971 prediction rows plus the header row **ID,TARGET**. A `sample_submission.csv` is provided with the dataset, and the starter notebook generates a correctly-formatted `submission.csv` for you. Refer to the Kaggle competition page for the daily submission limit.

10 Ablations

Ablations are performed by the team to get an idea about different parameters and hyperparameters tuning and how the models would work alongside getting the best model.

If planned properly by the team, ablations would be very helpful and handy when it comes to tuning the model.

An example of an ablation sheet: [Ablation Sheet](#)

11 Hyperparameter Tuning

Below are a few variations in hyperparameters that might help you reach the low cutoff.

Hyperparameters	Values
Number of Layers	2-8
Activations	ReLU, LeakyReLU, softplus, tanh, sigmoid
Batch Size	64, 128, 256, 512, 1024, 2048
Architecture	Cylinder, Pyramid, Inverse-Pyramid, Diamond
Dropout	0-0.5, Dropout in alternate layers
LR Scheduler	Fixed, StepLR, ReduceLROnPlateau, Exponential, CosineAnnealing
Weight Initialization	Gaussian, Xavier, Kaiming(Normal and Uniform), Random, Uniform
Context	0-50
Batch-Norm	Before or After Activation, Every layer or Alternate Layer or No Layer
Optimizer	Vanilla SGD, Nesterov's momentum, RMSProp, Adam
Regularization	Weight Decay
LR	0.001, you can experiment with this
Normalization	You can try Cepstral Normalization

Table 1: Hyperparameter Tuning

Weight initialization is an important aspect before training a neural network. It refers to defining initial values for the neural network prior to training. The weights have to be appropriately assigned to ensure good performance of the model.

Along with these, [R-Drop](#): Regularized Dropout might also lead to better performance. Additionally, masking might also lead to better performance.

Masking is the technique to mask or block certain segments of the audio data to enhance the model during training. The main two types of masking include the following:

1. Time masking: block the time frames of the audio signal
2. Frequency masking: block the frequency bands of the audio signal

Masking would help reduce noise in the data and help better generalization for the model.

12 Optional Reading

Go through this [link](#) to understand how log-mel spectrograms are generated.

A Checkpoint Architecture

Checkpoint Cutoff Architecture (60%) This is the default architecture shipped in your starter notebook. It should cross the 60% checkpoint cutoff within a few epochs when trained on **train-clean-100**.

Context: 30

Hidden layers: 4 (2048, 2048, 2048, 512)

Activation: ReLU

Learning rate: 1e-3

Optimizer: AdamW

Batch size: 2048

Loss: CrossEntropyLoss

B Dataloader

Data is the backbone of any Deep Learning system and Pytorch makes it easy to do this using the Dataloader functionality. Before beginning we would strongly recommend watching the dataloader recitation.

Having watched the recitation, you know the dataset class is crucial since this can speed up your training significantly and if you do not implement it efficiently it could be the bottleneck.

B.1 Data

Our data consists of a folder containing log-mel spectrograms (**log-mel**, one **.npy** array per utterance), and another containing the transcripts (**transcript**, one **.txt** file per utterance). We have 26624 recordings and corresponding 26624 transcripts, each recording is of different lengths. Which basically means for each timestep, we have a translated phoneme. For instance, the first recording ("103-1240-0000") has shape (1408,32) and the transcript is of length 1408, exactly one phoneme label per frame.

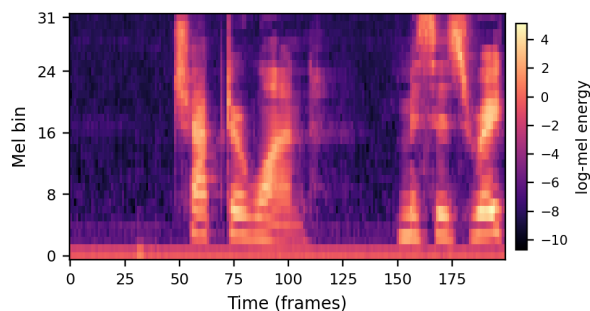


Figure 7: Visualization of a log-mel spectrogram.

B.2 Context

As explained above, to be able to understand what phoneme a timestep corresponds to, we need some "context" and for that we look for some information before the timestep and some information after the timestep. To make this clearer, let's consider some timestep t , and some context c , and further let the phoneme corresponding to the this timestep be p . For our network we want the dataloader to take the information from $[t - c, t + c]$ (our input) and return the phoneme (our target).

For example, let's consider a short example. Say we considered frames as text for the sake of simplicity in illustration, and we had the following recording -

```
feats = [A B C D E]
```

Now in your `__getitem__` function, if we wanted to get the frame at index 0, without any context, we would get `feats[0]=A`. Now we want to pad this recording with context zeros at the beginning and the end, to provide the MLP with window frames, instead of one frame at a time, to predict the phoneme at index `i`. Let's consider context as 3. So the `feats` array now becomes:

```
feats = [ 0 0 0 A B C D E 0 0 0]
```

Now if we wanted to access the first frame at index `i = 0`, we would get `feats[0] = 0`, which is not intended, as what we really want to access is A, with 3 frames behind it, and 3 frames in front.

So your start and end indices for slicing a window of frames would theoretically look like this first:

```
start = i - context
end = i + context + 1
window_of_interest = feats[start:end] = feats[i-context:i+context+1]
```

But because we pad to get context around the first and last frame, we have to now offset the padding to access the intended frame of interest. The way we would access A again would be using `frame[0+context]`. We have to offset our starting index for a frame at index `i` because we padded the beginning and end of our recording. We do the same for the ending index. So the right way to access our frame window now becomes:

```
start = i - context + context = i
end = i + context + context + 1 = i + 2*context + 1
window_of_interest = feats[start:end] = feats[i:i+2*context+1]
```

This way, if you are trying to access A again in the padded `feats` array, window of interest would correspond to `feats[0:7]`, returning `[0 0 0 A B C D]`, which gives our frame of interest A three context frames on each side. Here the three on the left are the zero padding.

B.3 Dataset class

For this homework, you will be implementing the `getitem()`, `len()` and the `init()` functions of our `Dataset` class(`AudioDataset`). We want to load all our data in the constructor(`init`), so that we can access the information at a given index using the `getitem(index)`. The `len()` just gives the length of the dataset.

The `getitem` will return the input and target for training, so the input will be information at the index but with context on both sides. While, the target will be the phoneme. We have given you the list of phonemes, but neural networks only return numerical datatypes(double, float, int etc.) so we want you to convert or map the given input to a numerical value which you can convert or map back to the original value while testing.

All this computation which is common and not dependent on a specific index should always be done in the constructor and not the `getitem()` or it would be called every time data is queried.

Now, lets come to some finer details of the `getitem()` function and its implications on the padding the data. We are taking context in each frame, how will this affect the edges of the data?

We will have to pad it to be able to get the data on the edges. This will mean that `i=0` of X (the log-mel features) and Y (the transcript) does not match?

Your job is to figure out how do we access the data such that we get aligned input and labels.