Homework 1

An Introduction to Neural Networks

11-785: Introduction to Deep Learning (Spring 2025)

OUT: **January 17, 2025**
Early Deadline/HW1P2 MCQ Deadline: **January 24, 2025, 11:59 PM EST**
DUE: **February 7, 2025, 11:59 PM EST**
Code Submission Deadline: **February 14, 2025, 11:59 PM EST**

Start Here

- **Collaboration policy:**
  - 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
  - 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.

- **Overview:**
  - **Part 2**: This section of the homework is an open ended competition hosted on Kaggle.com, a popular service for hosting predictive modeling and data analytics competitions. The competition page can be found here.
  - **Part 2 Multiple Choice Questions**: You need to take a quiz before you start with HW1-Part 2. This quiz can be found on Canvas under **HW1P2: MCQ (Early deadline)**. It is **mandatory** to complete this quiz before the early deadline for HW1-Part 2.
  - **Part 2 Code Submission**: You need to submit your notebook on Autolab after the final deadline. More information will be shared on Piazza.

- **Submission:**
  - **Part 2**: See the the competition page for details.
  - **Part 2 Code**: Autolab page for the code submission will be shared on Piazza after the deadline.

- **Note:**
  - The maximum number of parameters allowed for the models for this HW is 20 million (including ensembles).

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 Homework 1 Quiz
    - ☐ Download starter notebook and set up virtual environment (optional)
    - ☐ Install Kaggle API and create a directory
    - ☐ Download dataset .npz files 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 training loop *train_model()*
    - ☐ 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 inference loop *evaluate_model()*
    - ☐ Set model to 'Evaluation Mode' and create 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 initial submission before early submission deadline
    - ☐ Use Weights and Biases to log metrics for each epoch
    - ☐ Make sure the model is saved after every few epochs
    - ☐ Try changing the parameters listed in Section-10 to improve results
    - ☐ Attend office hours if you need help :)

Executive Summary

Objectives

The main goal of this homework is to explore neural networks for speech recognition, mainly focusing on phoneme state 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 Mel spectrograms (more about it ahead in the writeup), with shapes of $(T, 28)$, where T is the utterance duration in seconds. 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 phoneme states 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.

Workflow

Prepare the Data

- Use the provided starter notebook.
- Load and preprocess data into a suitable format, with optional context padding for each frame.

Model Setup

- Build a feedforward neural network **constrained to only MLPs,** without incorporating any other techniques such as CNNs, LSTMs, attention mechanisms, etc.
- Input: Frame (plus optional context) of dimension $((1 + 2 \times \text{context size}) \times 28)$. If context frames are not added, the input dimension is (1×28).
- Output: Class probabilities for 40 phoneme states.

Evaluation

Accuracy is calculated as the percentage of correctly predicted phoneme states across all test frames.

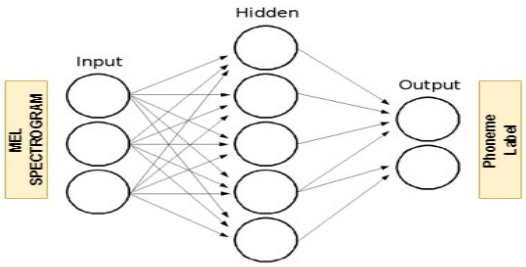
Hints for Better Performance

- Add context frames (e.g., 5 preceding and 5 following frames) for improved accuracy.
- Perform Cepstral Normalization to reduce channel effects.
- Experiment with hyperparameters (e.g., learning rate, number of layers).
- Use smaller subsets of the dataset for testing and debugging before training on the full dataset (e.g., 10% subset)

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 state (subphoneme) labels. The data comes from articles published in the Wall Street Journal (WSJ) 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.



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 state labels for each frame in the test set. Grade cut-offs are released after the early 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 (raw mel spectrogram frames)
- Frame-level phoneme state labels

The test data comprises of:

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

Your job is to identify the phoneme state 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.

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.

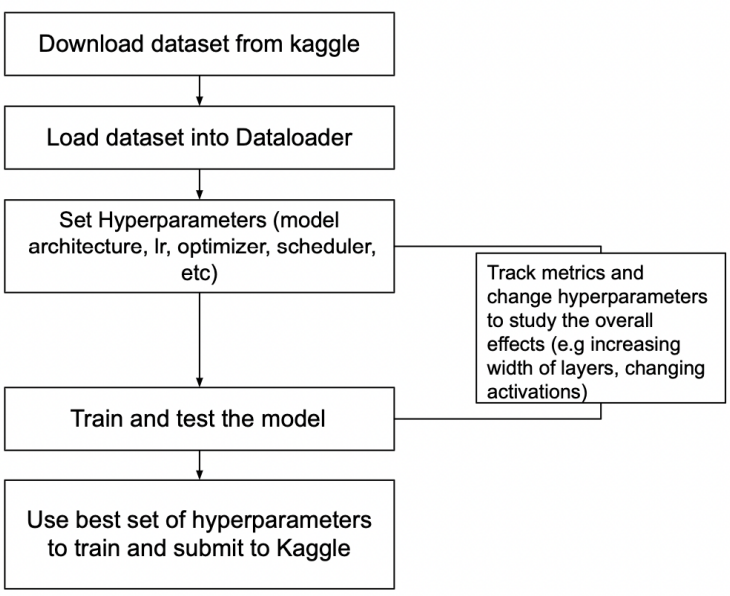


Figure 1: How to use starter notebooks

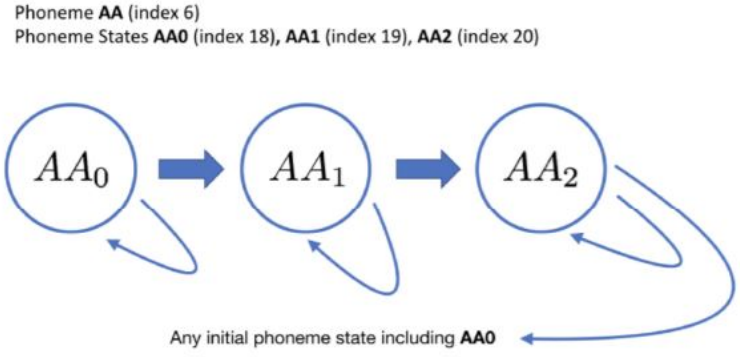
3 Phonemes and Phoneme States

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.

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:

Example: [”+BREATH+”, ”+COUGH+”, ”+NOISE+”, ”+SMACK+”, ”+UH+”, ”+UM+”, ”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”, ”SIL”, ”T”, ”TH”, ”UH”, ”UW”, ”V”, ”W”, ”Y”, ”Z”, ”ZH”]



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 mel spectrogram frames using a neural network that takes a frame (plus optional context) and outputs class probabilities for all 40 phoneme states. Performance on the task will be measured by classification accuracy on a held out set of labeled mel spectrogram frames. Training/dev labels are provided as integers [0-39].

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 **"melspectrogram"** 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 28-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, 28) known as the **melspectrogram**. Note that in the dataset provided to you, we have already done all of this pre-processing and provided the final **(*, 28) shaped melspectrograms** 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 melspectrograms, and phoneme labels for each 28-dimensional vector in the melspectrogram. The task in this assignment is to predict the label of a particular 28-dimensional vector in an utterance.

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:

```
data
├── train-clean-100
│   ├── mfcc (28539, )
│   └── transcript (28539, )
├── dev-clean
│   ├── mfcc (2703, )
│   └── transcript (2703, )
└── test-clean
    └── mfcc (2620, )
```

We recommend that you run you code and ablations using a fraction of the train-clean-100 training set, and then 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

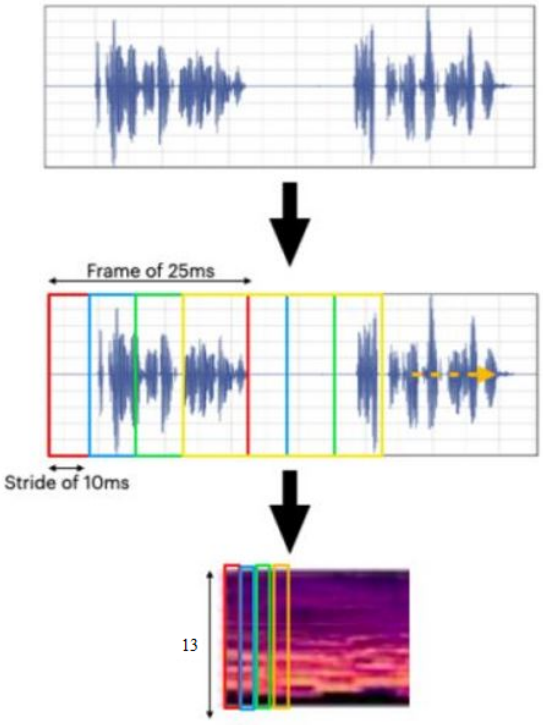


Figure 2: Feature extraction

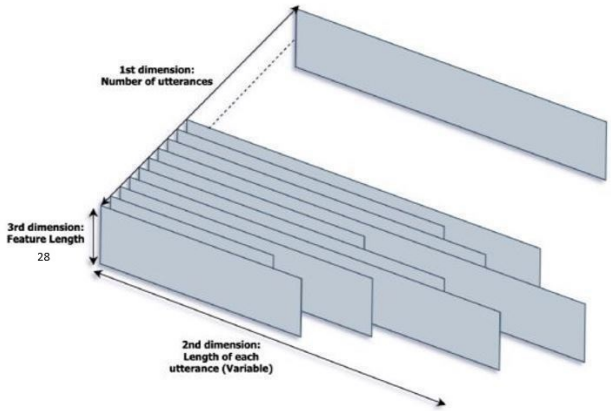


Figure 3: Data representation

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, 28) 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.

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, 28) 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, 28). Subsequently, the input layer will have (1+2*context_size)*28 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

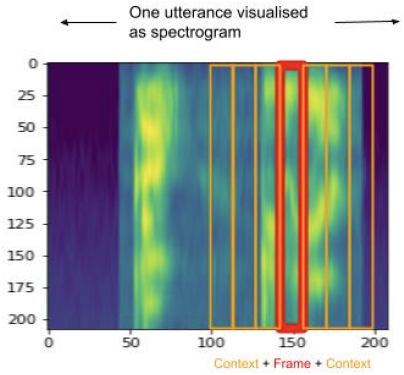


Figure 4: Context: Movement of spectrum from adjacent frames on either side has cues about what the frame in question could represent

5.3 Cepstral Normalization [1]

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] · 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] · 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:

$$Y_i[q] = H[q] + X_i[q] \tag{1}$$

By taking the average over all frames we get:

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

[1] https://dsp.stackexchange.com/questions/19564/cepstral-mean-normalization

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

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

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

The evaluation metric for this competition is frame-level accuracy. There are a total of 1934138 frames in the test set. You will be ranked by unweighted accuracy on those phoneme state labels. This homework is worth 100 points. The distribution of the points is as follows:

- **3 points:** Homework 1 Quiz - MCQs (due January 24 2025)
- **7 points:** Preliminary Submission (due January 24 2025)
- **90 points:** High Cutoff
- **70 points:** Medium Cutoff
- **50 points:** Low Cutoff
- **30 points:** Very Low Cutoff

7 Submission Format

Submission files should contain two columns: Id and Label.

Id: The 0-based index of the frame in the test set [0-1934137] (data type: int).

Label: The predicted label of the phoneme [0-39] (The Phoneme).

Id=0 is the first frame in the first utterance.

Id=1934137 is the last frame in the last utterance.

A sample submission file is available on the Piazza post for HW1P2. Please submit your prediction/submission files here. You will be allowed a maximum of 10 submissions every day.

You need to make atleast one submission (of a basic model) to Kaggle before the early deadline.

7.1 Mandatory Preliminary Submission

There is a mandatory preliminary submission and an associated MCQ that, together, are worth 10% of the points for the homework. The deadline for this preliminary submission is posted on the course website and piazza. This submission is intended to get you started quickly on the homework. We provide starter code in the form of a notebook that should, hopefully, make it relatively straightforward to make this submission.

For the preliminary submission, you must download the data, train a very simple preliminary model with it using the sampled (10%) dataset, using the (code in the) notebook provided. You must then process both the validation and evaluation data with the obtainded model, and submit the evaluation results to Kaggle. You must also answer the preliminary-submission MCQ on canvas, which queries you about the homework,

the values of specific data instances, and the loss obtained while training the preliminary model. The MCQ is worth 3 points. The actual preliminary submission is worth 7 points. You must get an accuracy of at least 60% on the preliminary submission to get the 7 points. If you do not make the cutoff you get 0 points for the submission.

You will find attached with this writeup, a Python notebook to get you started with the homework. It is not mandatory to use this notebook, and you are free to write your own code. With the starter code and the sampled dataset, you will be able to reach the early submission cutoff. **Note:** You need to submit your results that crosses the preliminary cutoff before the early deadline. Not doing so will cost you 7 points. The preliminary cutoff is set much lower than the final cutoff, and running the starter notebook should be enough to reach this cutoff.

7.2 Usage

- Download data from Kaggle: the data in the toy problem is the same as the one for the main dataset, although a very small portion of the main dataset. Try to implement the interface with Kaggle, data downloading and decompressing.
- Implement the data loader and network model. You can implement a simple one layer network for toy problem to assist testing other parts of codes and enlarge your network once training on the main dataset.
- Set up your training workflow and make sure it is running on toy dataset.
- Since the toy dataset is small and not representative, the trained model on toy dataset cannot accurately indicate the performance of your network.

Notice that the toy problem is used for making sure your codes runnable, checking the type and shape of the data, learning to interface with the Kaggle. The accuracy of the model on the toy problem cannot indicate the performance of your network on the main dataset.

7.3 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

8 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. Additonally, 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.

9 Optional Reading

Go through this link and Piazza Post @6 to understand how Mel-Spectrograms are generated.

Appendices

A Early Submission Architecture

Very Low Cutoff Architecture: (60%) This is the straightforward architecture in your starter notebook and consists of no hidden layers. (should cross cutoff within a few epochs when trained on `train-100`)

Context: 20
Hidden layers: 0
Activation: RELU
Learning rate: 1e-3
Optimizer: Adam

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 MFCCs, another consisting of the transcripts. We have 28539 recordings and corresponding 28539 transcripts, each recording is of different lengths. Which basically means for each timestep, we have a translated phoneme. For instance, the first recording("0019-000198-000") has shape (196,28) and the transcript is of length 198(why the extra 2 phonemes?).

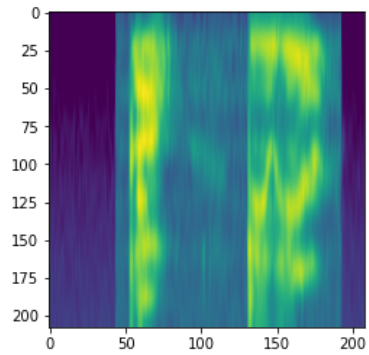


Figure 5: Visualized MFCC

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 -

`mfcc = [A B C D E]`

Now in your `__get_item__` function, if we wanted to get the frame at index 0, without any context, we would get `mfcc[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 mfcc now becomes:

```
mfcc = [ 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 `mfcc[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 = mfcc[start:end] = mfcc[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 = mfcc[start:end] = mfcc[i:i+2*context+1]
```

This way, if you are trying to access `A` again in the padded mfcc, window of interest would correspond to `mfcc[0:7]`, returning `[0 0 0 A B C D]`, which has 3 zero contexts before and after our frame of interest `A`.

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(mfccs) and Y(transcript) does not match? Your job is to figure out how do we access the data such that we get aligned input and labels.