Performance of Modified Greedy Decoding Model Shallow Fused with a 3-Gram and Performance of Data Augmentations

Williston Luo, Anisha Narurkar Department of Bioengineering
UCLA
405 Hilgard Avenue, Los Angeles, CA 90095
`willistonluo@ucla.edu`, `anishanarurkar@g.ucla.edu`
LINK: `https://github.com/wlucla/NSP2025_GRU_FINALPROJECT.git`

Abstract

Given a baseline GRU $model^0$ that maps from features (threshold crossings TX & spikeband powerSBP) to phonemes and characters performing at approximately 23% character error rate (CER) at baseline, attempts were made to decrease this initial CER. To this end, an external 3-gram Language Model was shallow fused with the baseline GRU model. Specifically, instead of just focusing on Connectionist Temporal Classification (CTC) loss, we wished for the model to consider English context, and for each predicted sequence of phonemes, we allowed for the natural context of the English language to alter the greedy decoding method implemented in the baseline model. Lastly, dimensionality reduction techniques and data augmentations, including rolling Z scores, injection of Poisson variation into spike counts, and feature masking for the duration of an entire trial. As we will show, the shallow fusion with the 3-Gram did NOT come with a beam search decoder; in other words, we STILL used a greedy decoding mindset during shallow fusion, and though we thought it to be feasible, it significantly worsened the CER by $\tilde{9}\%$. Next, a few data augmentations were attempted. First, a rolling Z-score with the past spike band power and threshold crossings was implemented. Next, for the threshold crossings features, we attempted to perturb them with Poisson noise, taking each TX feature as the mean of the Poisson and randomly picking a new TX out of this Poisson distribution. Lastly, masking was also implemented to zero out all features of a single channel across all time (aka zeroing out the SBP and TX data for random channels for all time). Here, we present how our Shallow Fusion still requires hyperparameter tuning if given more compute, and how our data augmentations produced insignificant effects on the baseline model performance.

1 Introduction

As previously mentioned, the motivation behind our tampering with the baseline model was to impart upon our model some knowledge of the English language. To do this, we implemented a 3-Gram by having it parse a phoneme dictionary and figure out a probabilistic distribution of a phoneme given the 2 previous phonemes in an ordered sequence. This would allow us to give any arbitrary sequence of phonemes an LM score (a score from the 3-Gram). Now, at each timestep of the GRU is an output that represents the probability of each phoneme based on CTC. Instead of purely choosing the phoneme with the highest probability (greedy decoding), what if we could introduce the LM scoring from the 3-Gram somehow? To do this, we decided to modify the greedy decoding algorithm a bit. Instead of choosing a single top CTC-scoring phoneme at each timestep of the GRU (highest logit),

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

we will choose an arbitrary number of top CTC-logit phonemes at each timestep (i.e, 3, 5). Then, to each of these phonemes' logits, we will add the phoneme's LM-Score scaled by a positive constant less than 1. This is the shallow fusion step. Next, among these scores, we will select the phoneme with the highest shallow-fusion-score. By doing this, I hope that our model gains some context of the English language. Next, we hoped to implement some of the data augmentations mentioned in class. First of all, the most intuitive data augmentation is a rolling Z-score of feature vectors in one trial within a batch. For a 256D feature vector, this implies that given a window, we must make the overall statistics of the feature vectors within this window 0 mean and unit variance. Hopefully, this will dampen any effects of large fluctuations in neural activity. The next augmentation attempted was an addition of Poisson Noise to the TX features; the reason for this augmentation was the fact that in class, we were taught that spike counts for a duration of time are a Poisson Process. Lastly, a feature masking was implemented to model potentially dysfunctional electrodes that record nothing throughout an entire trial; the probability that an electrode is dysfunctional is an input parameter.

2 Methods

2.1 Obtaining Predictive Phoneme Model (3-Gram)[1]

Before anything, it's important to realize the GRU baseline model has no idea what language is. It is purely mapping from neural features to phonemes without help. 'Help' is defined as any factor that imparts upon the model knowledge of the English language, and can discourage very bizarre phoneme sequences (aka rare/nonexistent in the English vocabulary). Thus, would it not be logical to define some function that can 'score' an arbitrary phoneme sequence based on the English language? First, a phoneme dictionary was acquired as a basis to train a model that can use context to predict the subsequent $phonemes^1$. In this case, the choice was a 3-gram LM, meaning the model should be learning a distribution of the subsequent phoneme given 2 preceding phonemes. This way, the 3-gram LM will learn the context of English when forced to scan an English phoneme dictionary. The Phoneme Dictionary chosen was organized in rows, where each row is a word followed by a phoneme sequence representing the pronunciation. The technique used here will be memory-based learning. Specifically, the 3-gram Model implementation will have a function to scan each row of the dictionary and update a member dictionary where the keys are tuples of 2 elements (ordered phonemes), and the key's value is a counter object that records the number of appearances of individual phonemes after the tuple in the key. This will be used in scoring an arbitrary list of phonemes later. The phoneme dictionary used happens to have integers within each phoneme representing which sound to place stress on. Because in our formatCompetitionData.py, it is clear that NO integers are in our phoneme definitions, I've added lines of code to strip away the integers that appear in any phoneme. Next, the 3-gram model will contain a function to calculate the score of an arbitrary sequence of phonemes. These phoneme sequences will appear as sequences of integers as mandated by formatCompetitionData.py, for easy integration into neural_decoder_trainer.py later on. Note that it was never said that the probability of an arbitrary sequence of phonemes would be calculated. In fact, I actually don't know how to do that with a 3-gram model, so only the SCORE of an arbitrary sequence of phonemes can be calculated. For example, if a sequence is 5 phonemes long, I can only calculate the probability of the 3rd, 4th, and 5th phonemes given the previous two based on the training results from having the 3-gram LM read the English dictionary. I will always be missing the probabilities of the first 2 phonemes. Thus, I can only calculate an approximate score of a sequence of phonemes, not the full probability of the sequence. In fact, later on, we will see that the first 2 phonemes of any sequence of phonemes will be obtained purely via greedy decoding of selecting the most probable phoneme outputted at that timestep in the GRU. For now, only note that the 3-gram score of a sequence of phonemes is calculated as below (example shown for a 5-phoneme sequence).

$$Score(Phoneme_0, Phoneme_1, Phoneme_2, Phoneme_3, Phoneme_4,) = \\ ln(P(Phoneme_2|Phoneme_0, Phoneme_1)) + ln(P(Phoneme_3|Phoneme_1, Phoneme_2)) \\ +ln(P(Phoneme_4|Phoneme_2, Phoneme_3))$$

Here, 'Score' is our LM's rating of the validity of the sequence of phonemes. It is not a rigorous probability as much as it is a hand-wavy score of how likely the LM thinks the Phoneme sequence occurs in the English language. Also, notice that I am summing log probabilities, which is robust against numerical underflow. Some people may question the validity of our conditional probability: what if "Phoneme0, Phoneme1" never appear in this order? To safeguard against this, within the natural log operator, I've inserted a very small number to prevent ln(0) from ever occurring.

2.2 Modified Greedy Decoding [5,6,7]

I will attempt to implement a modified greedy Decoding algorithm presented in the lecture, such that at every timestep where logits are outputted to model a distribution over phonemes, instead of choosing the phoneme with the highest probability, I'll choose the top 3 phonemes with the highest logits at each output timestep of the GRU. Now, since I implemented a 3-gram LM, I can, for each of these 3 phonemes, calculate the score that the LM gives them (note that the first 2 phonemes in time will have a score of 0 from the LM because there are no '2 previous phonemes'). After that, I will multiply the score from the LM by a factor alpha and add it to the 3 individual log-CTC-probabilities, and select the greatest phoneme with the highest shallow-fused score. This is my best attempt at imparting contextual knowledge of the English language onto our model. From this modified greedy decoding algorithm (modded_greedy function in neural_decoder_trainer.py), we obtain a decoded sequence that performs better than a decoded sequence from only CTC loss considerations. We can use this modded_greed function to modify the 'decodedSeq' variable in neural_decoder_trainer.py.

2.3 Autocorrection

The autocorrect function is a very aggressive usage of the 3-Gram LM. For a decodedSeq of phonemes, it will perform an iterative autocorrection of each phoneme based on the 2 previous phonemes (note that the first 2 phonemes never get autocorrected because they don't have 2 previous phonemes). The iterative process is as follows. For each phoneme (after the first 2) in decodedSeq, based on the last 2 phonemes, the LM will modify the current phoneme to a phoneme that is most likely given the trained 3-Gram's prediction by iteratively testing phonemes and obtaining the LM score, and choosing the phoneme that maximizes the LM score.. As we will see later, this performed very poorly AND was extremely computationally expensive.

2.4 Rolling Z Score[7]

In the lectures, a rolling Z-score was said to improve the decoding performance. It functions as follows. Within each batch, there are trials that contain 256D features in chronological order. The first 128D are allocated to TX, and the latter 128D are allocated to SBP. The argument for the Rolling Z Score is an integer that indicates how many feature vectors of the past you'd like to Z-score over to make sure your current vector is zero mean, unit variance among the current and past feature vectors. This Data augmentation was by far the most difficult, and slight bugs in code (i.e, accidentally dividing by variance instead of standard deviation) cause catastrophic errors, such as a CER of $\tilde{8}$0%.

2.5 Injection of Poisson Noise [3]

This augmentation is motivated by components learned in lecture. Specifically, spike counts within a period of time are Poisson distributed, which motivated the thought that perhaps it's logical to perturb the TX features by re-choosing a TX for each threshold crossing feature. To do this, a Poisson distribution was defined for each TX feature, and a new TX was randomly drawn out of this distribution. Whether or not a batch was to be perturbed with Poisson Noise was controlled by a probabilistic parameter p. If p=0.1 the there is a 10% chance of triggering Poisson noise augmentation for a batch. In hindsight, this data augmentation seems silly because TX are inherently already drawn out of a Poisson distribution, so there is really little point in re-drawing each TX out of a Poisson distribution. Nevertheless, this Data augmentation was directly motivated by what was taught in lecture.

2.6 Feature Masking (For entire Trial)

This Data augmentation was motivated by the assumption that individual electrodes may fail with some probability. A further assumption was made based on the fact that the first 128D of a feature vector correspond to TX data and the last 128D of a feature vector correspond to SBP data. Specifically, this assumption was that, if we lay these two 128D entities side by side, corresponding TX and SBP elements are taken from the same electrode. This means, for example, in the 256D feature vector, the 0th and 128th indices are from the same electrode, the 1st and the 129th indices are from the same electrode, etc. This gives us all we need to perform feature masking. For each of the 128 electrodes, we input a probability that the electrode fails. When it does fail, we zero out all data from this electrode for an entire batch. This can be done by generating a mask first filled with random values between 0 and 1. Then, apply a Boolean operation elementwise. If an element is less than the probability of electrode failure (argument to the feature masking function), then that element becomes 0. Or else it will become 1. Then, we will do an element-wise multiplication of our mask with the variable that represents a batch of data.

2.7 Time Masking[2]

Sometimes, neural recordings may be invalid because of poor signal quality (causes: packet loss, generally poor signal quality, etc.). In the lecture, neural responses have variability across repeated presentations of the same stimulus, and Dr. Kao even mentioned the possibility that the implanted invasive electrode records no useful signals, necessitating another surgery. The goal of Time Masking is thus to, in the temporal domain, define a window width **L** and starting from a feature index in time, zero out all TX and SBP data in the aforementioned window. We include a hyperparameter to toggle the amount of data that gets deleted, *p*, a probabilistic parameter that controls the probability that an index gets selected as the "starting index" we begin zeroing out features at. Thus, should Training data contain missing/non-optimal data sometimes, then this augmentation should in theory make the trained model more generalizable.

2.8 Time-Feature Masking[4]

Time-Feature masking was motivated by the observation that neural inputs change over time due to electrode drift, channel failure, etc, as illustrated in lecture 11. This augmentation models the scenario where a population of electrodes doesn't work for only an amount of time that can be less than the entire trial (i.e, Time-Feature Masking is a less aggressive data masking than Feature Masking). The probability that an electrode is zeroed out is controlled again by the probability parameter *p* and the maximum time window you are willing to zero out.

2.9 Time Shift[8]

The time shift augmentation was motivated by the trial-to-trial variability in neural response timing seen in lecture. Perhaps temporal jittering may occur in the timing of neural signals; for example, the timing of your neural signals when pronouncing the same phoneme may not necessarily be identical across multiple attempts at pronunciation. To model this, the time-shift augmentation will take a feature and push it forward in time by a parameter that defines the maximum magnitude of your time shift (i.e if the parameter is 5, then the data augmentation will at most shift the feature 5 indices forward in time or 5 indices back in time). Clearly, the endpoints of a trail will suffer data loss; we will just append zeros for now.

2.10 Principle Component Analysis[9]

As shown in lecture, there is a possibility that the speech neural signals are closely spaced on a lower-dimensional manifold, which might be able to project our 256D feature vectors onto in order to speed up the training of our model. However, when projected into 2 lower dimensions, the feature vectors occupy an irregularly shaped blob with no clear manifold like the ones presented in lecture for the delayed reach tasks. (Figure 1). Thus, we will not be using PCA as data preprocessing.

3 Results

3.1 Scoring From 3-Gram Language Model

Prior to a shallow fusion of my LM scores at each time step of the GRU, I must make sure that the LM scoring is performing as expected. In Figure 2, we clearly see that the scores of more anecdotally common words/sounds obtain a higher LM score. For example, 'S EH L' (i.e sell, cell) sounds like a word more than 'Z ZH V' does, and thus 'S EH L' is liked by our LM model more, obtaining a higher score. This score will be the component that is shallow fused with the CTC logits at each timestep of the GRU, which can hopefully grant our model some context of the English language to deter it from decoding gibberish.

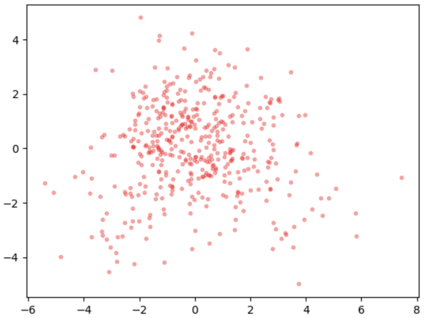
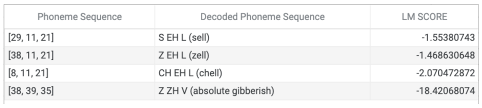


Figure 1: PCA of an arbitrary Trial into 2D. No visible Manifold like the ones shown in lecture can be seen.



Phoneme Sequence	Decoded Phoneme Sequence	LM SCORE
[29, 11, 21]	S EH L (sell)	-1.55380743
[38, 11, 21]	Z EH L (zell)	-1.468630648
[8, 11, 21]	CH EH L (chell)	-2.070472872
[38, 39, 35]	Z ZH V (absolute gibberish)	-18.42068074

Figure 2: The trained LM will assign a score to a sequence of phonemes. Common words and sub-words get a higher score in general compared to gibberish.

3.2 Results of Greedy Decoding Algorithm (Shallow fusion of GRU and 3-Gram)

In short, the standalone shallow fusion with 3-Gram with no data augmentations (except for the ones provided by the base implementation) performed relatively poorly relative to the baseline GRU. I ran out of money at the end to train my model and was cut off after 4̃0000 batches, but was able to copy the output terminal in the local host Jupyter IDE and plot the CER. The final CER was 33.1%. Looking at the plot (Figure 3), I really feel that if I altered the learning rate scheduler to not decrease as quickly and if I increased my number of batches, a better CER could be found, especially because I changed NO hyperparameters in this shallow fusion (except for the number of batches to train). Even better would be an extensive hyperparameter search, which is definitely beyond my computational budget.

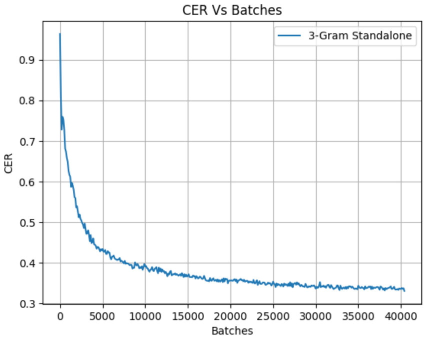


Figure 3: Shallow fusion of the 3-gram with the baseline GRU model results in a 3̃3.1% CER after 40000 batches. Using baseline hyperparameters.

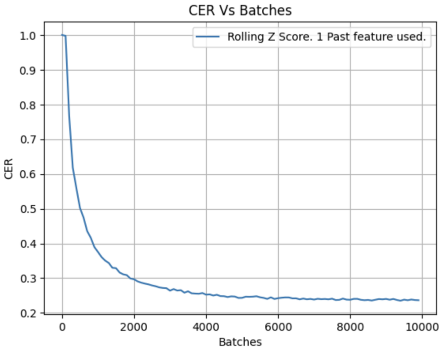


Figure 4: CER vs Batches with rolling Z score over 2 feature vectors in time. There is no 3-gram shallow fusion here.

3.3 Rolling Z Score

A rolling Z-score function that takes an integer number of past feature vectors to use was implemented. However, any time I tried to use more than 1 window, the code would freeze up and not run. As you can see in my codebase, I've made various versions of a rolling z-score function in hopes of increasing computational efficiency. But in the end, I could only manage to test the rolling Z-score function with a window size of 2 (ie, using only 1 past feature vector). Unfortunately, there was little difference with the baseline GRU model when using only this rolling Z score with a window size of 2. (Figure 4) The final CER was 23.6%, underperforming the baseline model.

3.4 Time Masking, Time Feature Masking, Time Shift

We have extensively double-checked our code, but for some reason, our Time Masking, Time Feature Masking, and Time Shift data augmentations fail to produce a significant effect on training, even with relatively aggressive parameter inputs. Specifically, Time masking was done with a maximum of 50 indices in time in a window to be zeroed with a 10% probability; Time feature Masking was done with a maximum of 100 indices in time in a window to be zeroed out with 20% probability, and Time shifting was done with a maximum time shift of 5 indices forward or backward in time with a probability of 10% probability that a feature is shifted. However, we were disappointed to see in Figure 5, these data augmentations produce little to no effect at all. At 10000 batches, the Time Masking augmentation has a CER of 23.6%, the Time Feature Masking augmentation had a CER of 23.5%, and the Time Shift augmentation had a CER of 23.9% (Figure 5,6,7).

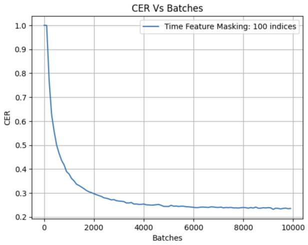


Figure 5: Time Feature Masking.

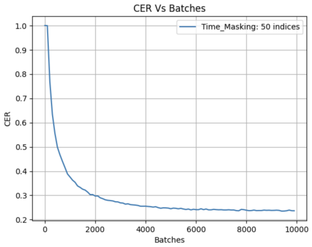


Figure 6: Time Masking

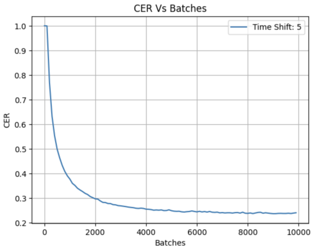


Figure 7: Time Shift

3.5 Time Masking For Entire Trial & Poisson Disturbances to Some Batches

In this data augmentation for any given trials, each electrode had a probability of being out of commission for the entire trial. We chose (arbitrarily) a 7% probability that any electrode would be out of commission. Also, for any arbitrary batch, there would be a probability that all TX features to be disturbed by Poisson noise. We arbitrarily defined that probability to be 10%. The results are as shown in Figure 8: a final CER of 21.88%.

4 Discussion

We successfully obtained a 3-Gram model that could predict the probability distribution of the next phoneme given the previous 2 phonemes. However, in the code, we could have easily made this a 2-gram, 4-gram, or 5-gram as we wished. Thus, the 'N' of our N-gram is clearly a hyperparameter in this project. Even though shallow fusion of baseline GRU with the 3-Gram was successful, the various hyperparameters remain untouched. Naively, a random hyperparameter search with just a few

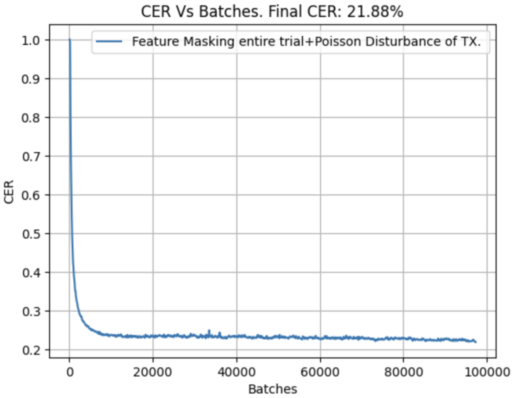


Figure 8: Feature Masking of entire trials in time domain combined with Poisson Disturbance of TX features. 97400 batches. Final CER=21.88%

hyperparameters could be performed, such as learning rate, batch size, dropout, and L2 regularization. However, the results in Figure 3 alone (baseline hyperparameters) took $\tilde{8}$ hours to obtain. Thus, an extensive hyperparameter search is not reasonable. Nevertheless, I believe that the modified greedy search performs as expected. Namely, at each timestep in the GRU, in addition to choosing the phoneme with the largest logit, we instead (0) choose the 3 phonemes with the largest logits, (1) score each of these 3 phonemes with our 3-Gram LM, (2) shallow-fuse the logit with the score from our 3-Gram LM. The operation appears as follows, with index 1 corresponding to the phoneme with the largest phoneme, index 2 corresponding to the phoneme with the second largest logit, and index 3 corresponding to the phoneme with the third largest logit. *Note that for a 3-gram, an LM score can only be calculated for a phoneme at a given time step in the GRU if there are at least 2 preceding phonemes in time.*

$$max_i\{\,\alpha\ \cdot Logit_1 + (1-\alpha)LM_1, \alpha \cdot Logit_2 + (1-\alpha)LM_2, \alpha \cdot Logit_3 + (1-\alpha)LM_3\}$$

Hence, yet another hyperparameter appears in our implementation α. For completeness, notice that our modified greedy search could very well have chosen, in general, at each timestep of the GRU, the top **N** phonemes with the greatest logits and proceed to implement the above algorithm for **N** different phonemes. Thus, yet another hyperparameter would be this integer **N**.

Although we double checked the logic for the Feature masking (entire trial), Poisson Noise augmentation, and the Rolling Z scorer, we cannot seem to get any significant improvements in the CER, as seen in all figures. Next, our series of time-domain data augmentations was meant to model the fact that neural signals and their recording are subject to packet loss, temporal distortion, and timing variances. To this end, time masking, time-feature masking, and time shifting were implemented. The CER vs batches training curves did not significantly differ for any of these data augmentations, converging to a final CER of 24%. Such consistent results suggest that perhaps the decoder relies mostly on broad temporal neuronal populations' firing patterns rather than our localized data augmentations in time (i.e, decoding does change even when local portions of the signal are deleted, corrupted, etc). Furthermore, neural activity in one electrode is heavily correlated with the closer the signals are in time (resulting from same action/effect), thus time shifts may not produce significant losses of information. Lastly, these augmentations where thought of while thinking of TX instead of SBP; a future direction may be to write data augmentations using more rigorous EE concepts surrounding the recording hardware.

5 Conclusion

We believe that the 3-Gram shallow fusion for a modified greedy decoding has the potential for higher performance. Because it is computationally expensive to extensively hyper-tune, we have not had the chance to do so. Furthermore, our data augmentations might have been too aggressive, resulting in final CERs that were not significantly different than that of the baseline model (perhaps even worse). Future directions should focus more on systematic tuning and augmentation refinement to improve the performance of the proposed methods.

References

[0]cffan. neural_seq_decoder. [Baseline GRU Model]. GitHub repository. `https://github.com/cffan/neural_seq_decoder`

[1] Rudnicky, A. (2015). *CMUdict (Version 0.7b) Phoneme Dictionary*. GitHub. https://github.com/Alexir/CMUdict/blob/master/cmudict-0.7b

[2] Kao, J. (2025). *Lecture 5: Firing rate and spike statistics* [Lecture slides]. UCLA ECE C143/C243A. Slide 13.

[3]Kao, J. (2025). *Lecture 6: Poisson modeling, discrete classification.* [Lecture slides]. UCLA ECE C143/C243A. Slide 10.

[4] Kao, J. (2025). *Lecture 11: Neural Networks in BCI* [Lecture slides]. UCLA ECE C143/C243A. Slides 64-68.

[5]Feghhi, E. (2025) *Lecture 13: Beam Search and External Language Model Integration* [Lecture slides]. UCLA ECE C143/C243A. Slides 25-31.

[6]Feghhi, E. (2025) *Lecture 13: Beam Search and External Language Model Integration* [Lecture slides]. UCLA ECE C143/C243A. Slide 44.

[7]Feghhi, E. (2025) *Lecture 12: Baseline Machine Learning Framework for Neural Speech Decoding.* [Lecture slides]. UCLA ECE C143/C243A. Slide 23.

[8] Kao, J. (2025). *Lecture 7: Neural Classification* [Lecture slides]. UCLA ECE C143/C243A. Slide 20.

[9] Kao, J. (2025). *Lecture 15: Dim. reduction + neural dynamics* [Lecture slides]. UCLA ECE C143/C243A. Slide 10.