ECE C147/C247, Winter 2025 **Project Guidelines**
Neural Networks & Deep Learning Prof. J.C. Kao
UCLA ECE TAs: B. Qu, K. Pang, S. Dong, S. Rajesh, T. Monsoor, X. Yan

Developed thanks to: Joe Lin, Kaifeng Pang, Bruce Qu
Due: Friday March 14, 2025 (Friday of Week 10)

This final project is intended to give you an opportunity to apply the skills you've learned in the course to a practical research application, with an emphasis on exploration of architectures beyond CNNs. In this project, you may explore – in addition to CNNs – RNNs, CNNs+RNNs, and other architectures we haven't explicitly covered in class (but are easy to implement with `PyTorch`, `JAX`, etc.) including those based on transformers. In this project, you may use any tools to aid your implementation, including `PyTorch`, `PyTorch Lightning`, `JAX`, and any other software libraries. Our philosophy here is that you already understand the principles of training these networks, including backpropagation and optimization, in which case you are now well-equipped to use these packages.

This document will describe a "default" project to work on. However, you also have the option of pursuing your own project. Note that if you are planning to do a custom project, please email Prof. Kao with a description of the project, ***__including what post-CNN architectures you will use__***. You must submit your request to do a custom project by Feb 24, 2025. In general, as long as the project involves exploring post-CNN content (e.g., RNNs or transformers), it will be approved. **Custom projects submitted for approval after Feb 24, 2025, will not be approved.**

Students may work in groups of up to (and including) 4 people. To find teammates, you may use the **Search for Teammates** functionality on Piazza. For more information, see `https://support.piazza.com/support/solutions/articles/48001158117-search-for-teammates`. You may also do the project individually.

Predicting Keystrokes from Electromyography Signals

Introduction

In class, we have worked extensively with computer vision data. What about temporal data, such as neural signals? It turns out we can also train effective models with similar design paradigms when it comes to sequential temporal data. In this project, we will explore the task of predicting typing (i.e., `QWERTY` keystrokes) given electromyography (EMG) signals. While we will give a brief background and description of the `emg2qwerty` dataset, further details can be found in the paper published alongside this dataset: `https://arxiv.org/abs/2410.20081` [3].

Background

Surface electromyography (sEMG) is a non-invasive technique that records electrical activity from the muscles, and is capable of measuring motor unit action potentials. These signals can be recorded even in amputees and people with paralysis, making sEMG a promising signal for use in non-invasive neural interfaces that restore movement and communication. Recently, Meta Re-

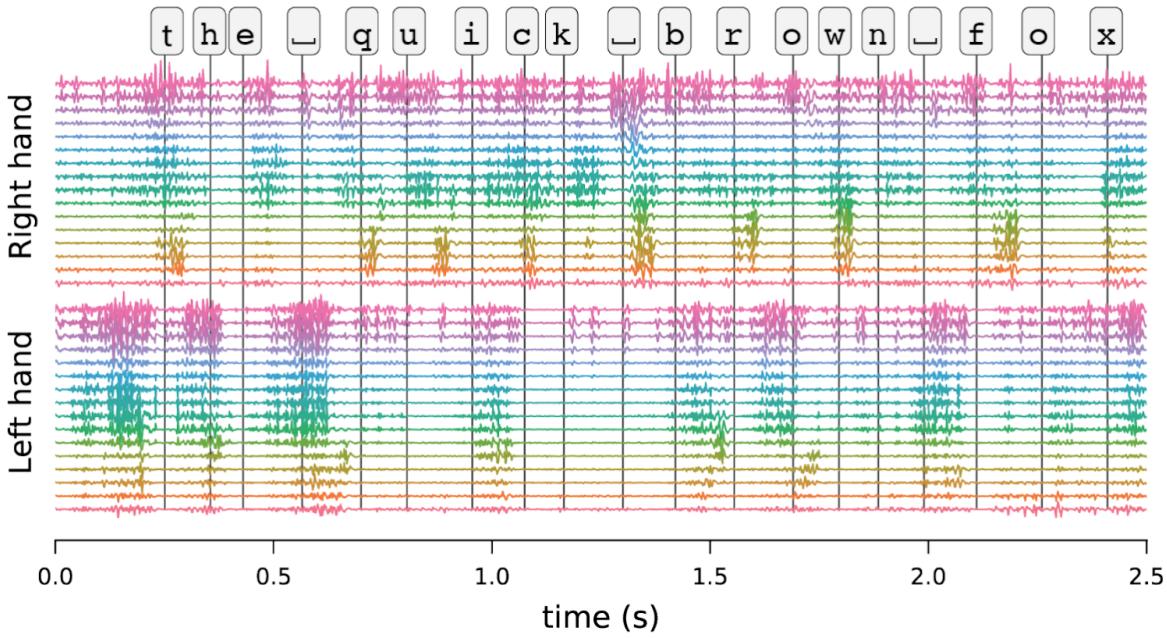

Figure 1: Visualization of *emg2qwerty* data, which includes sEMG signals from both hands and ground truth `QWERTY` key presses. [3]

ality Labs released a large dataset, `emg2qwerty`, containing simultaneously recorded sEMG signals and corresponding ground-truth keystrokes from a `QWERTY` keyboard [3].

The `emg2qwerty` dataset

The `emg2qwerty` dataset is composed of preprocessed sEMG signals recorded from both wrists and the corresponding ground-truth key logger sequence. During each experiment, a participant wore sEMG recording bands on their left and right wrists. Each band recorded from 16 sEMG electrode channels at a 2 kHz sampling rate (i.e., 2000 samples per second). Participants were then instructed to type while wearing the bands. The keys they struck, and the timestamps of when they were struck, were also recorded. This dataset therefore contains simultaneous sEMG signals from 32 total channels and corresponding key presses, illustrated in Figure 2. These data are released by Meta as hdf5 files. Using this data, it is therefore possible to **decode sEMG activity** to **predict what a person is typing**.

Although the entire `emg2qwerty` dataset is very large (108 users and 346 total hours of recording), we recognize that many students do not have compute resources to process the entire dataset. The "baseline" project therefore only requires you to train with a single subject's data, and doing solid work on this dataset can earn you a maximum score on the project. But you are welcome to use data from other participants, or the entire dataset if you wish. As we also describe later, for those who achieve excellent performance or generalization similar or related benchmarks in [3], it is also possible to do a project with the goal of turning it into a conference paper.

The files associated with the single user that the "baseline" project focuses on are located at `https://ucla.box.com/s/3xc4nwpfjfpo6ydjs94t0v2kuq37d5eg`. This data coresponds to subject #89335547. To be clear, you can score maximum points on this project only analyzing the data from subject #89335547. If you'd like to use other subjects from the dataset, you can access them by following the instructions from the github repository (linked below). The entire dataset is over 200 GB, and so for your convenience, we also split up the data and uploaded it here in smaller chunks:`https://ucla.box.com/s/e54bnjvy6hl33ao3jhvz7711q5iwkws5`.

Note that each `hdf5` file corresponds to a **single experiment session**.

Task inputs and outputs

The primary objective is to correctly predict typed keystrokes given a sequence of sEMG signals. You will therefore be designing a deep learning pipeline that takes input sEMG signals $x \in \mathbb{R}^{T_{\text{in}} \times N \times B \times C}$ and predicts an output character sequence $\hat{y} \in \mathbb{R}^{T_{\text{out}} \times N \times \text{num\_classes}}$. Here, T_{in} is the temporal length of the input sEMG signal, N is the batch size, B is the number of EMG recording bands ($B = 2$), C is the number of electrode channels in each band ($C = 16$), and T_{out} is the temporal length of the model output. The output character sequence may contain both typed characters, as well as the `null` character (indicating nothing was typed at that time).

The ground-truth labels $y \in \mathbb{R}^{T \times N}$ represent the *class indices* for the T non-`null` characters present within the input time window T_{in}. Because in general, $T \neq T_{\text{out}}$, this is not a simple classification task. Hence, we recommend using the **CTCLoss** [2], a specialized loss function for situations like this. We provide some background below.

Model Evaluation

Your models are to be evaluated using the **Character Error Rate (CER)** metric. The CER computes:

$$\text{CER} = \frac{S + D + I}{N},$$

where S is the number of substitutions, D is the number of deletions, I is the number of insertions, and N is the total number of characters.

CTC Loss

Proposed in 2006 [2], the **Connectionist Temporal Classification** (CTC) loss aims to address classification under settings with **unsegmented sequential data**. This refers to situations like ours where the labeled sequence consists of characters typed at *arbitrary timestamps*. For comparison, in a segmented setup, we would instead create uniform time intervals and provide a ground truth character for each time segment, which is unideal for our task. The CTC loss is already implemented in the codebase, however, we encourage you to read more indepth about it for a fuller understanding (e.g., `https://distill.pub/2017/ctc/`).

Training and evaluating a baseline model

The GitHub repository containing the implementation of the baseline model can be found at `https://github.com/joe-lin-tech/emg2qwerty`. We also provide an intial setup notebook for `Google Colab` in this repository. To train the base TDS model (see [3]) on a single subject of data, please follow the notebook `Colab_setup.ipynb`. In our experiments, the training on a single subject (with the most amount of data) took approximately 1 hour on `Google Colab` for 40 epochs and achieved a validation CER of 30.

"Baseline" project suggested directions (using single or few user data)

Below, we detail suggested directions you should pursue for the "baseline" project. This project is what we expect most students will do, and it will certainly be possible to score the maximum points on the project following these baseline project guidelines. We suggest you to explore the following directions:

1. Train, evaluate, and subsequently compare different architectures to reduce the test CER on the provided single subject, ID #89335547. You must experiment with at least 1 recurrent architecture (e.g. RNN, LSTM, GRU). Projects that evaluate more architectures, such as RNN+CNN hybrids, transformers, or other architectures, will receive more creativity points in the rubric below.

   Please note that we have already split the data for subject #89335547 into train/val/test splits at `https://github.com/joe-lin-tech/emg2qwerty/blob/main/config/user/single_user.yaml`. Please use these splits for the project.

2. Experiment with various data pre-processing or augmentation techniques.

3. How many channels are needed to achieve good decoding performance? Investigate relationship between the number of electrode channels and CER.

4. How much data is needed to achieve good decoding performance? Investigate relationship between the amount of training data and CER.

5. How fast does sEMG data need to be sampled for good performance? Investigate the relationship between sampling rate and CER.

Feel free to innovate beyond the components above to earn points for creativity and insight. Extra insight points may also be rewarded for explaining how these different approaches result in better or worse performance.

SOTA project directions

As this is a newly released dataset, there are opportunities to achieve state-of-the-art (SOTA) performance or generalization. If you are interested in potentially turning this project into a conference paper, you should attempt the following research directions. To support this, Prof. Kao and his lab are offering to help advise very promising projects to a conference paper, or even pursue further experiments within the lab for real-time demonstrations. If you would like to do the SOTA project

Table 3: A comparison of test subject performance across model benchmarks. Mean and standard deviation aggregates of character error rates (CER) across test subjects are reported. Lower is better. The reported test CER improvements arising out of personalization as well as the inclusion of the language model (LM), all have $p < .005$.

	No LM		6-gram char-LM	
Model benchmark	Val CER	Test CER	Val CER	Test CER
Generic (no personalization)	55.57 ± 4.40	55.38 ± 4.10	52.10 ± 5.54	51.78 ± 4.61
Personalized (random-init)	15.65 ± 5.95	15.38 ± 5.88	11.03 ± 4.45	9.55 ± 5.16
Personalized (finetuned)	11.39 ± 4.28	11.28 ± 4.45	8.31 ± 3.19	6.95 ± 3.61

Figure 2: Table 3 from [3].

and want some input, you can email my PhD student, Nima Hadidi (`nhadidi@g.ucla.edu`) and CC me (`kao@seas.ucla.edu`). Depending on the amount of interest in the SOTA project, we may have limited bandwidth to reply or may choose to answer questions more generally on Piazza. Irrespective of if you reach out to us before project submission, if you are doing a project of this nature and are interested in our help to bring this to a full paper, please put two asterisks in the title of your project before the beginning of your project name, e.g., "**A new method to significantly improve zero-shot generalization of sEMG typing." Note that adding the **'s does not mean we will advise the project (as we have only finite bandwidth) but if we find your project meets our threshold for novelty and impact, we will reach out. Of course, you are also welcome to pursue these SOTA directions independently without our help – simply leave the **'s out of the project title.

The following represent **research directions** that you can tackle instead of doing the baseline project.

1. Consider zero-shot generalization performance (see Table 3 replicated from [3], "Generic (no personalization)"). Improve zero-shot generalization by using better data preprocessing or utilizing other generalization techniques (e.g., SWAD [1]). You may explore other architectures, but keep in mind that these architectures are likely to be deployed in a real-time system with quick inference, and so should be causal and not unwieldy.

2. Improve fine-tuning performance (see Table 3).

3. Improve fine-tuning performance with *minimal* data from specific subjects. For example, good zero-shot generalization may be challenging, but if with only 30 seconds or 1 minute of user-specific data, one can achieve CER close to personalized decoders, that would be impactful.

4. Prof. Kao is releasing a custom dataset he collected recorded from the right hand. It is recorded from a different arm band with 8 sEMG channels recorded at 200 Hz. The data is in a csv file, where a timestamp is followed by the activity on all 8 recorded channels. This is a standard classification task with 6 labels (one for each finger on the right hand and the "rest" state). Performance can be measured via the accuracy of predicting each label at each time point. Maximize performance on this dataset (e.g., show that pre-training on the Meta dataset can improve performance on this dataset), or maximize performance with only minimal data (30 seconds to 1 minute of data). This data is hosted here: `https:`

//ucla.box.com/s/p6qrl0l74ajhc60sqe4p93vt6lcuxlf4. This project, if successful, could lead to further real-time experiments in our lab.

Project Logistics

Submittables

Each group should submit a writeup of their project work, exceeding no more than 7 pages for "baseline" projects and no more than 9 pages for SOTA or custom projects. References are excluded from this page count. It is fine to be below the page limit; this is the *maximum*. We will also ask you to submit your code, so that we can validate your results. If you have a project where you cannot submit your code, please notify us so we can proceed accordingly.

The writeup must adhere to the following template: `https://media.neurips.cc/Conferences/NeurIPS2024/Styles.zip` – so that we can judge all writeups in the same manner without having to worry about different font sizes, etc. To remove the line numbers, modify the following line at the beginning of the `.tex` file to include the `final` option: `\usepackage[final]{neurips_2024}`.

Writeup

In the writeup, there should be the following sections:

1. **Abstract** – A brief description of what you did in the project and the results observed.
2. **Introduction** – If you are doing the `emg2qwerty` project, do not use the introduction to formulate the general problem of sEMG decoding, as we are all familiar with the problem. Instead, use the introduction to set up and motivate the question and techniques you pursued. For example, if you focused on minimizing CER in a single subject, or maximized generalization performance using minimal amounts of data, you may motivate why this is important in the introduction. If you are doing a custom project (not `emg2qwerty`) from your own research, please give us brief background and establish other baselines we should be comparing your results to.
3. **Methods** – State the methods of your project (such as the architectures, data preprocessing, or other techniques used).
4. **Results** – State the results of your experiments.
5. **Discussion** – Discuss insights gained from your project, e.g., what resulted in good performance, and any hypotheses for why this might be the case.
6. **References** – List references used in your writeup.

Grading

Here we outline the criterion by which we will grade the project. Note that some projects will be more creative than others; some projects will achieve higher performance than others. We will

provide room for extraordinary work in one category to compensate for deficiencies in another category. These are the general areas we will look into. Concretely, the final project will be graded on a scale of 20 points, but each section is assigned points so that the sum total can exceed 20 points. Your final project score will be capped at 20 points. You should aim to do a good job in all areas.

1. **Creativity (7 points)**

   - How creative and/or diverse is the approach taken by the student(s)?
   - Do the student(s) implement and try various algorithms?
   - Are multiple architectures compared?

   An example of what may be considered creative is comparing CNN, RNN, and RNN + CNN architectures in prediction performance. Creativity may also result from how one tackles the design of these algorithms, the types of data preprocessing or augmentations, or the approach taken to solve a problem like zero-shot generalization or fine-tuning with little data.

2. **Insight (7 points)**

   - Does the project reveal some insight about why approaches work or did not work?
   - Is there reasonable insight, explanation, or intuition into the results? (i.e. you should not just blindly apply different algorithms to a problem and compare them.)

3. **Performance (6 points)**

   - Does the project achieve relatively good performance on the problem, given that the students are training with limited resources?
   - How do different algorithms compare?
   - If the project is related to one's research, how do results compare to the literature? (i.e. you should not just train a few different algorithms without optimizing them reasonably)

   We do recognize that students may not have access to GPUs beyond Google Colab. If this is a problem for you, state it clearly in your results that you believe performance could be increased with more time. You can show this, for example, with a loss function plot that has physical time on the x-axis (e.g. showing that after some number of hours, the loss had decreased, but still had a long way to go). We will account for hardware acceleration limitations in grading your performance.

4. **Write-up (4 points)**

   - Are the approach, insight, and results clearly presented and explained?

   Dissemination of work is an important component to any project.

References

[1] Junbum Cha, Sanghyuk Chun, Kyungjae Lee, Han-Cheol Cho, Seunghyun Park, Yunsung Lee, and Sungrae Park. Swad: Domain generalization by seeking flat minima, 2021.

[2] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural 'networks. volume 2006, pages 369–376, 01 2006.

[3] Viswanath Sivakumar, Jeffrey Seely, Alan Du, Sean R Bittner, Adam Berenzweig, Anuoluwapo Bolarinwa, Alexandre Gramfort, and Michael I Mandel. emg2qwerty: A large dataset with baselines for touch typing using surface electromyography, 2024.