Motion-Based Handwriting Recognition and Word Reconstruction

Junshen Kevin Chen, Wanze Xie, Yutong He | {jkc1,wanzexie,kellyyhe}[at]stanford.edu

Video Presentation Link: https://youtu.be/sYK3rznT5nI

Overview

This project attempts **motion-based handwriting recognition** to explore an alternative to a vision-based approach, for various use cases where there is no convenient surface to write on (e.g. VR).

For this project, we build the collection **hardware** with Arduino and motion sensor, collect our own **original dataset** of written individual letters as training data, and written words as test data.

We solve this problem by building a **word reconstruction pipeline** that splits a given sequence to segments, search through combinations of segments for optimal trajectories, then auto-correct to produce a production word. Finally, we experiment with **domain adaptation** to handle unseen data distribution.

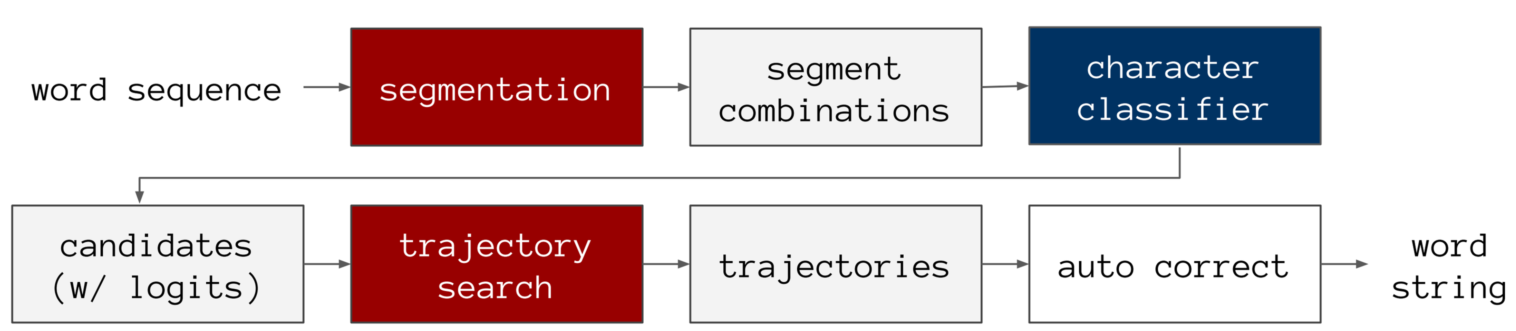
Data Augmentation

With a small dataset of individually written letter, we augment the training set to approach the distribution continuous writing:

- **shape modification** adding noise, stretching, rotating
- **prepend / append** frames from other letter samples
- **trimming** off small number of frames off current sample

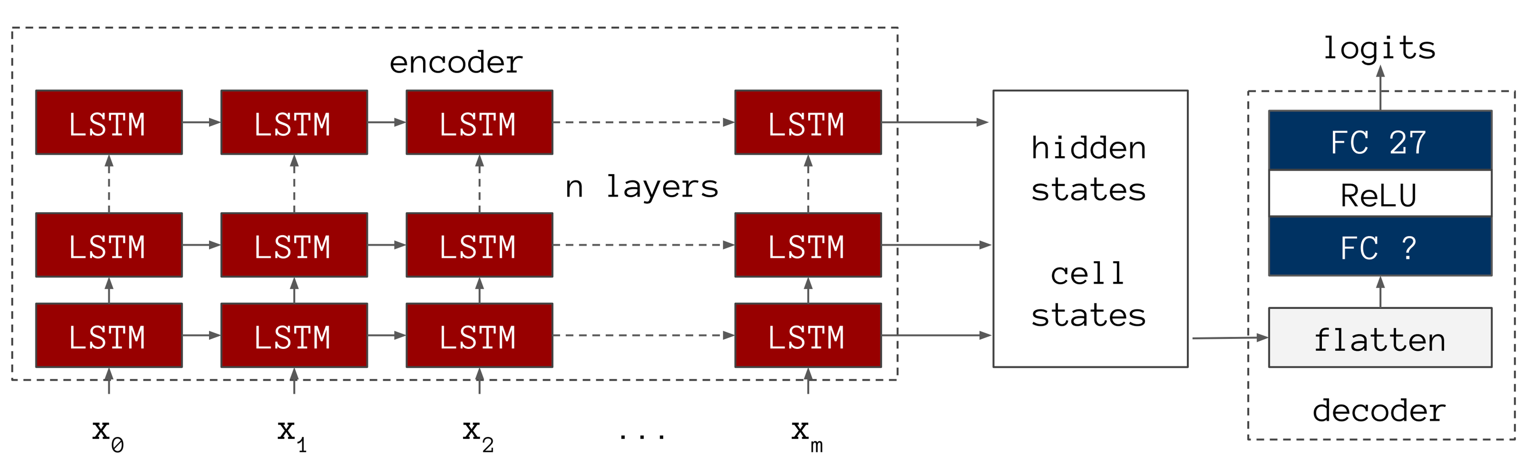
We also create **non-class** labeled samples by taking a combination of noise and partial letter samples.

Word Reconstruction Pipeline



Character Classifier

The character classifier is an **encoder-decoder**, with a LSTM encoder that encodes frames of motion data into states, then a feed-forward network decodes into a character class.

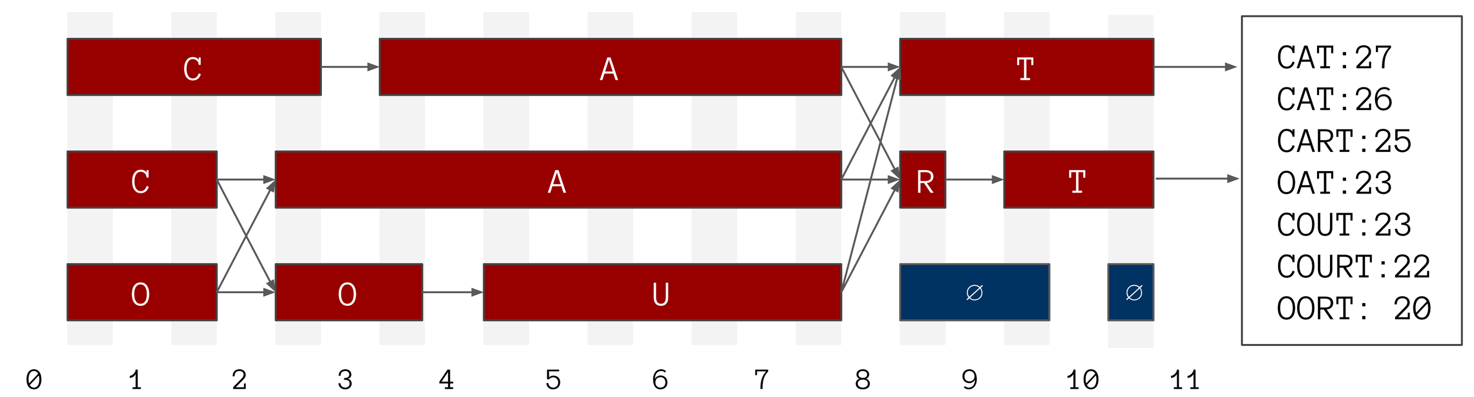


All combinations of segments are forwarded through the model in a batch to parallelize computation. The character classifier produces 26 **candidate letter predictions ranked by logits** for each segments. Non-class candidates are discarded.

Trajectory Search

We search through all candidates of all segments to form **trajectories**, combination of candidates through the sequence.

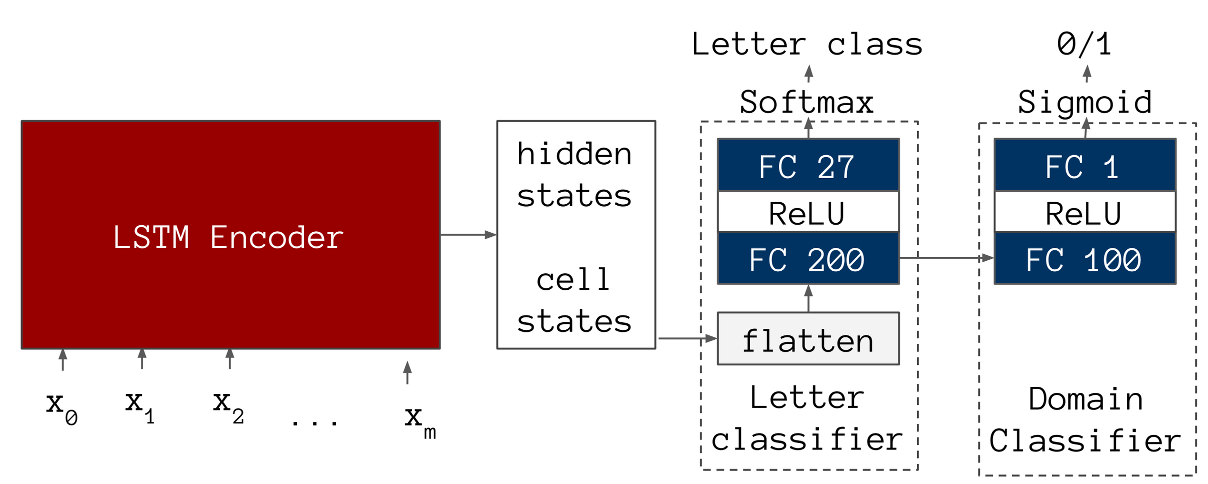
At each split-point, we keep the top ranking sub-trajectories ("beams") by average logit over constituting candidates. Trajectory search is a **dynamic programming** problem, where optimal sub-trajectories starting at a split-point must be optimal for all sub-trajectory arriving at this split-point.



Auto Correction

We implemented our auto-correction model based on Symmetric Delete spelling correction (**SymSpell**) algorithm. However, instead of directly auto-correcting the Top 1 result from the Trajectory Search, we auto-correct all available predictions from word search and use the confidence c_i from the word search, the frequency f_i and edit distance d_i from auto-correction lookup result, to pick the final predicted word. We experiment with four different techniques, where we finalize word based on **MaxVote** (max # of repeated occurrence), **SumConf** (max sum confidence c_i), **Division Combination** (max sum of $\alpha_i = c_i \cdot \frac{\log(f_i)}{\beta \cdot d_i + 1}$), or **Power Combination** (max sum of $\alpha_i = c_i \cdot \log(f_i)^{\frac{\beta}{d_i+1}}$). Note that we sum these properties only when the auto-correction outputs belong to the same word.

Domain Adaptation



To compensate the lack of generalized dataset, we use domain adaptation to transfer the knowledge and feature extractor the model learned from the limited dataset into any new user of the device. We train the model with the following loss function:

$$\mathcal{L}_{chr}(\theta_{chr}) = -\sum_{i=0}^{n}\sum_{c=1}^{27} y_c^{(i)} \log(P(x^{(i)}|c, \theta_{chr}))$$

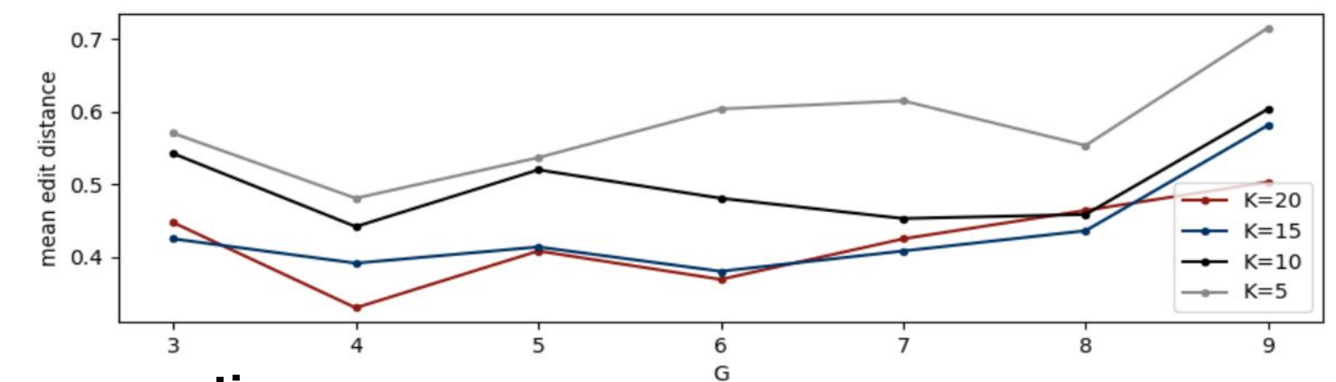
$$\mathcal{L}_{dom}(\theta_{dom}) = -\sum_{i=0}^{n} y_d^{(i)} \log(P(x^{(i)}|\theta_{dom})) + (1 - y_d^{(i)})(1 - \log(P_{dom}(x^{(i)}|\theta_{dom})))$$

$$\mathcal{L}(\theta_{chr}, \theta_{dom}) = \mathcal{L}_{chr}(\theta_{chr}) - \lambda \mathcal{L}_{dom}(\theta_{dom})$$

Experiment Results

Character classifier hyperparameter search: we perform a random search to find the best performing model has **8** LSTM layers of **275** dimensional hidden states, and **88** FC hidden units.

Word reconstruction hyperparameter search: we perform a grid search on number of splits per letter (**G**), and number of beams for trajectory search (**K**), and evaluate on average edit distance:



Auto-correction:

	Top 1	MaxVote	SumConf	D.C.	P.C.
OOD	0.229	0.208	0.229	**0.292**	0.260
ID-1	0.674	0.640	0.685	**0.832**	0.787
ID-2	0.678	0.711	0.733	**0.944**	0.900
ID-avg	0.676	0.676	0.710	**0.888**	0.844

Out-of-Domain Character Classification:

	Train Acc	Dev Acc	Test Acc
Original	-	-	0.13725
Fine-Tuning	0.96323	0.49057	0.49020
Domain Adaptation	**0.99185**	**0.64780**	**0.70588**

Analysis and Discussion

- **Character classifier** with a complex LSTM encoder and simple FC decoder works better, to encode complex features from raw data, while avoiding overfitting.
- **Trajectory search** with a higher number of beams produces higher accuracy by producing more "confident options" to auto-correct, while a higher number of split worsens performance by introducing noise and more false predictions.
- **Auto-correction** with the Direct Combination (D.C.) technique is able to produce the most accurate prediction, and contributes to our final word reconstruction pipeline's accuracy of 0.888.
- **Domain adaptation** is able to boost accuracy significantly for the character classifier, but is unable to improve sufficiently for accurate trajectory search. With a base model trained with better data, DA is a data-efficient way to apply the model to real world.

Future Work

- Apply multi-threading to parallelize trajectory search
- Improve data collection strategy, and collect more, cleaner data to train a better base model
- Experiment with other auto-correction and domain adaptation techniques