Transfer and Skip Hybrid Models for sEMG Keystroke Decoding

Colton Rowe
University of California, Los Angeles
Samueli Master of Engineering
ECE 247A, Dr. Jonathan Kao
Los Angeles, CA 90095
coltonrowe@g.ucla.edu

Abstract

This is a comparative analysis of the character error rate (CER) performance differences between recurrent networks in the framework of Facebook's Time-Depth Separable (TDS) Encodings [1]. The dataset used is the emg2qwerty dataset, which translates surface electromyography (sEMG) recordings into typed keys. The baseline Encoding + CNN model achieves a 22.15 testing CER after epoch 41. I then test architectures which substitute the multilayer CNN for an RNN and LSTM. To overcome computation limits, these tests were scaled back: each model was trained for 10 epochs with additional reduced settings to be computed on the CPU. The CNN achieves a loss of 2.37, the RNN achieves a loss of 2.99 before the loss explodes, and the LSTM achieves a loss of 2.83, ranking CNN $>$ LSTM $>$ RNN. An additional model that combines a CNN and LSTM is trained for 30 epochs and compared with the standard CNN, both on reduced settings. I then introduce a "Transfer Hybrid" model, which freezes the model parameters of a previously trained CNN, and trains a LSTM on its output. Neither of these hybrid architecture surpass the performance of the baseline CNN, but the transfer hybrid model shows more promise for future experiments. Lastly, I propose a skip hybrid model, which attempts to address the shortcomings of the transfer hybrid model.

1 Introduction

1.1 Background

Accessible human-computer interfaces have the potential to be revolutionized with surface electromyography (sEMG). Traditional computer-brain interfaces require a direct electrical connection with neurons, but sEMG provides a non-invasive alternative which could give amputees safe and independent access to computers. In the emg2querty dataset, sEMG is used to record electrical impulses in the forearms of participants while recording their keystrokes. The 2 x 16 electrical sensors on the participants forearms become 32 channels, which allows the convolutional layer to predict a character in this context window. Convolutional neural networks are typically used to process image data, where layers care about the relative distance between features in the input. Treating time as a spacial distance allows the convolutional layer to make accurate predictions on sequential data [1]. More typically, recurrent architectures like RNNs and LSTMs are used to process time series data. We would hope that a recurrent network could store memories about it's own prediction for the last keystroke, which could inform it's prediction about the next keystroke. Vanilla RNNs do not tend to perform as well as other recurrent architectures because of vanishing and exploding gradients, which can cause the loss to become unstable. Long Short Term Memory Networks are more stable than vanilla RNNs because the activations pass through forget gates which selectively throw away unnecessary information [2]. LSTMs can be used to generate natural language that sounds somewhat

sensible while following grammar rules. LSTMs don't require a large amount of data to produce impressive results. These properties make LSTMs ideal for assisting in next-letter prediction for keystrokes. Today, transformers dominate language related tasks, but attention mechanisms are much more data intensive than LSTMs, so I decide against using them in this exploration.

1.2 Methods Motivation

Here, I would like to outline specifically why I chose to implement each model in the next section.

1.2.1 Recurrent models

Three models were implemented to investigate how recurrent architectures perform on the sEMG data. I chose to train a TDS + CNN, a TDS + RNN, and a TDS + LSTM. This gave me a starting point to compare the differences between the recurrent and convolutional networks, and led me to try hybrid approaches in an attempt to increase performance.

1.2.2 Hybrid Model

A hybrid model was trained by combining a TDS + CNN + LSTM. Because the LSTM was the highest performing recurrent model, I thought combining it with the established CNN model could lead to higher performance. My intuition was that the TDS + CNN could decode the information in the scrolling window, and the LSTM can add additional context about letter relationships to inform the model of the best character class. The hope is that the CNN predicts the class of the current token to the best of it's ability, and the LSTM uses that prediction in tandem with it's memory to refine and modify the prediction.

1.2.3 Transfer Hybrid Model

Finally, I train a "transfer hybrid" model. The intuition for this model comes from transfer learning: we train the CNN model to convergence, then add an LSTM to the last layer, freezing the previously trained layers. The reason I think transfer learning could increase performance is because in the original hybrid model, the CNN and LSTM need to train together, which adds unnecessary recurrent information into the CNN through the gradient of the LSTM. In the transfer hybrid model, the CNN is allowed to learn the data independently, and the LSTM uses the CNN's prediction to adjust its output. Like the vanilla hybrid model, the hope is that the LSTM learns something about next-character prediction in the context of the previous TDS + CNN. Feeding the softmax into the LSTM has advantages over feeding in the raw character. For example, if the TDS + CNN model predicts that letter j has a probability of 30% of having been pressed and letter h has only a 20% chance of having been pressed, but the the last letter was a t, the LSTM could infer that the keystroke was really an h, not a j. Recurrent relationships like these could inform the model of the intended key press.

2 Methods

2.1 Dataset

In the emg2querty dataset, surface electromyography records signals in the forearms of participants while they type on a keyboard. The 2 x 16 electrical sensors on the participants forearms become 32 channels of data, capturing muscle movements which map to typed keys. The data is sampled at 2kHz with a window of 8000 - about 4 seconds of recorded EMG signal.

2.2 Baseline Model

The baseline model is a TDS + CNN as described in the Facebook paper [1] and used in the original emg2querty Github [3]. This is trained with 384 MLP features, four block channels of size 24, and with a batch size of 32. The model was trained for 41 epochs on a T4 GPU. For all of the models in this report, the Adam optimizer was used with learning rate of 1e-3.

2.3 Recurrent Architectures

A TDS + CNN, a TDS + RNN, and a TDS + LSTM were each structured by substituting the convolutional layer in the baseline model with a recurrent layer. Each of these models were trained with 192 MLP features, four block channels of size 12, and a batch size of 8 over 8 epochs. The reduction in parameter count, batch size, and epochs allowed the models to be trained on a CPU.

2.4 Hybrid model

A hybrid model was structured by combining a TDS + CNN with an LSTM, with 192 MLP features, four block channels of size 12, and a batch size of 8 over 30 epochs. It was compared to a baseline TDS + CNN structured the same way. Specifically, the hybrid model was structured as a LogSpectrogram - MLP - CNN - Affine - LSTM - Affine - Softmax.

2.5 Transfer Hybrid model

The Transfer Hybrid was created by reloading the TDS + CNN into a local variable, freezing each parameter in this model, then feeding the output into a sequence of Affine - LSTM - Affine - Softmax. The TDS + CNN separately continued training to compare with. Again, these models were trained with 192 MLP features, four block channels of size 12, and a batch size of 8 over 30 epochs, totaling 58 epochs for each model (a checkpoint at epoch 28 was used for the frozen CNN). This choice was because the baseline TDS + CNN model with reduced settings outperformed the baseline with higher settings.

3 Results

Model	BatSz	ChnSz	MLPSz	Epchs	Loss	Test CER
Baseline CNN	32	24	384	42	0.765	22.15
CNN	8	12	192	9	2.37	>=100
RNN	8	12	192	9	2.99	>=100
LSTM	8	12	192	9	2.83	>=100
CNN Long	8	12	192	30	0.665	21.82
Hybrid	8	12	192	30	0.815	25.99
Extended CNN Long	8	12	192	30	0.392	21.59
Transfer Hybrid	8	12	192	30	0.611	22.54

Table 1: Model Specifications and Tabular Results

3.1 Baseline Model

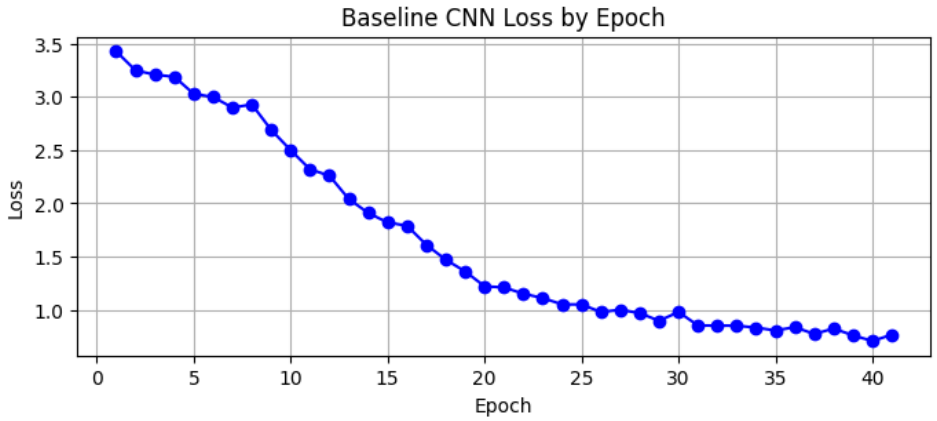

Figure 1: Loss by Epoch for Baseline CNN Model, 42 epochs.

[1]

The Baseline CNN with high settings trained for 42 epochs on a Colab T4 GPU and acheived a test CER of 22.15, with a loss of 0.76. Of the tested models, the CNNs performed the best across different parameters and trained as expected.

3.2 Recurrent Models

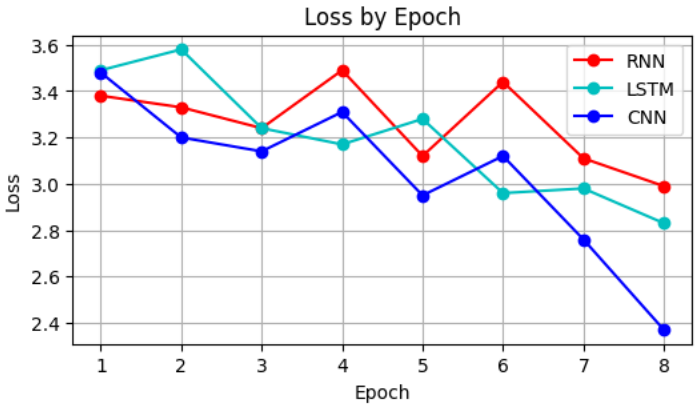

Figure 2: Loss by Epoch for CNN, RNN, and LSTM with reduced settings, 9 epochs.

The recurrent models were trained on reduced parameters with 192 MLP features, four block channels of size 12, and with a batch size of 8. The models were trained for 42 epochs on a Colab CPU. The exact CER couldn't be recorded for all of these models, but it's likely that the CER did not get below 100 because of the reduced amount of epochs. The structure of the RNN and LSTM models replaced the CNN layer in the original model with an RNN and LSTM layer respectively.

Comparing the reduced parameter models, the RNN did the worst, with a final loss of 2.99. Not shown in the diagram, the loss for the RNN was NaN on Epoch 9, likely due to a exploding gradient which would quickly increase the size of the parameters in the model.

The LSTM did better than the vanilla RNN, scoring a final loss of 2.83. It's expected that the LSTM should outperformed the RNN because the LSTM mitigates the problem of exploding gradients, and can handle long-range dependencies better than the Vanilla RNN.

The CNN performed the best, with a final loss of 2.37. CNNs are proven to be supreme in handling spacial data, so treating time as a spacial dimension proves convolutional layers to be extremely effective. A CNN is limited by its context window, which would suggest that it could useful as a feature-extractor for long-term dependencies introduced by an LSTM.

3.3 Hybrid Model

The hybrid CNN + LSTM model under performs the baseline, converging slower and reaching a final loss of 0.815 compared to the CNN's loss of 0.665. The CER of the hybrid model also under performs the CNN, with a CER of 25.99 compared to the CNN's CER of 21.82.

3.4 Transfer Hybrid Model

The transfer hybrid model under performs the CNN + extended training with a final loss of 0.611 compared to the CNN's loss of 0.392. The CER of the transfer hybrid model also underperforms the extended CNN, with a CER of 21.59 compared to the CNN's CER of 22.54.

[1]The loss in Epoch 0 for each model was over 100, so Epoch 0 was omitted to make the graphs more legible.

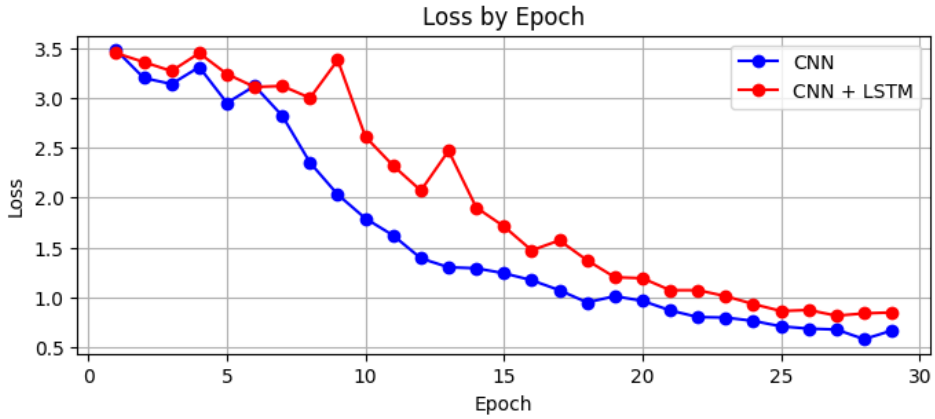

Figure 3: Loss by Epoch for CNN and the CNN + LSTM, 30 epochs.

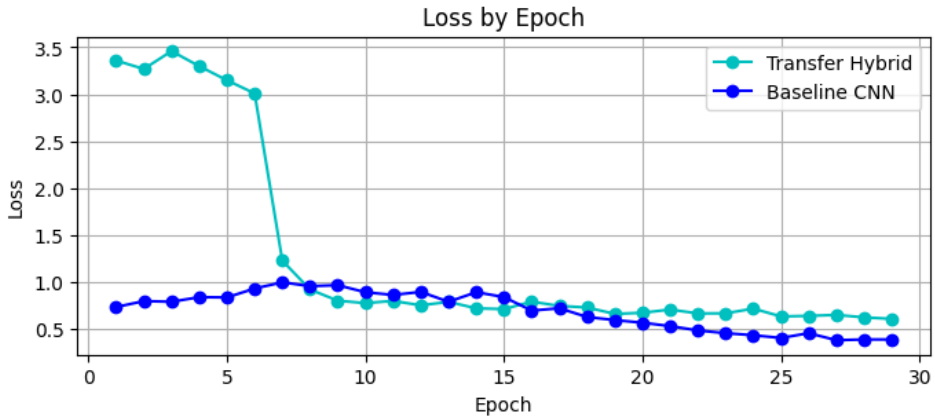

Figure 4: Loss by Epoch for the transfer hybrid model and baseline CNN with continued training.

4 Discussion

4.1 Highest Performing Model: Baseline CNN

Of the tested models, the baseline CNN consistently performed the best. I believe this is because it adds just enough complexity and information to the model without adding unnecessary bloat. I think that given more epochs, the CNN + LSTM could have outperformed the baseline CNN as the recurrence becomes more important in prediction, but this would likely take many more epochs. It's interesting that the reduced setting model outperformed the baseline - perhaps at lower epochs a simpler model is preferred, or that that more stochasticity improves performance.

4.2 Recurrent models

The vanilla RNN performed the worst in the recurrence comparison test, which was expected because of the vanishing and exploding gradient problems common in RNNs. The loss of the vanilla RNN exploded at only 9 epochs, which shows the importance of the forget gates in the LSTM. In the future, it would be interesting to compare additional architectures such as a GRU to the LSTM, because in fewer epochs the GRU would be expected to converge faster.

4.3 Transfer Hybrid Model

The fact that the transfer hybrid model has lower loss than it's input could suggest that it's learning something about next-character prediction in the context of the previous TDS + CNN. However, the

best CER it achieved did not beat out the CER of it's input model. The transfer hybrid model also does not beat out the baseline CNN when total trained epochs are compared. I believe two factors prevent this model from beating the baseline. The first is that the Transfer Hybrid model needs to spend time learning the basics: that the softmax probabilities from the CNN have predictive power for the output. The transfer hybrid model has a breakthrough between 8 and 10 epochs where the loss rapidly decreases and the model learns the basics. The second problem with this model is that the frozen CNN model can hinder the predictive capabilities of the LSTM. To be fully effective, the input to the LSTM should be trained until complete convergence. Without a converged model, we can't hope for the LSTM to add additional sequential context to the prediction of a keystroke that outperforms the CNN simply continuing to improve. The frozen CNN caps the maximum performance that the transfer hybrid model can achieve.

4.4 Future Investigation: Proposed Architectures for a Skip Hybrid Model

The transfer hybrid model has the problems of the CNN not having fully converged and needing to relearn the 'basics' of the CNN's prediction (the large drop off in the loss). To be fully effective, the input to the LSTM in the transfer hybrid model should be independently good at predicting a keystroke. My original motivation behind the transfer hybrid model was to have the CNN train independent of the LSTM, because the CNN shouldn't care too much about the gradient of the recurrent layer, it should care mostly about making the best possible prediction given the information from the encoder. I believe is why the baseline CNN trains better than the Vanilla CNN + LSTM Hybrid: because the hybrid model has the weights in the CNN updated based on the recurrence in the network, which the CNN shouldn't care about. A "skip hybrid" model could pass the loss around the recurrent layer into the CNN directly, which would address the potential downside of the recurrent information updating the CNN. This is similar to the idea of a gradient highway as used in architectures like RESNet. An architecture like this would subvert the disadvantages of the transfer hybrid and vanilla hybrid models, while adding valuable recurrent information to the output of the baseline model. In Figure 5, I outline two possible architectures for a skip-hybrid model. Notably, both architectures sever the backward pass of the gradient from the LSTM block to the TDS + CNN block.

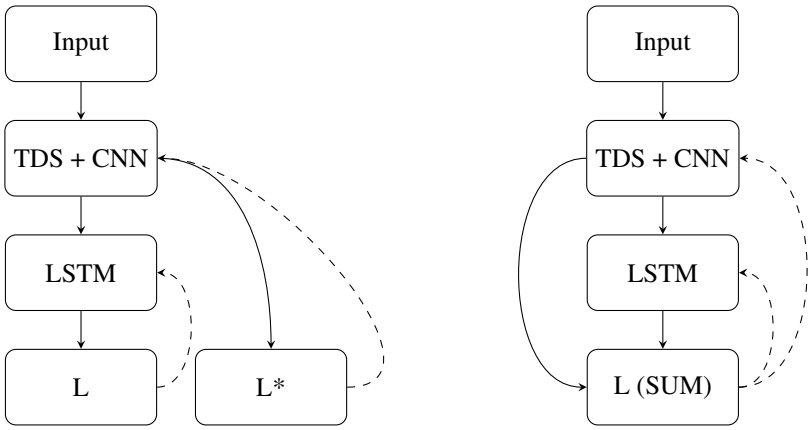

Figure 5: Proposed Architectures for a skip hybrid model. The dotted lines represent the flow of gradients backwards through the network and the solid lines represent the forward pass.

References

[1] Awni Hannun, Ann Lee, Qiantong Xu, and Ronan Collobert. Sequence-to-sequence speech recognition with time-depth separable convolutions. 2019.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[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.