Ablation Report HW3P2

Nicholas Magal

1 Introduction

In Homework 3 Part 2 we are given the challenge of mapping utterances to phonemes. During my experiments the best performing model achieved a 9.49 average Levenshtein distance. The following different architectures and configurations were attempted:

- Pyramidal LSTM Architecture
- GRUs
- DropConnect
- NT-ASGD optimizer
- Ensembling
- Variable Data Augmentation and Resnet Embedding
- Hyperparameter Search

Below the performance and rationale of each of these methods are described. The notebook that contains all of these experiments can be found here.

2 Configurations

2.1 Baseline

Below contains the baseline model, which scored a 9.49 Levenshtein distance. All other model configurations build upon this baseline, and only differ from this baseline in the outlined difference. For example, Pyramidal LSTM used the same hyperparameters as our baseline and only differs in its architecture.

Table 1: Hyperparameters

Learning Rate	0.0006
Optimizer	Adam
Batch Size	64
Schedular	ReduceLROnPlateau
Weight Decay	5e-6
LSTM Hidden Nuerons	512
LSTM Layers	4
Dropout	0.25
Patience	5
Beam Width	1
Epochs	52

The architecture used for this was: Conv1d –> ReLU –> BatchNorm1d –> Conv1d –> ReLU –> BatchNorm1d –> Dropout –> biLSTM –> biLSTM –> ReLU –> Linear Layers –> ReLU –> Dropout –> Linear Layers.

2.2 Pyramidal LSTM Architecture

The pyramidal architecture used is similar to the one used by Chan et al. in [1]. Since there are many time steps for each utterance, and theoretically multiple time-steps for each phoneme, ideally a pyramidal LSTM will improve performance by reducing the total number of time steps by a factor of $1/(2*l)$, where l corresponds to the highest level of the pyramidal LSTM.

Empirically, the pyramidal LSTM performed worse then our baseline. In fact, when we created 3 levels, the model was unable to learn and gave us nan for our training and validation loss.

This is probably due to the network being able to predict correctly at each time-step the relevant phoneme, and thus reducing the amount of time steps actually negatively impacts the network by taking away valuable information. A pyramidal LSTM would probably be better suited for tasks such as utterance to word.

Table 2: Pyramidal LSTMs Performance

Levels	Training Loss	Validation Loss	Lev. Distance	Epochs
1	0.0007205	0.008384	10.1	51
2	0.0009438	0.007369.9	11.14	51
3	nan	nan	-	1

2.3 GRUs

The GRU is a variant of the LSTM that typically trains faster and contains less learnable parameters than the LSTM. Because of this, it is usually a good idea to try out GRUs on tasks where LSTMs perform well.

Switching out the LSTMS for the GRUs resulted in worse performance. Furthermore, surprisingly enough the GRUs converged much slower then did the LSTMs. This shows that even though GRUs are newer then LSTMs, LSTMs still remain a good choice for sequential data.

Table 3: GRUs Performance

Training Loss	Validation Loss	Lev. Distance	Epochs
0.00568	0.008694	14.94	100

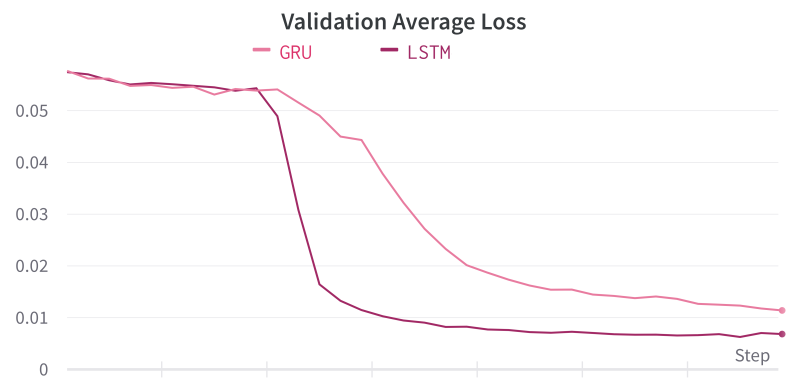

Figure 1: GRU Validation Average Loss vs LSTM for 30 Epochs

2.4 DropConnect

From [2], DropConnect is a regularization method for RNN's that helps increase a model's ability to generalize by adding a dropout operation on recurrent hidden to hidden weight matrices. The recurrent weights are reused through time-steps, and in this sense the same individual dropped weights are dropped across time-steps [2]. For DropConnect, the open source library called Pytorch-NLP was used [3].

Empirically adding DropConnect did not improve the model's performance. This could be because Pytorch's built in dropout in nn.LSTM is already applying sufficient dropout to the network.

Table 4: DropConnect Performance

Training Loss	Validation Loss	Lev. Distance	Epochs
0.002692	0.006639	10.27	80

2.5 NT-ASGD Optimizer

NT-ASGD is a variant of ASGD and is proposed first by Merity et al. in [2]. NT-ASGD works by using a "non-monotonic criterion that conservatively triggers the averaging when the validation metric fails to improve for multiple cycles." Code from [4] was used that provided a Pytorch implementation of this optimizer.

While training with NT-ASGD, I observed after 16 epochs my loss dramatically increased. One possibility of this problem is the exploding gradient. In order to combat this, gradient clipping was used. Gradient clipping works by ensuring the gradient of a parameter falls between a specific range, denying a exploded gradient the opportunity to throw off a models parameters. Once gradient clipping was applied, training converged smoothly, although performed worse than the Adam optimizer.

Table 5: NT - ASGD Performance

Training Loss	Validation Loss	Lev. Distance	Epochs
0.00203	0.006875	10.91	92

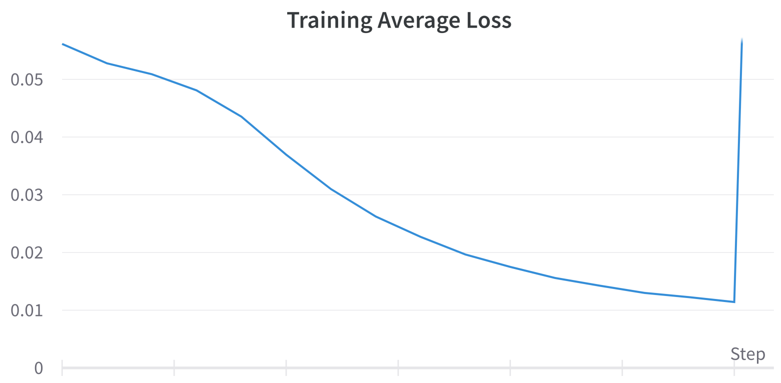

Figure 2: Training Loss After 16 Epochs, Exploding Loss

2.6 Ensembling

Ensembling works by combining the results of different models into one result. By combining the results of multiple models, where some models may perform better on some data and others may perform better on other data, theoretically ensembling should be able to better generalize to data. Ensembling works best when there is low correlation between the different models being ensembled.

The following models were used for ensembling: Baseline, Baseline without Convolution Layers, DropConnect Baseline, and NT-ASGD Baseline. These models were chosen for both their relative high performance and also dissimilarity to each other.

Table 6: Ensembling Performance

Linear Interpolation Lev. Distance	Log Linear Interpolation Lev. Distance
78.2	92.2

2.7 Variable Data Augmentation and Resnet Embedding

Adding more sophisticated CNN architectures for embedding could potentially increase the capacity of a model and allow it to better generalize. Three Resnet blocks were used in order to test this.

Alongside this, variable data augmentation (DA) was used as the model was trained. Variable DA allows a model to continue to learn new patterns of data as it learns and becomes more sophisticated. Rather then to apply DA all at once, this allows a model to better keep up with learning and not challenge it too much from the start of training.

For the first 10 epochs, no DA was used. For epochs 10-30, frequency and time masking with parameters of 5 and 10 were used. For epochs 30-40, frequency and time masking with parameters 10 and 20 were used. And finally, for epochs 40-60 frequency and time masking with parameters 15 and 30 were used.

The following results are from a model employing both of these tactics.

Table 7: Resnet and Variable DA Performance

Lev. Distance	Epochs
11.14	60

2.8 Increasing Hidden Dimension

In order to increase capacity of the network, the hidden dimension of 512 was doubled to be 1024. After applying this two interesting things were observed. The model's training and validation loss became more unstable, and the model also learned at a much slower rate.

Although ultimately this may have improved the performance, the model trained prohibitively slow, and was preemptively ended after the 52nd epoch.

Table 8: 1024 Hidden Dimension Performance

Training Loss	Validation Loss	Lev. Distance	Epochs
0.07475	0.0703	80.57	52

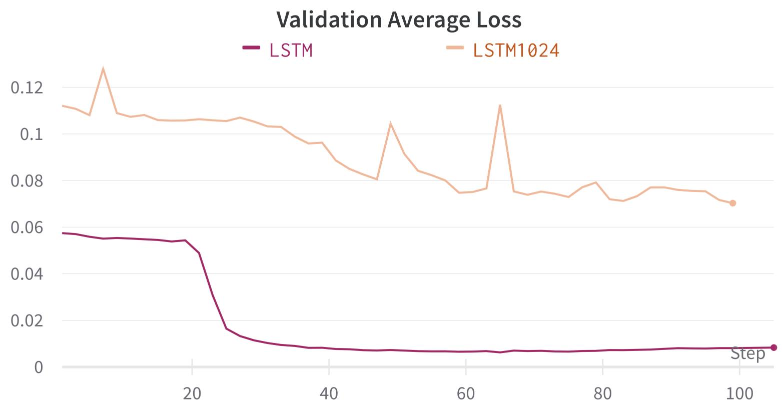

Figure 3: LSTM with 512 Hidden Dimension vs 1024 Hidden Dimension After 52 Epochs

3 Conclusion

Although many different tactics were employed, none were able to beat the baseline performance of 9.49. The author understands that other solutions to HW3P2 acheived below a score of 8.0. Further exploration into this problem may investigate how data is being read into the model as a possible source error.

References

[1] Chan, W., Jaitly, N., Le, Q., ,Vinyals O., (2015). Listen, Attend, and Spell *ArXiv, https://arxiv.org/pdf/1508.01211.pdf*

[2] Merity, S., Keskar, N., Socher, R., (2017) Regularzing and Optimizng LSTM Models *ArXiv, https://arxiv.org/pdf/1708.02182.pdf*

[3] *Pytorch - NLP* Retrieved from https://github.com/PetrochukM/PyTorch-NLP.git

[4] *Awd - LSTM* Retrived from https://github.com/ahmetumutdurmus/awd-lstm.git