CS512: Advanced Machine Learning.

Assignment 3: Adversarial Training on Sequence Classification

Garima Gupta: ggupta22@uic.edu

Sai Teja Karnati: skarna3@uic.edu

Shubham Singh: ssing57@uic.edu

Wangfei Wang: wwang75@uic.edu

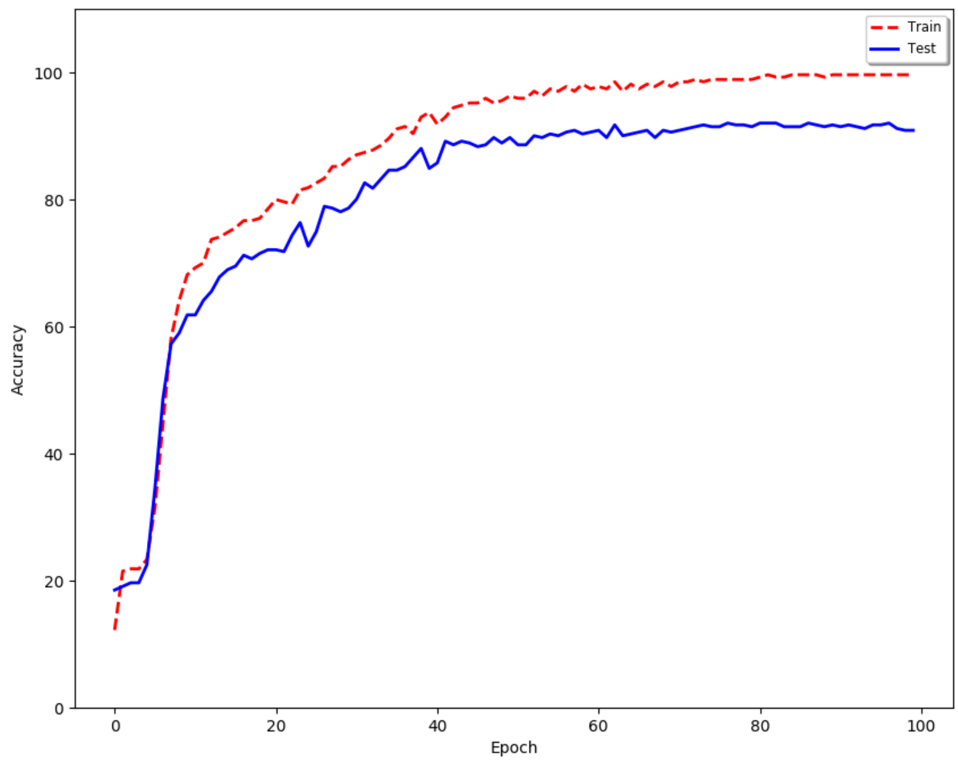
April 16, 2020

1 Introduction

2 (15 points) Training the Basic Model

Hyperparameters values:

`batch_size = 27`, `hidden_size = 10`, `basic_epoch = 100`, `out_channels = 64`, `kernel_size = 10`, `stride = 3`, `lr = 1e-3` (learning rate), `weight_decay = 1e-3`.



3 (10 points) Save and Load Pretrained Model

See code in `training.py` under the comment `Part 3, Save and Load model`.

4 (25 points) Adversarial Training as Regularization

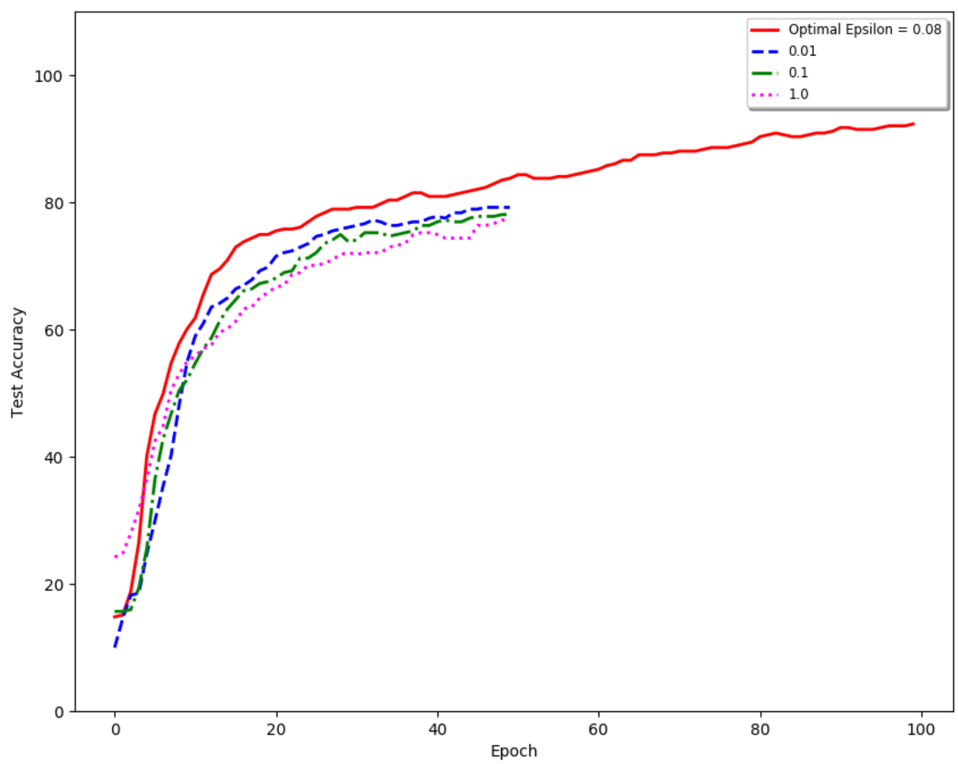
a **(10 points)** See the `compute_perturbation` function in `training.py`.

b **(5 points)** See the branch `mode` = ‘AdvLSTM’ in `LSTMClassifier` in `Classifier.py`.

c **(10 points)**

Among the ϵ's we have tried ($\epsilon = [0, 2, 4, 6, 8, 10, 0.001, 0.01, 0.08, 0.1, 1, 10, 100, 1000]$), $\epsilon = 0.08$ gives the optimal performance at the end of 100 epochs. The other hyperparameters were set the same as those in the basic model. The adversial training improved the test accuracy although this improvement is not significant. The basic model test accuracy is about 92.02%, while with adversarial training, the test accuracy reached 92.3%. The output of basic model test accuracy and test accuracy with adversarial training were stored in `BasicModel_test.txt`, and first 100 rows of `AdvModel_acc.txt` in folder `Figures`.

As shown in the figure, the performance of the model changes slightly with the change of ϵ between $[0.01, 0.1, 1]$, meaning our model is pretty robust to disturbance. At the end of epoch 50, $\epsilon = 0.01$ seems to give the best test accuracy among $\epsilon = [0.01, 0.1, 1]$.

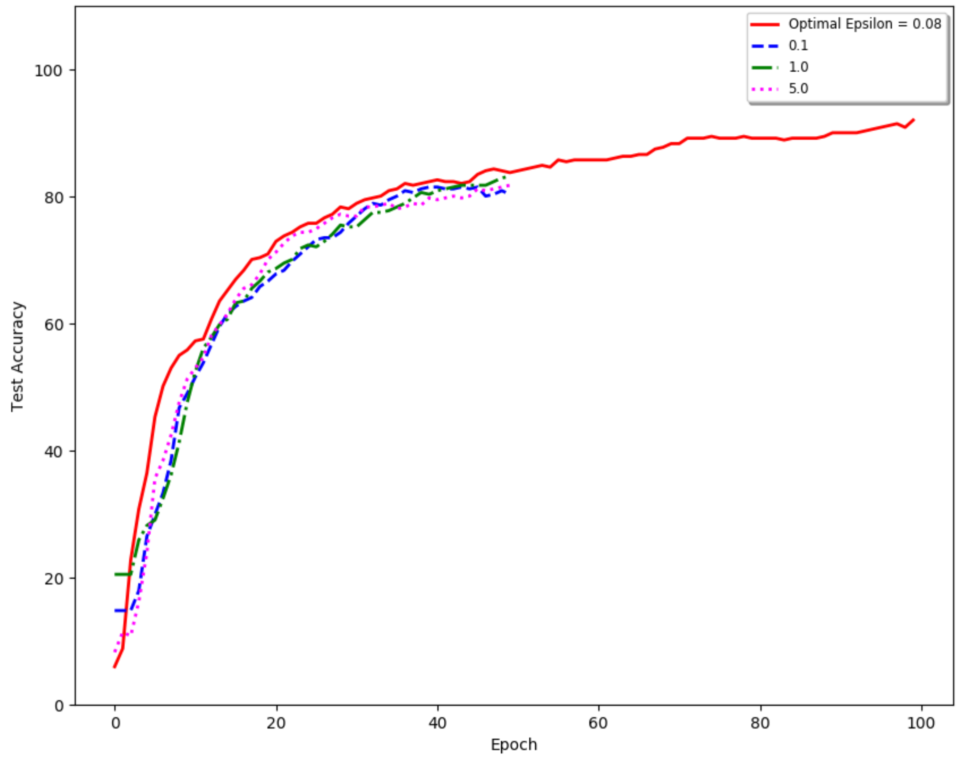


5 (40 points) Adversarial Training as Proximal Mapping

a **(30 points)** We have implemented the ProxLSTMCell in `ProxLSTM.py`. We also have implemented `forward` pass and `backward` pass. See code.

b **(10 points)** We have written a branch in `LSTMClassifier` that can handle `mode = 'ProxLSTM'`. See code.

Among the $\epsilon = \lambda^{-1}\sigma^2$ we have tried, $\epsilon = 0.08$ also performed the best. The performance of the model (test accuracy $= 92.02\%$) did not improve significantly from the previous models.



This is probably because of the small dataset. We also notice that the small change of ϵ did not change the performance significantly. Among $\epsilon = [0.1, 1.0, 5.0]$, $\epsilon = 1.0$ seems to perform the best.

6 (10 points) Dropout and Batch Normalization

a **(5 points)** We have initiated a dropout layer in `Classifier.py` and we use it with a flag `apply_dropout`. When the flag is set to `True`, we apply the dropout before the convolution layer. Our finding was that by adding a dropout layer in `Classifier.py` did not help regularize the convolution parameters, and improve the test accuracy. It actually made our model underfit compared to any of the previous models (test accuracy dropped to around 87%). The results were stored in `ProxModel_acc_dropout.txt`.

b **(5 points)** We have implemented a batch normalization layer in `Classifier.py` and like dropout, we have a flag `apply_batch_norm`, which when set to `True`, is applied before the ProxLSTM layer. By adding the batch normalization layer, the test accuracy is around 91.5%. It doesn't seem to help improve test accuracy greatly. The results were stored in `ProxModel_acc_batchnorm.txt`.