CS512: Advanced Machine Learning.

Assignment 2: Conditional Random Fields and Convolutions

Garima Gupta: ggupta22@uic.edu

Sai Teja Karnati: skarna3@uic.edu

Shubham Singh: ssing57@uic.edu

Wangfei Wang: wwang75@uic.edu

April 10, 2020

1 (20 points) Convolution

(3a) **(20 points)** We implemented Conv layer and the get_conv_features(x) function in the starter code, with accommodation of different strides and an option of zero padding. The implementation was tested.

For the following X and K with unit stride and zero padding:

$$X = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}; \quad K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

the result of convolving the X with K is:

$$\hat{X} = \begin{bmatrix} 2 & 2 & 3 & 1 & 1 \\ 1 & 4 & 3 & 4 & 1 \\ 1 & 2 & 4 & 3 & 3 \\ 1 & 2 & 3 & 4 & 1 \\ 0 & 2 & 2 & 1 & 1 \end{bmatrix};$$

See `code` folder for `conv_test.py`.

2 (50 points) CRF

(4a) We implemented the forward, backward pass and loss inside `crf.py`.
See code.

(4b) **(20 points)**

Parameters we are using:

Batch size = 64

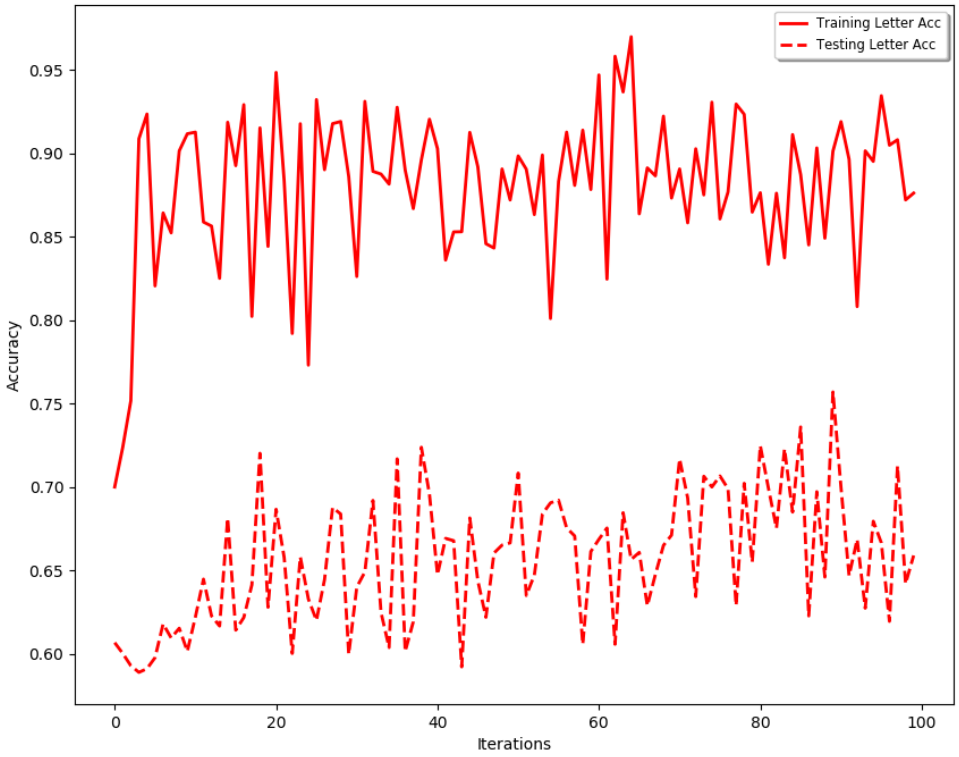
Number of iterations = 100

C = 1000 for CRF

LBFGS lr = 0.1

Zero-padded

(1) **letter-wise prediction accuracy**.

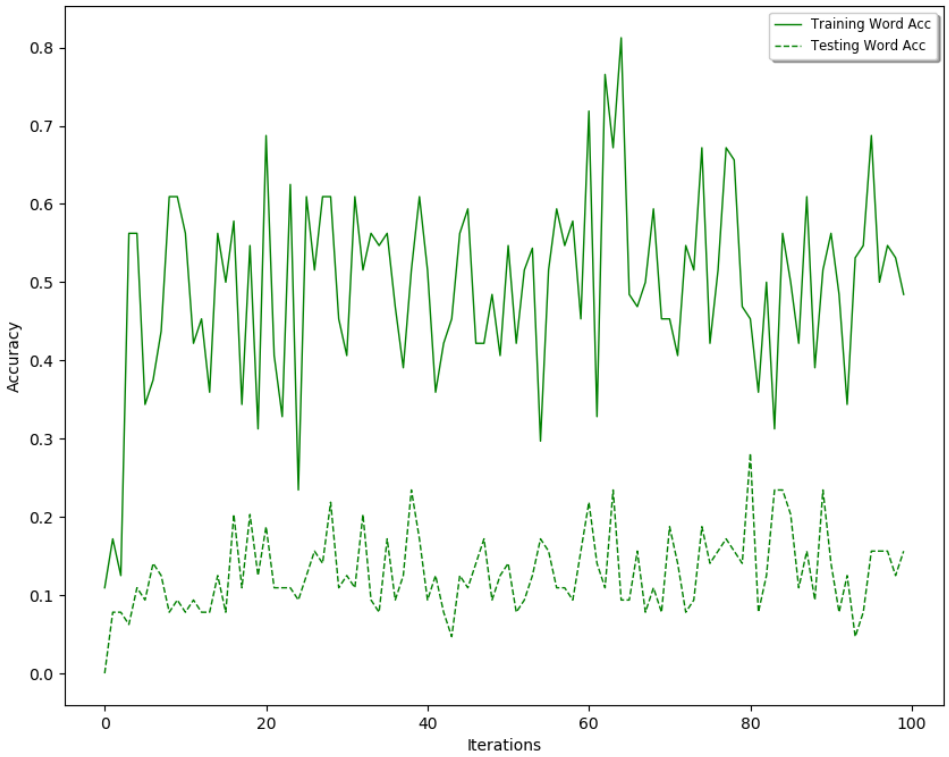


(2) **word-wise prediction accuracy**.

Average letter accuracy for training set is 88.0%

Average letter accuracy for test set is 65.7%.

Average word accuracy for training set is 49.4%



Average word accuracy for test set is 12.9%

(4c) **(20 points)** Repeat experiments in (4b) with the following convolution layers. Set stride and zero/no padding to optimize the test performance.

The other parameters used in this case are:

Batch size 64

Iterations 100

C = 1000 for CRF

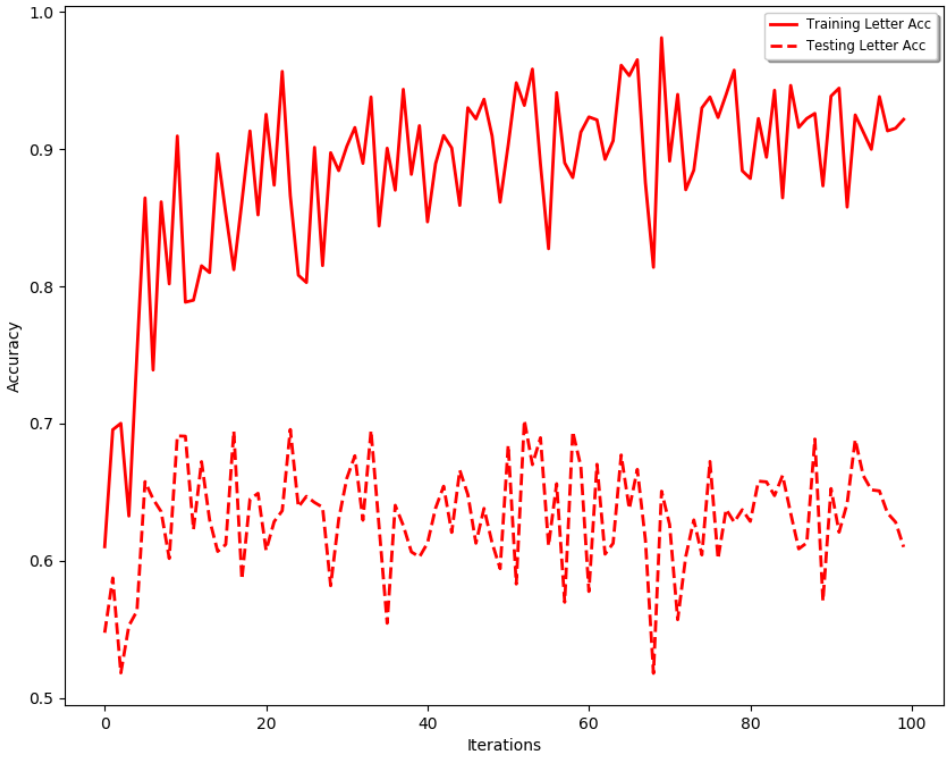
LBFGS learning rate = 0.1

Zero-padded

2-CNN CRF:

1. A Layer with 5×5 filter matrix
2. A Layer with 3×3 filter matrix

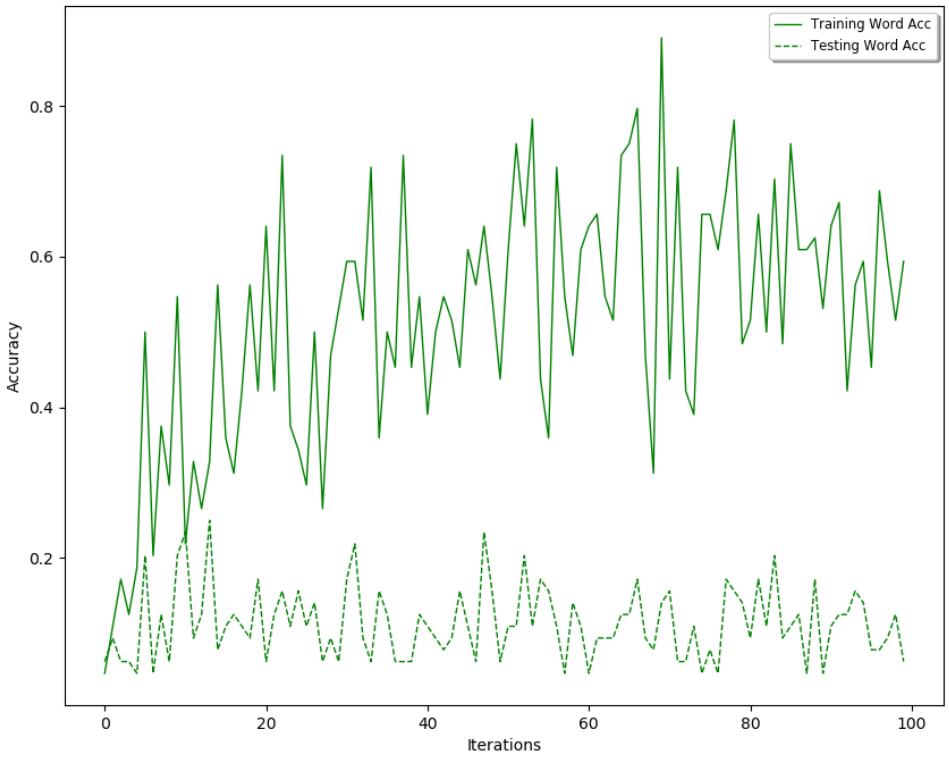
(1) **letter-wise prediction accuracy**,



(2) **word-wise prediction accuracy**.

Average letter accuracy for training set is 88.3%

Average letter accuracy for test set is 63.1%



Average word accuracy for training set is 51.4%

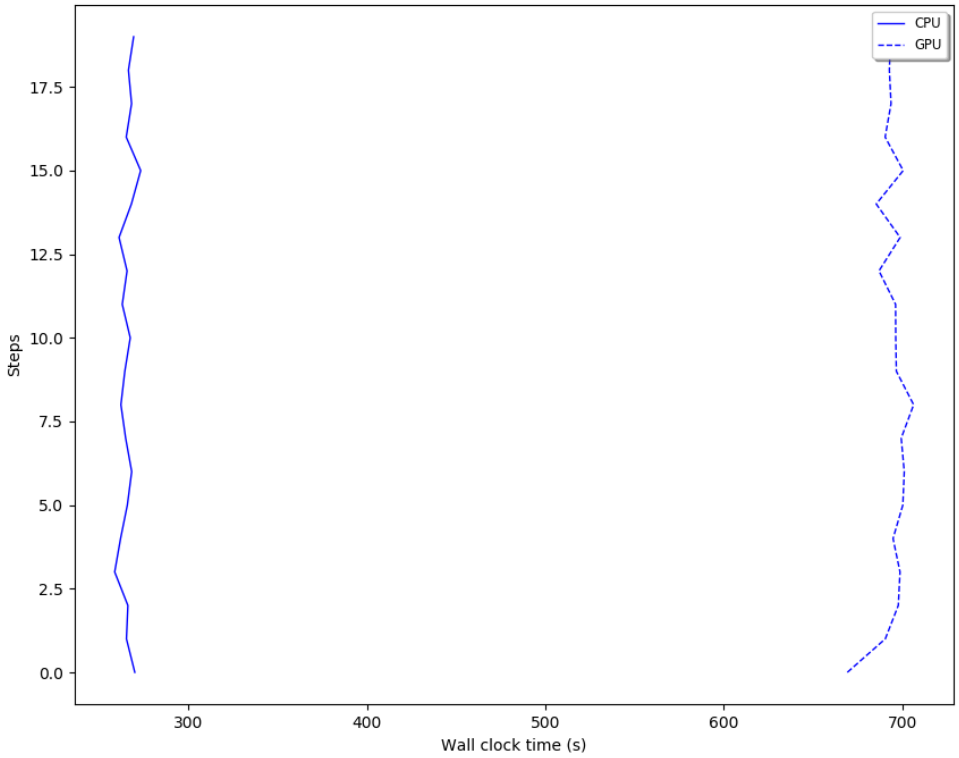
Average word accuracy for test set is 11.4%

Our accuracies fluctuates a little bit between iterations.

1. This could mean that our model is subjective to small noises. And on test set, if the accuaracy fluctuates, it usually means that our model is overfitting. We've played with hyper-parameters (e.g., learning rate and value of C) to try to increase the regularization. However, the result still fluctuates.
2. We've also noticed that the dataset we were given include some words that were repetitive. That could also cause the unstableness of our model.

Comparing 4b and 4c, the accuracies are pretty similar. It's very like both of the models in 4b and 4c are overfitting and therefore, adding an extra layer in 4c did not improve accuracies a lot.

(4d) **(10 points)** Enable GPU in your implementation. Does it lead to significant speedup? You can test on the network in 4c. Make sure your plot uses wallclock time as the horizontal axis.



The GPU isn't much faster than CPU. In our case, GPU is even slower than CPU. We think it's first because that function dp_infer() in inference.py is quite slow and not optimal for GPU. Second, we think somewhere in our code is using CPU that is not compatible with the GPU we were using, which caused communication problem between CPU and GPU.

We also noticed that using matrix operations instead of for loop in dp_infer() could speed up the code.