**CS 512: Advanced Topics in Machine Learning** **Due: 05:00, April 17, 2019**

# Assignment 3: Conditional Random Fields and Convolutions

*Student:* *Email:*

**Group Members:**
Omid Memarrast <omemar2@uic.edu>
Sanket Gaurav <sgaura2@uic.edu>
Raj Shekhar <rshekh3@uic.edu>
Sarit Adhikari <sadhik6@uic.edu>

## 1 (20 points) Convolution

(3a) Code provided for this part.

## 2 (50 points) CRF

(4a)

(4b) **(20 points)**
For Two Convolution layer:
Training accuracy is plotted in Figure 1
Letter-wise prediction Test accuracy: 80.68%
Word-wise prediction Test accuracy: 49.12%

(4c) **(20 points)**
For Two Convolution layer:
Training accuracy is plotted in Figure 2
Letter-wise prediction Test accuracy: 81.29%
Word-wise prediction Test accuracy: 49.98%

(4d) **(10 points)**
Yes, it leads to significant improvement in speeding up the training process of the network. Due time scarcity we could not have time to go back and print the time stamp. On an average the training time on GPU is significantly reduced by 6 times of CPU usage.

## 3 (30 points) Comparison with Deep Learning

(5a) **(10 points)** If you designed your own DNN model, then report the implementation details of it, along with the model architecture, loss functions etc. If you picked an existing model (say VGG), explain each of the layers inside the network and its purpose for the task at hand. That is, why you picked it and what functions (layers) were useful in the solution to the problem. In addition, look into the source code and sketch its structure within 150 words.

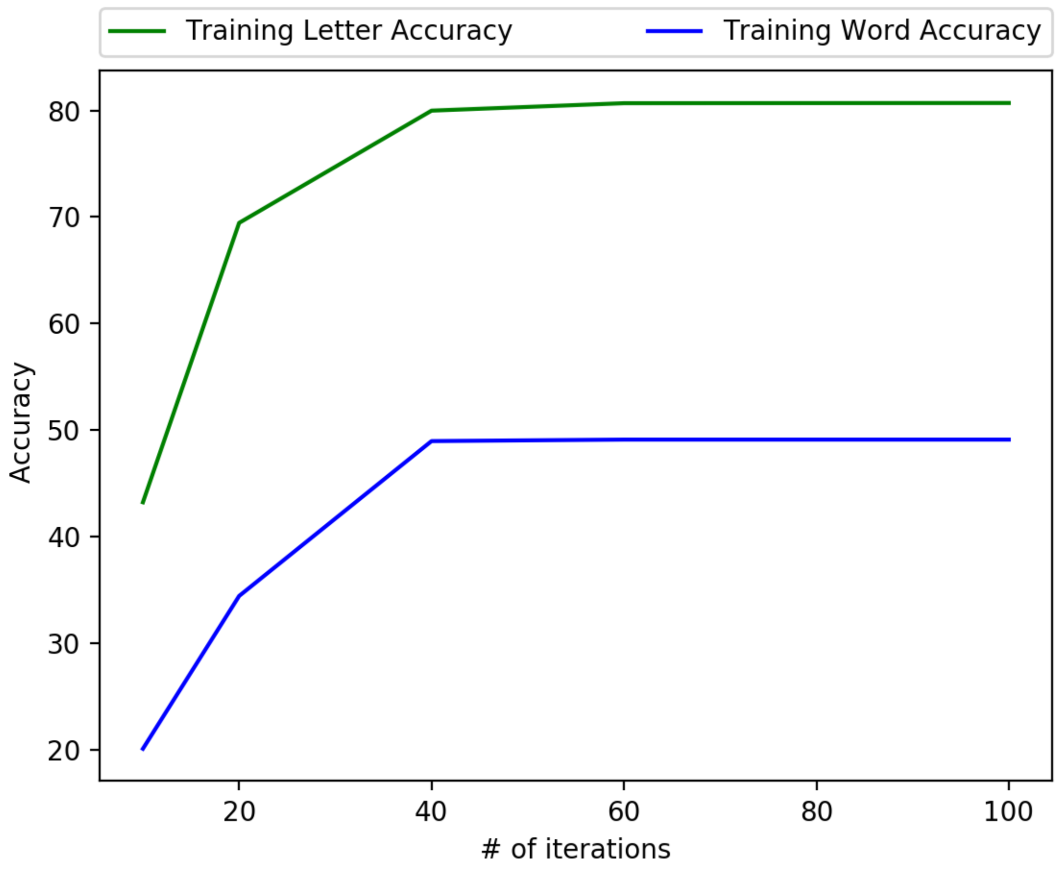

**Figure 1.** Letter-wise accuracy and Word-wise accuracy for one convolution layer

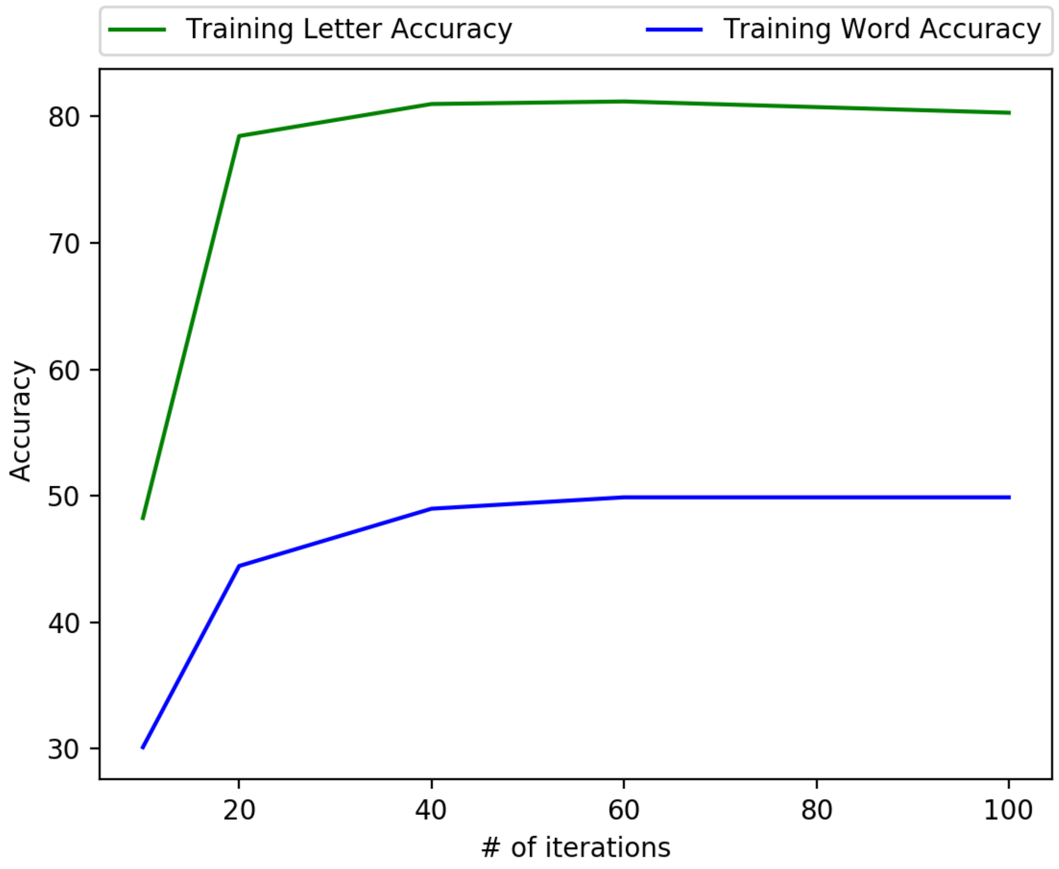

**Figure 2.** Letter-wise accuracy and Word-wise accuracy for two convolution layer

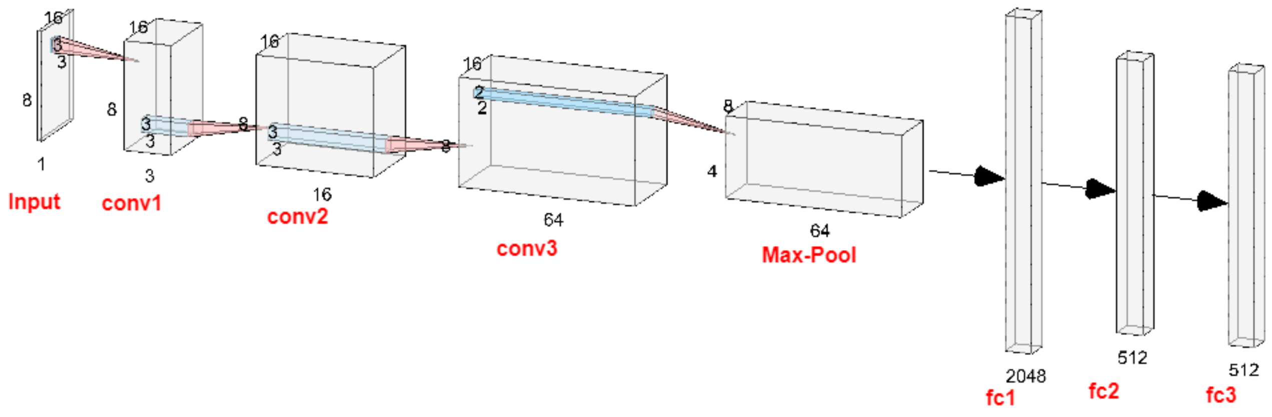

**Figure 3.** CNN architecture used for benchamarking

We designed our own DNN model with the main components as shown in figure 3. This DNN model is a simpler version of AlexNet. Our DNN architecture contains 3 convolutional layers and 3 fully connected layers. Dropout is applied after the first and the second fully connected layer and batch-normalization is applied after every convolutional layer. There is one max-pooling layer before feeding flattened feature to 3 fully conncted layers. Relu is applied after every convolutional and fully connected layer.

Like AlexNet our architecture uses of ReLU (Rectified Linear Unit) Non-linearity. Tanh or sigmoid activation functions used to be the usual way to train a neural network model. AlexNet showed that using ReLU nonlinearity, deep CNNs could be trained much faster than using the saturating activation functions like tanh or sigmoid. So ReLU plays an important role in this architecture.

As loss function, we used nn.CrossEntropyLoss() class in PyTorch. It is useful when training a classification problem with C classes and that is the case in this project.

(5b) **(5 points)** Plot the letter-wise and word-wise prediction accuracies on both training and test data ($y$ axis) over 100 iterations ($x$ axis) (You might have to implement these). Compare this with your CNN+CRF results and report your analysis (which model fared better? and why?). You may use the hyperparameter that yields the best performance for your CNN+CRF model. If it has not converged in 100 iterations, you may increase the number of iterations.

In CNN+CRF we achieved test accuracy of 81.29% for letter-wise and also 49.89% for word-wise. These numbers are on par with best performance that we could get with CRF in last assignment. What we expected was to get best results when adding CNN to CRF model. However, our own designed CNN model which does not have CRF layer can achieve the highest accuracy. DNN model with tuned hyperparameters can achieve 0.98 accuracy on letter-wise and almost 0.90 accuracy on word-wise prediction. The optimizer used for this part is L-BFGS, so it produces high accuracy but it takes a long time for model to converge (about one hour). DNN used for this project is much more complex than CNN+CRF model because it has much more layers and nodes. This complexity allows model to learn so many patterns that exist in the training data, so that model outperforms the simple CNN+CRF model.

(5c) **(5 points)** Change the optimizer from *LBFGS* to *ADAM*. Repeat the experiments in (5b) and report the letter-wise and word-wise accuracies, with $x$-axis as the #iterations. Does *ADAM* find a better solution eventually, and does it converge faster than *LBFGS*?

Fig 6 and Fig 7 show letter-wise accuracy and word-wise accuracy when using Adam optimizer respectively. After convergence letter-wise accuracy is 0.932 and word-wise accuracy is 0.629. Using Adam optimizer makes DNN to converge much faster and it our experiment it takes around only 5 minutes to get the desired result. Although L-BFGS achieves higher accuracy (around 0.98 for letter-wise) it is super slow in compare to Adam optimizer. One main reason is that L-BFGS compute second order gradient and needs more computation and memory. As far as we know, almost everyone in deep learning community uses Adam optimizer because it is very fast and also we normally don't need accuracy near to 1 on training data because it will make our model to overfit.

(5d) **(10 points)** Why did you choose this model (again it could be your own design or an off-the-shelf model)? More precisely you should explain every design decision (use of batchnorm layer, dropout layer etc) and how it helped in the task at hand, in your report.

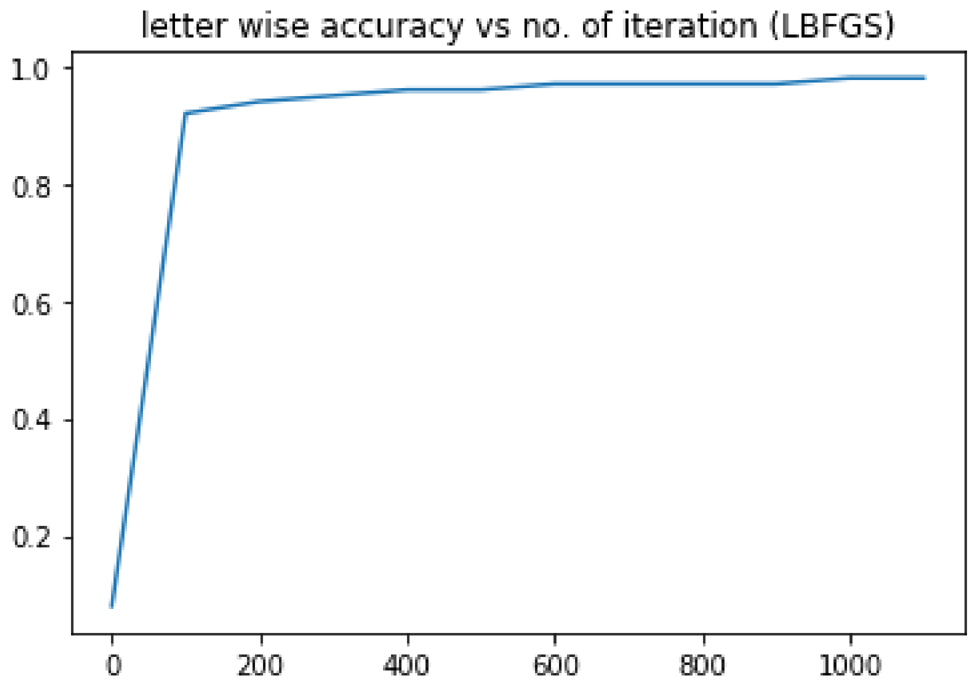

**Figure 4.** Letter-wise accuracy in DNN when using L-BFGS optimizer

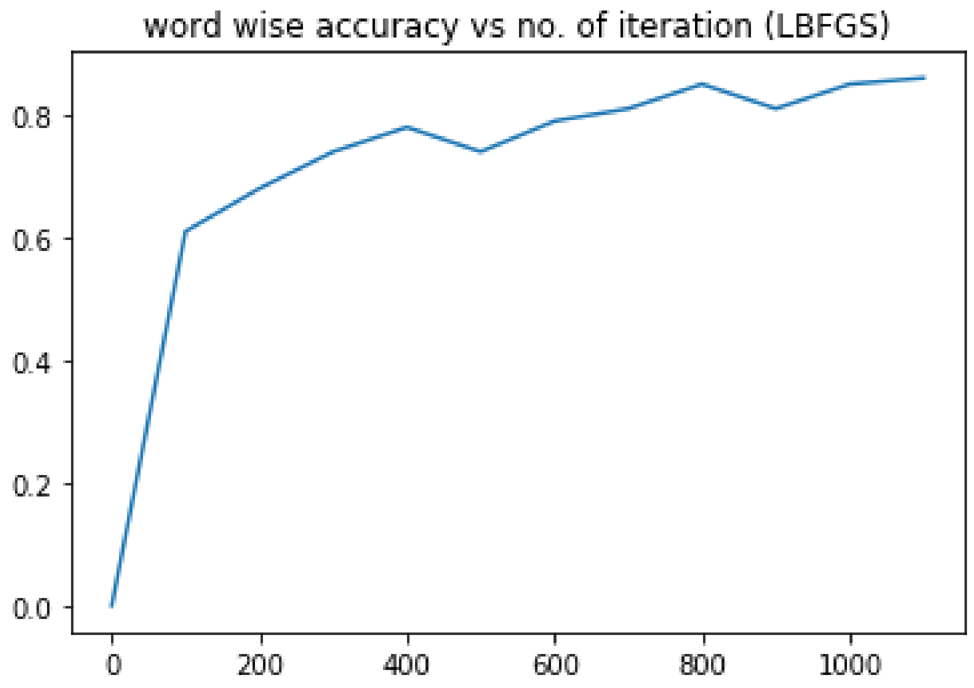

**Figure 5.** Word-wise accuracy in DNN when using L-BFGS optimizer

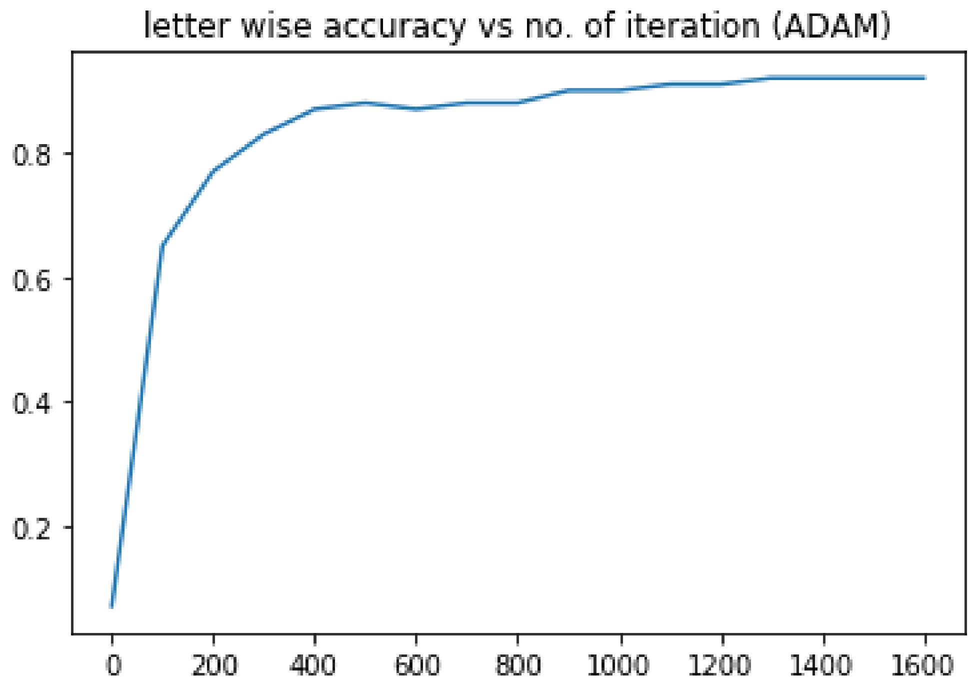

**Figure 6.** Letter-wise accuracy in DNN when using Adam optimizer

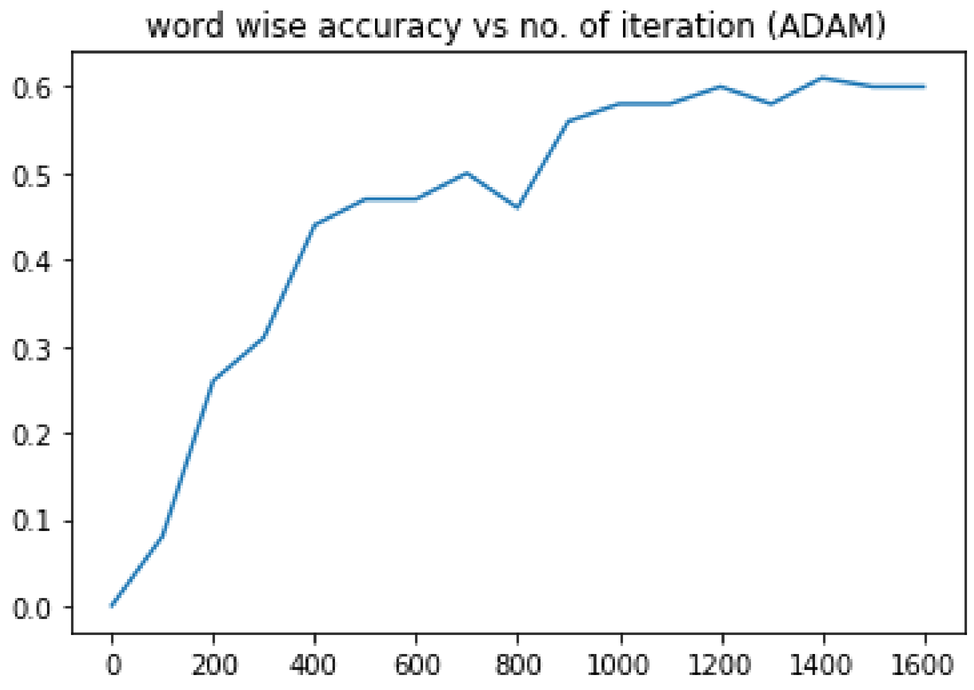

**Figure 7.** Word-wise accuracy in DNN when using Adam optimizer

We based our model on AlexNet becasue it is one of the simplest state-of-art architecture which famously won the 2012 ImageNet LSVRC-2012 competition by a large margin (15.3% VS 26.2% (second place) error rates). We further simplified the version because in ImageNet task we may have different objects in a single picture and the classifier should recognize and classify these objects which is more complex task than this project. Since we only need to identify letters ,we can achieve desirable results with simpler network in much less training time. Moreover, AlexNet's interface in pytorch uses different image sizes and number of channels than our OCR data which is smaller in size and is grayscale. We could preprocess the data to replicate image across 3 channels and scale the image to 224 X 224 , but we thought it's better to design custom architecture which doesn't require preprocessing the data.

We used relu activation function because it improves the training speed to attain same level of accuracy as other activation functions. Among various methods to prevent overfitting we used dropout layers. On increasing dropout from 0.2 to 0.5 , we found it prevented overfitting of data and we attained higher test accuracy. In order to address, internal covariate shift, we used batch normalization layers. One pooling layer was applied to downsample the output from convolution layers which serves two purposes - prevent overfitting and increase computational efficiency.