CS 512: Advanced Topics in Machine Learning **Due: 17:00, April 14, 2019**

Assignment 3: Conditional Random Fields and Convolutions

Student: Your Name(s) *Email: NetID(s)@uic.edu*

Deadline: 5 PM, April 14, 2019

In this project we will continue to explore Conditional Random Fields (CRFs), but we will use additional image level features such as convolutions to aid the training. We will use PyTorch to implement our CRF model and convolutions. You will get a chance to implement an end-to-end machine learning solution to a problem in PyTorch. In the process, you will also pick up tools to do differentiable layer-wise programming, which is common across all the popular deep learning frameworks that exist today.

You must submit the following TWO files on Blackboard:

1. A PDF report with answers to the questions outlined below. **Your report should include the name and NetID of *all* team members.** The LaTeX source code of this document is provided with the package, and you may write up your report based on it.
2. A tarball or zip file which includes your source code. Your code should be well commented. If you changed the format of command line, then include a short readme.txt file that explains how to run your code.

Please submit **TWO files separately**, and do NOT include the PDF report in the tarball/zip.

1 Introduction

Dataset. We will use Taskar's OCR dataset for this project(the same dataset which was used in Assignment2). The dataset description is repeated here again for your convenience.

The original dataset is downloaded from http://ai.stanford.edu/ btaskar/ocr/letter.data.gz. It contains the image and label of 6,877 words collected from 150 human subjects, with 52,152 letters in total. To simplify feature engineering, each letter image is encoded by a 128 (=16*8) dimensional vector, whose entries are either 0 (black) or 1 (white). Note in this dataset, only lowercase letters are involved, *i.e.* 26 possible labels. Since the first letter of each word was capitalized and the rest were in lowercase, the dataset has removed all first letters.

Minibatches. When training a model in PyTorch, it is common to send minibatches of training examples to the optimizer. A minibatch is a small subset of the training data (the size of the minibatch is usually a tunable hyperparameter).

In the starter code that is provided, the `DataLoader` is already setup to process the input dataset as batches. The entire dataset is also divided evenly into training and test data.

Figure 1. Example word image

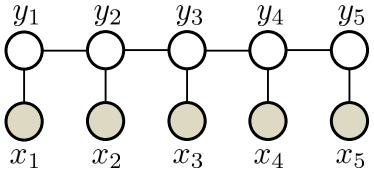

Figure 2. CRF for word-letter

However, when implementing the CRF and Convolution layer, you will have to be mindful of the fact that the input to your module will be a minibatch. The following is the shape of the data after batching.

$$X \in \mathbb{R}^{\text{batch\_size} \times \text{max\_word\_length} \times 128} \tag{1}$$

So every row of a batch corresponds to a word (sequence). Here max_word_length is the maximum word length of all the words in the dataset. For words whose length is less than max_word_length, zero padding is added.

Conditional Random Fields. The CRF model is the same as what was defined in Assignment 2. However, there is a difference in the input that is passed to the CRF model. To recall the details of the OCR dataset - the training set consists of n words. The image of the t-th word can be represented as $X^t = (\mathbf{x}_1^t, \ldots, \mathbf{x}_m^t)'$, where $'$ means transpose, t is a superscript (not exponent), and each *row* of X^t (*e.g.* $\mathbf{x}_m^t$) represents a letter. Here m is the number of letters in the word, and $\mathbf{x}_j^t$ is a 128 dimensional vector that represents its j-th letter image. To ease notation, we simply assume all words have m letters. The sequence label of a word is encoded as $\mathbf{y}^t = (y_1^t, \ldots, y_m^t)$, where $y_k^t \in \mathcal{Y} := \{1, 2, \ldots, 26\}$ represents the label of the k-th letter. So in Figure 1, $y_1^t = 2$, $y_2^t = 18$, $\ldots$, $y_5^t = 5$.

In this assignment, the CRF model will instead take *convolutional features*, given by a function g, which you will implement. The details of the convolution operation is given in Section 3.

Using this (new) notation, the Conditional Random Field (CRF) model for this task is a sequence shown in Figure 2, and the probabilistic model for a word/label pair $(X, \mathbf{y})$ can be written as

$$p(\mathbf{y}|X) = \frac{1}{Z_X} \exp\left(\sum_{s=1}^{m} \langle \mathbf{w}_{y_s}, g(\mathbf{x}_s)\rangle + \sum_{s=1}^{m-1} T_{y_s, y_{s+1}}\right) \tag{2}$$

$$\text{where} \quad Z_X = \sum_{\hat{\mathbf{y}} \in \mathcal{Y}^m} \exp\left(\sum_{s=1}^{m} \langle \mathbf{w}_{\hat{y}_s}, g(\mathbf{x}_s)\rangle + \sum_{s=1}^{m-1} T_{\hat{y}_s, \hat{y}_{s+1}}\right). \tag{3}$$

$\langle \cdot, \cdot \rangle$ denotes inner product between vectors. Two groups of parameters are used here:

- **Node weight:** Letter-wise discriminant weight vector $\mathbf{w}_k \in \mathbb{R}^{128}$ for each possible letter label $k \in \mathcal{Y}$;
- **Edge weight:** Transition weight matrix T which is sized 26-by-26. T_{ij} is the weight associated with the letter pair of the i-th and j-th letter in the alphabet. For example $T_{1,9}$ is the weight for pair ('a', 'i'), and $T_{24,2}$ is for the pair ('x', 'b'). In general T is not symmetric, *i.e.* $T_{ij} \neq T_{ji}$, or written as $T' \neq T$ where T' is the transpose of T.

Given these parameters (*e.g.* by learning from data), the model (2) can be used to predict the sequence label (*i.e.* word) for a new word image $X^* := (\mathbf{x}_1^*, \ldots, \mathbf{x}_m^*)'$ via the so-called maximum a-posteriori (MAP) inference:

$$\mathbf{y}^* = \underset{\mathbf{y} \in \mathcal{Y}^m}{\operatorname{argmax}}\, p(\mathbf{y}|X^*) = \underset{\mathbf{y} \in \mathcal{Y}^m}{\operatorname{argmax}} \left\{ \sum_{j=1}^{m} \langle \mathbf{w}_{y_j}, g(\mathbf{x}^*)_j \rangle + \sum_{j=1}^{m-1} T_{y_j, y_{j+1}} \right\}. \tag{4}$$

2 PyTorch

You will use the popular PyTorch deep learning framework to implement all the algorithms in this assignment. For a comprehensive introduction to PyTorch please refer to this link PyTorch Tutorial.

The above tutorial is a beginner-friendly introduction to the basic concepts of PyTorch. You will need to be comfortable with all the concepts in that link to successfully complete this assignment. In particular, pay attention to the `nn.module` class. This is a standard way computational units are programmed in PyTorch. You will implement the `CRF` layer and the `Conv` layer, in the code, as a subclass of the `nn.module` class. Refer to the starter code for more details.

3 (20 points) Convolution

Different from the previous assignment, we are going to feed in convolutional features of the input image of a letter to the CRF model. Your task is to **implement the convolution layer in PyTorch**. Note that PyTorch implements its own convolution layer (`nn.conv2d`). You are required to provide your own implementation and **NOT** use PyTorch's implementation. However, you may use PyTorch's implementation of convolution as a reference to check the correctness of your implementation.

Convolution operation. Convolution is a commonly used image processing technique, applying various types of transformations on an image. Convolutional Neural Networks (CNNs) employ multiple layers of convolutions to capture fine-grained image features, which are further used downstream in learning several computer vision tasks such as object detection, segmentation etc.

A convolution operation takes in an image matrix X and a filter matrix K and computes the following function as detailed in Eq 9.6 of [GBC]:

$$\hat{X}(i,j) = \sum_{k,l} X(i+k, j+l) K(k,l). \tag{5}$$

(3a) **(20 points)** Implement the `Conv` layer and the `get_conv_features(x)` function, in the starter code. Once convolution is implemented, the CRF's `forward` pass and `loss` functions use the convolution features as inputs (the code for this is set up already).

Your implementation also needs to accommodate different strides, along with an option of zero padding or not.

Testing your implementation. Your implementation of the `Conv` layer will contain the implementation of the convolution operation (similar to `nn.functional.conv2d` of PyTorch). It is crucial to get the implementation of the convolution operation correct first. Consider this simple example of an input matrix X and filter matrix K, with unit stride and zero padding. Report the result of convolving the X with K. You must write this as a test case for the grader to run.

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

Implement your test case inside a file `conv_test.py`. It should run as a standalone test (with all the dependencies, imports in place), and print the result on the screen.

Note. In PyTorch input to conv2d are 4-D tensors - (`batch_size` $\times$ `channel_size` $\times$ `height` $\times$ `weight`) for both the input image X and the filter K. In our dataset, we use a single channel input (`channel_size = 1`).

4 (50 points) CRF

Now, you will (re)-implement the CRF model in PyTorch. Note that this version is designed to use convolutional features and NOT the raw pixels to the CRF model (recall in Assignment 2 we used raw images pixels as the input features $\mathbf{x}_j^t$ which is a 128 dimensional vector). Here, they will be replaced by convolutional features. However, the CRF implementation should remain almost the same; except for changes in the input and output shapes and the fact that it needs to be implemented as a layer (`nn.module`) in PyTorch.

The CRF model is implemented as a `class` in the `crf.py` file in the provided starter code.

(4a) Implement the forward, backward pass and loss inside `crf.py`. This would amount to (1) re-implementing the *inference* procedure using dynamic programming (decoder) (2) dynamic programming algorithm for *gradient computation* (3) loss - which is the negative log-likelihood of the CRF objective. Once again, place holders for all these are provided in the starter code (in the `crf.py` file). This question will be graded through the subsequent questions.

(4b) **(20 points)** Implement and display performance measures on the CRF model - we will use the same performance measures as the previous assignment (1) **letter-wise prediction accuracy**, (2) **word-wise prediction accuracy**. Using a batch size of 64 plot the letter-wise and word-wise prediction accuracies on both training and test data over 100 iterations (x axis). (Place holder provided in the startup code). Use a 5×5 filter matrix for this experiment, and set stride and zero/no padding to optimize the test performance. If it has not converged (function value changes little), you may increase the number of iterations.

(4c) **(20 points)** It is common to use more than one convolution layer in modern deep learning models. Convolution layers typically capture local features in an image. Stacked convolutions (multiple layers of convolutions put one after the other) help capture higher level features in an

image that has shown to aid classification significantly. Repeat experiments in (4b) with the following convolution layers. Set stride and zero/no padding to optimize the test performance.

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

(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.

5 (30 points) Comparison with Deep Learning

Compare your new CRF model, with convolution features, with a convolution based Deep Neural Network (DNN) model of your choice, also known as Convolutional Neural Networks (CNN). You are free to design your own DNN model or pick one of the popular model architectures that have been published. Some popular choices would be,

1. VGG [3]
2. ResNet [1]
3. AlexNet [2]
4. GoogLeNet [4]

The input to the CNN model will of course be the original train and test dataset. You will have to report the following in your report.

(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.

(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.

(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*?

(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.

Bibliography

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[3] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition.

[4] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions.