Assignment 4 – Domain Adversarial Adaptation

Due Date: Dec 03, 2022 Midnight

In this assignment we will implement Domain Adaptation using the Domain Adversarial Network (DANN paper https://arxiv.org/pdf/1505.07818.pdf). We will use the SVHN digit dataset (http://ufldl.stanford.edu/housenumbers/ use the Format 2) as the source domain and the MNIST dataset as the target domain. The source domain (SVHN) is trained with labels and the target domain (MNIST) is trained without labels. We will use MNIST labels only for evaluating the test accuracy. We will be training a single network to classify both the source and target datasets. The classifier is trained with labeled source data and adapted to classify the target images. The `digits.py` file provided with Assignment 1 downloads the MNIST and the SVHN datasets. The MNIST images are 28x28 pixels grayscale (1 channel). The SVHN are 32x32 pixels color images. (3 channels). When you load MNIST dataset, convert the images to 3 channels by copying (repeating) the grayscale channel 2 more times. The source and target images must be the same format (number of channels) even though the size (height and width) may vary by a few pixels. The `digits.py` file does this preprocessing for you.

The network consists of a (1) Feature_Extractor, (2) Label_Classifier and (3) Domain_Classifier. The Label_Classifier is used to classify the source and target digits. The Domain_Classifier is trained to distinguish between source and target domains. The Feature_Extractor extracts features from the source and target images and feeds them to the Label_Classifier and the Domain_Classifier. The architecture of the neural network is as follows:

Convolution Layers are represented as 'Conv(ch)', where 'ch' is the number of output channels. All convolutions are same convolutions with stride=3 and padding=1 Linear layers are represented as 'Linear(fts)', where 'fts' is the number of output features. Every Convolution and Linear Layer is followed by ReLU activation except if it is the last layer before the Loss function. MaxPool layers are represented as 'MaxPool(fs)' where 'fs'=2 is the filter size with stride=2 BatchNorm layers are represented as 'BatchNorm(ch)', where 'ch' is the number of channels or inputs feature dimensions. For Convolution layers we use 'BacthNorm2d(ch)' and for linear layers we use 'BatchNorm1d(ch)'

FeatureExtractor (Input = Batch_Size x 3 x 32 x 32 or Batch_Size x 3 x 28 x 28):
Conv(32) → Conv(32) → MaxPool(2) →BatchNorm(32) → Conv(64) → Conv(64) → MaxPool (2) → BatchNorm(64) → Conv(128) → Conv(128) → AdaptiveAvgPool2d(1) → Linear(128) → BatchNorm(128)

LabelClassifier (Input = Batch_Size x 128):
Linear(64) → BatchNorm → Linear(64) → SoftmaxCrossEntropyLoss

DomainClassifier (Input = Batch_Size x 128):
Gradient_Reverse_Layer() → Linear(64) → BatchNorm → Linear(64) → BatchNorm → Linear (64) → SigmoidBinaryCrossEntropyLoss

The Gradeint_Reverse layer is used to train the Feature_Extractor to align the source and target features. We will add it at the beginning of the Domain_Classifier. In the forward pass it does not manipulate the data. In the backward pass, it will multiply the gradient by a negative/minus (-) sign to reverse the gradient to the Feature_Classifier. Here is an implementation in PyTorch.

```
class GradReverse(Function):
    @staticmethod
    def forward(ctx, x, lamda):
        ctx.lamda = lamda
        return x.view_as(x)

    @staticmethod
    def backward(ctx, grad_output):
        output = (grad_output.neg() * ctx.lamda)
        return output, None
```

Lamda is a constant that controls the importance of the Domain_Classifier loss. In the initial stages of training we set it to 0 so as to ensure the Label_Classifier gets trained. We will gradually increase its value using the following schedule as the training progresses. Refer to Sec 5.2.2 in the DANN paper for more details.

$$\lambda = \frac{2}{1 + \exp(-\gamma . p)} - 1$$

where, γ=10 and $p = \frac{iteration + epoch*no\_iterations\_per\_epoch}{no\_epochs*no\_iterations\_per\_epoch}$ is the fraction of total iterations completed in the training.

Initially train the Feature_Extractor and Label_Classifier using the labeled source data (SVHN). Then, use this pretrained network to classify target (MNIST) and verify the accuracy. You will get a relatively poor performance which we will improve using Adversarial domain adaptation.

Now, apply Adversarial Domain Adaptation using the Domain Classifier. This will train the Feature Extractor to align the source and target features. Evaluating the MNIST test data with this Domain aligned Feature_Extractor + Label_Classifier should give you better accuracies. The accuracy should improve by 10% points.

Submission Format and Grading:

1. Implement the assignment in Python and submit a Jupyter Notebook file with the following name format `Assignment4_FirstName_LastName.ipynb`
2. Convert your notebook to pdf after saving it with all the outputs and save the file as `Assignment4_FirstName_LastName.pdf`
3. Do not save images of your code in the pdf file. The pdf file has to run through the plagiarism check and code needs to be in text format, not image.

4. Implementation of Feature_Extractor (10 pts)
5. Implementation of Label_Classifier (10 pts)
6. Implementation of Domain_Classifier (10 pts)
7. Accuracy on Source (SVHN) test data using classifier trained on Source labeled data (no domain adaptation) is > 90%. Make a note of Target (MNIST) data accuracy with this classifier. (20 pts)
8. Implementation of Lamda schedule (20pts)
9. Accuracy of Target (MNIST) data after domain alignment is improved by at least 10% points. (30 pts)

Your submissions will be executed to verify your solutions. The code and report will be evaluated for plagiarism. Ensure the notebook will execute on a generic Google Colab environment without the need for change/debugging. If it is not possible to execute your code, it is not possible to verify your results. The submission will also be evaluated for the quality of images in each of these outputs.