Computer Vision Assignment 1

Saurabh Vaishampayan
svaishampaya@student.ethz.ch
21-950-977

October 2021

1 Notes on code additions

The places where the code for the relevant question is implemented is given as follows for ease of examination as line numbers in relevant files:

1. Question 2: 2D classifier
    - Question 2.1:
        (a) Reading .npz file: L19 to L23, L51 to 55 in dataset.py
        (b) get item method: L32, L33, L64, L65 in dataset.py
    - Question 2.2: Linear Classifier: L16 in networks.py
    - Question 2.3: Training: L23 to L34 in train.py
    - Question 2.4: MLP: L37 to L39 in networks.py. For running, uncomment L86 in train.py
    - Question 2.5: Feature engineering: L78 in dataset.py. For running uncomment L65 and L76.
2. Question 3:
    - Question 3.1: Normalisation: L68 in dataset.py
    - Question 3.2: Training: L25 to L36 in train.py
    - Question 3.3: Multilayer perceptron:
        (a) For Linear Network: Uncomment L18 and Comment L21 to L24 in networks.py. Uncomment L83 and comment L84 in train.py
        (b) For MLP: Comment L18 and uncomment L22 to 25. Uncomment L83 and comment L84 in train.py
    - Question 3.4: CNNs: L46 to L57 in networks.py. Uncomment L84 in train.py
    - Question 3.6: Confusion matrix: L42 to L58 in plot_confusion_matrix.py

2 Simple 2D Classifier

2.1 Linear Classifier

As part of question 2.2 we were asked to implement a Linear Classifier to classify a 2D input into 2 categories. Upon training the classifier on the given data for a binary cross entropy (BCE) loss, we obtained the following results, after 10 training epochs:

- BCE loss on training data: 0.6931
- Accuracy on validation data: 47.8175 percent

In both the train and validation dataset, we found that the fraction of points belonging to cluster 1 are 0.5. So from the accuracy results (which are close to 0.5) we can conclude that the linear classifier is not able to separate the points belonging to cluster 1 and 0. This is because the distribution is determined by the radial distance of points and is rotationally invariant. Hence no hyperplane will be able to separate the points into the 2 clusters and the linear classifier is no better than random allocation of points to either of the clusters.

2.2 Multilayer perceptron

As part of question 2.4 we are asked to implement and train a multilayer perceptron (MLP) for classifying the data. The MLP had 2 hidden layers of 16 neurons each with a ReLU nonlinearity. We obtain the following results for accuracy and BCE loss after 10 training epochs:

- BCE loss on training data: 0.0421
- Accuracy on validation data: 99.8016 percent

We had earlier explained why the linear classifier struggles in this particular classification task. The MLP succeeds because it allows us to transform the feature space during training because of the presence of nonlinearities. The feature space transformation is done by the network automatically during training phase. The distribution is determined by radial distance of points and is rotationally invariant. MLP can optimize the decision boundary to separate the 2 clusters by virtue of nonlinearities, while Linear Classifier struggles since it can only find a hyperplane decision boundary.

2.3 Feature engineering

Observing that the distribution of points into categories seems to be determined by the radial distance, we propose the coordinate system change:

$$\begin{aligned} & (x, y) \rightarrow (r, \theta) \\ \text{where} \quad & r = \sqrt{x^2 + y^2} \\ & \theta = atan(\frac{y}{x}) \end{aligned}$$

We implemented *atan* using the **angle** function from **numpy** which calculates the phase taking into account which quadrant the point lies in. Here (x, y) are first and second coordinates of the given 2D dataset. We transform the dataset using the above feature transform and then use a linear classifier for classification. We obtain the following accuracy and BCEloss metrics (see note below on reproducibility and number of epochs). We have averaged the below scores for 10 different runtimes:

- BCELoss on the training data: 0.4931
- Accuracy on validation data: 79.37 percent

Upon implementation of this we observed that the loss and accuracy vary a lot each time we run. This is because of initialised weights are random. We expect close to zero weight for θ (phase of the data point) in the optimized network due to rotation invariance. But because the network starts with random weights, it may take more epochs for the weightage for θ (phase) to go to 0 and this greatly affects the performance, since we are only training for 10 epochs. This is the reason we get different results on different iterations. **When the number of epochs are increased above 10, we observed that the final "steady state" accuracy metrics are roughly initialisation independent.** This is also the reason why one may end up with lesser accuracy with this feature space engineering w.r.t MLP if one only observes for 10 epochs. When we changed the number of epochs to 100, both networks achieve near perfect accuracy with the feature engineering beating the MLP.

3 Question 3

3.1 Question 3.3

1. Performance metrics for Linear Classifier:
   - Cross entropy loss on training data: 0.3541
   - Accuracy on validation data: 90.02 percent
2. Performance metrics for Multilayer Perceptron with a single hidden layer of dimension 32 and ReLU nonlinearity.
   - Cross entropy loss on training data: 0.3570
   - Accuracy on validation data: 90.41

We observe that MLP and Linear Classifier both give similar performance with an accuracy close to 90 percent. This indicates that this high dimensional (784 pixel values) data is reasonably well approximated by hyperplane decision boundaries.

3.2 Question 3.4

Performance metrics for Convolutional Neural Networks:

- Cross entropy loss on training data: 0.0764
- Accuracy on validation data: 97.6402

The convolutional neural network performs significantly better than the Multilayer perceptron (error rate is 1/4 of MLP). This is because CNNs take advantage of the spatial structure of "natural" image data, and multiple layers of convolutions allow a large receptive field enabling structured and increasing order of feature identification. MLPs have much larger number of parameters and can be prone to overfitting. We have compared the number of parameters of the two networks in the next question.

3.3 Question 3.5

Comparison of number of parameters. The image size is 28 x 28 with one channel (because grayscale image). We have to classify the image as one of the digits, so the output layer should have 10 neurons.

3.3.1 MLP

Consider number of parameters for a neural network with n_i input nodes and n_o output nodes. Each input node is connected to every output node by a weight parameter. Every output node has a bias associated with it.

$$\begin{aligned} \text{Number of weights} &= n_i \times n_o \\ \text{Number of biases} &= n_o \\ \text{Parameters} &= n_o \times (n_i + 1) \end{aligned} \tag{1}$$

We have 28 x 28 inputs and 32 neurons in hidden layer. We have 10 neurons in output layer. Therefore number of parameters N_p is

$$N_p = 32 \times (28 \times 28 + 1) + 10 \times (32 + 1) = 25450$$

Hence number of parameters is equal to 25450. We also verified by counting number of differentiable parameters (gradient=True) in the pytorch model.

3.3.2 CNN

As seen in lecture, if the input layer is of dimensions $W_i \times H_i \times C_i$, Kernel size is $K \times K$ and output has C_o channels then the number of parameters (including biases) is given by

$$N = C_o(K \times K \times C_i + 1)$$

If padding is P and stride is s then output dimensions is given by:

$$W_o = \lfloor \frac{W_i + 2P - K}{s} + 1 \rfloor$$
$$H_o = \lfloor \frac{H_i + 2P - K}{s} + 1 \rfloor$$

After max pooling with stride s and kernel size K, the output image has dimensions:

$$W = \lfloor \frac{W_o + -K}{s} + 1 \rfloor$$
$$H = \lfloor \frac{H_o + -K}{s} + 1 \rfloor$$

We calculate the number of parameters for our system below. For all convolution operations we have zero padding and stride of 1. No trainable parameters involved in maxpooling, although image dimension is changed:

1. Conv2d with output channels = 8 and kernel size of 3. Input is 28 x 28 x 1.
   - Number of parameters = $8 \times (3 \times 3 \times 1 + 1) = 80$
   - Image size = $(28 - 3 + 1) \times (28 - 3 + 1) = 26 \times 26 \times 8$
2. MaxPooling with stride = Kernel size = 2.
   - Image size = $(\frac{[26-2]}{2} + 1) \times (\frac{[26-2]}{2} + 1) = 13 \times 13 \times 8$
3. Conv2d with output channels = 16 and kernel size of 3. Input is 13 x 13 x 8.
   - Number of parameters = $16 \times (3 \times 3 \times 8 + 1) = 1168$
   - Image size = $(13 - 3 + 1) \times (13 - 3 + 1) = 11 \times 11 \times 16$
4. MaxPooling with stride = Kernel size = 2.
   - Image size = $(\lfloor \frac{[11-2]}{2} + 1 \rfloor) \times (\lfloor \frac{[11-2]}{2} + 1 \rfloor) = 5 \times 5 \times 16$
5. Conv2d with output channels = 32 and kernel size of 3.
   - Number of parameters = $32 \times (3 \times 3 \times 16 + 1) = 4640$
   - Image size = $(5 - 3 + 1) \times (5 - 3 + 1) = 3 \times 3 \times 32$
6. Now Adaptive maxpooling changes it into 1 x 1 x 32 image.
7. Finally we have a neural layer with 32 inputs and 10 outputs. Hence number of parameters $= 10 \times (32 + 1) = 330$.

Adding them all up (note that maxpooling does not have trainable parameters, we only used it to obtain image sizes), we have Number of Parameters = 80+1168+4640+330 = 6218. This matches with the length of the list of differentiable parameters (gradient = True) in the pytorch model.

3.4 Question 3.6

Plots of confusion matrices for MLP and CNN.

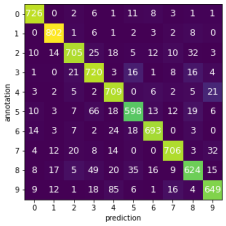

Figure 1: Confusion matrix for Multilayer Perceptron

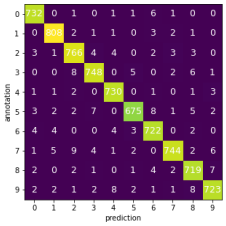

Figure 2: Confusion matrix for CNN

3.4.1 Observations

The sum of off diagonal entries divided by total entries is the error rate. So higher count for off diagonal entries indicates more error. We can see that for CNNs, the misclassification counts (off diagonal counts in the confusion matrix) are lower than MLP. This is because CNNs take advantage of the spatial structure of "natural" image data, and multiple layers of convolutions allow a large receptive field enabling structured and increasing order of feature identification. MLPs have much larger number of parameters and can be prone to overfitting. We have compared the number of parameters of the two networks in the previous question. This explains the better performance of CNNs.