Computer Vision Assignment 1

Nicolas Wicki

October 2021

1 Environment setup

All good.

2 Simple 2D classifier

2.1 Dataset

The last line of comment of the class *Simple2DDataset* requests the following: *# Samples should be an* `NxNx2` *numpy array. Annotations should be* `Nx1`. While you could actually reshape the y-vector to `Nx1` (using the `.reshape` functionality of numpy), we do not actually have to do anything here, since a simple array is sufficient.

2.2 Linear Classifier

All good.S

2.3 Training Loop

Running the script for 10 epochs leads to a converging accuracy at around 48%. This is more or less expected. Achieving a high expressiveness in a clustering task using only a linear classifier is really hard or even impossible. We have no non-linearity built into our layer, and it therefore only allows us to divide the space linearly.

2.4 Multi-layer perceptron

The accuracy achieved using the multi-layer perceptron network using again 10 epochs is around 99.6%. The results are much better since the non-linearity helps to separate clusters inseparable by lines.

2.5 Feature transform

To transform our samples, we use a second-degree polynomial mapping as follows: $\phi(x, y) = (x^2, \sqrt{2}xy, y^2)$. This way we create a higher dimensional space where a linear separation might be possible. For this we adjust the `transform function` and also adjust the input dimension of our linear classifier network. Running the model for 10 epochs then delivers an accuracy of around 88%. The accuracy certainly improved, and it shows that the applied transformation helps to separate the data points.

3 Digit classifier

3.1 Data normalization

All good.

3.2 Training loop

All good.

3.3 Multi-layer perceptron

Running the `MLPCLassifier` for 5 epochs, we achieve an accuracy of 84.2%. Adding a hidden layer with the non-linear activation by `ReLU`, we get an accuracy of around 94.35%.

3.4 Convolutional Network

This architecture achieves an accuracy of around 98% when training for 5 epochs.

3.5 Comparison of number of parameters

We base our computation on the formulas presented on slide 135 of lecture 3. Activation layers are omitted since they do not influence the number of parameters.

MLP For this the network architecture looks as follows:

1. Input Layer: 28×28 input neurons and 1 bias
2. Fully Connected Layer: 32 neurons and 1 bias
3. Output Layer: 10 output neurons

The formula for the number of parameters for fully connected layers is given by: $W_{out} * H_{out} * C_{out} * (W_{in} * H_{in} * C_{in} + 1)$. This then gives us $32 * 1 * (784 * 1 + 1) = 25120$ parameters from the input layer to the hidden layer and $10 * 1(32 * 1 + 1) = 330$ parameters from the hidden layer to the output layer. Totally, we have $25120 + 330 = 25450$ parameters to learn.

CNN For this the network architecture looks as follows:

1. Input Layer: 28×28 input neurons and 1 bias
2. Convolution Layer: 8 kernels of size 3×3
3. Max-pooling Layer: 2×2 kernel and stride of 2
4. Convolution Layer: 16 kernels of size 3×3
5. Max-Pooling Layer: 2×2 kernel and stride of 2
6. Convolution Layer: 32 kernels of size 3×3
7. Adaptive Average-pooling Layer: Pools image down to size $1x1$ and keeps 32 feature channels
8. Output Layer: 10 output neurons

The formula for the number of parameters for fully convolutional layers is given by: $C_{out}*(K_w*K_h*C_{in}+1)$. Since convolutions and max-pooling layers influence the layer size, we will first compute all layer sizes as given by the formula described on slide 165 of lecture 3: $(\lfloor \frac{W_{in}+2P-K_w}{s} \rfloor + 1) * (\lfloor \frac{H_{in}+2P-K_h}{s} \rfloor + 1) = W_{out} \times H_{out}$ where $W_{in/out}$ is the input/output width, $H_{in/out}$ the height of the input/output, P the size of the padding, K_w the width of the kernel, K_h the height of the kernel, and s the size of the used stride.

1. Input Layer: $28 \times 28 = 784$ input neurons
2. Convolution Layer: $(\lfloor \frac{28+2*0-3}{1} \rfloor + 1) \times (\lfloor \frac{28+2*0-3}{1} \rfloor + 1) = 26 \times 26$, with 8 channels we get: $26 \times 26 \times 8$
3. Max-pooling Layer: $(\lfloor \frac{26+2*0-2}{2} \rfloor + 1) \times (\lfloor \frac{26+2*0-2}{2} \rfloor + 1) = 13 \times 13$, with 8 channels we get: $13 \times 13 \times 8$
4. Convolution Layer: $(\lfloor \frac{13+2*0-3}{1} \rfloor + 1) \times (\lfloor \frac{13+2*0-3}{1} \rfloor + 1) = 11 \times 11$, with 16 channels we get: $11 \times 11 \times 16$
5. Max-Pooling Layer: $(\lfloor \frac{11+2*0-2}{2} \rfloor + 1) \times (\lfloor \frac{11+2*0-2}{2} \rfloor + 1) = 5 \times 5$, with 16 channels we get: $5 \times 5 \times 16$
6. Convolution Layer: $(\lfloor \frac{5+2*0-3}{1} \rfloor + 1) \times (\lfloor \frac{5+2*0-3}{1} \rfloor + 1) = 3 \times 3$, with 32 channels we get: $3 \times 3 \times 32$
7. Adaptive Average-pooling Layer: $1 \times 1 \times 32$

8. Output Layer: 10 output neurons

Now, to compute the number of parameters we can simply put the calculated layer sizes into the above formulas:

1. Input Layer - Convolution Layer: $8 * (3 * 3 * 1 + 1) = 80$
2. Max-pooling Layer - Convolution Layer: $16 * (3 * 3 * 8 + 1) = 1168$
3. Max-pooling Layer - Convolution Layer: $32 * (3 * 3 * 16 + 1) = 4640$
4. Adaptive Average-pooling Layer - Output Layer: $1*1*10*(1*1*32+1) = 330$

In total, we arrive at the following number of parameters to be learned: $80 + 1168 + 4640 + 330 = 6218$. This is considerably less than the 25k parameters we had to learn to use the MLP classifier. Note that the steps from convolution layers to max-pooling layers does not involve learning any parameters, and neither does the application of the activation functions. Also, the layer sizes are mostly irrelevant, since the amount of parameters in convolution layers is only dependent on channels and kernel sizes.

3.6 Confusion Matrix

As depicted in Figure 1, we can see that most predictions were correct, which is also identifiable by a strong tendency towards a purely diagonal matrix. These results are based on the previously mentioned basic CNN architecture trained for 5 epochs and a validation accuracy of 98%. Certainly better accuracy and an even stronger tendency towards a diagonal matrix could be achieved by training for more epochs.

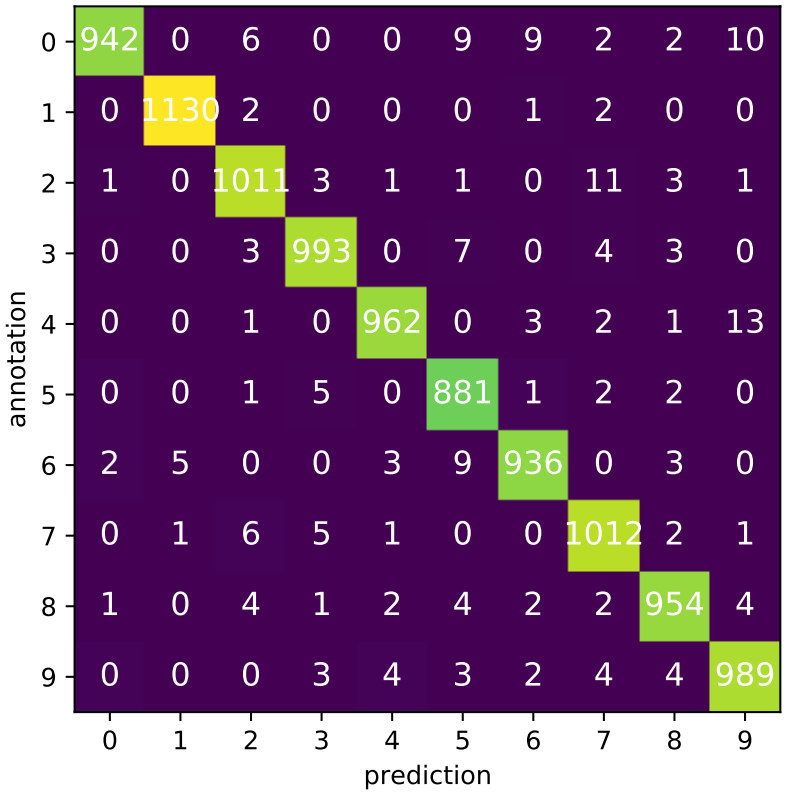

Figure 1: Confusion matrix where entry (i, j) represents the number of predictions with class i as the result and j being the actual ground truth class.