Homework problem 4 – Classifiers

Deliverable: One document (MS Word or PDF) containing the answers to all questions and the figures specified in the task assignments below. You should not include all figures produced by the code. You do not have to include the modified code.

Part 1: Using MATLAB to build a classifier that identifies hand-written digits

In the first part of this exercise, we will explore MATLAB's ANN functionality. We will use an image data set provided by MATLAB similar to the MINST data set of hand-written digits. Our goal is to build a CNN that can classify these images with high accuracy.

Run Section 1 of the code ("NTP640_HW4_Classifiers_Part1.m"). You will need MATLAB's Deep Learning Toolbox and Image Processing Toolbox to run the code successfully. Those are available with the UW software library MATLAB installation.

analyzeNetwork(layers)
Outputs a schematic representation of the CNN we defined with the "layers" variable.

While training the network, you should also be able to see a MATLAB window labeled "Training Process."

Q_1.1: What are the receptive field sizes of the three convolution layers? How many filters do we have in each convolution layer? Compare the dimensions of the subsequent ReLu layers and their total number of neurons (include the numbers in your document). What is going on there? What does it mean that the total number of neurons in the Relu layers is decreasing? (5 pts)

The network is trained via stochastic gradient descent with momentum (SGDM). SGD evaluates the loss only at a random subset of data samples from the full training set, called a "mini-batch", instead of all the training samples. SGD often performs better, as the random sampling avoids overfitting and getting stuck in potential local minima of the loss function. The momentum can dampen oscillations in the gradient descent and can thus lead to faster convergence.

Now run Section 2 of the provided code.
Task 1.1: Include the figure of the filter weights of the first convolution layer in your document. (1 pt)
Task 1.2: Select a different input image out of the 10,000 examples. Run Section 2 again. Include the figure that outputs the three rows of 8 images. What are the different rows? (Look at the code). What explains is the difference between the second row and the third row? (3 pts)

Q_1.2: Are any of the filters in the first convolution layer interpretable to you? See if you can explain one of the activation images by the properties of one of your filters. (1 pt)

Q_1.2: Include the figure that shows the activations of the fully connected layer. What does the output of the fully connected layer represent? Does the output fit your input image? (1 pt)

Task 1.3: Record the final training accuracy of the original network in your document. Now delete the 3rd convolution layer (plus the associated batchNormalization and reluLayer). Train the new network and note down the accuracy. Now delete the second convolution layer also (plus its batchNormalization and reluLayer). Train again and note down the accuracy. Restore the network to the original network, but then take out all reluLayers. Record the accuracy of this linear network. What did you observe? Do you think these numbers are representative? Why?

(If you are not sure you can train the network multiple times to see how much the training accuracy varies with each training run). (5 pts)

(Total points part 1: 16)

Part 2: Comparing a normative encoding model to real V1 neurons

In this task, we will build and train an artificial neural network for orientation discrimination. The network should be able to tell whether a given grating stimulus is tilted to the "right" or "left"; that is, whether its angle relative to the vertical is positive or negative, respectively. We will compare the representations of our optimized encoding model to neural activity recorded in response to these very same stimuli, courtesy of Stringer et al 2019 (https://doi.org/10.1016/j.cell.2021.03.042).

We will use a CNN that performs two-dimensional convolutions on the raw stimulus image (which is a 2D matrix of pixels). The particular CNN we will use here has three layers of import:

1. a *convolutional layer*, which convolves the images with a set of six filters
2. a *pooling layer,* which combines the information of the convolutional filters
3. a *fully connected layer*, which transforms the output of this convolution into a 10-dimensional representation

Different from the model in Part 1, here we will initialize the convolution layer with a set of six predefined filters that correspond to idealized neural receptive fields as found in LGN or V1 (i.e., center-surround and Gabor filters).

The first section of the code plots example stimuli and creates the six filters (2 center surround and 4 Gabor filters) with which we will initialize the CNN. Gabor filters have a positive region next to a negative region and the orientation of these regions of the filter determine the orientation of edges to which they respond. Simple cells in V1 have receptive fields that resemble Gabor filters.
Next, we will define the layers of the CNN and create 1000 training stimuli. Note the “Weights” input to the convolution layer.

Q_2.1: How many neurons (also called “activations”) are in each convolution channel? How many weights does the convolution layer have? How many weights would it have if it were a fully connected layer? (3 pts)

In the second section of the code, we will train the network, plot the final weights of the convolution layer filters, and also plot the activations of the convolution layer for five sample orientations.

Q_2.2: Look at the activations for the different filters. What do the first two filters seem to respond to? What do filters 3-6 seem to respond to? (2 pts)

Task 2.1: Copy the output of the network training into your document. (1 pt)

In section 3 of the code, we will compare individual activations of the pooling layer and the fully connected layer to the responses of real V1 neurons. We will also compute the representational dissimilarity matrices (RDMs) of the V1 data and the layer activations and compare their pairwise correlations. The idea is that we can say that a brain area and a model use a similar representational scheme if stimuli that are represented (dis)similarly in the brain are represented (dis)similarly in the model as well.

Q_2.3: What is the size of the “hidden_activity_pool” variable and why? (1 pt)
Q_2.4: In which way do the pooling activations resemble V1 responses? What are the FC layer neurons selective for? (2 pt)
Q_2.5: What are the resulting correlation coefficients between the RDM’s of V1 neurons and the two model layers? Which layer’s representation scheme is closer to that of V1 neurons? (2 pt)

Task 2.2: Copy the figure comparing the activations of V1 neurons with the pool and FC layer and the figure with the three RDMs into your document. (2 pts)

Task 2.3: Change the code so the network is initialized with random weights instead of the predefined filters. Train the network again and copy the training output into your document. Compare the mini-batch accuracy of the first and last epoch with the previous training output. Explain. (3 pts)

Task 2.4: Include a figure of the six filters resulting from random weight initialization in your document. (1 pt)

Task 2.5: Now compare the pooling layer and FC layer of this CNN to V1. Include all figures resulting from rerunning section 3 of the code with the updated CNN. What are the correlations between the V1 RDM and model RMDs now? What can you conclude from those values compared to the previous correlation values? (4 pts)

(Total points part 2: 21)