Probabilistics Models of Visual Cortex

Homework 2

Feitong Yang

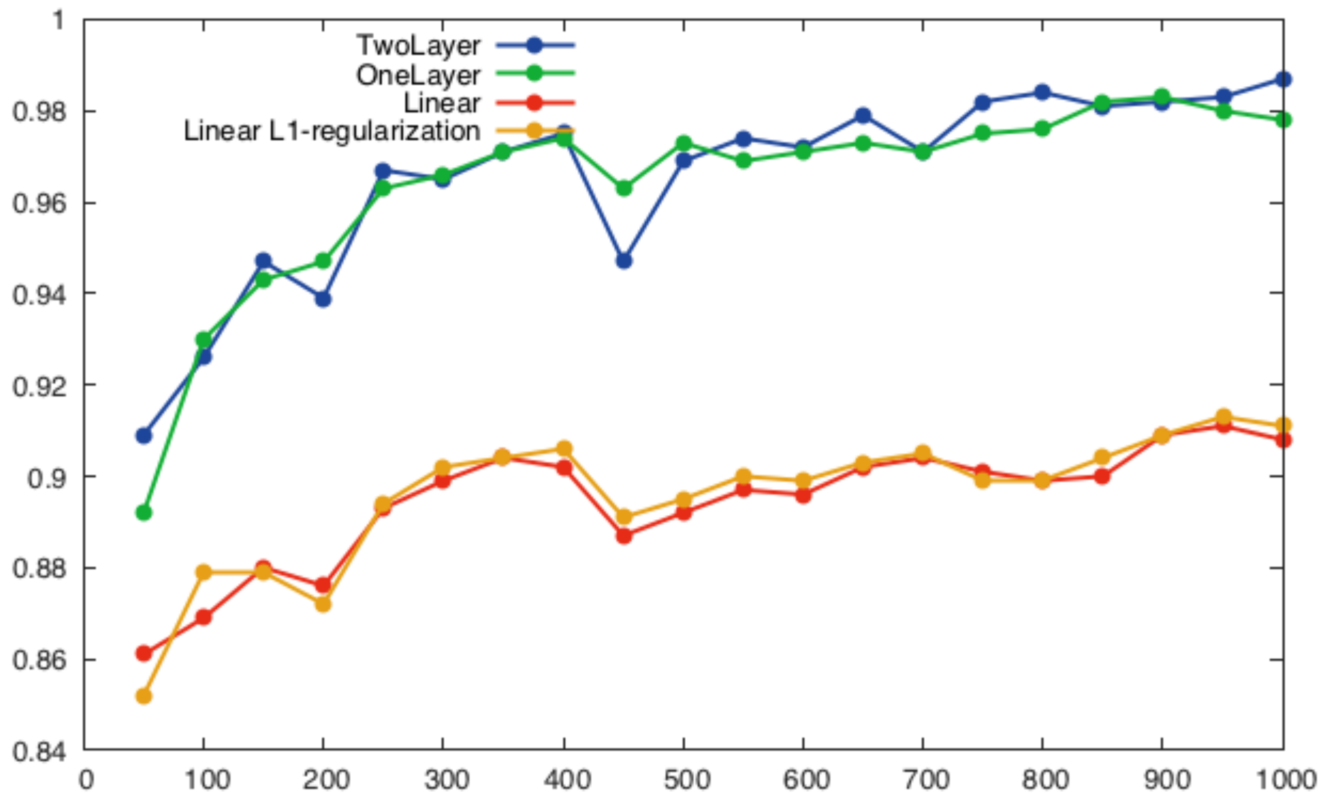
Department of Psychological and Brain Sciences

1 Comparing network architectures

Please refere to the code attached below, for both the architectures of models and the training process in `train.lua`

2 Comparing performance

The learning curves looks like



> Question: Comment on the plot above. Why does it look how it does? Think about both the final accuracy the model achieves and the dynamics of how it gets there.

Look at the learning curve, we can see all models gradually improve the performance in classification, just like human does. However, there are some small flucturations in accuracy. However, when a person take the same test again and again, she won't make the same mistakes, because she can learn from the test. However, in the experiment, the classifiers were tested on the same test dataset, but their performances do not monotonously increase strictly. This is because we assume a classifier does not learn anything in the test dataset, and forget about it after it is tested. So even though a classifier was tested on the test dataset, whenever it looks at the test dataset, it treat dataset as if it has neven seen it before. So the flucturation is reasonable.

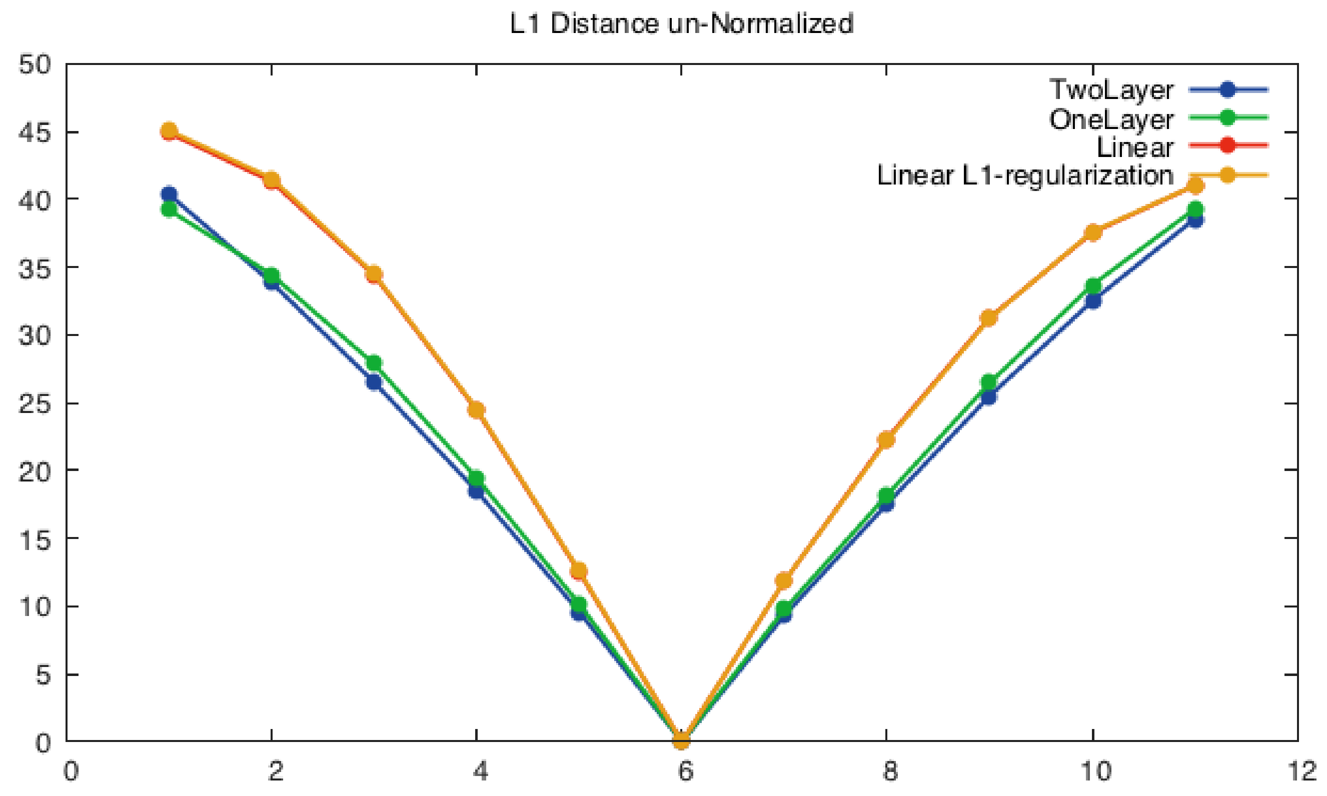
The longer the training, the more images are seen by the classifier. Each iteration, a model see 100 random images, and the model learns from these images in a supervised way. The more images a model sees, the better it can understand the "general properties" of a letter regardless of its specific shape. By learning these "general properties", the model can recognize an unseen letter in the future, which results in improved recognition performance in testing dataset. However, sometimes, the model may overtrained itself in a specific training set, and overfits its weights basedon on such training set, thus cannot be generalized to other dataset. In this case, the model may score a low accuracy when tested on an independent testing dataset. This may be a reason fo the performance dip in the figure, because all three models have a performance dip at around 450 iterations, and regain the performance afterwards.

Comparing different models, we can see that the 2-layer and the 1-layer CNN perform better than the linear classifier. The 2-layer CNN seems to be better than the 1-layer because it performance is better in the begining and reached a higher accuracy in the first 50 iterations. This suggests that the 2-layer CNN learns faster at the begining. However, the later performanes of 2-layer CNN and the 1-layer CNN does not differ from each other very much. Their performances, however, have already reached 98% accuracy, and there is not much room to improve

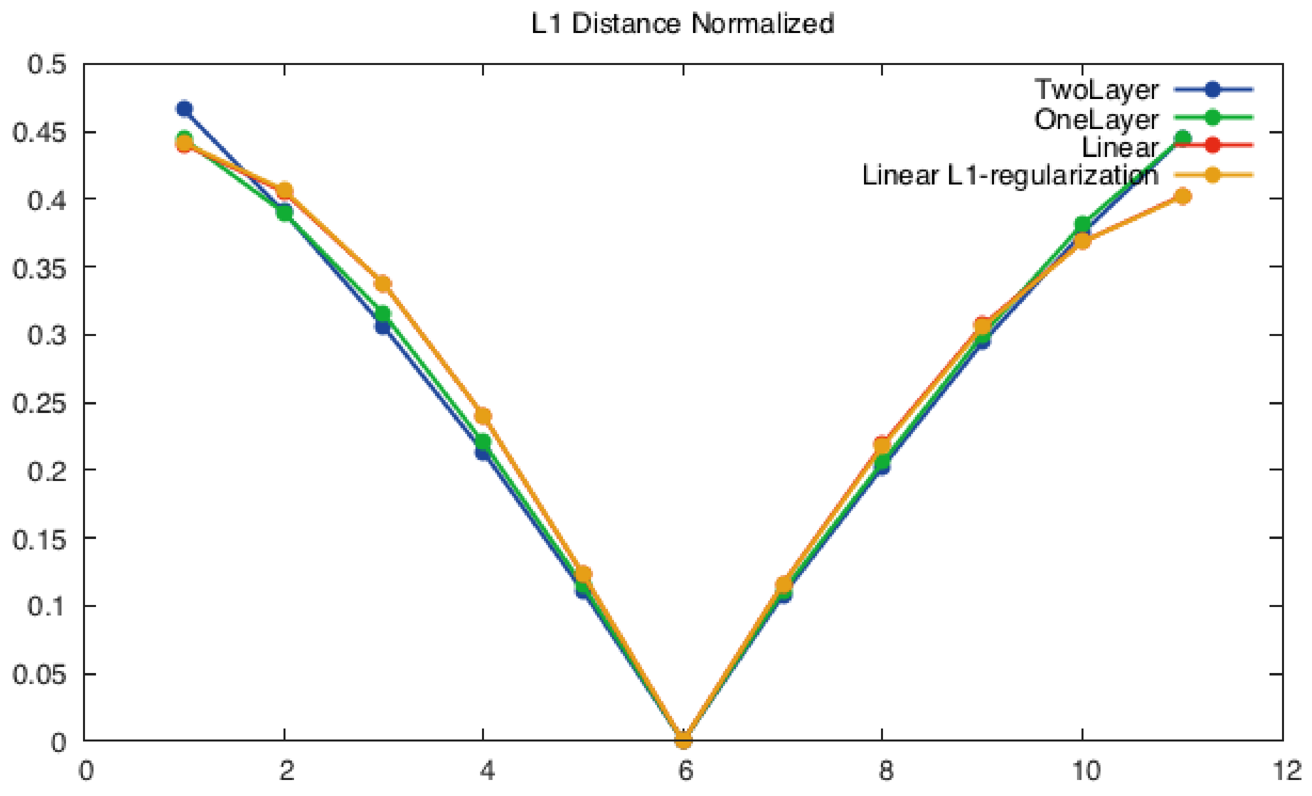
3 Comparing Invariance

Code, please refer to `analysis.lua`

Invariance performance looks like



after the normalization, it looks like



> Question: When you compare the invariance data for the three networks, what do you observe? Is it what you would expect from the networks' structure? How would you go about building a network with more invariance?

When comparing three models, I found that, for the raw score distance, the 2-layer and 1-layer CNN has smaller distances in different translation conditions than the linear classifier. However, when I normalized the distance, the performances of these three models look quite similar, which is unexpected.

Because the 2-Layer and 1-Layer CNN has the max-pooling process, it should provide better translation invariance for the model. So, I would expect the 2-Layer CNN has the lowest translational distance (benefited from its two max-pooling layer), and the 1-Layer CNN has slightly higher distance, and the classifier has the highest distance. This is what we observed in the results before we normalize the score vector.

Also, because the linear classifier lacks of the max-pooling part, we would expect its score distane increases linearly as the translation pixel increases. In contrast, the score distances of CNNs should gradually plateau, and become sublinear pattern as the translation pixel increases. This is also not observed in the figure, which is weird.

There is, however, a slightly smaller distances for the CNNs when they are compared with the linear classifier, but I would expect a larger difference.

To gain more invariance, we can have multiple layers of max-pooling process. This is similar to have a neuron at a higher layer to look at a large area of the original image because it pools the information from other neurons in a lower layer. Similarly, in human visual cortex, a neuron in higher visual area has a larger receptive field, and such a neuron can be activated by a stimulus in a large area in the visual field, regardless of its location. That is, such a neuron has a better translational invariance property.

Code

`train.lua`

```lua
require "nn"
require "optim"
require "batches"

function initialize_model_1()
    --container
    model = nn.Sequential()
    --first layer
    ----Convolution
    model:add(nn.SpatialConvolutionMM(1, 32, 5, 5))
    ----Nonlinearity
    model:add(nn.ReLU())
    ----Max-pooling
    model:add(nn.SpatialMaxPooling(2, 2))
    --Second layer
    ----Convolution
    model:add(nn.SpatialConvolutionMM(32, 64, 5, 5))
    ----Nonlinearity
    model:add(nn.ReLU())
    ----Max-Pooling
    model:add(nn.SpatialMaxPooling(2, 2))
    ----Reshape Results from the second layer
    model:add(nn.Reshape(64 * 5 *5))
    --Output layer: linear classification
    model:add(nn.Linear(64 * 5 * 5, 10))

    return model
end

function initialize_model_2()
    --container
    model = nn.Sequential()
    --first layer
    ----Convolution
    model:add(nn.SpatialConvolutionMM(1, 32, 5, 5))
    ----Nonlinearity
    model:add(nn.ReLU())
    ----Max-pooling
    model:add(nn.SpatialMaxPooling(2, 2))
    ----Reshape Results from the second layer
    model:add(nn.Reshape(32 * 14 * 14))
    --Output layer: linear classification
    model:add(nn.Linear(32 * 14 * 14, 10))

    return model
end

function initialize_model_3()
    model = nn.Sequential()
    model:add(nn.Reshape(1 * 32 * 32))
    model:add(nn.Linear(1 * 32 * 32, 10))
    return model
end

-- load data
```

```lua
mnistTrain = torch.load("./data/trainingData.t7")
mnistTest  = torch.load("./data/testData.t7")
-- training parameter
training_step = 1000
train_batch_size = 100
test_batch_size  = 1000
inspection_step = 50
learning_rate = 0.05

testImages, testLabels = mnistTest:getNextBatch(test_batch_size)
function getModelAccuracy (data, model)
    local preds = model:forward(testImages)
    return accuracy(preds, testLabels)
end

function trainModel (data, model, batch_size, learning_rate)
    --load image
    images, labels = data:getNextBatch(batch_size)
    --feedforward
    scores = model:forward(images)
    --define cross entropy criterion
    crit = nn.CrossEntropyCriterion()
    --use crit to calculate teh loss function
    loss = crit:forward(scores, labels)

    --backward
    --calculate gradient of loss w.r.t scores
    dScores = crit:backward(scores, labels)
    --find the rest of the gradients
    model:backward(images, dScores)
    --update parameters
    model:updateParameters(learning_rate)
    --zero the gradients
    model:zeroGradParameters()
    --return model
    return model
end

function trainModel_L1 (data, model, batch_size, learning_rate)
    collectgarbage()
    --load image
    images, labels = data:getNextBatch(batch_size)
    --retrieve the model's parameters and gradients
    parameters,gradParameters = model:getParameters()
    --zero the gradients
    model:zeroGradParameters()

    --feedforward
    scores = model:forward(images)
    --define cross entropy criterion
    crit = nn.CrossEntropyCriterion()
    --use crit to calculate teh loss function
    loss = crit:forward(scores, labels)

    --backward
    --calculate gradient of loss w.r.t scores
    dScores = crit:backward(scores, labels)
    --find the rest of the gradients
    model:backward(images, dScores)
    --update parameters

    local feval = function(x)
        collectgarbage()
        -- get new parameters
        if x ~= parameters then
            parameters:copy(x)
        end
        -- define L1 coef
        coefL1 = 1e-6;
        local norm,sign= torch.norm,torch.sign
        --print('old loss'..loss)
        loss = loss + norm(coefL1 * parameters,1)
        --print('adjusted loss: '..loss)
        gradParameters:add( sign(parameters):mul(coefL1) )
        --print('L1 grad: '..norm(gradParameters,1))
        return loss, gradParameters
    end;

    sgdState = {
            learningRate = learning_rate,
            momentum = 0,
            learningRateDecay = 5e-7
         }
    optim.sgd(feval, parameters, sgdState)

    --return model
    return model
end
```

```lua
print('Hello, which model do you want to train?')
print('1: 2-layer CNN; 2: 1-layer CNN; 3: linear classifier')
model_id = io.read("*n")
-- Repeated Code lines, could be further cleaned
if model_id == 1 then

    --Test Model 1
    model = initialize_model_1()
    accuracy_inspection = torch.zeros(training_step / inspection_step)
    for i = 1,training_step do
        model = trainModel_L1(mnistTrain, model, train_batch_size, learning_rate)
        if (i % inspection_step == 0) then
            accuracy_inspection[i / inspection_step] = getModelAccuracy(mnistTest, model)
            print('['..tonumber(i / training_step * 100)..'%]')
            print(accuracy_inspection[i / inspection_step])
        end
    end
    trained_model_1 = model
    trained_model_1_acc = accuracy_inspection
    torch.save('model1.t7', trained_model_1)
    torch.save('model1_acc.t7', trained_model_1_acc)

elseif model_id == 2 then

    --Test Model 2
    model = initialize_model_2()
    accuracy_inspection = torch.zeros(training_step / inspection_step)
    for i = 1,training_step do
        model = trainModel_L1(mnistTrain, model, train_batch_size, learning_rate)
        if (i % inspection_step == 0) then
            accuracy_inspection[i / inspection_step] = getModelAccuracy(mnistTest, model)
            print('['..tonumber(i / training_step * 100)..'%]')
            print(accuracy_inspection[i / inspection_step])
        end
    end
    trained_model_2 = model
    trained_model_2_acc = accuracy_inspection
    torch.save('model2.t7', trained_model_2)
    torch.save('model2_acc.t7', trained_model_2_acc)

elseif model_id == 3 then
    --Test Model 3
    model = initialize_model_3()
    accuracy_inspection = torch.zeros(training_step / inspection_step)
    for i = 1,training_step do
        --model = trainModel_L1(mnistTrain, model, train_batch_size, learning_rate)
        model = trainModel(mnistTrain, model, train_batch_size, learning_rate)
        if (i % inspection_step == 0) then
            accuracy_inspection[i / inspection_step] = getModelAccuracy(mnistTest, model)
            print('['..tonumber(i / training_step * 100)..'%]')
            print(accuracy_inspection[i / inspection_step])
        end
    end
    range = torch.range(inspection_step, training_step, inspection_step)
    trained_model_3 = model
    trained_model_3_acc = accuracy_inspection
    torch.save('model3.t7', trained_model_3)
    torch.save('model3_acc.t7', trained_model_3_acc)
else
    print("Please only choose from 1 to 3! See you next time")
end
```

`analysis.lua`

```
require "nn"
require 'gnuplot'
require 'util'

training_step = 1000
inspection_step = 50

range = torch.range(inspection_step, training_step, inspection_step)

--load models
model_1 = torch.load('model1.t7')
model_1_acc = torch.load('model1_acc.t7')

model_2 = torch.load('model2.t7')
model_2_acc = torch.load('model2_acc.t7')

model_3 = torch.load('model3.t7')
model_3_acc = torch.load('model3_acc.t7')

model_4 = torch.load('model3_L1.t7')
model_4_acc = torch.load('model3_L1_acc.t7')

--plot learning curve
gnuplot.figure(1)
gnuplot.plot({'TwoLayer',range, model_1_acc,'+-'},
    {'OneLayer',range, model_2_acc,'+-'},
    {'Linear',range, model_3_acc,'+-'},
    {'Linear L1-regularization',range, model_4_acc,'+-'})
gnuplot.movelegend('left','top')

--load image
image_center = torch.load('./data/translations/center.t7')
image_left   = torch.load('./data/translations/leftShifts.t7')
image_right  = torch.load('./data/translations/rightShifts.t7')

function getScore(model, image)
    model:forward(image)
    scores = model.output:clone()
    return scores
end
function getDiff(model)
    score_center = getScore(model,image_center)
    score_all  = {}
    diff       = {}
    diff_image = {}
    invariance = torch.zeros(11)
    for i = 1,5 do
        score_all[6 - i] = getScore(model,image_left[i])
    end
    score_all[6] = score_center
    for i = 1,5 do
        score_all[i + 6] = getScore(model,image_right[i])
    end
    for i = 1,11 do
        invariance[i] = (score_all[i] - score_center):norm(1, 2):mean()
    end;
    return invariance
end
function getDiff_L2(model)
    score_center = getScore(model,image_center)
    score_all  = {}
    diff       = {}
    diff_image = {}
    invariance = torch.zeros(11)
    for i = 1,5 do
        score_all[6 - i] = getScore(model,image_left[i])
    end
    score_all[6] = score_center
    for i = 1,5 do
```

```
        score_all[i + 6] = getScore(model,image_right[i])
    end
    for i = 1,11 do
        invariance[i] = avgDistance(score_all[i], score_center)
    end
    return invariance
    --return score_all, score_center
end

--plot unnormalized invariance
invariance_1 = getDiff_L2(model_1)
invariance_2 = getDiff_L2(model_2)
invariance_3 = getDiff_L2(model_3)
invariance_4 = getDiff_L2(model_4)
gnuplot.figure(2)
gnuplot.plot({'TwoLayer',invariance_1,'+-'},
    {'OneLayer', invariance_2,'+-'},
    {'Linear', invariance_3,'+-'},
    {'Linear L1-regularization', invariance_4,'+-'})
gnuplot.title('L2 Distance un-Normalized')

--normalize
function normalize(t)
    t:cdiv(t:norm(2,1):expandAs(t))
    return t
end


--plot normalized invariance
gnuplot.figure(3)
invariance_1 = normalize(invariance_1)
invariance_2 = normalize(invariance_2)
invariance_3 = normalize(invariance_3)
invariance_4 = normalize(invariance_4)


tick = torch.range(1,11,1)
gnuplot.plot({'TwoLayer',invariance_1,'+-'},
    {'OneLayer', invariance_2,'+-'},
    {'Linear', invariance_3,'+-'},
    {'Linear L1-regularization', invariance_4,'+-'})

gnuplot.title('L2 Distance Normalized')

--plot unnormalized invariance
```

```
invariance_1 = getDiff(model_1)
invariance_2 = getDiff(model_2)
invariance_3 = getDiff(model_3)
invariance_4 = getDiff(model_4)
gnuplot.figure(4)
gnuplot.plot({'TwoLayer',invariance_1,'+-'},
    {'OneLayer', invariance_2,'+-'},
    {'Linear', invariance_3,'+-'},
    {'Linear L1-regularization', invariance_4,'+-'})
gnuplot.title('L1 Distance un-Normalized')

--plot normalized invariance
gnuplot.figure(5)
invariance_1 = normalize(invariance_1)
invariance_2 = normalize(invariance_2)
invariance_3 = normalize(invariance_3)
invariance_4 = normalize(invariance_4)

tick = torch.range(1,11,1)
gnuplot.plot({'TwoLayer',invariance_1,'+-'},
    {'OneLayer', invariance_2,'+-'},
    {'Linear', invariance_3,'+-'},
    {'Linear L1-regularization', invariance_4,'+-'})
gnuplot.title('L1 Distance Normalized')

io.read("*n")
```