Project-3 of "introduction to Statistical Learning and Machine Learning"

Yanwei Fu

November 22, 2018

Abstract

(1) This is the third project of our course. The deadline is 5:00pm, Dec 22nd, 2018. (and happy New Year, by the way). Please send the report to xieyu9332@gmail.com.

(2) The goal of your write-up is to document the experiments you've done and your main findings. So be sure to explain the results. The report can be written by Word or Latex. Generate a single pdf file of your mini-projects and turned in along with your code. package your code and a copy of the write-up pdf document into a zip or tar.gz file and named as Project3-*your-student-id*_your_name.[zip|tar.gz]. Only include functions and scripts that you modified. Also put the names and Student ID in your paper. To submit the report, email the pdf file to xieyu9332@gmail.com.

(3) About the deadline and penalty. In general, you should submit the paper according to the deadline of each mini-project. The late submission is also acceptable; however, you will be penalized 10% of scores for each week's delay.

(4) Note that if you are not satisfied with the initial report, the updated report will also be acceptable, nevertheless given the necessary score penalty of late submission. The good thing is that we will compare and choose the higher score from several submissions as your final score of this project.

(6) OK! That's all. Please let me know if you have any additional doubts of this project. Enjoy!

1 Neural Network

In this problem we will investigate handwritten digit classification. The inputs are 16 by 16 grayscale images of handwritten digits (0 through 9), and the goal is to predict the number of the given the image. If you run *example_neuralNetwork* it will load this dataset and train a neural network with stochastic gradient descent, printing the validation set error as it goes. To handle the 10 classes, there are 10 output units (using a $\{-1, 1\}$ encoding of each of the ten labels) and the squared error is used during training. Your task in this question is to modify this training procedure architecture to optimize performance.

Report the best test error you are able to achieve on this dataset, and report the modifications you made to achieve this. Please refer to previous instruction of writing the report.

1.1 Hint

Below are additional features you could try to incorporate into your neural network to improve performance (the options are approximately in order of increasing difficulty). You do not have to implement all of these, the modifications you make to try to improve performance are up to you and you can even try things that are not on the question list in Sec. 1.2. But, let's stick with neural networks models and only using one neural network (no ensembles).

1.2 Questions (10 points each question)

1. Change the network structure: the vector *nHidden* specifies the number of hidden units in each layer.

2. Change the training procedure by modifying the sequence of step-sizes or using different step-sizes for different variables. That momentum uses the update

$$w^{t+1} = w^t - \alpha_t \nabla f\left(w^t\right) + \beta_t \left(w^t - w^{t-1}\right)$$

   where α_t is the learning rate (step size) and β_t is the momentum strength. A common value of β_t is a constant 0.9.

3. You could vectorize evaluating the loss function (e.g., try to express as much as possible in terms of matrix operations), to allow you to do more training iterations in a reasonable amount of time.

4. Add l_2 regularization (or l_1-regularization) of the weights to your loss function. For neural networks this is called *weight decay*. An alternate form of regularization that is sometimes used is *early stopping*, which is stopping training when the error on a validation set stops decreasing.

5. Instead of using the squared error, use a softmax (multinomial logistic) layer at the end of the network so that the 10 outputs can be interpreted as probabilities of each class. Recall that the softmax function is

$$p\left(y_i\right) = \frac{\exp\left(z_i\right)}{\sum_{j=1}^{J} \exp\left(z_j\right)}$$

   you can replace squared error with the negative log-likelihood of the true label under this loss, $-\log p\left(y_i\right)$

6. Instead of just having a bias variable at the beginning, make one of the hidden units in each layer a constant, so that each layer has a bias.

7. Implement "dropout", in which hidden units are dropped out with probability p during training. A common choice is $p = 0.5$.

8. You can do 'fine-tuning' of the last layer. Fix the parameters of all the layers except the last one, and solve for the parameters of the last layer exactly as a convex optimization problem. E.g., treat the input to the last layer as the features and use techniques from earlier in the course (this is particularly fast if you use the squared error, since it has a closed-form solution).

9. You can artificially create more training examples, by applying small transformations (translations, rotations, resizing, etc.) to the original images.

10. Replace the first layer of the network with a 2D convolutional layer. You will need to reshape the USPS images back to their original 16 by 16 format. The Matlab *conv2* function implements 2D convolutions. Filters of size 5 by 5 are a common choice.

2 Practice: Dimensionality Reduction

This practical session is an opportunity to explore data using the methods. We have provided two data sets:

- freyface.mat – images of Brendan Frey's face in a variety of expressions: start with this one
- swiss roll – There is also MATLAB code to generate the artificial "swiss roll" data. Feel free to use other data sets if you have them. We have also provided code to help with visualisation, as well as LLE and Isomap.

Matlab code for other dimensionality reduction techniques is available from: https://lvdmaaten.github.io/drtoolbox/

2.1 freyface.mat

X contains 1965 images of Brendan Frey's face. It is stored in integer format, although MATLAB does not support this very well. It is best to convert to double:

```
>> load freyface.mat
>> X = double(X);
```

The function *showfreyface* renders the image(s) in its argument. Try *showfreyface(X(:,1:100))*.

2.2 PCA

Find the eigenvectors of $X * X^T/N$, both with and without first removing the mean:

```
>> N = size(X, 2);
>> [Vun, Dun] = eig(X*X'/N);
>> [lambda_un, order] = sort(diag(Dun));
>> Vun = Vun(:, order);
>> Xctr = X - repmat(mean(X, 2), 1, N);
>> [Vctr, Dctr] = eig(Xctr*Xctr'/N);
>> [lambda_ctr, order] = sort(diag(Dctr));
>> Vctr = Vctr(:, order);
```

Which of these corresponds to PCA?

1. Look at the *eigenspectra* (i.e. plot the λ). What might be a good choice for k? Is it easy to tell?
2. Look at the top 16 eigenvectors in each case (the sorted output of *eig* above placed eigenvalues in increasing order, so this would be *showfreyface(V(:,end-15:end))*; or you can use *eigs* to obtain just the top 16). Can you interpret them? How do they differ?
3. Project the data onto the top two eigenvectors, and plot the resulting 2D points. You can use the function *explorefreymanifold* to explore this space. Does what you see make sense?

```
>> Y = V(:,end-1:end)' * X;
>> plot(Y(1,:), Y(2,:), '.');
>> explorefreymanifold(Y, X);
```

(note: V here could be wither *Vun* or *Vctr*: how do the manifolds differ?)

4. Try reconstructing a face from an arbitrary point in this space. That is, choose a point y within the space and compose the corresponding projected vector $\hat{x}$. Remember to add back in the mean when working with eigenvectors in the centered data. Does it look reasonable?

5. Try adding noise to a face (help *randn* if you don't know how), projecting to the manifold (i.e. find the corresponding y) and then reconstructing (i.e. find $\hat{x}$).

2.3 swissroll.m

This is code provided by Roweis and Saul to generate "swiss roll" data and run LLE. You may want to extract the data generation part of it and try running other algorithms on the same data set. Please show the images you visualized in the report.

3 Gaussian Mixture Model

In this assignment, you will experiment with the mixture of Gaussians model. Some code that implements mixture of Gaussians model will be provided for you (both MATLAB and Python).

You will be working with the following dataset:

Digits: The file digits.mat contains 6 sets of 1616 greyscale images in vector format (the pixel intensities are between 0 and 1 and were read into the vectors in a raster-scan manner). The images contain centered, handwritten 2's and 3's, scanned from postal envelopes. train2 and train3 contain examples of 2's and 3's respectively to be used for training. There are 300 examples of each digit, stored as 256300 matrices. Note that each data vector is a column of the matrix. valid2 and valid3 contain data to be used for validation (100 examples of each digit) and test2 and test3 contain test data to be used for final evaluation only (200 examples of each digit).

3.1 EM for Mixture of Gaussians

Let us consider a Gaussian mixture model:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k)$$

Consider a special case of a Gaussian mixture model in which the covariance matrices Σ_k of the components are all constrained to have a common value Σ. In other words $\Sigma_k = \Sigma$, for all k. Derive the EM equations for maximizing the likelihood function under such a model.

3.2 Mixtures of Gaussians

The Matlab file *mogEM.m* implements the EM algorithm for the MoG model. The file *mogLogProb.m* computes the log-probability of data under a MoG model. The file *kmeans.m* contains the k-means algorithm. The file *distmat.m* contains a function that efficiently computes pairwise distances between sets of vectors. It is used in the implementation of k-means.

Similarly, *mogEM.py* implements methods related to training MoG models. The file *kmeans.py* implements k-means.

As always, read and understand code before using it.

3.3 Training

The Matlab variables *train2* and *train3* each contain 300 training examples of handwritten 2's and 3's, respectively. Take a look at some of them to make sure you have transferred the data properly. In Matlab, plot the digits as images using *imagesc(reshape(vector,16,16))*, which converts a 256-vector to an 16x16 image. You may also need to use colormap(gray) to obtain grayscale image. Look at *kmeans.py* to see an example of how to do this in Python.

For each training set separately, train a mixture-of-Gaussians using the code in *mogEM.m*. Let the number of clusters in the Gaussian mixture be 2, and the minimum variance be 0.01. You will also need to experiment with the parameter settings, e.g. randConst, in that program to get sensible clustering results. And you'll need to execute *mogEM* a few times for each digit, and see the local optima the EM algorithm nds. Choose a good model for each digit from your results.

For each model, show both the mean vector(s) and variance vector(s) as images, and show the mixing proportions for the clusters within each model. Finally, provide $\log P(TrainingData)$ for each model.

3.4 Initializing a mixture of Gaussians with k-means

Training a MoG model with many components tends to be slow. People have found that initializing the means of the mixture components by running a few iterations of k-means tends to speed up convergence. You will experiment with this method of initialization. You should do the following.

- Read and understand *kmeans.m* and *distmat.m* (Alternatively, *kmeans.py*).
- Change the initialization of the means in *mogEM.m* (or *mogEm.py*) to use the k-means algorithm. As a result of the change the model should run k-means on the training data and use the returned means as the starting values for mu. Use 5 iterations of k-means.
- Train a MoG model with 20 components on all 600 training vectors (both 2's and 3's) using both the original initialization and the one based on k-means. Comment on the speed of convergence as well as the final *log-prob* resulting from the two initialization methods.

3.5 Classification using MoGs

Now we will investigate using the trained mixture models for classification. The goal is to decide which digit class d a new input image x belongs to. We'll assign $d = 1$ to the 2's and $d = 2$ to the 3's.

For each mixture model, after training, the likelihoods $P(x \mid d)$ for each class can be computed for an image x by consulting the model trained on examples from that class; probabilistic inference can be used to compute $P(d \mid x)$, and the most probable digit class can be chosen to classify the image.

Write a program that computes $P(d = 1 \mid x)$ and $P(d = 2 \mid x)$ based on the outputs of the two trained models. You can use *mogLogProb.m* (or the method *mogLogProb* in *mogEm.py*) to compute the log probability of examples under any model.

You will compare models trained with the same number of mixture components. You have trained 2's and 3's models with 2 components. Also train models with more components: 5, 15 and 25. For each number, use your program to classify the validation and test examples.

For each of the validation and test examples, compute $P(d \mid x)$ and classify the example. Plot the results. The plot should have 3 curves of classification error rates versus number of mixture components (averages are taken over the two classes):

- The average classification error rate, on the training set;

- The average classification error rate, on the validation set;
- The average classification error rate, on the test set.

Provide answers to these questions:

1. You should nd that the error rates on the training sets generally decrease as the number of clusters increases. Explain why.
2. Examine the error rate curve for the test set and discuss its properties. Explain the trends that you observe.
3. If you wanted to choose a particular model from your experiments as the best, how would you choose it? If your aim is to achieve the lowest error rate possible on the new images your system will receive, which model (number of clusters) would you select? Why?