Model Answers to "The Next Pixel Prediction Task"

December 21, 2015

1. (Data preprocessing and visualization, 8 marks)

   (a) *Solution.* In Algorithm 1 we are told that the data was discretized to 64 grey scale values $0, \ldots, 63$, hence this sets a natural scale for the problem. Figure 1 shows the histogram, which tells that a large portion of patches have small standard deviation, i.e., flat.

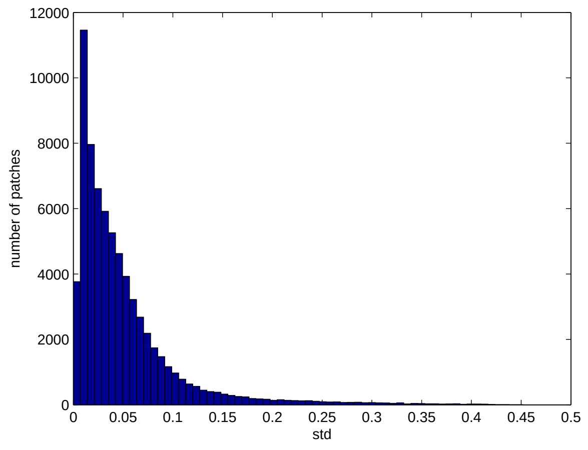

Figure 1: Histogram of the standard deviation of each patch for the training set.

   (b) *Solution.* For flat patches, a simple way of predicting the next pixel is using the mean of its preceding pixels as a predictor.

   (c) *Solution.* Figure 2 shows examples of flat and non-flat patches, where the pixels after the target pixels are set as zero values.

2. (Linear regression with adjacent pixels, 10 marks)

   (a) *Solution.* Figure 3 shows 3-D plot of $x(j, \text{end})$ (the pixel west of target pixel $y(j)$), $x(j, \text{end} - 34)$ (the pixel north of target pixel $y(j)$) and $y(j)$,

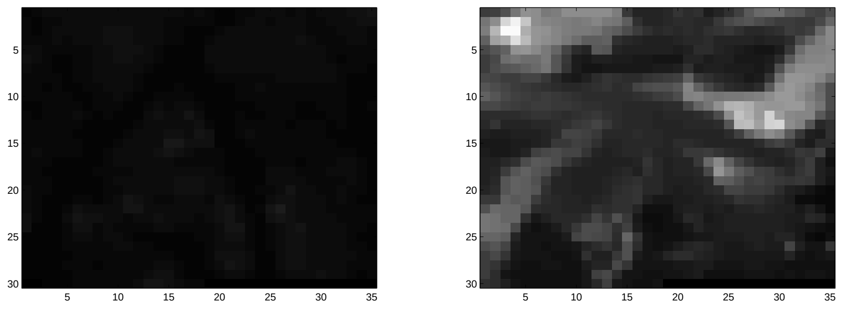

Figure 2: Examples of flat and non-flat patches on the left and right, respectively.

where only $10,000$ points are plotted to avoid overprinting. From this visualization, we can easily observe that most of data points concentrate around the diagonal line from (0,0,0) to (1,1,1), i.e., the target pixel's values are quite similar to its neighbouring pixels. There are some 'outliers' deviating from the diagonal. The plot also shows that the distribution of (normalized) pixel values is skewed towards lower values (say 0.1 to 0.3).

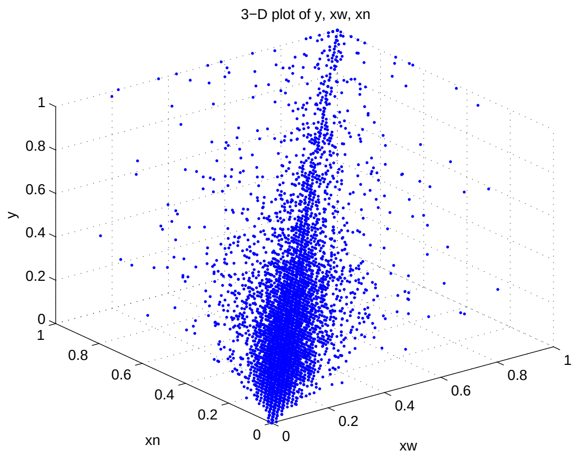

Figure 3: 3D plot of the target pixel value y against its two neighbours xn and xw.

(b) *Solution.* $\hat{\mathbf{w}} = (X^T X)^{-1} X^T \mathbf{y}$.

(c) *Solution.*

$w = [0.4606, 0.5241, 0.0026]$. RMSE on training set = 0.0506, RMSE on test set = 0.0503. The RMSE values of the training and test sets are close, which we would expect given that the linear regression model only estimates 3 parameters, and that there are over 17,000 training examples. To get a sense of the variability expected, we can compute the standard error of the training and testing RMSEs. The standard error of the MSE can be computed as per qu 2 in tutorial 5. To obtain the standard error of the RMSE, one method is to use a Taylor expansion to obtain $f(x \pm \delta) = f(x) \pm \delta f'(x)$ with $f(x) = \sqrt{x}$. Applying this gives standard errors of 0.0010 and 0.0017 respectively. Thus we can see that the difference between the training and test RMSEs can be attributed to sampling variability.

Figure 4 shows the linear regression surface plot as well as 3-D plot of test points.

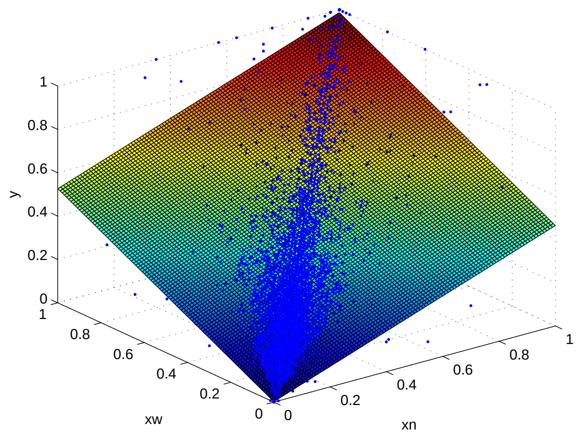

Figure 4: Plot of the linear regression prediction surface, and a 3-D plot of test points.

Code fragment:

```
feature_matrix = [xtr_nf(:, [1032-34, 1032]),...
 ones(number_of_non_flat, 1)];
w = (feature_matrix'*feature_matrix) \ (feature_matrix' * ytr_nf );

ypre_tr = feature_matrix * w;
rmse_LRadj_tr = sqrt(mean(((ytr_nf - ypre_tr).^2)))

ypre = [xte_nf(:, [1032-34, 1032]), ones(size(xte_nf,1),1)] * w;
rmse_LRadj_te = sqrt(mean(((yte_nf - ypre).^2)))

figure,
[dim1, dim2] = meshgrid(0:0.01:1,0:0.01:1);
ysurf = [[dim1(:), dim2(:)], ones(numel(dim1),1)]*w;
surf(dim1, dim2, reshape(ysurf, size(dim1)));

hold on
plot3(xte_nf(:, 1032), xte_nf(:,1032-34), yte_nf, 'b.');
grid on;
xlabel(xn);
ylabel(xw);
zlabel(yte);
hold off
```

3. (RBF regression with adjacent pixels, 8 marks)

   (a) *Solutions.* Figure 5 shows the cross-validation RMSE with respect to different number of basis functions. *Note that due to the random partition of the data during cross-validation, you will not obtain exactly the same results across multiple runs.* The optimal number of basis functions for this run is 10, although looking at the y-axis in the plot we see that the differences in RMSE appear to be very small for different number of basis functions. Running the cross-validation procedure a number of times with different random numbers gave quite a lot of variability in the resulting plot, although the optimal number of basis functions was generally 10 or 15.
   Code fragment:

```
nhidden_all = [5:5:30];
nout = 1; % Number of outputs.
nin = 2; % Number of inputs.

options = foptions;
options(1) = 1; % Display EM training
options(14) = 5; % number of iterations of EM

for ind_nh = 1:numel(nhidden_all)
    net = rbf(nin, nhidden_all(ind_nh), nout, 'gaussian');
    predfun = @(XTRAIN, YTRAIN,  XTEST) ...
        rbffwd(rbftrain(net, options, XTRAIN, YTRAIN), XTEST);
    mse(ind_nh) = crossval('mse', xtr_nf(:, [1032-34, 1032]), ...
        ytr_nf, 'Predfun', predfun);
end
figure, plot(nhidden_all, sqrt(mse), 'r-o');
xlabel('Number of Basis')
ylabel('RMSE of 10-fold CV')
```

   (b) *Solutions.* Based on the answer above, 10 basis functions were selected, and the RBF model was trained on all of the training data, giving results of RMSE on training set = 0.0504, and RMSE on test set = 0.0500. The test performance is slightly better than for linear regression (0.0503) although the difference is very small relative to the standard error.
   Code fragment:

```
% find the optimal number of hidden neurons
nhidden_opt = nhidden_all(mse == min(mse));

% using the optimal nhidden to train RBF on the whole training set
net = rbf(nin, nhidden_opt, nout, 'gaussian');
options = foptions;
options(1) = 1; % Display EM training
options(14) = 5; % number of iterations of EM
net = rbftrain(net, options, xtr_nf(:, [1032-34, 1032]), ytr_nf);

ypre_tr = rbffwd(net, xtr_nf(:, [1032-34, 1032]));
rmse_RBFadj_tr = sqrt(mean(((ytr_nf - ypre_tr).^2)))
```

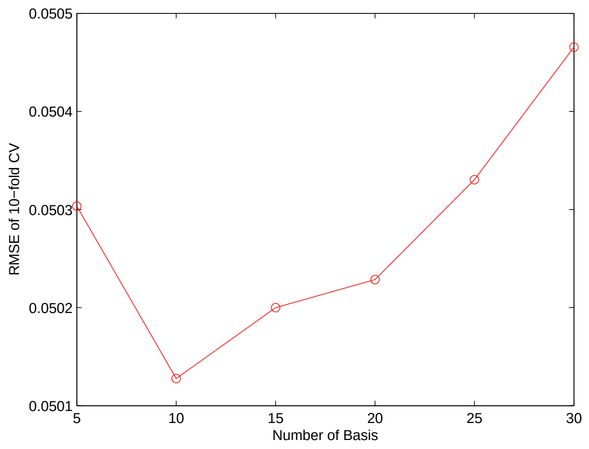

Figure 5: Cross-validation RMSE with respect to different numbers of RBFs.

```
ypre = rbffwd(net, xte_nf(:, [1032-34, 1032])); %
rmse_RBFadj_te = sqrt(mean(((yte_nf - ypre).^2)))
```

4. (Linear regression with all pixels, 6 marks)

   Solutions. RMSE on training set = 0.0371, RMSE on test set = 0.0456.

   The first observation is that larger contexts help to improve the prediction compared with the solutions using only adjacent pixels, comparing the test RMSE 0.0456 with the RBF result of 0.0500. However, we also note that there is now a large difference between the training and test results. There are 1033 entries in the weight vector $\mathbf{w}$ (1032 pixels plus the bias term). One would expect that it is relatively safe to estimate this number of parameters from over 17,000 training examples. However, the 1032 pixels are clearly highly correlated, which can make the weights poorly determined. Checking of the matrix $X^T X$ which is inverted in the linear regression solution shows that its minimum eigenvalue is larger than 1 and that the condition number (approx 7×10^5) is not extreme; hence the matrix inversion should be stable.

   Going beyond what would be expected in most students' answers, a useful way to think about what is going on is to consider the eigenvectors and eigenvalues of $X^T X$. The eigenvectors with small eigenvalues correspond to "checkerboard" type patterns in pixel space, while those with large eigenvalues correspond to smooth, slowly varying patterns. It seems unlikely that these checkerboard patterns correspond to meaningful structure in the data. Computing a prediction $\hat{\mathbf{w}}^T \mathbf{x}$ implicitly involves computing the dot product of $\mathbf{x}$ with each of the eigenvectors. Using ridge regression with a ridge parameter substantially larger than the smallest eigenvalue can kill off the "checkerboard" contributions, and narrow the gap between training and test RMSE.

The code is very similar to question 2(c).

5. (Neural Network with all pixels, 10 marks)

   (a) *Solutions.* RMSE on training set = 0.0333, RMSE on test set = 0.0473. We can observe two points: 1) the test error of MLP is significantly higher than the training error; 2) MLP has smaller training error but higher test error than the simpler model, linear regression. Relative to the linear regression model we can see that the MLP involves an overfitting issue. Note that the MLP model contains over 10,000 weights (there are 10 hiddens each looking at 1033 inputs), and this is relatively large wrt the number of training examples (over 17,000).

   (b) *Solutions.* 5 runs: RMSE on training set = $\{0.0503, 0.0467, 0.0485, 0.0481, 0.0489\}$, RMSE on test set = $\{0.0514, 0.0498, 0.0515, 0.0515, 0.0526\}$. These different solutions on the same (reduced) dataset show the issue of multiple local minima in neural network in training neural networks. Another observation is that using only subset of the data for training produces much worse prediction results compared with usage of full data, implying a complex model (such as MLP) normally needs sufficient data to be well trained for prediction.

```
%% Neural Network with all pixels
rng(2015,'twister')
% Set up the network
nhid = 10; % number of hidden units
net = mlp(size(xtr_nf,2), nhid, 1, 'linear');

% Set up vector of options for the optimiser.
options = zeros(1,18);
options(1) = 1;             % This provides display of error values.
options(9) = 1;             % Check the gradient calculations.
options(14) = 200;         % Number of training cycles.

% Train using scaled conjugate gradients.
[net, options] = netopt(net, options, xtr_nf(1:5000,:), ytr_nf(1:5000,:), 'scg');
toc


% RMSE on training set
ypred_tr = mlpfwd(net, xtr_nf);
rmse_NNsuball_tr = sqrt(mean(((ytr_nf - ypred_tr).^2)))

% RMSE on test set
ypred = mlpfwd(net, xte_nf);
rmse_NNsuball_te = sqrt(mean(((yte_nf - ypred).^2)))
```

6. (Discussion, 8 marks)

   Solutions. The results from the experiments are summarized in Table 1.

   Some observations are:

Method	Training RMSE	Test RMSE
Lin regression neighbours	0.0506	0.0503
RBF neighbours	0.0504	0.0500
Lin regression all	0.0371	0.0456
NN all	0.0333	0.0473

Table 1: Table summarizing results from the experiments.

- Linear regression and the RBF network using the two neighbouring pixels are computationally simple, but have worse predictions results than the models using all the pixels. Using all the pixels to train the models yields better results but with higher computational complexity.
- As above we have discussed the potential for overfitting when using all of the context pixels; this is clearly seen with the results for the neural network relative to linear regression.

Possible next experiment: it is a large step to go from 2 to 1032 pixels of context; it would be interesting to consider different larger contexts, growing the neighbourhood size above and to the left of the target pixel.