Computational Intelligence, SS11

http://www.igi.tugraz.at/lehre/CI

Gerhard Neumann [1]

1 Homework: Linear Regression and Gradient Descent

[Points: 12.5 , Issued: 2011/03/18 , Deadline: 2011/04/01 , Tutor: TBA [2]; Infohour: TBA , 15:30-16:30 , HS i11 ; Einsichtnahme: TBA , 15:30-16:30 , HS i11 ;]

1.1 Linear Regression with Non-Linear Basis-Functions [6 + 3* points]

In this task you have to fit a different basis function models $f(x; \mathbf{w}) = \sum_{k=1}^{d} \Phi_k(x) w_k)$ to a given dataset. Download the file *linearregression_homework.mat*. The file contains the training set (*x_train*, *y_train*), consisting of 60 data point and a test set (*x_test*, *y_test*), consisting of 10000 data points. The target values *y_train* and *y_test* are noisy estimates of the real target function. The real (without noise) target values for the test set are stored in *y_target*. Use this variable only for plotting.

1.1.1 Polynomial Basis functions(3 points)

In this task you have to fit a l-degree polynomial. Here, we have $l+1$ basis functions, $\Phi_k(x) = x^{k-1}$.

- Train a $l = 0 : 18$ degree polynomial for the given data set.
- Plot the training data-points, the target function (*y_target*) and the learned function for each l (in the same plot).
- For $l = 18$ degrees, plot the basis functions as a function of x. Describe your results with respect to the slope of the basis functions
- Plot the mean squared error (mse) on the training and on the test set as a function of l. Explain your results. Which value of l would you choose? Why is it getting worse for large ls ?

1.1.2 Radial Basis functions (RBF) (3 points)

Here, the kth basis function is given by $\Phi_k(x) = \exp(-(x - \mu_k)^2/\sigma^2)$, which is equivalent to a Gaussian bell curve, also called receptive field. μ_k is the center of the receptive field and σ^2 corresponds to its bandwidth. We will again evaluate the approach for a $d = 2 : 18$ basis functions. We will distribute centers μ_k uniformly in the input space by setting $\mu = \text{linspace}(-1, 1, d)$;. For σ we will use the value $2/d$.

- Plot the training data-points, the target function (*y_target*) and the learned function for each d (in the same plot).
- For $d = [6, 12, 18]$ receptive fields, plot the basis functions $\Phi_k(x)$ as a function of x.
- Plot the mean squared error (mse) on the training and on the test set as a function of d. Explain your results. Which value of d would you choose?

[1] mailto:neumann@igi.tu-graz.ac.at
[2] mailto:neumann@igi.tu-graz.ac.at

1.1.3 Local linear models (3*points)

This approach is similar to the RBF approach. However, for each receptive field we now use two basis functions, $\Phi_k(x) = \exp(-(x-\mu_k)^2/\sigma^2)$ and $\Phi_{2k}(x) = \exp(-(x-\mu_k)^2/\sigma^2)x$. We will again evaluate the approach for a $l = 2:18$ reptive fields (resulting into $d = 4:36$ basis functions...). Use the same values for μ and σ as before.

- Plot the training data-points, the target function (*y_target*) and the learned function for each d (in the same plot).
- For $l = [6, 12, 18]$ receptive fields, plot the basis functions $\Phi_k(x)$ as a function of x.
- Plot the mean squared error (mse) on the training and on the test set as a function of l. Explain your results. Which value of l would you choose?

Compare the Polynomial, the RBF and the Local linear models (*) with respect to their performance on the test and training set.

1.1.4 Regularized Polynomial Regression [8 (+*1) points]

For this task we will use regularized regression. Indifference to standard error function, for regularized regression we minimize the error function:

$$E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \sum_{k=1}^{d}\Phi_k(x_i)w_k)^2 + \alpha^2\sum_{k=1}^{d}w_k^2$$

.

- Write the error function in matrix form. Explicitely state the dimensions of the vectors and matrices.
- Derive a closed form solution for the optimal weight vector. (Hint : Use the identity $\alpha^2\mathbf{w} = \alpha^2\mathbf{I}\mathbf{w}$, $\mathbf{I}$ being the identity matrix.

Implement your learning rule in matlab for all basis function models you have used (+1* point for local linear models) (use always the settings with the highest number of basis functions). For all basis function models, do :

- Train the model with different α values, use $\alpha = \text{logspace}(-10, 0, 200)$
- Plot the mean squared error of the training and of the test set for the given αs. Use a logarithmic scale of the x-axis (matlab command : semilogx). What is the best value of α? Interpret your results.
- Plot the learned functions for the lowest, the highest and the best α. Also include the training points and the target function in your plot.
- Plot the mean absolute weight values for the given α (again use a logarithmic scale for the x-axis).
- Interpret your results, can you see a relation between the mean squared error on the test set and the mean absolute weight values?

Also compare the results between the different basis function models!

1.1.5 Decomposition into structural and approximation error [5* points]

In this task we want to investigate the decomposition of the expected error in the structural error and the approximation error. For this task you have to download the file *linearregression_homework_80.mat*. It contains 100 datasets with 80 datapoints (*x_train*, *y_train*, stored in a cell array) and one dataset with 10000 datapoints *x_test*, *y_test*. The structural error is given by $\langle (y - f(x; \mathbf{w}^*))^2 \rangle$ ($\langle \dots \rangle$ denotes the expectation with respect to the data points x, y) and the approximation error is given by $\langle (f(x; \hat{\mathbf{w}}) - f(x; \mathbf{w}^*))^2 \rangle$, where $\mathbf{w}^*$ is the optimal weight vector and $\hat{\mathbf{w}}$ is an weight vector which we infer from a limited amount of training data (in our case we have 80 data points).

The optimal weight vector $\mathbf{w}^*$ can be estimated by calculating $\mathbf{w}$ on the large data set (*x_test*, *y_test*). Also use this data set to calculate the expectations given by the $\langle\rangle$ brackets. Also use the large dataset to estimate the structural error. The approximation error and the expected error have to be calculated by averaging over several training data sets. Therefore, we estimate $\hat{\mathbf{w}}$ on each training data set und calculate the mean error $(1/N \sum_i (y_i - \sum_k \Phi_k(x_i) w_k)^2)$ and the mean squared deviation from the optimal hypthesis $(1/N \sum_i (\sum_k \Phi_k(x_i) w_k^* - \sum_k \Phi_k(x_i) \hat{w}_k)^2)$. Both values have to be averaged over the different training sets to get the expected and the approximation error.

- Calculate the expected error, the structure error and the approximation error of a 18-degree polynomial using regularized regression with $\alpha = \text{logspace}(-10, 0, 25)$
- Plot the expected error, the structure error, the approximation error and the sum of structure and approximation error. Check wether your expected error equals the sum of strcuture and approximation error. Again use a semilogx plot to get a logarithmic scale of the x-axis. Interpret your results, what is the best value of α? Why is there a difference to Example 1.1.4?

1.1.6 Hints

- If implemented efficiently, the scripts for this homework run in less than a minute on a standard modern computer. However, if implemented ineffeciently, the scripts may run for several hours. If you have problems with your execution time think about a more efficient solution.

2 Gradient Descent

2.1 Gradient Descent, Impulse term and adaptive learning rates... [7 points]

In this task you have to implement and evaluate different gradient descent algorithms. Consider the following function:

$$f(w_1, w_2) = -2 \exp(-20(0.5(w_1 - 1)^2 + (w_2 - 1)^2)) - \exp(-0.1(4(w_1 + 1)^2 + 0.5(w_2)^2))$$

1. Calculate the gradient of f and state the gradient descent update rules for w_1 and w_2.

2.1.1 Standard Gradient Descent (2 points)

1. Implement a gradient descent minimization algorithm for f in matlab. Therefore you are required to implement the following function :

```
function [weights, error] = gradientDescentHw2(w0, numIter, eta),
```

where *weights* is a numIter + 1 × 2 matrix representing the weight vector for each iteration and *error* is a numIter × 1 vector returning the value of $f(w_1, w_2)$ after each iteration. *w0* is the initial weight vector, *numIter* represents the number of iterations for the gradient descent algorithm and, parameter *eta* represents the learning rate.

2. Plot the evolution of the weightvector for $w0 = [2, 0.5]$ with different learning rates ($\eta = [0.2, 0.15, 0.1, 0.05]$). Always use numIter = 100 iterations. For plotting the evolution of the weight vector you can use the script plotErrorFunction.m[3]. Also plot the evolution of the error for each of the learning rate (in a single plot!).

3. Repeat the previous experiment for $w0 = [-0.2, -0.5]$ and the same learning rates. Interpret your results! What kind of problems occur for the different learning rates? When do we find a local and when a global minimum?

2.1.2 Impulse Term (2 points)

Implement the gradient descent algorithm with impulse term. Therefore, implement the following function:

```
function [weights, error] = gradientDescentImpulseHw2(w0, numIter, eta, alpha),
```

where α denotes the smoothing factor for the impulse term. The rest of the parameters remain the same.

1. Repeat the experiments from Section 2.1.1 with $\alpha = 0.5$, compare your results with the previuous Section.

2.1.3 Adaptive learning rate (1 point)

Extend the gradient descent with impulse term by an adaptive learning rate. If the error decreases, increase the learning rate by the factor 1.05, otherwise descrease the learning rate by 0.7. In addition, reject the gradient update if the error increased!

```
function [weights, error] = gradientDescentAdaptiveHw2(w0, numIter, eta, alpha),
```

1. Repeat the experiments from Section 2.1.1 with $\alpha = 0.5$, compare your results with the previuous Sections.

2.1.4 Local Minima (2 points)

1. Now we want to evaluate the effect of local mimima. For this task we will use the standard gradient descent algorithm with a learning rate of $\eta = 0.1$. Determine the function value minf $= f(\hat{w}_1, \hat{w}_2)$, where $\hat{w}_1$ and $\hat{w}_2$ are the weight values returned by the gradient descent algorithm. This has to be done for the intial weight vectors $w0 = [-2+0.1i, -2+0.1j]$ where $0 \leq i \leq 40$ and $0 \leq j \leq 40$. I.e. we get a 2-dimensional matrix of minf values.

2. Create a surface plot of these minf values. In which area do we find the global and in which area only the local minimum?

3. Calculate the fraction of the number of initial weight vectors which end in the global minimum.

[3] http://www.igi.tugraz.at/lehre/CI/homework/data/plotErrorFunction.m

2.2 Linear regression with gradient descent [4* points]

Implement a gradient descent learning algorithm of your choice (standard/impulse/adaptive learning rate) for linear regression. Use your algorithm to train a 9-degree polynomial using the dataset of Section 1.1.1. Show that your gradient descent algorithm always (from different (random) initial conditions) finds the same solution as the closed form solution. How many iterations are needed?