Computational Intelligence, SS12

https://www.spsc.tugraz.at/courses/computational-intelligence

Stefan Habenschuss [1]

Homework 2: Backpropagation and Neural Networks

[Points: 20 , Issued: 2012/05/04 , Deadline: 2012/05/25 , Tutor: Christian Knoll [2]; Infohour: TBA , , ; Einsichtnahme: TBA , , ;]

1 Backpropagation [12 + 10* points]

1.1 Backpropagation for a simple network [12 points]

Consider a 2-layer neural network with 2 inputs, 4 hidden neurons and 1 output unit. In both layers the sigmoid activation function

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

is used (Note that $\sigma'(x) = (1 - \sigma(x))\sigma(x)$). In addition to the weights for the inputs, each neuron j has an offset weight w_{j0}. Hence, the whole network has $4 * 3 + 5 = 17$ weights. Your task is to implement the backpropagation algorithm for this network.

- Implement the function

  ```
  function [output, z, a1, a2] = feedforwardNN(w, x)
  ```

  which, given the weight vector $\mathbf{w}$ and the (2-dimensional input) $\mathbf{x}$, returns the output of the neural network as well as the outputs of the hidden layer (z, 4-dimensional).

- Implement the function

  ```
  function [dw] = backpropNNSingle(w, x, y)
  ```

  which calculates the gradient of the squared error of a single example $\mathbf{x}$ and y.

- Implement the function

  ```
  function [dw] = backpropNNFull(w, X, Y)
  ```

  which calculates the gradient of the squared error of N examples which are stored in the matrices X and Y (each row corresponds to an example).

- Use these functions to implement a gradient descent algorithm with an adaptive learning rate. Train your network using the dataset *backprop.mat*, use an initial learning rate of 1.0 and 1000 iterations. Create your initial weight vector $\mathbf{w}_0$ by using a normal distribution (randn(17, 1)).

- Repeat the training for 10 different initial weight vectors and plot the evolution of the error function for each of these weight vectors (in the same plot). Do you get different local minima?

- Create a surface plot of the learned function after 0, 10, 100, 300 and 1000 iterations for the worst and for your best learning trial.

[1] mailto:habenschuss@igi.tugraz.at
[2] mailto:christian.knoll@student.tugraz.at

1.2 Backpropagation for RBF networks[10 *points]

Consider the following 2-layer feedforward neural network with a 2-dimensional input, K outputs a M hidden units. The kth output is given by :

$$o_k(x) = h(\sum_{j=1}^{M} w_{kj} \exp((\mathbf{x} - \boldsymbol{\mu}_j)^2 / b_j^2)),$$

where h is sigmoid function $\sigma(x)$. Your task is to derive a weight update rule for the weights w_{kj}, $\boldsymbol{\mu}_j$ and b_j using the backpropagation algorithm. Only consider the mean squared error (mse) of a single example $(\mathbf{x}, \mathbf{y})$. Always use the same formalism and notation as discussed in the lecture/practical course.

- Define the neuron input $a_j^{(1)}$ (input to the activation function), the layer output $z_j^{(1)}$ and the activation function of the neurons $h_j^{(1)}$ in the hidden layer.
- Define the neuron input $a_k^{(2)}$ and the layer output o_k and the activation function. Use the definitions from the hidden layer to simplify your equations.
- Calculate $\delta_k^{(2)}$ for the output neurons and the resulting weight update Δw_{kj} for the weights in the output layer.
- Calculate $\delta_j^{(1)}$ for the hidden neurons and the resulting weight updates $\Delta \boldsymbol{\mu}_j$ and Δb_j for the parameters in the hidden layer [6* points up to here].
- Implement your rule in matlab using the same set of functions as defined in 1.1. Use $K = 1$ and $M = 4$. Also use the same dataset as in 1.1. Note that b_j is the bandwidth of the RBF centers which should not get smaller than 0.05. Just fix b_j at this value if it gets smaller than this value.
- Repeat the training for 10 different initial weight vectors and plot the evolution of the error function for each of these weight vectors (in the same plot). Do you get different local minima? Compare to 1.1.
- Create a surface plot of the learned function after 0, 10, 100, 300 and 1000 iterations for the worst and for your best learning trial.

2 Regression with Neural Networks [8 points]

2.1 Simple Regression with Neural Networks [3 points]

In this task a simple 1-dimensional function should be learned with feed-forward neural networks. Use the same dataset (*linearregression_homework.mat*) as for Homework 1.

- Train a neural network with $n = [1, 2, 3, 4, 6, 8, 12, 20, 40]$ neurons. Use the training algorithm 'trainscg', train for 700 epochs.
- Plot the mean squared error of the training and of the test set for the given number of neurons. For the test set, plot the mean squared error (mse).
- Interpret your results. What is the best value of n?
- Plot the error on the test set and on the training set for $n = 2$, $n = 8$ and $n = 40$ during training (you already get this plot as output of the neural network toolbox, or you can get the relevant data from the performance structure returned by the train function). Interpret your result, is the error on the training and test set always decreasing?

- Plot the learned functions for $n = 2$, $n = 8$ and $n = 40$. Interpret your results, refer to results from the previous plots!
- Compare the results to Homework 1. Is there any connection of n and the degree of the polynomial?

2.2 Regularized Neural Networks [5 points]

Now we want to investigate different regularization methods for neural networks, i.e. weight decay and early stopping. Use the same data-set as before.

- **Weight Decay**: Train a neural network with $n = 40$ hidden neurons. Use the training algorithm 'trainscg', train for 700 epochs. Use the regularized error function *msereg* as *net.performFcn*. This performance function is equivalent to the standard loss function used for weight decay:
$$\text{msereg} = \alpha\text{mse} + (1 - \alpha)\sum_i w_i^2$$
Use different regularization factors (α resp. *net.performParam.ratio* in matlab) of $\alpha = [0.9, 0.95, 0.975, 0.99, 0.995, 1.0]$;
- Plot the mean squared error of the training and of the test set for the given regularization factors.
- Interpret your results, i.e. explain the course of both plots. What is the best value of α?
- Compare your results to the regularization term used in Homework 1! Is there any relation between these regularization terms (also compare the equations)?
- Plot the learned functions for the lowest, the highest and the best value of α.
- **Early Stopping:** Now we want to test the performance of early stopping. Again train a neural network with 40 hidden neurons, use the standard *mse* function as performance function. Train a neural network for 700 epochs and determine at which epoch $epoch_{ES}$ the error on the test-set reaches the minimum. Retrain a second neural network starting from the same initial weights as the first network, but this time only train for $epoch_{ES}$ epochs.
- Determine the error on the test set and plot the learned function. Compare the error and the plotted function to the fully trained network.
- Compare the performance (error on the testset) and the learned functions for early stopping, weight decay and for the different number of hidden neurons. Which type of regularization would you prefer? What are the advantages/disadvantages of these methods ?

2.3 Hints

- You can easily use the training record `tr` returned by the *train* function to get the error on the training set. If you supply the test set to the *train* function via the *TV* parameter, i.e. `train(net,P,T,[],[],[],TV)`, you will also find the error on the test set in that structure. See `doc train` for more info.
- For weight decay you can't use the performance structure returned by the *train* function because it returns the regularized error function and not the *mse*. Instead, you need to determine the mse explicitely (e.g. you can use the function *mse(errorvector)*).
- Always normalize your training data to zero-mean and unit variance (use the function *mapstd*).