Artificial Neural Networks and Bias/Variance with PYTHON

Objective: The objective of today's exercise is to understand (i) regularization in linear regression (regularized linear regression), (ii) Introduce ANNs and get a feeling of how neural networks can be used for data modelling and the role of hidden units in neural networks (iii) understand how ANNs and multinomial regression (the generalization of logistic regression to multiple classes) can be applied to multi-class problems.

Material: Lecture notes *"Introduction to Machine Learning and Data Mining"* C12, C13 as well as the files in the exercise 8 folder available from Campusnet.

Preparation: Exercises 1-7

Part 1: Group discussion (max 15 min)

For the group discussion, each group should have selected a *discussion leader* at the previous exercise session. The purpose of the discussion leader is to ensure all team members understands the answers to the following two questions:

Multiple-Choice question: Solve and discuss **problem 13.1** from chapter 13 of the lecture notes. Ensure all group members understand the reason why one of the options is true and why the other options can be ruled out. (After today's exercises make sure to complete the remaining multiple-choice problems listed as part of the preparation for week 8 on the course homepage).

Discussion question: Discuss the following question in the group

- Suppose you wished to apply an ANN to classify if a 16×16 hand-written digit was three or not and suppose you had 20 units in the hidden layer. How many neurons would you have in the input and output layers? How many weights would be attached to each unit? How many weights would there be in total? How would the value at each hidden unit be computed in practice? How would the value at the output unit be computed? (explain dimensions of matrices/vectors that enter in the computation) How difficult would it be for you to implement a neural network that computed the output value? (don't consider training)

Part 2: Programming exercises

Piazza discussion forum: You can get help by asking questions on Piazza: https://piazza.com/dtu.dk/fall2017/02450

Software installation: Extract the Python toolbox from Campusnet. Start Spyder and add the toolbox directory (`<base-dir>/02450Toolbox_Python/Tools/`) to `PYTHONPATH` (Tools/PYTHONPATH manager in Spyder). Remember the purpose of the exercises is not to re-write the code from scratch but to work with the scripts provided in the directory `<base-dir>/02450Toolbox_Python/Scripts/`

Representation of data in Python:

	Python var.	Type	Size	Description
	`X`	numpy.array	$N \times M$	Data matrix: The rows correspond to N data objects, each of which contains M attributes.
	`attributeNames`	list	$M \times 1$	Attribute names: Name (string) for each of the M attributes.
	`N`	integer	Scalar	Number of data objects.
	`M`	integer	Scalar	Number of attributes.
Regression	`y`	numpy.array	$N \times 1$	Dependent variable (output): For each data object, `y` contains an output value that we wish to predict.
Classification	`y`	numpy.array	$N \times 1$	Class index: For each data object, `y` contains a class index, $\mathtt{y}_n \in \{0, 1, \ldots, C-1\}$, where C is the total number of classes.
	`classNames`	list	$C \times 1$	Class names: Name (string) for each of the C classes.
	`C`	integer	Scalar	Number of classes.
Cross-validation				All variables mentioned above appended with `_train` or `_test` represent the corresponding variable for the training or test set.
	`⋆_train`	—	—	Training data.
	`⋆_test`	—	—	Test data.

8.1 Regularized linear regression

An approach to control for the complexity of the model is to regularize the parameters in the objective function. For instance for linear regression we can regularize the parameters $\boldsymbol{w}$ by the Frobenius norm, i.e.

$$\begin{aligned} C &= \sum_n \|y_n - \boldsymbol{x}_n^\top \boldsymbol{w}\|_F^2 + \lambda \|\boldsymbol{w}\|_F^2 \qquad (1) \\ &= \sum_n (y_n - \boldsymbol{x}_n^\top \boldsymbol{w})^2 + \lambda \boldsymbol{w}^\top \boldsymbol{w} \qquad (2) \end{aligned}$$

Solving the equation $\frac{\partial C}{\partial \boldsymbol{w}} = 0$ we obtain $\boldsymbol{w} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$, see also the lecture slides for today.

8.1.1 Inspect and run the script `ex8_1_1.py`. The script analyses the body data with the regularized least squares regression method. In the inner loop of the script 10-fold cross-validation is used to select for the optimal value of regularization λ. In the outer loop 5-fold cross-validation is used to evaluate the performance of the optimal selected value for λ. What is plotted on each of the generated plots? How is the optimal value of the regularization (λ) chosen? What do you think may be the advantages and disadvantages of regularized linear regression compared to the sequential feature selection?

8.2 Artificial Neural Networks

In this part of the exercise we will use neural networks to classify data. We will consider networks with an input layer, one layer of hidden units and an output layer. If there is one unit in the hidden and output layer and the transfer function of each layer is linear, i.e. $g^{(hidden)}(t) = t$ and $g^{(output)}(t) = t$ it can be shown that the output y^{est} of the neural network is equivalent to linear regression, i.e. $y^{est} = w_0 + \sum_k w_k x_k$, while if the transfer function of the hidden layer ($g^{(hidden)}$) is given by the sigmoid or tanh while the transfer function of the output layer ($g^{(output)}$) is linear the network is closely related to logistic regression. Artificial Neural Networks (ANN) with more than one hidden units can be much more flexible than linear and logistic regression, however, ANN are not guaranteed to find the optimal solutions. Therefore ANN are normally trained multiple times with different random initialization and the trained network with best training error selected.

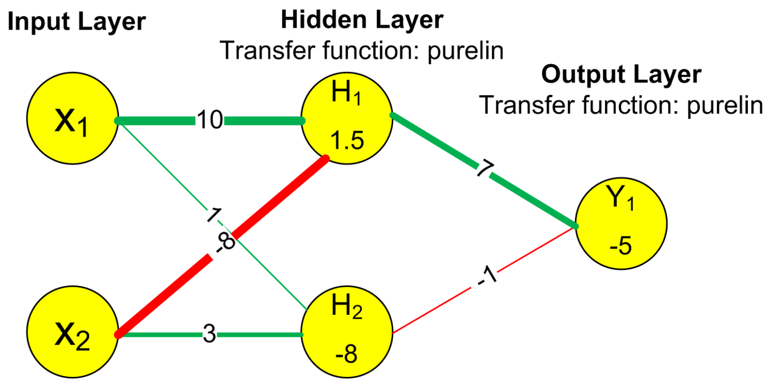

Figure 1: A learned network with two inputs, two hidden units and one outputs. Positive weights are indicated in green, negative in red. The bias, i.e. the constant for each unit is given by the numeric value inside the hidden and output unit. The transfer functions of the hidden layer and output layer are here linear.

In the following exercises we shall use a simple package offering neural networks functionality in Python. You can find the source code, installation hints, documentation and examples at the following site:

- Neurolab `http://code.google.com/p/neurolab/`

8.2.1 In figure 1 a network with two input units, two hidden units and one output unit has been trained based on linear functions as transfer function, i.e. the output of the first hidden unit is $h_1 = g^{(hidden)}(1.5 + 10x_1 - 8x_2)$ and the output of the second hidden unit is $h_2 = g^{(hidden)}(-8 + 1x_1 + 3x_2)$ with ($g^{(hidden)}(t) = t$). The final output of the neural network is given by $y^{est} = g^{(Output)}(-5 + 7h_1 - 1h_2)$ with ($g^{(output)}(t) = t$). What is the output of the network if $x_1 = 1$ and $x_2 = 0$ and what is the output if $x_1 = 0.5$ and $x_2 = 0.5$?)

8.2.2 We will use a simple data set to demonstrate the neural network. Use the `loadmat()` function to load `Data/xor.mat` Matlab datafile into Python. After examining the loaded dataset, examine and run the script `ex8_2_2.py`, which fits a neural network and uses k-fold crossvalidation to estimate the classification error. Observe the decision boundaries and learning curves.

Note that you will get quite different results every time you run the script. Can you explain why? Since there is no certainty a network will converge, you might need to rerun the script.
Usually, you will need to train multiple randomly initiated networks and use crossvalidation to select the best ones, as well as to optimize hyper-parameters (i.e. learning rate, number of hidden nodes). This process can be computationally expensive, hence in practice it is often executed in parallel on multi-processor machines and computational grids.

8.2.3 Try increase the number of hidden units to `n_hidden_units = 2`. Does the classification performance improve? Can you explain why?

8.2.4 Try change the number of hidden units in the network to `n_hidden_units = 10`. What happens to the decision boundaries of the learned neural networks? What are the benefits and drawbacks of including many hidden units in the network?

8.2.5 Load the wine data (load `Data/wine2` for data with outliers removed) and try and classify wine as red or white using the neural network classifier. Use the script `ex8_2_5.py` as a reference. What is the performance when `n_hidden_units=2`? Can you interpret how wine is classified as red and white wine? Try run the model with `n_hidden_units=1`, can you now interpret how the network predicts the type of wine?

8.2.6 Neural networks can also be used for regression (i.e. continuous output). Examine and run the script `ex8_2_6.py`. Try predict the alcohol content of wine for `n_hidden_units=2`. How well is the network able to predict the alcohol content of wine? Try and see if you can interpret how the trained networks determines the alcohol content of the wine? Try set the n_hidden_units=5, how well does the network predict the alcohol content of the wine?

8.3 Multiclass problems using ANNs and multinomial regression

We have previously used NaïveBayes, K-Nearest Neighbor and Decision trees for the classification of data with multiple classes. Logistic regression and artificial neural networks (ANN) can however also be extended to multiple classes by use of the softmax function given by $f_c(\boldsymbol{o}) = \frac{\exp(o_c)}{\sum_{c'} \exp(o_{c'})}$. Thus, o_c corresponds to the predictions made for the c^{th} class and all predictions are tied together through the softmax function. When training the model such that the output of the function $f_c(\boldsymbol{o})$ can be interpreted as the probability that the observation belongs to the c^{th} class. The softmax link function is for two class problems equivalent to the logit link function and this particular extension of logistic regression to multi-class is called multinomial regression.

8.3.1 Examine and run the scripts `ex8_3_1.py` and `ex8_3_2.py` . The scripts loads a synthetic multi-class classification data set and fits respectively a neural network to training data using the multi-class ANN and multinomial regression model based on tieing the outputs by the softmax function. Next, it computes the outputs of the trained classifier on the test data and classifies it according to the class having the highest probability. The script then computes the error rate and plots the decision boundaries of the trained model.

For evaluation, you can use one of the synthetic multiclass data sets as before, (`Data/synth1` ... `Data/synth4`).
Optional: As an additional task, try to use `ex8_3_2.py` to fit a multinomial regression model to the dataset `Data/synth3`. Notice the multinomial regression model is able to solve this problem; look at the weights and explain *how* this is accomplished (n.b. notice the vertical dimension does not matter).

8.4 Tasks for the report

Classify your data by an Artificial neural networks (ANN). Use cross-validation to select the number of hidden units in the hidden layer of the ANN. Try also to analyze your data in terms of a regression problem using ANN (select again for the optimal number of hidden units using cross-validation). If you have a multi-class problem use the multi-class ANN and multinomial regression (instead of logistic regression) based on tieing the outputs by the softmax function.

References