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 artificial neural networks (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"* as well as the files provided from DTU Learn.

8.1 Regularization

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.

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

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.

When regularizing a model as above we apply the same regularization strength to all features, and so it is important to standardize the features. Note that if we include a bias term (offset/intercept) we do not regularize it.

In the two following scripts, we will see how to regularize a model for predicting a continous target variable (using *linear* regression), and a model for classification (predicting a binary class label with *logistic* regression) using the above regularization method (also called ridge regression, or L2-regularization).

8.1.1 *Linear* regression model: inspect and run the script `ex8_1_1.py`. The script investigates 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? Try to inspect the weights obtained when using a very high regularization strength. What has happend to the value of the coefficients and what does the bias now represent? (try to compare the mean of the target variable in the training set to the parameters that have been fitted)

8.1.2 *Logistic* regression model: inspect and run the script `ex8_1_2.py`. The script loads the wine data (`Data/wine2`). The script standardizes the data based on the training set feat means and standard deviations. The data is then used to train a model that predicts whether a data point is a red or a white wine (see Exercise 5.2.6, for the unregularized counterpart). A suitable regularization strength is determined using hold-out cross-validation. The Wine dataset is relatively large, so we train on only a small part of the dataset (10 percent) to illustrate the effect of regularization on small datasets. You can try to change the percent used for training (try e.g. 50 % and 99 %). Run the script multiple times to see how much the subset affects the performance when reducing the training set size. How is the magnitude of the fitted parameters affected by the regularization strength?

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), or just neural networks, 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, ANNs are normally trained multiple times with different random initialization and the trained network with best training error selected.

We can use ANNs for both classification and regression. When we train an ANN to do regression, we do not a apply a tranfer function to the output node(s), and we train the network using a mean-square-error loss. However, when we train classification problems, we use a transfer function for the output nodes and a different loss. For instance, when doing binary classification, we can use a sigmoidal tranfer function in the last layer, and we would use a binary cross entropy loss. When doing multi-class classification, we would use e.g. a softmax transfer function and apply a cross entropy loss.

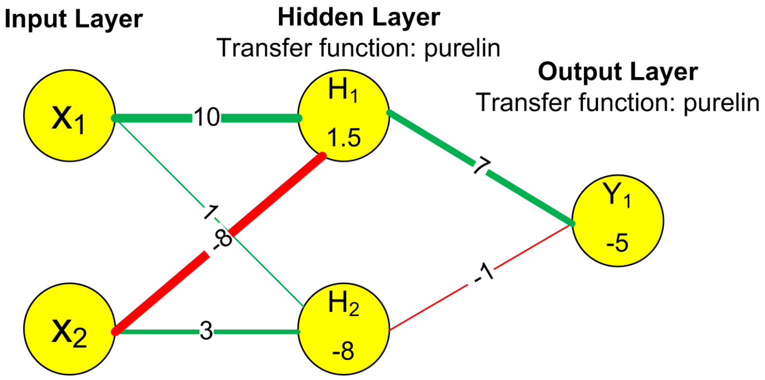

Figure 1: A learned network with two inputs, two hidden units and one outputs. Positive weights are indicated in green, negative in red. The bias, i.e. the constant for each unit is given by the numeric value inside the hidden and output unit. The transfer functions of the hidden layer and output layer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

In the following exercises we shall use an open source library offering neural networks functionality in Python called PyTorch. You can find the source code, installation hints, documentation and examples at: `https://pytorch.org/tutorials/` We recommend to install PyTorch with no GPU-support (no CUDA) by running the install commands outlined at `https://pytorch.org/` according to your operating system with the option "CPU" as "Compute Platform" (this will create a minimal installation without the cudatoolkit that requires quite a bit of space).

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 $x_2 = 0.5$?
Change the transfer function of the hidden layer to be the rectified linear function, i.e. $g^{(hidden)}(t) = \left\{ \begin{array}{l} t \text{ if } t > 0 \\ 0 \text{ otherwise} \end{array} \right\}$. This function is also called a rectifier, and when a node has this tranfer function we call it a rectified linear unit (ReLU). What is the output now 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 (loss as a function of training steps).
Script details:

- *We can import PyTorch using* `import torch`
- *To define a neural network we use* `torch.nn.Sequential`*. Each argument to this function is a part of a sequentially applied network. We can get a layer that maps M features to H hidden units by using* `torch.nn.Linear(M, H)`*. We can then apply a transfer function to the output of that layer by having the next argument be* `torch.nn.Tanh()` *(applying a hyperbolic tangent function). To train a binary classifier, we need a suitable final transfer function, which is for instance* `torch.nn.Sigmoid()`*.*
- *By using a lambda function we can get a new copy of the network each time we call* `model()`*, which we need since we're doing K-fold cross-validation and multiple replicates within each fold.*
- *Neural networks are trained by iteratively reducing a cost function/loss. We can define a binary cross entropy loss suitable for binary classification problems by using* `torch.nn.BCELoss()`*.*
- *PyTorch operates on objects called tensors, which are simply multi-dimensional matrices, and the way they are defined within PyTorch enables e.g. processing on GPUs, which speeds up training of large networks. We wont be using GPUs for training, but we need to define the inputs to the neural network as tensors, which we do using* `torch.tensor()`*. We can define the inputs to be of type floats by specyfing the datatype:* `dtype=torch.float`*.*
- *Training the specified network is done by calling* `train_neural_net`*. Type* `help(train_neural_net)`*. You should have a look at how the training is done by writing* `edit train_neural_net`*—this will take you to where the function is defined in the 02450 toolbox.*
- *The outputs of the network are the activation of the final unit passed through the sigmoidal tranfer function, resulting in a number between zero and one. To obtain class-estimates we can threshold the value at 0.5.*
- *Since PyTorch operates on PyTorch-tensors, we need to convert back to "regular" arrays before using e.g. numpy-operations, we can get a numpy version of a tensor a by* `b = a.data.numpy()`*.*
- *We can visualize the decision boundary of a classifier by using* `visualize_decision_boundary()`*. The function needs a lambda-function, which is function that takes as input a 2-dimensional dataset and outputs predictions.*
- *We can visualize a diagram of a trained network similar to Figure 1 by calling* `draw_neural_net`*. This functions needs to be supplied with the weights, biases and transfer functions supplied, which we can get by extracting the values from the* `torch.nn.Sequential`*-object that is returned from* `train_neural_net`*. These diagrams quickly become of little use with increasing input features and increasing layer sizes (number of hidden units).*
- *We control how many replicates/restarts we do when training a network through* `n_replicates`*, and we can set the maximum number of training steps that is done during training by* `max_iter`*.*

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 to 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? Script details:

- *When doing regression, we do not apply a transfer-function to the final layer. If we have a one-dimensional regression target, the last argument to* `torch.nn.Sequential` *when defining the model should be a* `torch.nn.Linear(n_hidden_units, n_out)`*, where* `n_out` *is equal to one.*
- *Similarly, when doing regression we do not train the network based on a cross entropy loss, but on a mean-square-error-loss (MSE loss). We can define such a loss function by using* `torch.nn.MSELoss()`*.*

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`. The scripts loads a synthetic multi-class classification data set and fits a neural network to training data using the multi-class ANN (`ex8_3_1.py`). 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`).

Script details:

- *When we train a network for C-classes classification problem, we set the last layer of the network to have C output neurons, that we also call logits.*
- *The softmax-transfer function can be used by* `torch.nn.Softmax`*. When defining this, we need to specify which dimension to normalize/link over. If its the second dimension (as is the case for the script), we specify* `dim=1`*—if we had used the first dimesion, the function would have normalized not over classes but over the number of observations.*
- *The appropriate loss function for a multi-class problem is a cross-entropy loss,* `torch.nn.CrossEntropyLo`

8.3.2 **Optional:** Use `ex8_3_2.py` to fit a multinomial regression model to the dataset. Multinomial regression models are based on tieing the outputs by the softmax function. Especially consider `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).

8.3.3 **Optional:** Just like we can regularize a linear and logistic regression, we can regularize multinomial regression. Use `ex8_3_3.py` to fit a regularized multinomial regression model. Investigate how a very high regularization strength affects the class label estimations (look at which class label is the mode in the training set and which label is estimated).

8.4 Tasks for the report

You are now in a position to address the remaining tasks for the report.
Regression, part a

1. Introduce a regularization parameter λ as discussed in chapter 14 of the lecture notes, and estimate the generalization error for different values of λ. Specifically, choose a reasonable range of values of λ (ideally one where the generalization error first drop and then increases), and for each value use $K = 10$ fold cross-validation (algorithm 5) to estimate the generalization error. Include a figure of the estimated generalization error as a function of λ in the report and briefly discuss the result.
2. Explain how the output, y, of the linear model with the lowest generalization error (as determined in the previous question) is computed for a given input $\mathbf{x}$. What is the effect of an individual attribute in $\mathbf{x}$ on the output, y, of the linear model? Does the effect of individual attributes make sense based on your understanding of the problem?

Regression, part b: In this section, we will compare three models: the regularized linear regression model from the previous section, an artificial neural network (ANN) and a baseline. We are interested in two questions: Is one model better than the other? Is either model better than a trivial baseline?. We will attempt to answer these questions with two-level cross-validation.

1. Implement two-level cross-validation (see algorithm 6 of the lecture notes). We will use 2-level cross-validation to compare the models with $K_1 = K_2 = 10$ folds[1]. As a baseline model, we will apply a linear regression model with no features, i.e. it computes the mean of y on the training data, and use this value to predict y on the test data.

   Make sure you can fit an ANN model to the data. As complexity-controlling parameter for the ANN, we will use the number of hidden units[2] h. Based on a few test-runs, select a reasonable range of values for h (which should include $h = 1$), and describe the range of values you will use for h and λ.

[1] If this is too time-consuming, use $K_1 = K_2 = 5$

[2] Note there are many things we could potentially tweak or select, such as regularization. If you wish to select another parameter to tweak feel free to do so.

Outer fold	ANN		Linear regression		baseline
i	h_i^*	E_i^{test}	λ_i^*	E_i^{test}	E_i^{test}
1	3	10.8	0.01	12.8	15.3
2	4	10.1	0.01	12.4	15.1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
10	3	10.9	0.05	12.1	15.9

Table 1: Two-level cross-validation table used to compare the three models

2. Produce a table akin to Table 1 using two-level cross-validation (algorithm 6 in the lecture notes). The table shows, for each of the $K_1 = 10$ folds i, the optimal value of the number of hidden units and regularization strength (h_i^* and λ_i^* respectively) as found after each inner loop, as well as the estimated generalization errors E_i^{test} by evaluating on $\mathcal{D}_i^{\text{test}}$. It also includes the baseline test error, also evaluated on $\mathcal{D}_i^{\text{test}}$. Importantly, you must re-use the train/test splits $\mathcal{D}_i^{\text{par}}, \mathcal{D}_i^{\text{test}}$ for all three methods to allow statistical comparison (see next section).

   Note the error measure we use is the squared loss *per observation*, i.e. we divide by the number of observation in the test dataset:

   $$E = \frac{1}{N^{\text{test}}} \sum_{i=1}^{N^{\text{test}}} (y_i - \hat{y}_i)^2$$

   Include a table similar to Table 1 in your report and briefly discuss what it tells you at a glance. Do you find the same value of λ^* as in the previous section?

3. Statistically evaluate if there is a significant performance difference between the fitted ANN, linear regression model and baseline using the methods described in chapter 11. These comparisons will be made pairwise (ANN vs. linear regression; ANN vs. baseline; linear regression vs. baseline). We will allow some freedom in what test to choose. Therefore, choose either:

   setup I (section 11.3): Use the paired t-test described in Box 11.3.4

   setup II (section 11.4): Use the method described in Box 11.4.1)

   Include p-values and confidence intervals for the three pairwise tests in your report and conclude on the results: Is one model better than the other? Are the two models better than the baseline? Are some of the models identical? What recommendations would you make based on what you've learned?

Classification: In this part of the report you are to solve a relevant classification problem for your data and statistically evaluate your result. The tasks will closely mirror what you just did in the last section. The three methods we will compare is a baseline, logistic regression, and **one** of the other four methods from below (referred to as *method 2*).

Logistic regression for classification. Once more, we can use a regularization parameter $\lambda \geq 0$ to control complexity

ANN Artificial neural networks for classification. Same complexity-controlling parameter as in the previous exercise

CT Classification trees. Same complexity-controlling parameter as for regression trees

Outer fold	Method 2		Logistic regression		baseline
i	x_i^*	E_i^{test}	λ_i^*	E_i^{test}	E_i^{test}
1	3	10.8	0.01	12.8	15.3
2	4	10.1	0.01	12.4	15.1
⋮	⋮	⋮	⋮	⋮	⋮
10	3	10.9	0.05	12.1	15.9

Table 2: Two-level cross-validation table used to compare the three models in the classification problem.

KNN k-nearest neighbor classification, complexity controlling parameter $k = 1, 2 \dots$

NB Naïve Bayes. As complexity-controlling parameter, we suggest the term $b \geq 0$ from section 11.2.1 of the lecture notes to estimate[3] $p(x = 1) = \frac{n^+ + b}{n^+ + n^- + 2b}$

1. We will compare logistic regression[4], *method 2* and a baseline. For logistic regression, we will once more use λ as a complexity-controlling parameter, and for *method 2* a relevant complexity controlling parameter and range of values. We recommend this choice is made based on a trial run, which you do not need to report. Describe which parameter you have chosen and the possible values of the parameters you will examine. The baseline will be a model which compute the largest class on the training data, and predict everything in the test-data as belonging to that class (corresponding to the optimal prediction by a logistic regression model with a bias term and no features).

2. Again use two-level cross-validation to create a table similar to Table 2, but now comparing the logistic regression, *method 2*, and baseline. The table should once more include the selected parameters, and as an error measure we will use the error rate:

$$E = \frac{\{\text{Number of misclassified observations}\}}{N^{\text{test}}}$$

Once more, make sure to re-use the outer validation splits to admit statistical evaluation. Briefly discuss the result.

3. Perform a statistical evaluation of your three models similar to the previous section. That is, compare the three models pairwise. We will once more allow some freedom in what test to choose. Therefore, choose either:

setup I (section 11.3): Use McNemar's test described in Box 11.3.2)

setup II (section 11.4): Use the method described in Box 11.4.1)

Include p-values and confidence intervals for the three pairwise tests in your report and conclude on the results: Is one model better than the other? Are the two models better than the baseline? Are some of the models identical? What recommendations would you make based on what you've learned?

4. Train a logistic regression model using a suitable value of λ (see previous exercise). Explain how the logistic regression model make a prediction. Are the same features deemed relevant as for the regression part of the report?

[3] In Python, use the `alpha` parameter in `sklearn.naive_bayes` and in R, use the `laplacian` parameter to `naiveBayes`. We do *not* recommend NB for Matlab users, as the implementation is somewhat lacking.

[4] in case of a multi-class problem, substitute logistic regression for multinomial regression

Discussion:

1. Include a discussion of what you have discovered in the regression and classification part of the report.
2. If your data has been analyzed previously (which will be the case in nearly all instances), find a study which uses it for classification, regression or both. Discuss how your results relate to those obtained in the study. If your dataset has not been published before, or the articles are irrelevant/unobtainable, this question may be omitted but make sure you justify this is the case.

References