Assignment 5

Submission deadline:

- problems 1-5: Tue 22.12, We 16.12 or Fr 18.12 depending on your group (essentially last class before Christmas).
- problems 6-12: Tue 12.1.16, We 6.1.16 or Fr 8.1.16 depending on your group (essentially first class after Christmas)

Points: 17 + 10 bonus

In this assignment we will study the effect of regularization on model performance.
Please note: this assignment comes with starter code.

1 Introduction

We are solving the problem of finding a general relation describing all of certain data (e.g. recognizing all handwritten digits) based on a finite dataset available to us. Therefore our goal is not to achieve a 100% perfect fit to the training data*. Instead, we are concerned about generalization, that is being able to find relations that are not only valid on the data set, but also on other, unknown to us data.

During lectures we have learned about two problems related to finding a general relation based on a training data set:

a) **bias** – which occurs when the model we are training cannot express the relation in the data because it is too weak, or in other words the hypothesis space we consider is too narrow.

   Intuitively, bias is similar to systematic errors in physical measurements – when the model has a high bias, it simply cannot express the correct relation.

b) **variance** – which is variability of the trained model due to variation in the data set, randomness in the training algorithm etc.

   When variance is high, repeated training of the model will result in very different solutions with varying generalization errors. Intuitively, variance is similar to the random errors in physical measurements - repeated model training (possibly on new training data) can help.

In the next few lectures we will discover that learning from data is not possible without bias, and that the art of machine learning is striking a balance between the error due to bias and the error due to variance. We must carefully constrain the model capacity, on the one hand ensuring that it is able to represent a sufficiently close approximation to the real relation in the data, while on the other hand maintaining low variability and randomness of the effects of training. We will call the action of constraining the capacity *model regularization*.

A very generic form of regularization, called *weight decay*, consists of penalizing the model for the magnitude of its weights. Intuitively weight decay says that we should prefer models (linear regression or neural networks) whose weights are small. Do you remember how increasing K in K-nearest neighbors smoothed out the output of a classifier because it was averaging several data points? Intuitively, weight decay also smooths the classifier's output. You can recall from the gradient backpropagation equations (please see e.g. the previous assignment) that the gradient with respect to the inputs is proportional to the weights connected to the inputs. Therefore, the smaller the weights the smaller the gradient, which in turn limits how fast the classifier's output can change for a change of its inputs.

Finally, you will need to tune a neural network to perform well on MNIST (the validation and test errors should be below 2%). To achieve this error rate you will need to tune:

a) number of layers and the number of neurons in each layer,

*In fact achieving 100% accuracy on the training data is easy – a 1-nearest neighbors classifier scores 100% on the training set.

b) initialization method,

c) learning rate and momentum constant schedule,

d) weight decay coefficient for neurons.

2 Ridge regression

Minimization of square of a model's weights is a very popular regularization, called by many names: weight decay, ridge regression, Tikhonov regularization, or L2 regularization.

Unregularized linear regression predicts the value $y \in \mathbb{R}$ as a function of $x \in \mathbb{R}^n$ expressed by the formula $y = \Theta^T x$. The parameters $\Theta \in \mathbb{R}^{n+1}$ (we assume that x is extended with $x_0 = 1$) are chosen to minimize:

$$\Theta^* = \arg\min_{\Theta} J(\Theta) = \arg\min_{\Theta} \frac{1}{m} \sum_{j=1}^{m} (y^{(j)} - \Theta^T x^{(j)})^2. \tag{1}$$

To regularize the model we add to the cost term $J(\Theta)$ a regularizing term $R(\Theta)$ penalizing the magnitude of parameters:

$$J_R(\Theta) = J(\Theta) + \lambda R(\Theta) = \frac{1}{m} \sum_{j=1}^{m} (y^{(j)} - \Theta^T x^{(j)})^2 + \lambda \sum_{i=0}^{n} \Theta_i^2, \tag{2}$$

where: λ is a constant typically set experimentally by cross-validation. Please note, that typically in a neural network one does not penalize the magnitude of biases.

Problem 1. **[2p]** Implement the closed form solution for ridge regularized linear regression.

Then analyze datasets sampled using the following procedure:

a) $x \propto U(0; 1)$: x is sampled uniformly from the $0 - 1$ range.

b) $y \propto \mathcal{N}(\mu = 1 + 2x - 5x^2 + 4x^3, \sigma = 0.1)$: then y is sampled from the Normal distribution with mean $\mu = 1 + 2x - 5x^2 + 4x^3$ and standard deviation 0.1

Repeat 30 times an experiment in which you sample a new training dataset, then fit polynomials of degree 0 to 14 and use λ value from the set $\{0, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$.

Plot the mean training and testing errors. Indicate regions of high bias and of high variance. What is the effect of increasing λ?

3 SVM

3.1 Installing dependencies

When using lab computers, you are all set. When using your home computer you need to install CVXOPT (Python Software for Convex Optimization):

a) for Linux Anaconda python you can install cvxopt running `conda install cvxopt`.

b) On Windows download a package from http://www.lfd.uci.edu/~gohlke/pythonlibs/#cvxopt. Under WinPython, you can use the WinPython control panel to intall the package. On Anconda simply unzip the downloaded file (it's a zip file in disguise) and place the contents somewhere under your PYTHONPATH environment variable.

3.2 Theory

A linear SVM assigns points $x^{(i)} \in \mathbb{R}^n$ to one of two classes, $y^{(i)} \in \{-1, 1\}$ using the decision rule:

$$y = \text{signum}(w^T x + b). \tag{3}$$

SVM training consists of finding weights $w \in \mathbb{R}^n$ and bias $b \in \mathbb{R}$ that maximize the separation margin. This corresponds to solving the following quadratic optimization problem:

$$\begin{aligned} &\min_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{m} \xi_i \\ &\text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i \ \ \forall_i \\ &\quad \xi_i \geq 0 \ \ \forall_i. \end{aligned} \tag{4}$$

Problem 2. **[2p]** Load the iris dataset. Implement a linear SVM that separates the Virginica from the Versicolor class using the petal length and petal width features. Use $C = 10$. Implement the SVM twice. First use the LIBSVM library function (on Python this is available as `sklearn.svm.SVM`). Plot the obtained decision boundary and the location and weight (coefficients α assigned to support vectors).

Then use `cvxopt.solvers.qp` quadratic programming solver. You will need to define the matrices that define the problem. Compare the obtained solutions. Extract the support vectors from the LIBSVM solution and plot the support vectors.

Problem 3. **[2p]** Repeat 100 bootstrap experiments to establish the effect of constant C on SVM.

For each experiment do the following:

a) Sample (with replacement) a bootstrap dataset equal in size to the training dataset. This will be this experiment's training dataset.

b) Prepare the experiment's testing dataset by using samples not inluded in the bootstrap dataset.

c) For all C from the set $\{10^{-4}, 10^{-3.5}, 10^{-3.}, \ldots, 10^{6}\}$ fit a nonlinear SVM (Gaussian kernel, called `rbf` in LIBSVM using the default γ) and record the training and testing errors.

Analyze boxplot of errors in function of C. Can you see its influence on the training and testing error, as well as on the testing error variability?

Problem 4. **[3p bonus]** Implement a nonlinear SVM by solving the dual problem using the qp solver. Compare results with LIBSVM.

Problem 5. **[2p bonus]** Compare two ways to implement a multi-class SVM: by training "1-vs-1" classifier for each class combination, and by training a "1-vs-rest" classifier for each clas. See http://www.csie.ntu.edu.tw/~cjlin/papers/multisvm.pdf for details.

4 Neural network regularization

You are provided starter code for a modular implementation of a feedforward neural network. Your general task is to fill in the blanks in the code, and train a network on MNIST that would achieve below 2% testing errors.

Problem 6. **[2p]** Fill in the blanks in the network implementation code. Make sure that the network can be trained on the Iris dataset to reach 100% training accuracy.

Problem 7. [**2p**] Implement momentum in the SGD code.

Problem 8. [**1p**] Implement learning rate scheduling in the SGD code.

Problem 9. [**1p**] Implement weight decay in the SGD code. One way to do it is to look at all parameter names, and it they are "W", add a term to the loss gradient with respect to the weight.

Problem 10. [**5p**] Tune the network to reach below 1.9% error rate on the validation set. This should result in a test error below 2%. To tune the network you will need to:

- choose the number of layers (more than 1, less than 5),
- choose the number of neurons in each layer (more than 100, less than 5000),
- pick proper weight initialization,
- pick proper learning rate schedule (need to decay over time, good range to check on MNIST is about 1e-2 ... 1e-1 at the beginning and half of that after 10000 batches),
- pick a momentum constant (probably a constant one will be OK).

Problem 11. [**2p bonus**] Implement norm constraints, i.e. limit the total norm of connections incoming to a neuron. In our case, this corresponds to clipping the norm of *rows* of weight matrices. An easy way of implementing it is to make a gradient step, then look at the norm of rows and scale down those that are over the threshold (this technique is called "projected gradient descent").

Problem 12. [**3p bonus**] Implement a *dropout* layer and try to train a network getting below 1.5% test error rates with dropout (the best result is below 1% for dropout!). Details: `http://arxiv.org/pdf/1207.0580.pdf`.