Solutions Exercise 2

Question 1

The gradient descent algorithm can be implemented with three simple functions: One computes the loss, another computes the gradient, and finally the algorithm itself just iteratively calls these. A practical implementation solving this problem is provided in Moodle.

Some practical hints and observations:

1. It is a good idea to write the loss and gradient so that they support computing losses with and without regularization
2. The loss for evaluating alternative choices for the regularization parameter is the validation loss *without* the regularizing part
3. You should not use early-stopping together with regularization, at least not in this exercise – both prevent overfitting so mixing them together makes it hard to understand what is happening
4. For early stopping you should stop optimization already when the validation loss decreases by small enough amount – waiting until it starts increasing might mean the algorithm never stops if there is no over-fitting

Figure 1 illustrates the convergence of the algorithm for ERM using the 10/90 and 90/10 training/validation split. The total number of iterations taken until convergence are listed in Table 1; early-stopping naturally takes less iterations in the first case where the model soon starts to overfit.

The results for the regularized risk minimization (RRM) algorithm are shown in Figure 2, plotting for both training/validation splits the corresponding losses for the set

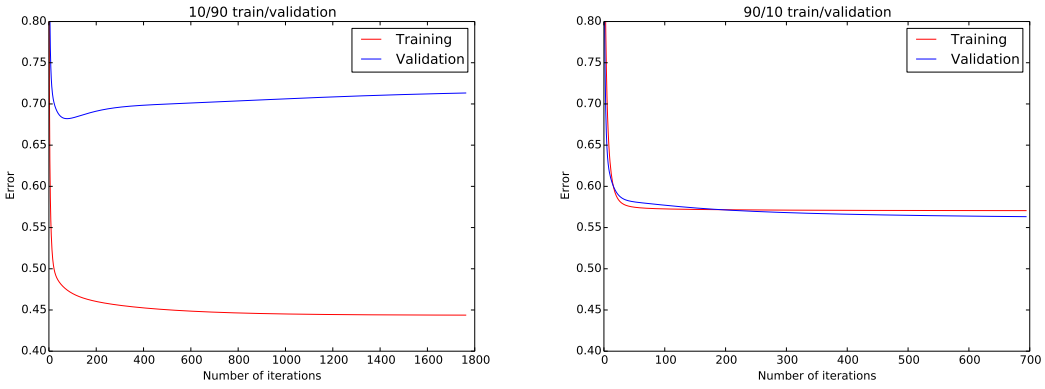

Figure 1: Empirical risk and the validation error as function of the iteration for the two splits. With small training data the model clearly overfits, with the larger one not.

spilt	ERM	ES	RRM
10/90	1762	76	229
90/10	694	694	491

Table 1: The numbers for the iterations taken until convergence. Your numbers for early-stopping might be slightly different if you used a different criterion for stopping.

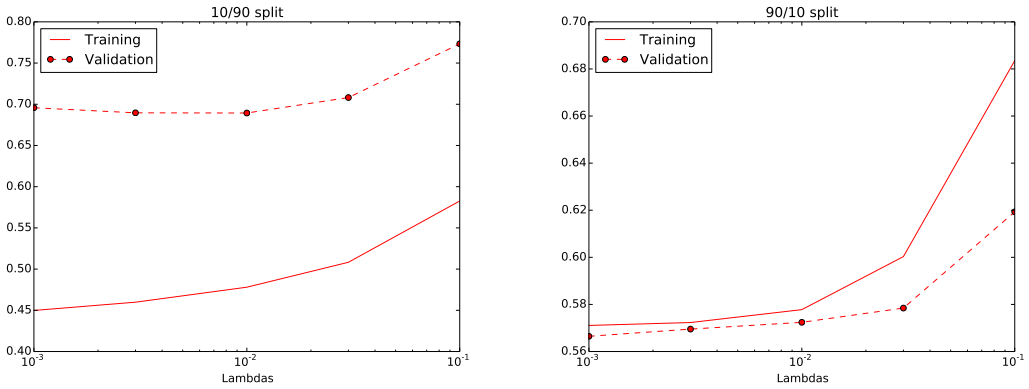

Figure 2: The validation losses for different values of the regularization parameter. The x-axis is drawn on logarithmic scale for improved visualization. The plot also shows the training errors – this was not requested and you would not typically look at them, but they are included here to show that with stronger regularization the model fits the training data worse.

% of data used for training	ERM	ES	RRM
10	0.827	0.791	0.796
90	0.702	0.702	0.702

Table 2: The test errors for the three different solutions for the two training/validation splits. The values in the table correspond to loss over test data $L((x_{test}, y_{test}), \theta)$, and lower loss means better accuracy.

of alternative regularization parameters. We see right away that with the small training data we need to regularize the solution, but with the bigger training data setting λ to a very small value is optimal.

Finally the estimated test errors on the separate test data are provided in Table 2. For small training data ERM overfits, but ES and RRM are able to regularize the problem and hence reach lower test error. For the larger training sets all methods are quite comparable, demonstrating that having more data help avoiding overfitting. Too small validation data (the case of 90/10) could result in noisy estimates for the validation error and could hence cause problems in choosing the regularization parameter or determining early-stopping, but in this example this did not really happen.

Question 2

In this exercise we studied different gradient-based optimization techniques. The implementations are linked from the exercise page.

First, we looked at the choice of the step-size for regular gradient descent, running the algorithm until convergence (determined with the same criterion as in the previous exercise). Table 3 shows the number of iterations required, showing that too small value results in excessively large number of iterations. However, with too large value the algorithm diverges.

Step-size	0.3	0.2	0.1	0.05	0.01	0.001
Iterations	NaN	45	93	187	878	9693

Table 3: The number of iterations taken until convergence for the standard gradient descent method. With $\alpha = 0.3$ the algorithm diverges, whereas with $\alpha < 0.001$ it converges very slowly. A good choice would be around 0.1 and we have chosen this step to visualize the contour plots.

Figure 3 illustrates gradient descent (with $\alpha = 0.1$) and the Newton's method. The Newton's algorithm converges in one step, since the constant Hessian matrix corresponds to a global transformation of the loss function into a sphere where the gradient points directly to the minimum and the step-size is automatically chosen to reach that point. The SGD algorithm is illustrated in Figure 4. With mini-batch of one the gradients are clearly noisier and Adagrad does not seem to really work in that case either. Using bigger mini-batches is always a good idea; the SGD algorithm that takes just one sample at a time is more of theoretical interest. Here we set $\alpha = 0.05$; SGD with one sample batches diverges already with $\alpha = 0.1$.

For gradient descent we recognized the solution as parameters that result in zero gradient. For SGD this does not hold – we do not have the whole gradient but just an estimate for it. In practice one can simply run SGD for some fixed number of iterations since the step-size should converge to zero. A reasonable alternative is to keep track of running averages of the gradient and stop optimization when that is sufficiently close to zero. Here we just ran SGD for 100 iterations.

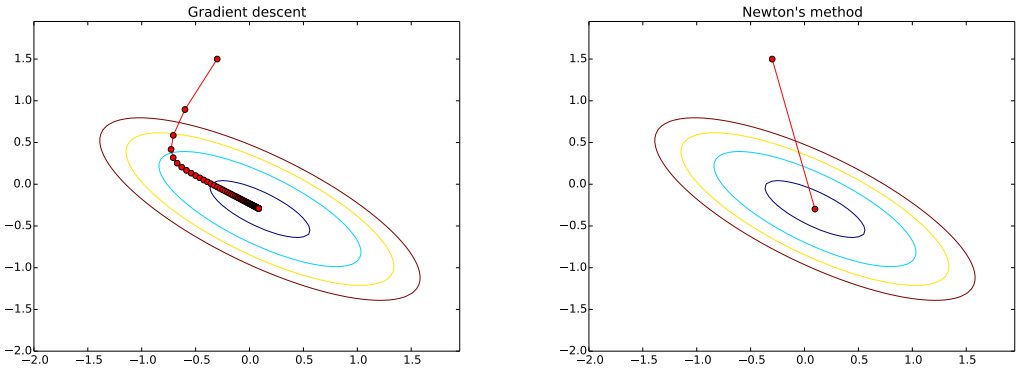

Figure 3: Regular gradient descent (left here $\alpha = 0.1$) converges steadily but slowly, whereas the Newtons' method (right) converges in a single step but requires computing the Hessian matrix.

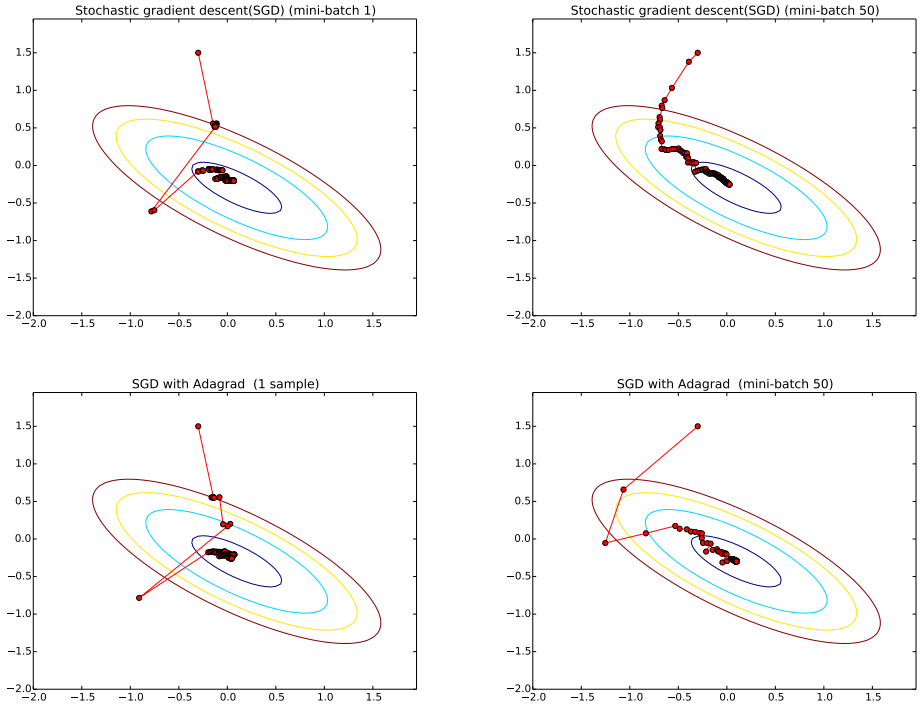

Figure 4: Top row: SGD with constant step-size. Bottom row: SGD with Adagrad. Using bigger mini-batch (right plots) clearly helps. SGD with constant step-size is not really a valid algorithm since the step-size should decrease during optimization to guarantee convergence, but is included here for reference – in practice we should anyway use Adagrad or other adaptive algorithm instead of simple decreasing scheme.

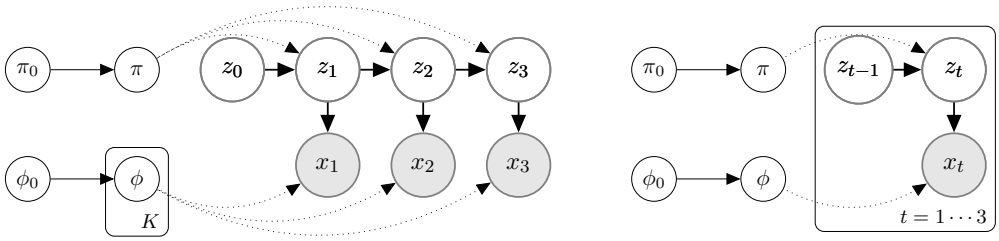

Figure 5: The left plot is how HMM would usually be drawn. The right plot is close to what many of you produced; it would typically be understood correctly as well.

Question 3

One of the goals of this exercise was to show that the rules of drawing plate diagrams are not always very clear. The most important thing is that all the dependencies are correctly depicted, and often slight modifications of the "correct" plate would also be understood correctly.

a) The model in Figure 5 is called Hidden Markov Model, which is a model for time series data that describes the dynamics at the level of latent variables z_t. The transition probabilities $P(z_t \mid z_{t-1})$ control the dynamics and the emission probabilities $P(x_t \mid z_t)$ tell how the observations are generated given the latent variables. The parameters of those are inside a plate over K since there are K discrete choices z_k makes.

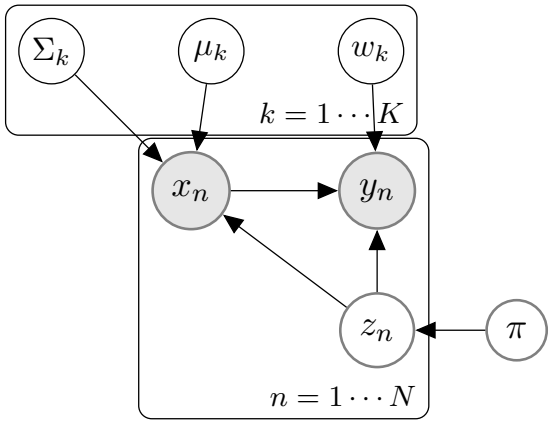

Figure 6: a) Mixture of supervised models. The latent variables z_n associated with each sample (x_n, y_n) pick which mixture component is used to model both the input x_n and the conditional relationship between the input and the output y_n.

b) The model in Figure 6 corresponds to a mixture of some supervised models. Based on the illustration alone we cannot tell whether the supervised models are regressors or classifiers. Note that there needs to be an arrow from z to both x and y; it is required

for specifying the correct distributions even though we did not explicitly condition on z in the probability distribution – this dependency comes from the part where we raise $p(x|\mu_k, \Sigma_k)$ (and other similar terms) to the power of $\mathbb{I}(z_n = k)$, which is one way of writing that z_n chooses which of the K alternatives to use. The same part makes the two products – over N and over K – to separate and hence we do not need two nested plates.

c) The illustration was taken from Klami et al. Group factor analysis (IEEE Transactions on Neural Networks and Learning Systems 2015), and corresponds to model with the same name. It is a factor analysis model for the join analysis of M groups of variables with additional structure for the factor-loading matrices $W^{(m)}$. The joint likelihood is written as

$$\begin{aligned} p(U, V, W, \alpha, \tau, x, z) =& p(U)p(V)p(\tau) \\ & \prod_{m=1}^{M} p(W^{(m)} \mid \alpha_m)p(\alpha_m \mid U, V) \\ & \prod_{n=1}^{N} \left[p(z_n) \prod_{m=1}^{M} p(x_n^{(m)} \mid z_n, \tau, W^{(m)}) \right] \end{aligned}$$

Note how $p(x_n^{(m)} \mid \cdot)$ needs to be inside products over N and M since x is inside two nested plates. Otherwise we get the right factorization by simply writing down the conditional densities with the parents on the right and child on the left, putting them inside products whenever needed.