5 Optimization algorithms[1]

5.1 Introduction

We saw in Chapter 4 that the core problem in machine learning is parameter estimation (aka model fitting). This requires solving an **optimization problem**, where we try to find the values for a set of variables $\boldsymbol{\theta} \in \Theta$, that minimize a scalar-valued **loss function** or **cost function** $\mathcal{L} : \Theta \to \mathbb{R}$:

$$\boldsymbol{\theta}^* \in \operatorname*{argmin}_{\boldsymbol{\theta} \in \Theta} \mathcal{L}(\boldsymbol{\theta}) \tag{5.1}$$

We will assume that the **parameter space** is given by $\Theta \subseteq \mathbb{R}^D$, where D is the number of variables being optimized over. Thus we are focusing on **continuous optimization**, rather than **discrete optimization**.

If we want to *maximize* a **score function** or **reward function** $R(\boldsymbol{\theta})$, we can equivalently minimize $-R$. We will use the term **objective function** to refer generically to a function we want to maximize or minimize. An algorithm that can find an optimum of an objective function is often called a **solver**.

In the rest of this chapter, we discuss different kinds of solvers for different kinds of objective functions, with a focus on methods used in the machine learning community. For more details on optimization, please consult some of the many excellent textbooks, such as [KW19b; BV04; NW06; Ber15; Ber16] as well as various review articles, such as [BCN18; Sun+19; PPS18].

5.1.1 Local vs global optimization

A point that satisfies Eq. (5.1) is called a **global optimum**. Finding such a point is called **global optimization**.

In general, finding global optima is computationally intractable [Neu04]. In such cases, we will just try to find a **local optimum**. For continuous problems, this is defined to be a point $\boldsymbol{\theta}^*$ which has lower (or equal) cost than "nearby" points. Formally, we say $\boldsymbol{\theta}^*$ is a **local minimum** if

$$\exists \delta > 0,\ \forall \boldsymbol{\theta} \in \Theta : ||\boldsymbol{\theta} - \boldsymbol{\theta}^*|| < \delta,\ \mathcal{L}(\boldsymbol{\theta}^*) \leq \mathcal{L}(\boldsymbol{\theta}) \tag{5.2}$$

A local minimum could be surrounded by other local minima with the same objective value; this is known as a **flat local minimum**. A point is said to be a **strict local minimum** if its cost is

1. This chapter benefited from contributions from Frederik Kunstner, Si Yi Meng, Aaron Mishkin, Sharan Vaswani, and Mark Schmidt.

Summary of Comments on pml1.pdf

Page: 107

Author: petercerno Subject: Comment on Text Date: 08.01.21, 12:26:39

Nit: This might look as if the RHS holds for all \theta.
I would change it to sth like this:
\forall \theta \in \Theta s. t. ... : ...

information. All of these algorithms require that the user specify a starting point $\boldsymbol{\theta}_0$. Then at each iteration t, they perform an update of the following form:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta_t \mathbf{d}_t \tag{5.7}$$

where η_t is known as the **step size** or **learning rate**, and $\mathbf{d}_t$ is a **descent direction**, such as the **gradient** $\mathbf{g}_t = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}_t}$. These update steps are continued until the method reaches a stationary point, where the gradient is zero.

5.2.1 Descent direction

We say that a direction $\mathbf{d}$ is a **descent direction** if there is a small enough (but nonzero) amount η we can move in direction $\mathbf{d}$ and be guaranteed to decrease the function value. Formally, we require that there exists an $\eta_{\max} > 0$ such that

$$\mathcal{L}(\boldsymbol{\theta} + \eta \mathbf{d}) < \mathcal{L}(\boldsymbol{\theta}) \tag{5.8}$$

for all $0 < \eta < \eta_{\max}$. The gradient at the current iterate,

$$\mathbf{g}_t \triangleq \nabla \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}_t} = \nabla \mathcal{L}(\boldsymbol{\theta}_t) = \mathbf{g}(\boldsymbol{\theta}_t) \tag{5.9}$$

points in the direction of maximal increase in f, so the negative gradient is a descent direction. It can be shown that any direction $\mathbf{d}$ is also a descent direction if the angle θ between $\mathbf{d}$ and $-\mathbf{g}_t$ is less than 90 degrees and satisfies

$$\mathbf{d}^T \mathbf{g}_t = ||\mathbf{d}||\, ||\mathbf{g}_t||\, \cos(\theta) < 0 \tag{5.10}$$

It seems that the best choice would be to pick $\mathbf{d}_t = -\mathbf{g}_t$. This is known as the direction of **steepest descent**. However, this can be quite slow. We consider faster versions later.

5.2.2 Step size (learning rate)

In machine learning, the sequence of step sizes $\{\eta_t\}$ is called the **learning rate schedule**. There are several widely used methods for picking this, some of which we discuss below. (See also Sec. 5.4.3, where we discuss schedules for stochastic optimization.)

5.2.2.1 Constant step size

The simplest method is to use a constant step size, $\eta_t = \eta$. However, if it is too large, the method may fail to converge, and if it is too small, the method will converge but very slowly.

For example, consider the convex function

$$\mathcal{L}(\boldsymbol{\theta}) = 0.5(\theta_1^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2 \tag{5.11}$$

Let us pick as our descent direction $\mathbf{d}_t = \mathbf{g}_t$. Fig. 5.4 shows what happens if we use this descent direction with a fixed step size, starting from $(0, 0)$. In Fig. 5.4(a), we use a small step size of $\eta = 0.1$; we see that the iterates move slowly along the valley. In Fig. 5.4(b), we use a larger step size $\eta = 0.6$;

Page: 111

Author: petercerno Subject: Comment on Text Date: 08.01.21, 12:27:55
Nit: negative gradient -g_t

Author: petercerno Subject: Comment on Text Date: 08.01.21, 12:28:24
-g_t

this case, the momentum term is updated using the gradient at the predicted new location,

$$\mathbf{m}_{t+1} = \beta \mathbf{m}_t - \eta_t \nabla \mathcal{L}(\boldsymbol{\theta}_t + \beta \mathbf{m}_t) \tag{5.29}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{m}_{t+1} \tag{5.30}$$

This explains why the Nesterov accelerated gradient method is sometimes called Nesterov momentum. It also shows how this method can be faster than standard momentum: the momentum vector is already roughly pointing in the right direction, so measuring the gradient at the new location, $\boldsymbol{\theta}_t + \beta \mathbf{m}_t$, rather than the current location, $\boldsymbol{\theta}_t$, can be more accurate.

The Nesterov accelerated gradient method is provably faster than steepest descent for convex functions when β and η_t are chosen appropriately. It is called "accelerated" because of this improved convergence rate, which is optimal for gradient-based methods using only first-order information when the objective function is convex and has Lipschitz-continuous gradients. In practice, however, using Nesterov momentum can be slower than steepest descent, and can even unstable if β or η_t are misspecified.

5.3 Second-order methods

Optimization algorithms that only use the gradient are called **first-order** methods. They have the advantage that the gradient is cheap to compute and to store, but they do not model the curvature of the space, and hence they can be slow to converge, as we have seen in Fig. 5.5. **Second-order** optimization methods incorporate curvature in various ways (such via the Hessian) which may yield faster convergence. We discuss some of these methods below.

5.3.1 Newton's method

The classic second-order method is **Newton's method**. This consists of updates of the form

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \mathbf{H}_t^{-1} \mathbf{g}_t \tag{5.31}$$

where

$$\mathbf{H}_t \triangleq \nabla^2 \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}_t} = \nabla^2 \mathcal{L}(\boldsymbol{\theta}_t) = \mathbf{H}(\boldsymbol{\theta}_t) \tag{5.32}$$

is assumed to be positive-definite to ensure the update is well-defined. The pseudo-code for Newton's method is given in Algorithm 1. The intuition for why this is faster than gradient descent is that the matrix inverse $\mathbf{H}^{-1}$ "undoes" any skew in the local curvature, converting a topology like Fig. 5.5a to one like Fig. 5.5b.

This algorithm can be derived as follows. Consider making a second-order Taylor series approximation of $\mathcal{L}(\boldsymbol{\theta})$ around $\boldsymbol{\theta}_t$:

$$\mathcal{L}_{\text{quad}}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}_t) + \mathbf{g}_t^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_t) + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_t)^\top \mathbf{H}_t (\boldsymbol{\theta} - \boldsymbol{\theta}_t) \tag{5.33}$$

The minimum of $\mathcal{L}_{\text{quad}}$ is at

$$\boldsymbol{\theta} = \boldsymbol{\theta}_t - \mathbf{H}_t^{-1} \mathbf{g}_t \tag{5.34}$$

Page: 116

Author: petercerno Subject: Comment on Text Date: 08.01.21, 12:28:46
as

This method is known as **stochastic gradient descent** or **SGD**. As long as the gradient estimate is unbiased, then this method will converge to a stationary point, providing we decay the step size η_t at a certain rate, as we discuss in Sec. 5.4.3.

5.4.1 Application to finite sum problems

SGD is very widely used in machine learning. To see why, recall from Sec. 4.3 that many model fitting procedures are based on empirical risk minimization, which involve minimizing the following loss:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} \ell(\mathbf{y}_n, f(\mathbf{x}_n; \boldsymbol{\theta}_t)) = \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_n(\boldsymbol{\theta}_t) \tag{5.59}$$

This is called a **finite sum problem**. The gradient of this objective has the form

$$\mathbf{g}_t = \frac{1}{N}\sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} \mathcal{L}_n(\boldsymbol{\theta}_t) = \frac{1}{N}\sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} \ell(\mathbf{y}_n, f(\mathbf{x}_n; \boldsymbol{\theta}_t)) \tag{5.60}$$

This requires summing over all N training examples, and thus can be slow if N is large. Fortunately we can approximate this by sampling a **minibatch** of $B \ll N$ samples to get

$$\mathbf{g}_t \approx \frac{1}{|\mathcal{B}_t|}\sum_{n\in\mathcal{B}_t} \nabla_{\boldsymbol{\theta}} \mathcal{L}_n(\boldsymbol{\theta}_t) = \frac{1}{|\mathcal{B}_t|}\sum_{n\in\mathcal{B}_t} \nabla_{\boldsymbol{\theta}} \ell(\mathbf{y}_n, f(\mathbf{x}_n; \boldsymbol{\theta}_t)) \tag{5.61}$$

where $\mathcal{B}_t$ is a set of randomly chosen examples to use at iteration t.[3] This is an unbiased approximation to the empirical average in Eq. (5.60). Hence we can safely use this with SGD.

Although the theoretical rate of convergence of SGD is slower than batch GD (in particular, SGD has a sublinear convergence rate), in practice SGD is often faster, since the per-step time is much lower [BB08; BB11]. To see why SGD can make faster progress than full batch GD, suppose we have a dataset consisting of a single example duplicated K times. Batch training will be (at least) K times slower than SGD, since it will waste time computing the gradient for the repeated examples. Even if there are no duplicates, batch training can be wasteful, since early on in training the parameters are not well estimated, so it is not worth carefully evaluating the gradient.

5.4.2 Example: SGD for fitting linear regression

In this section, we show how to use SGD to fit a linear regression model. Recall from Sec. 4.2.7 that the objective has the form

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{n=1}^{N} (\mathbf{x}_n^\top \boldsymbol{\theta} - y_n)^2 = \frac{1}{2N}||\mathbf{X}\boldsymbol{\theta} - \mathbf{y}||_2^2 \tag{5.62}$$

3. In practice we usually sample $\mathcal{B}_t$ without replacement. However, once we reach the end of the dataset (i.e., after a single training **epoch**), we can perform a random shuffling of the examples, to ensure that each minibatch on the next epoch is different from the last. This version of SGD is analyzed in [HS19].

Page: 123

Author: petercerno Subject: Comment on Text Date: 08.01.21, 12:29:31
\theta_t

An alternative to using heuristics for estimating the learning rate is to use line search (Sec. 5.2.2.2). This is tricky when using SGD, because the noisy gradients make the computation of the Armijo condition difficult [CS20]. However, [Vas+19] show that it can be made to work if the variance of the gradient noise goes to zero over time. This can happen if the model is sufficient flexible that it can perfectly interpolate the training set.

5.4.4 Iterate averaging

The parameter estimates produced by SGD can be very unstable over time. To reduce the variance of the estimate, we can compute the average using

$$\overline{\boldsymbol{\theta}}_t = \frac{1}{t}\sum_{i=1}^{t}\boldsymbol{\theta}_i = \frac{1}{t}\boldsymbol{\theta}_t + \frac{t-1}{t}\overline{\boldsymbol{\theta}}_{t-1} \tag{5.66}$$

where $\boldsymbol{\theta}_t$ are the usual SGD iterates. This is called **iterate averaging** or **Polyak-Ruppert averaging** [Rup88].

In [PJ92], they prove that the estimate $\overline{\boldsymbol{\theta}}_t$ achieves the best possible asymptotic convergence rate among SGD algorithms, matching that of variants using second-order information, such as Hessians.

This averaging can also have statistical benefits. For example, in [NR18], they prove that, in the case of linear regression, this method is equivalent to ℓ_2 regularization (i.e., ridge regression).

Rather than an exponential moving average of SGD iterates, **Stochastic Weight Averaging** (SWA) [Izm+18] uses an *equal* average in conjunction with a modified learning rate schedule. In contrast to standard Polyak-Ruppert averaging, which was motivated for faster convergence rates, SWA exploits the flatness in objectives used to train deep neural networks, to find solutions which provide better generalization.

5.4.5 Variance reduction

In this section, we discuss various ways to reduce the variance in SGD. In some cases, this can improve the theoretical convergence rate from sublinear to linear (i.e., the same as full-batch gradient descent) [SLRB17; JZ13; DBLJ14]. These methods reduces the variance of the gradients, rather than the parameters themselves and are designed to work for finite sum problems.

5.4.5.1 SVRG

The basic idea of **stochastic variance reduced gradient** (**SVRG**) [JZ13] is to use a control variate, in which we estimate a baseline value of the gradient based on the full batch, which we then use to compare the stochastic gradients to.

More precisely, ever so often (e.g., once per epoch), we compute the full gradient at a "snapshot" of the model parameters $\tilde{\boldsymbol{\theta}}$; the corresponding "exact" gradient is therefore $\nabla\mathcal{L}(\tilde{\boldsymbol{\theta}})$. At step t, we compute the usual stochastic gradient at the current parameters, $\nabla\mathcal{L}_t(\boldsymbol{\theta}_t)$, but also at the snapshot parameters, $\nabla\mathcal{L}_t(\tilde{\boldsymbol{\theta}})$, which we use as a baseline. We can then use the following improved gradient estimate

$$\mathbf{g}_t = \nabla\mathcal{L}_t(\boldsymbol{\theta}_t) - \nabla\mathcal{L}_t(\tilde{\boldsymbol{\theta}}) + \nabla\mathcal{L}(\tilde{\boldsymbol{\theta}}) \tag{5.67}$$

Page: 125

Author: petercerno Subject: Comment on Text Date: 08.01.21, 12:29:56
sufficiently

In the following sections, we briefly describe some of the theory and algorithms underlying constrained optimization. More details can be found in other books, such as [BV04; NW06; Ber15; Ber16].

5.5.1 Lagrange multipliers

In this section, we discuss how to solve equality contrained optimization problems. We initially assume that we have just one equality constraint, $h(\boldsymbol{\theta}) = 0$.

First note that for any point on the constraint surface, $\nabla h(\boldsymbol{\theta})$ will be orthogonal to the constraint surface. To see why, consider another point nearby, $\boldsymbol{\theta} + \boldsymbol{\epsilon}$, that also lies on the surface. If we make a first-order Taylor expansion around $\boldsymbol{\theta}$ we have

$$h(\boldsymbol{\theta} + \boldsymbol{\epsilon}) \approx h(\boldsymbol{\theta}) + \boldsymbol{\epsilon}^\top \nabla h(\boldsymbol{\theta}) \tag{5.85}$$

Since both $\boldsymbol{\theta}$ and $\boldsymbol{\theta} + \boldsymbol{\epsilon}$ are on the constraint surface, we must have $h(\boldsymbol{\theta}) = h(\boldsymbol{\theta} + \boldsymbol{\epsilon})$ and hence $\boldsymbol{\epsilon}^\top \nabla h(\boldsymbol{\theta}) \approx 0$. Since $\boldsymbol{\epsilon}$ is parallel to the constraint surface, $\nabla h(\boldsymbol{\theta})$ must be perpendicular to it.

We seek a point $\boldsymbol{\theta}^*$ on the constraint surface such that $\mathcal{L}(\boldsymbol{\theta})$ is minimized. We just showed that it must satisfy the condition that $\nabla h(\boldsymbol{\theta}^*)$ is orthogonal to the constraint surface. In addition, such a point must have the property that $\nabla \mathcal{L}(\boldsymbol{\theta})$ is also orthogonal to the constraint surface, as otherwise we could decrease $\mathcal{L}(\boldsymbol{\theta})$ by moving a short distance along the constraint surface. Since both $\nabla h(\boldsymbol{\theta})$ and $\nabla \mathcal{L}(\boldsymbol{\theta})$ are orthogonal to the constraint surface at $\boldsymbol{\theta}^*$, they must be parallel (or anti-parallel) to each other. Hence there must exist a constant $\lambda^* \in \mathbb{R}$ such that

$$\nabla \mathcal{L}(\boldsymbol{\theta}^*) = \lambda^* \nabla h(\boldsymbol{\theta}^*) \tag{5.86}$$

(We cannot just equate the gradient vectors, since they may have different magnitudes.) The constant λ^* is called a **Lagrange multiplier**, and can be positive, negative, or zero. This latter case occurs when $\nabla f(\boldsymbol{\theta}^*) = 0$.

We can convert Eq. (5.86) into an objective, known as the **Lagrangian**, that we should minimize:

$$\mathcal{L}(\boldsymbol{\theta}, \lambda) \triangleq \mathcal{L}(\boldsymbol{\theta}) - \lambda h(\boldsymbol{\theta}) \tag{5.87}$$

At a stationary point of the Lagrangian, we have

$$\nabla_{\boldsymbol{\theta},\lambda} \mathcal{L}(\boldsymbol{\theta}, \lambda) = \mathbf{0} \iff \lambda \nabla_{\boldsymbol{\theta}} h(\boldsymbol{\theta}) = \nabla \mathcal{L}(\boldsymbol{\theta}),\ h(\boldsymbol{\theta}) = 0 \tag{5.88}$$

This is called a **critical point**, and satisfies the original constraint $h(\boldsymbol{\theta}) = 0$ and Eq. (5.86).

If we have $m > 1$ constraints, we can form a new constraint function by addition, as follows:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \mathcal{L}(\boldsymbol{\theta}) - \sum_{j=1}^{m} \lambda_j h_j(\boldsymbol{\theta}) \tag{5.89}$$

We now have $D+m$ equations in $D+m$ unknowns and we can use standard unconstrained optimization methods to find a stationary point. We give some examples below.

Page: 130

Author: petercerno Subject: Comment on Text Date: 11.01.21, 12:53:38
Nit: This should be a vector.

5.5.1.1 Example: 2d Quadratic objective with one linear equality constraint

Consider minimizing $\mathcal{L}(\boldsymbol{\theta}) = \theta_1^2 + \theta_2^2 - 1$ subject to the constraint that $\theta_1 + \theta_2 = 1$.

(This is the problem illustrated in Fig. 5.15(a).) The Lagrangian is

$$L(\theta_1, \theta_2, \lambda) = \theta_1^2 + \theta_2^2 - 1 + \lambda(\theta_1 + \theta_2 - 1) \tag{5.90}$$

We have the following conditions for a stationary point:

$$\frac{\partial}{\partial \theta_1} L(\theta_1, \theta_2, \lambda) = 2\theta_1 + \lambda = 0 \tag{5.91}$$

$$\frac{\partial}{\partial \theta_2} L(\theta_1, \theta_2, \lambda) = 2\theta_2 + \lambda = 0 \tag{5.92}$$

$$\frac{\partial}{\partial \lambda} L(\theta_1, \theta_2, \lambda) = \theta_1 + \theta_2 - 1 = 0 \tag{5.93}$$

From Equations 5.91 and 5.92 we find $2\theta_1 = -\lambda = 2\theta_2$, so $\theta_1 = \theta_2$. Also, from Eq. (5.93), we find $2\theta_1 = 1$. So $\boldsymbol{\theta}^* = (0.5, 0.5)$, as we claimed earlier. Furthermore, this is the global minimum since the objective is convex and the constraint is affine.

5.5.2 The KKT conditions

In this section, we generalize the concept of Lagrange multipliers to additionally handle inequality constraints.

First consider the case where we have a single inequality constraint $g(\boldsymbol{\theta}) \leq 0$. To find the optimum, one approach would be to consider an unconstrained problem where we add the penalty as an infinite step function:

$$\hat{\mathcal{L}}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \infty \mathbb{I}(g(\boldsymbol{\theta}) > 0) \tag{5.94}$$

However, this is a discontinuous function that is hard to optimize.

Instead, we create a lower bound of the form $\mu g(\boldsymbol{\theta})$, where $\mu \geq 0$. This gives us the following Lagrangian:

$$\mathcal{L}(\boldsymbol{\theta}, \mu) = \mathcal{L}(\boldsymbol{\theta}) + \mu g(\boldsymbol{\theta}) \tag{5.95}$$

Note that the step function can be recovered using

$$\hat{\mathcal{L}}(\boldsymbol{\theta}) = \max_{\mu \geq 0} \mathcal{L}(\boldsymbol{\theta}, \mu) = \begin{cases} \infty & \text{if } g(\boldsymbol{\theta}) > 0, \\ \mathcal{L}(\boldsymbol{\theta}) & \text{otherwise} \end{cases} \tag{5.96}$$

Thus our optimization problem becomes

$$\min_{\boldsymbol{\theta}} \max_{\mu \geq 0} \mathcal{L}(\boldsymbol{\theta}, \mu) \tag{5.97}$$

Now consider the general case where we have multiple inequality constraints, $\mathbf{g}(\boldsymbol{\theta}) \leq \mathbf{0}$, and multiple equality constraints, $\mathbf{h}(\boldsymbol{\theta}) = \mathbf{0}$. The **generalized Lagrangian** becomes

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = \mathcal{L}(\boldsymbol{\theta}) + \sum_i \mu_i g_i(\boldsymbol{\theta}) + \sum_j \lambda_j h_j(\boldsymbol{\theta}) \tag{5.98}$$

Page: 131

Author: petercerno Subject: Comment on Text Date: 11.01.21, 12:56:06
Nit: Although it does not matter, according to (5.87) this should be -.

(We are free to change $-\lambda_j h_j$ to $+\lambda_j h_j$ since the sign is arbitrary.) Our optimization problem becomes

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\mu} \geq \mathbf{0}, \boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\mu}, \boldsymbol{\lambda}) \tag{5.99}$$

When $\mathcal{L}$, g and h are convex and satisfy a technical property called Slater's condition (see [BV04, Sec.5.2.3]) all critical points of this problem must satisfy the following criteria.

- All constraints are satisfied (this is called **feasibility**):

$$\mathbf{g}(\boldsymbol{\theta}) \leq \mathbf{0}, \ \mathbf{h}(\boldsymbol{\theta}) = \mathbf{0} \tag{5.100}$$

- The solution is a stationary point:

$$\nabla \mathcal{L}(\boldsymbol{\theta}^*) - \sum_i \mu_i \nabla g_i(\boldsymbol{\theta}^*) - \sum_j \lambda_i \nabla h_j(\boldsymbol{\theta}^*) = \mathbf{0} \tag{5.101}$$

- The penalty for the inequality constraint points in the right direction (this is called **dual feasibility**):

$$\boldsymbol{\mu} \geq \mathbf{0} \tag{5.102}$$

- The Lagrange multipliers pick up any slack in the inactive constraints, i.e., either $\mu_i = 0$ or $g_i(\boldsymbol{\theta}^*) = 0$, so

$$\boldsymbol{\mu} \odot \mathbf{g} = \mathbf{0} \tag{5.103}$$

  This is called **complementary slackness**.

To see why the last condition holds, consider (for simplicity) the case of a single inequality constraint, $g(\boldsymbol{\theta}) \leq 0$. Either it is **active**, meaning $g(\boldsymbol{\theta}) = 0$, or it is inactive, meaning $g(\boldsymbol{\theta}) < 0$. In the active case, the solution lies on the constraint boundary, and $g(\boldsymbol{\theta}) = 0$ becomes an equality constraint; then we have $\nabla \mathcal{L} = \mu \nabla g$ for some constant $\mu \neq 0$, because of Eq. (5.86). In the inactive case, the solution is not on the constraint boundary; we still have $\nabla \mathcal{L} = \mu \nabla g$, but now $\mu = 0$.

These are called called the **Karush-Kuhn-Tucker** (**KKT**) conditions. If $\mathcal{L}$ is a convex function, and the constraints define a convex set, the KKT conditions are sufficient for (global) optimality, as well as necessary.

5.5.3 Linear programming

Consider optimizing a linear function subject to linear constraints. We can write such a problem in **general form** as follows.

$$\min_{\boldsymbol{\theta}} \mathbf{c}^\top \boldsymbol{\theta} \quad \text{s.t.} \quad \mathbf{A}_{le}\boldsymbol{\theta} \leq \mathbf{b}_{le}, \ \mathbf{A}_{ge}\boldsymbol{\theta} \geq \mathbf{b}_{ge}, \ \mathbf{A}_{eq}\boldsymbol{\theta} = \mathbf{b}_{eq}, \tag{5.104}$$

This kind of optimization problem is known as a **linear program**. Let us now rewrite this in **standard form**, as follows:

$$\min_{\boldsymbol{\theta}} \mathbf{c}^\top \boldsymbol{\theta} \quad \text{s.t.} \quad \mathbf{A}\boldsymbol{\theta} \leq \mathbf{b}, \ \boldsymbol{\theta} \geq 0 \tag{5.105}$$

Page: 132

Author: petercerno Subject: Comment on Text Date: 11.01.21, 16:36:10
In (5.98) we were using + sign.

Hence the solution is

$$\boldsymbol{\theta}_* = (1,0)^\top, \boldsymbol{\mu}_* = (0.625, 0.375, 0, 0)^\top \tag{5.116}$$

Notice that the optimal value of $\boldsymbol{\theta}$ occurs at one of the vertices of the ℓ_1 "ball" (the diamond shape).

5.5.4.2 Applications

There are several applications of quadratic programming in ML. In Sec. 11.5, we show how to use it for sparse linear regression. And in Sec. 17.5, we show how to use it for SVMs (support vector machines).

5.5.5 Mixed integer linear programming

Integer linear programming or **ILP** corresponds to minimizing a linear objective, subject to linear constraints, where the optimization variables are discrete integers instead of reals. In standard form, the problem is as follows:

$$\min_{\boldsymbol{\theta}} \mathbf{c}^\top \boldsymbol{\theta} \quad \text{s.t.} \quad \mathbf{A}\boldsymbol{\theta} \leq \mathbf{b},\ \boldsymbol{\theta} \geq 0, \boldsymbol{\theta} \in \mathbb{Z}^D \tag{5.117}$$

where $\mathbb{Z}$ is the set of integers. If some of the optimization variables are real-valued, it is called a **mixed ILP**, often called a **MIP** for short. (If all of the variables are real-valued, it becomes a standard LP.)

MIPs have a large number of applications, such as in vehicle routing, scheduling and packing. They are also useful for some ML applications, such as formally verifying the behavior of certain kinds of deep neural networks [And+18], and proving robustness properties of DNNs to adversarial (worst-case) perturbations [TXT19].

5.6 Proximal gradient method

We are often interested in optimizing an objective of the form

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_s(\boldsymbol{\theta}) + \mathcal{L}_r(\boldsymbol{\theta}) \tag{5.118}$$

where $\mathcal{L}_s$ is differentiable (smooth), and $\mathcal{L}_r$ is convex but not necessarily differentiable (i.e., it may be non-smooth or "rough"). For example, $\mathcal{L}_s$ might be the NLL, and $\mathcal{L}_r$ might be an indicator function that is infinite if a constraint is violated (see Sec. 5.6.1), or $\mathcal{L}_r$ might be the ℓ_1 norm of some parameters (see Sec. 5.6.2), or $\mathcal{L}_r$ might measure how far the parameters are from a set of allowed quantized values (see Sec. 5.6.3).

One way to tackle such problems is to use the **proximal gradient method** (see e.g., [PB+14; PSW15]). Roughly speaking, this takes a step in the direction of $\mathcal{L}_s$, and then projects the resulting update into a space that respects $\mathcal{L}_r$. More precisely, the update is as follows

$$\boldsymbol{\theta}_{t+1} = \text{prox}_{\eta_t \mathcal{L}_r}(\boldsymbol{\theta}_t - \eta_t \nabla \mathcal{L}_s(\boldsymbol{\theta}_t)) \tag{5.119}$$

where $\text{prox}_{\eta f}(\boldsymbol{\theta})$ is the **proximal operator** of $\mathcal{L}_r$ (scaled by η) evaluated at $\boldsymbol{\theta}$:

$$\text{prox}_{\eta f}(\boldsymbol{\theta}) \triangleq \underset{\boldsymbol{\theta}_0}{\text{argmin}} \left(f(\boldsymbol{\theta}_0) + \frac{1}{2\eta} ||\boldsymbol{\theta}_0 - \boldsymbol{\theta}||_2^2 \right) = \underset{\mathbf{z}}{\text{argmin}}\, f(\boldsymbol{\theta}_0) \ \text{ s.t. } \ ||\boldsymbol{\theta}_0 - \boldsymbol{\theta}||_2 \leq \rho \tag{5.120}$$

Page: 135

Author: petercerno Subject: Comment on Text Date: 11.01.21, 17:00:46
Should be vector ?

Author: petercerno Subject: Comment on Text Date: 11.01.21, 17:00:36
\theta_0 ?

This method is known as **projected gradient descent**. See Fig. 5.15 for an illustration.

For example, consider the box constraints $\mathcal{C} = \{\boldsymbol{\theta} : \mathbf{l} \leq \boldsymbol{\theta} \leq \mathbf{u}\}$. The projection operator in this case can be computed elementwise by simply thresholding at the boundaries:

$$\text{proj}_{\mathcal{C}}(\boldsymbol{\theta})_d = \begin{cases} l_d & \text{if } \theta_k \leq l_k \\ x_d & \text{if } l_k \leq \theta_k \leq u_k \\ u_d & \text{if } \theta_k \geq u_k \end{cases} \tag{5.125}$$

For example, if we want to ensure all elements are non-negative, we can use

$$\text{proj}_{\mathcal{C}}(\boldsymbol{\theta}) = \boldsymbol{\theta}_+ = [\max(\theta_1, 0), \ldots, \max(\theta_D, 0)] \tag{5.126}$$

See Sec. 11.5.9.2 for an application of this method to sparse linear regression.

5.6.2 Proximal operator for ℓ_1-norm regularizer

Consider a linear predictor of the form $f(\mathbf{x}; \boldsymbol{\theta}) = \sum_{d=1}^{D} \theta_d x_d$. If we have $\theta_d = 0$ for any dimension d, we ignore the corresponding feature x_d. This is a form of **feature selection**, which can be useful both as a way to reduce overfitting as well as way to improve model interpretability. We can encourage weights to be zero (and not just small) by penalizing the ℓ_1 norm,

$$||\boldsymbol{\theta}||_1 = \sum_{d=1}^{D} |\theta_d| \tag{5.127}$$

This is called a **sparsity inducing regularizer**.

To see why this induces sparsity, consider two possible parameter vectors, one which is sparse, $\boldsymbol{\theta} = (1, 0)$, and one which is non-sparse, $\boldsymbol{\theta}' = (1/\sqrt{2}, 1/\sqrt{2})$. Both have the same ℓ_2 norm

$$||(1,0)||_2^2 = ||(1/\sqrt{2}, 1/\sqrt{2})||_2^2 = 1 \tag{5.128}$$

Hence ℓ_2 regularization (Sec. 4.4.3) will not favor the sparse solution over the dense solution. However, when using ℓ_1 regularization, the sparse solution is cheaper, since

$$||(1,0)||_1 = 1 < ||(1/\sqrt{2}, 1/\sqrt{2})||_1 = \sqrt{2} \tag{5.129}$$

See Sec. 11.5 for more details on sparse regression.

If we combine this regularizer with our smooth loss, we get

$$\mathcal{L}(\boldsymbol{\theta}) = \text{NLL}(\boldsymbol{\theta}) + \lambda ||\boldsymbol{\theta}||_1 \tag{5.130}$$

We can optimize this objective using proximal gradient descent. The key question is how to compute the prox operator for the function $f(\boldsymbol{\theta}) = ||\boldsymbol{\theta}||_1$. Since this function decomposes over dimensions d, the proximal projection can be computed componentwise. We can solve each 1d problem as follows:

$$\text{prox}_{\lambda f}(\theta) = \underset{z}{\text{argmin}}\, \lambda |z| + \frac{1}{2}(z - \theta)^2 \tag{5.131}$$

Page: 137

Author: petercerno Subject: Comment on Text Date: 11.01.21, 17:06:49
Replace k by d (here and below)

Author: petercerno Subject: Comment on Text Date: 11.01.21, 17:06:19
\theta

Author: petercerno Subject: Comment on Text Date: 11.01.21, 17:17:40
This is slightly different from (5.120), where \mu was in denominator: 1 / 2 \mu and instead of z one used \theta_0

In Sec. 11.5.3, we show that the solution to this is given by

$$\text{prox}_{\lambda f}(\theta) = \begin{cases} -\lambda & \text{if } \theta \geq \lambda \\ 0 & \text{if } |\theta| \leq \lambda \\ \theta + \lambda & \text{if } \theta \leq -\lambda \end{cases} \tag{5.132}$$

This is known as the **soft thresholding operator**, since values less than λ in absolute value are set to 0 (thresholded), but in a continuous way. Note that soft thresholding can be written more compactly as

$$\text{SoftThreshold}(\theta, \lambda) = \text{sign}(\theta)\left(|\theta| - \lambda\right)_+ \tag{5.133}$$

where $\theta_+ = \max(\theta, 0)$ is the positive part of θ. In the vector case, we perform this elementwise:

$$\text{SoftThreshold}(\boldsymbol{\theta}, \lambda) = \text{sign}(\boldsymbol{\theta}) \odot \left(|\boldsymbol{\theta}| - \lambda\right)_+ \tag{5.134}$$

See Sec. 11.5.9.3 for an application of this method to sparse linear regression.

5.6.3 Proximal operator for quantization

In some applications (e.g., when training deep neural networks to run on memory-limited **edge devices**, such as mobile phones) we want to ensure that the parameters are **quantized**. For example, in the extreme case where each parameter can only be -1 or $+1$, the state space becomes $\mathcal{C} = \{-1, +1\}^D$.

Let us define a regularizer that measures distance to the nearest quantized version of the parameter vector:

$$\mathcal{L}_r(\boldsymbol{\theta}) = \inf_{\boldsymbol{\theta}_0 \in \mathcal{C}} ||\boldsymbol{\theta} - \boldsymbol{\theta}_0||_1 \tag{5.135}$$

(We could also use the ℓ_2 norm.) In the case of $\mathcal{C} = \{-1, +1\}^D$, this becomes

$$\mathcal{L}_r(\boldsymbol{\theta}) = \sum_{d=1}^{D} \inf_{[\theta_0]_d \in \{\pm 1\}} |\theta_d - [\theta_0]_d| = \sum_{d=1}^{D} \min\{|\theta_d - 1|, |\theta_d + 1|\} = ||\boldsymbol{\theta} - \text{sign}(\boldsymbol{\theta})||_1 \tag{5.136}$$

Let us define the corresponding quantization operator to be

$$q(\boldsymbol{\theta}) = \text{proj}_{\mathcal{C}}(\boldsymbol{\theta}) = \text{argmin}\, \mathcal{L}_r(\boldsymbol{\theta}) = \text{sign}(\boldsymbol{\theta}) \tag{5.137}$$

The core difficulty with quantized learning is that quantization is not a differentiable operation. A popular solution to this is to use the **straight-through estimator**, which uses the approximation $\frac{\partial \mathcal{L}}{\partial q(\boldsymbol{\theta})} \approx \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$ (see e.g., [Yin+19b]). The corresponding update can be done in two steps: first compute the gradient vector at the quantized version of the current parameters, and then update the unconstrained parameters using this approximate gradient:

$$\tilde{\boldsymbol{\theta}}_t = \text{proj}_{\mathcal{C}}(\boldsymbol{\theta}_t) = q(\boldsymbol{\theta}_t) \tag{5.138}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla \mathcal{L}_s(\tilde{\boldsymbol{\theta}}_t) \tag{5.139}$$

Page: 138

Author: petercerno Subject: Comment on Text Date: 11.01.21, 17:18:49
\theta - \lambda ?

When applied to $\mathcal{C} = \{-1, +1\}^D$, this is known as the **binary connect** method [CBD15].

We can get better results using proximal gradient descent, in which we treat quantization as a regularizer, rather than a hard constraint; this is known as **ProxQuant** [BWL19]. The update becomes

$$\tilde{\boldsymbol{\theta}}_t = \text{prox}_{\eta_t \lambda \mathcal{L}_r}(\boldsymbol{\theta}_t - \eta_t \nabla \mathcal{L}_s(\boldsymbol{\theta}_t)) \tag{5.140}$$

In the case that $\mathcal{C} = \{-1, +1\}^D$, one can show that the proximal operator is a generalization of the soft thresholding operator in Eq. (5.134):

$$\text{prox}_{\eta_t \lambda \mathcal{L}_r}(\boldsymbol{\theta}) = \text{SoftThreshold}(\boldsymbol{\theta}, \lambda, \text{sign}(\boldsymbol{\theta})) \tag{5.141}$$

$$= \text{sign}(\boldsymbol{\theta}) + \text{sign}(\boldsymbol{\theta} - \text{sign}(\boldsymbol{\theta})) \odot (|\boldsymbol{\theta} - \text{sign}(\boldsymbol{\theta})| - \lambda)_+ \tag{5.142}$$

This can be generalized to other forms of quantization; see [Yin+19b] for details.

5.7 Bound optimization

In this section, we consider a class of algorithms known as **bound optimization** or **MM** algorithms. In the context of minimization, MM stands for **majorize-minimize**. In the context of maximization, MM stands for **minorize-maximize**. We will discuss a special case of MM, known as **expectation maximization** or **EM**, in Sec. 5.7.2.

5.7.1 The general algorithm

In this section, we give a brief outline of MM methods. (More details can be found in e.g., [HL04; Mai15; SBP17; Nad+19],) To be consistent with the literature, we assume our goal is to *maximize* some function $L(\boldsymbol{\theta})$, such as the log likelihood, wrt its parameters $\boldsymbol{\theta}$. The basic approach in MM algorithms is to construct a **surrogate function** $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t)$ which is a tight lowerbound to $L(\boldsymbol{\theta})$ such that $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) \leq L(\boldsymbol{\theta})$ and $Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}^t) = L(\boldsymbol{\theta}^t)$. If these conditions are met, we say that Q minorizes L. We then perform the following update at each step:

$$\boldsymbol{\theta}^{t+1} = \text{argmax}_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) \tag{5.143}$$

This guarantees us monotonic increases in the original objective:

$$\ell(\boldsymbol{\theta}^{t+1}) \geq Q(\boldsymbol{\theta}^{t+1}, \boldsymbol{\theta}^t) \geq Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}^t) = \ell(\boldsymbol{\theta}^t) \tag{5.144}$$

where the first inequality follows since $Q(\boldsymbol{\theta}^t, \boldsymbol{\theta}')$ is a lower bound on $\ell(\boldsymbol{\theta}^t)$ for any $\boldsymbol{\theta}'$; the second inequality follows from Eq. (5.143); and the final equality follows the tightness property. As a consequence of this result, if you do not observe monotonic increase of the objective, you must have an error in your math and/or code. This is a surprisingly powerful debugging tool.

This process is sketched in Fig. 5.16. The dashed red curve is the original function (e.g., the log-likelihood of the observed data). The solid blue curve is the lower bound, evaluated at $\boldsymbol{\theta}^t$; this touches the objective function at $\boldsymbol{\theta}^t$. We then set $\boldsymbol{\theta}^{t+1}$ to the maximum of the lower bound (blue curve), and fit a new bound at that point (dotted green curve). The maximum of this new bound becomes $\boldsymbol{\theta}^{t+2}$, etc.

Page: 139

Author: petercerno Subject: Cross-Out Date: 11.01.21, 17:22:37

Author: petercerno Subject: Cross-Out Date: 11.01.21, 17:22:54

Author: petercerno Subject: Comment on Text Date: 01.02.21, 14:07:37
This implies that in the expression: Q(x, y) the arguments x and y are of the same type (dimension etc.).
However, in Section 5.7.2 on EM algorithm we have Q(\theta, {q_n}), i.e. the arguments are of different types.

Author: petercerno Subject: Comment on Text Date: 25.01.21, 14:08:11
L (here and below)

Author: petercerno Subject: Comment on Text Date: 01.02.21, 13:50:58
t+1

Author: petercerno Subject: Comment on Text Date: 01.02.21, 13:51:07
t+1

5.7.2.2 E step

We see that the lower bound is a sum of N terms, each of which has the following form:

$$Q(\boldsymbol{\theta}, q_n) = \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{y}_n, \mathbf{z}_n|\boldsymbol{\theta})}{q_n(\mathbf{z}_n)} \tag{5.150}$$

$$= \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n|\mathbf{y}_n, \boldsymbol{\theta}) p(\mathbf{y}_n|\boldsymbol{\theta})}{q_n(\mathbf{z}_n)} \tag{5.151}$$

$$= \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n|\mathbf{y}_n, \boldsymbol{\theta})}{q_n(\mathbf{z}_n)} + \sum_{\mathbf{z}_n} q_n(\mathbf{z}_n) \log p(\mathbf{y}_n|\boldsymbol{\theta}) \tag{5.152}$$

$$= -\mathbb{KL}\left(q_n(\mathbf{z}_n) \| p(\mathbf{z}_n|\mathbf{y}_n, \boldsymbol{\theta})\right) + \log p(\mathbf{y}_n|\boldsymbol{\theta}) \tag{5.153}$$

where $\mathbb{KL}(q\|p) \triangleq \sum_z q(z) \log \frac{q(z)}{p(z)}$ is the Kullback-Leibler divergence (or KL divergence for short) between probability distributions q and p. We discuss this in more detail in Sec. 6.2, but the key property we need here is that $\mathbb{KL}(q\|p) \geq 0$ and $\mathbb{KL}(q\|p) = 0$ iff $q = p$. Hence we can maximize the lower bound $Q(\boldsymbol{\theta}, \{q_n\})$ wrt $\{q_n\}$ by setting each one to $q_n^* = p(\mathbf{z}_n|\mathbf{y}_n, \boldsymbol{\theta})$. This is called the **E step**. This makes the lower bound tight, in the sense that $Q(\boldsymbol{\theta}, \{q_n^*\}) = \sum_n \log p(\mathbf{y}_n|\boldsymbol{\theta}) = \ell(\boldsymbol{\theta})$.

5.7.2.3 M step

In the M step, we need to maximize $Q(\boldsymbol{\theta}, \{q_n^t\})$ wrt $\boldsymbol{\theta}$, where q_n^t are the distributions computed in the E step at iteration t. Since the entropy terms $\mathbb{H}(q_n)$ are constant wrt $\boldsymbol{\theta}$, so we can drop them in the M step. We are left with

$$\ell^t(\boldsymbol{\theta}) = \sum_n \mathbb{E}_{q_n^t(\mathbf{z}_n)} \left[\log p(\mathbf{y}_n, \mathbf{z}_n|\boldsymbol{\theta})\right] \tag{5.154}$$

This is called the **expected complete data log likelihood**. If the joint probability is in the exponential family (Sec. 12.2), we can rewrite this as

$$\ell^t(\boldsymbol{\theta}) = \sum_n \mathbb{E}\left[\mathbf{t}(\mathbf{y}_n, \mathbf{z}_n)^\top \boldsymbol{\theta} - A(\boldsymbol{\theta})\right] = \sum_n (\mathbb{E}\left[\mathbf{t}(\mathbf{y}_n, \mathbf{z}_n)\right]^\top \boldsymbol{\theta} - A(\boldsymbol{\theta})) \tag{5.155}$$

where $\mathbb{E}[\mathbf{t}(\mathbf{y}_n, \mathbf{z}_n)]$ are called the **expected sufficient statistics**.

In the M step, we maximize the expected complete data log likelihood to get

$$\boldsymbol{\theta}^{t+1} = \arg\max_{\boldsymbol{\theta}} \sum_n \mathbb{E}_{q_n^t}[]\log p(\mathbf{y}_n, \mathbf{z}_n|\boldsymbol{\theta})] \tag{5.156}$$

In the case of the exponential family, the maximization can be solved in closed-form by matching the moments of the expected sufficient statistics (Sec. 12.2.4).

We see from the above that the E step does not in fact need to return the full set of posterior distributions $\{q(\mathbf{z}_n) : n = 1 : N\}$, but can instead just return the sum of the expected sufficient statistics, $\sum_n \mathbb{E}_{q(\mathbf{z}_n)}[\mathbf{t}(\mathbf{y}_n, \mathbf{z}_n)]$. This will become clearer in the examples below.

Page: 142

Author: petercerno Subject: Comment on Text Date: 27.01.21, 16:32:38
{ q_n }, to be consistent with previous notation

Author: petercerno Subject: Comment on Text Date: 01.02.21, 14:15:26
Nit: { q_n^t }

Author: petercerno Subject: Comment on Text Date: 27.01.21, 16:41:06
q_n

The current expression might give an impression that there are only N possiblities for z: z_1, ..., z_N, which is misleading, as z might be real-valued vector.

5.7.3 Example: EM for a GMM

In this section, we show how to use the EM algorithm to compute MLE and MAP estimates of the parameters for a Gaussian mixture model (GMM).

5.7.3.1 E step

The E step simply computes the **responsibility** of cluster k for generating data point n, as estimated using the current parameter estimates $\boldsymbol{\theta}^{(t)}$:

$$r_{nk}^{(t)} = p(z_n = k|\boldsymbol{\theta}^{(t)}) = \frac{\pi_k^{(t)} p(\mathbf{y}_n|\boldsymbol{\theta}_k^{(t)})}{\sum_{k'} \pi_{k'}^{(t)} p(\mathbf{y}_n|\boldsymbol{\theta}_{k'}^{(t)})} \tag{5.157}$$

5.7.3.2 M step

The M step maximizes the expected complete data log likelihood, given by

$$\ell^t(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_n \log p(z_n|\boldsymbol{\pi}) + \log p(\mathbf{y}_n|z_n,\boldsymbol{\theta})\right] \tag{5.158}$$

$$= \mathbb{E}\left[\sum_n \log\left(\prod_k \pi_k^{z_{nk}}\right) + \log\left(\prod_k \mathcal{N}(\mathbf{y}_n|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)^{z_{nk}}\right)\right] \tag{5.159}$$

$$= \sum_n \sum_k \mathbb{E}\left[z_{nk}\right] \log \pi_k + \sum_n \sum_k \mathbb{E}\left[z_{nk}\right] \log \mathcal{N}(\mathbf{y}_n|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k) \tag{5.160}$$

$$= \sum_n \sum_k r_{nk}^{(t)} \log(\pi_k) - \frac{1}{2} \sum_n \sum_k r_{nk}^{(t)} \left[\log|\boldsymbol{\Sigma}_k| + (\mathbf{y}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\mathbf{y}_n - \boldsymbol{\mu}_k)\right] + \text{const} \tag{5.161}$$

where $z_{nk} = \mathbb{I}(z_n = k)$ is a one-hot encoding of the categorical value z_n. This objective is just a weighted version of the standard problem of computing the MLEs of an MVN (see Sec. 4.2.6). One can show that the new parameter estimates are given by

$$\boldsymbol{\mu}_k^{(t+1)} = \frac{\sum_n r_{nk}^{(t)} \mathbf{y}_n}{r_k^{(t)}} \tag{5.162}$$

$$\boldsymbol{\Sigma}_k^{(t+1)} = \frac{\sum_n r_{nk}^{(t)} (\mathbf{y}_n - \boldsymbol{\mu}_k^{(t+1)})(\mathbf{y}_n - \boldsymbol{\mu}_k^{(t+1)})^\top}{r_k^{(t)}}$$

$$= \frac{\sum_n r_{nk}^{(t)} \mathbf{y}_n \mathbf{y}_n^\top}{r_k^{(t)}} - \boldsymbol{\mu}_k^{(t+1)} (\boldsymbol{\mu}_k^{(t+1)})^\top \tag{5.163}$$

where $r_k^{(t)} \triangleq \sum_n r_{nk}^{(t)}$ is the weighted number of points assigned to cluster k. The mean of cluster k is just the weighted average of all points assigned to cluster k, and the covariance is proportional to the weighted empirical scatter matrix.

Page: 143

Author: petercerno Subject: Comment on Text Date: 27.01.21, 16:51:41
-> "z_n, "

The M step for the mixture weights is simply a weighted form of the usual MLE:

$$\pi_k^{(t+1)} = \frac{1}{N}\sum_n r_{nk}^{(t)} = \frac{r_k^{(t)}}{N} \tag{5.164}$$

5.7.3.3 Example

An example of the algorithm in action is shown in Fig. 5.18 where we fit some 2d data with a 2 component GMM. The data set, from [Bis06], is derived from measurements of the Old Faithful geyser in Yellowstone National Park. In particular, we plot the time to next eruption in minutes versus the duration of the eruption in minutes. The data was standardized, by removing the mean and dividing by the standard deviation, before processing; this often helps convergence.

We start with $\boldsymbol{\mu}_1 = (-1, 1)$, $\boldsymbol{\Sigma}_1 = \mathbf{I}$, $\boldsymbol{\mu}_2 = (1, -1)$, $\boldsymbol{\Sigma}_2 = \mathbf{I}$. We color code points such that blue points come from cluster 1 and red points from cluster 2. More precisely, we set the color of the n'th point to color$(n) = r_{n1}$blue$ + r_{n2}$red, so ambiguous points appear purple. We see that initially, the cluster centers move, and their shapes change, quite quickly, and then EM slowly converges to a local maximum. (For a recent analysis of the convergence rate of EM, see [KKS20].)

For more details on applying GMMs for clustering, see Sec. 21.4.1.

5.7.3.4 MAP estimation

Computing the MLE of a GMM often suffers from numerical problems and overfitting. To see why, suppose for simplicity that $\boldsymbol{\Sigma}_k = \sigma_k^2\mathbf{I}$ for all k. It is possible to get an infinite likelihood by assigning one of the centers, say $\boldsymbol{\mu}_k$, to a single data point, say $\mathbf{y}_n$, since then the likelihood of that data point is given by

$$\mathcal{N}(\mathbf{y}_n|\boldsymbol{\mu}_k = \mathbf{y}_n, \sigma_k^2\mathbf{I}) = \frac{1}{\sqrt{2\pi\sigma_k^2}}e^0 \tag{5.165}$$

Hence we can drive this term to infinity by letting $\sigma_k \to 0$, as shown in Fig. 5.19(a). We call this the "collapsing variance problem".

An easy solution to this is to perform MAP estimation. Fortunately, we can still use EM to find this MAP estimate. Our goal is now to maximize the expected complete data log-likelihood plus the log prior:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) = \left[\sum_n\sum_k r_{nk}\log\pi_{nk} + \sum_n\sum_k r_{nk}\log p(\mathbf{y}_n|\boldsymbol{\theta}_k)\right] + \log p(\boldsymbol{\pi}) + \sum_k \log p(\boldsymbol{\theta}_k) \tag{5.166}$$

Note that the E step remains unchanged, but the M step needs to be modified, as we now explain.

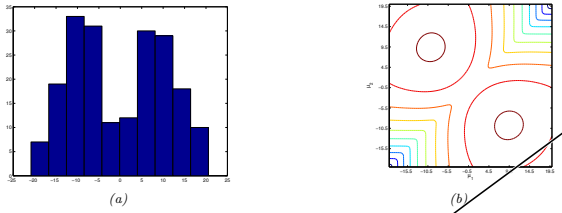
For the prior on the mixture weights, it is natural to use a Dirichlet prior (Sec. 7.2.2.2), $\boldsymbol{\pi} \sim \mathrm{Dir}(\boldsymbol{\alpha})$, since this is conjugate to the categorical distribution. The MAP estimate is given by

$$\pi_k = \frac{r_k + \alpha_k - 1}{N + \sum_k \alpha_k - K} \tag{5.167}$$

If we use a uniform prior, $\alpha_k = 1$, this reduces to the MLE.

Page: 144

Author: petercerno Subject: Comment on Text Date: 01.02.21, 15:13:09
I^t (\theta) as in (5.161) ?
Note that \theta^old is not used



(a) (b)

Figure 5.20: Left: $N = 200$ data points sampled from a mixture of 2 Gaussians in 1d, with $\pi_k = 0.5$, $\sigma_k = 5$, $\mu_1 = -10$ and $\mu_2 = 10$. Right: Likelihood surface $p(\mathcal{D}|\mu_1, \mu_2)$, with all other parameters set to their true values. We see the two symmetric modes, reflecting the unidentifiability of the parameters. Generated by mixGaussLikSurfaceDemo.m.

5.7.4 Example: EM for an MVN with missing data

In Sec. 4.2.6, we explained how to compute the MLE for an MVN (multivariate normal) when we have a fully observed data matrix $\mathbf{Y}$. In this section, we consider the case where we have **missing data** or **partially observed data**. For example, we can think of the entries of $\mathbf{Y}$ as being answers to a survey; some of these answers may be unknown.

To model the missing data, let $\mathbf{M}$ be an $N \times D$ matrix of binary variables, where $M_{nd} = 1$ if feature d in example n is missing, and $M_{nd} = 0$ otherwise. Let $\mathbf{y}_n$ be the visible entries for case n (where $M_{nd} = 1$) $\mathbf{z}_n$ be the hidden entries (where $M_{nd} = 0$), and $\mathbf{y}_n = (\mathbf{y}_n, \mathbf{z}_n) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ be all the entries. If we assume $p(\mathbf{M}|\mathbf{Y}, \boldsymbol{\phi}) = p(\mathbf{M}|\boldsymbol{\phi})$, we say the data is **missing completely at random** or **MCAR**. If we assume $p(\mathbf{M}|\mathbf{Y}, \boldsymbol{\phi}) = p(\mathbf{M}|\mathbf{Y}, \boldsymbol{\phi})$, we say the data is **missing at random** or **MAR**. If neither of these assumptions hold, we say the data is **not missing at random** or **NMAR**.

In the MCAR and MAR cases, we can ignore the missingess mechanism, since it tells us nothing about the hidden data. However, in the NMAR case, we need to model the **missing data mechanism**, since the lack of information may be informative. For example, the fact that someone did not fill out an answer to a sensitive question on a survey (e.g., "Do you have COVID?") could be informative about the underlying value. See e.g., [LR87; Mar08] for more information on missing data models.

In this section, we make the MAR assumption, for simplicity. Under the MAR assumption, the log likelihood of the visible data has the form

$$\log p(\mathbf{Y}|\boldsymbol{\theta}) = \sum_n \log p(\mathbf{y}_n|\boldsymbol{\theta}) = \sum_n \log \left[\int p(\mathbf{y}_n, \mathbf{z}_n|\boldsymbol{\theta}) d\mathbf{z}_n \right] \tag{5.173}$$

Unfortunately, this objective is hard to maximize. since we cannot push the log inside the expectation. Fortunately, we can easily apply EM, as we explain below.

Page: 147

Author: petercerno Subject: Comment on Text Date: 01.02.21, 15:18:49
M_nd = 1 is the missing case.

Author: petercerno Subject: Comment on Text Date: 01.02.21, 15:20:15
LHS and RHS are exactly the same, i.e. this the equality always holds ...

5.7.4.1 E step

Suppose we have $\boldsymbol{\theta}^{t-1}$. Then we can compute the expected complete data log likelihood at iteration t as follows:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) = \mathbb{E}\left[\sum_{n=1}^{N} \log \mathcal{N}(\mathbf{y}_n|\boldsymbol{\mu}, \boldsymbol{\Sigma})|\mathcal{D}, \boldsymbol{\theta}^{t-1}\right] \tag{5.174}$$

$$= -\frac{N}{2}\log|2\pi\boldsymbol{\Sigma}| - \frac{1}{2}\sum_n \mathbb{E}\left[(\mathbf{y}_n - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{y}_n - \boldsymbol{\mu})\right] \tag{5.175}$$

$$= -\frac{N}{2}\log|2\pi\boldsymbol{\Sigma}| - \frac{1}{2}\mathrm{tr}(\boldsymbol{\Sigma}^{-1}\sum_n \mathbb{E}\left[(\mathbf{y}_n - \boldsymbol{\mu})(\mathbf{y}_n - \boldsymbol{\mu})^\top\right] \tag{5.176}$$

$$= -\frac{N}{2}\log|\boldsymbol{\Sigma}| - \frac{ND}{2}\log(2\pi) - \frac{1}{2}\mathrm{tr}(\boldsymbol{\Sigma}^{-1}\mathbb{E}\left[\mathbf{S}(\boldsymbol{\mu})\right]) \tag{5.177}$$

where

$$\mathbb{E}\left[\mathbf{S}(\boldsymbol{\mu})\right] \triangleq \sum_n \left(\mathbb{E}\left[\mathbf{y}_n\mathbf{y}_n^\top\right] + \boldsymbol{\mu}\boldsymbol{\mu}^\top - 2\boldsymbol{\mu}\mathbb{E}\left[\mathbf{y}_n\right]^\top\right) \tag{5.178}$$

(We drop the conditioning of the expectation on $\mathcal{D}$ and $\boldsymbol{\theta}^{t-1}$ for brevity.) We see that we need to compute $\sum_n \mathbb{E}\left[\mathbf{y}_n\right]$ and $\sum_n \mathbb{E}\left[\mathbf{y}_n\mathbf{y}_n^\top\right]$; these are the expected sufficient statistics.

To compute these quantities, we use the results from Sec. 3.5.3. Specifically, consider case n, where components v are observed and components h are unobserved. We have

$$p(\mathbf{z}_n|\mathbf{y}_n, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}_n|\mathbf{m}_n, \mathbf{V}_n) \tag{5.179}$$

$$\mathbf{m}_n \triangleq \boldsymbol{\mu}_h + \boldsymbol{\Sigma}_{hv}\boldsymbol{\Sigma}_{vv}^{-1}(\mathbf{y}_n - \boldsymbol{\mu}_v) \tag{5.180}$$

$$\mathbf{V}_n \triangleq \boldsymbol{\Sigma}_{hh} - \boldsymbol{\Sigma}_{hv}\boldsymbol{\Sigma}_{vv}^{-1}\boldsymbol{\Sigma}_{vh} \tag{5.181}$$

Hence the expected sufficient statistics are

$$\mathbb{E}\left[\mathbf{y}_n\right] = (\mathbb{E}\left[\mathbf{z}_n\right]; \mathbf{y}_n) = (\mathbf{m}_n; \mathbf{y}_n) \tag{5.182}$$

where we have assumed (without loss of generality) that the unobserved variables come before the observed variables in the node ordering.

To compute $\mathbb{E}\left[\mathbf{y}_n\mathbf{y}_n^\top\right]$, we use the result that $\mathrm{Cov}\left[\mathbf{y}\right] = \mathbb{E}\left[\mathbf{y}\mathbf{y}^\top\right] - \mathbb{E}\left[\mathbf{y}\right]\mathbb{E}\left[\mathbf{y}^\top\right]$. Hence

$$\mathbb{E}\left[\mathbf{y}_n\mathbf{y}_n^\top\right] = \mathbb{E}\left[\begin{pmatrix}\mathbf{z}_n\\ \mathbf{y}_n\end{pmatrix}\begin{pmatrix}\mathbf{z}_n^\top & \mathbf{y}_n^\top\end{pmatrix}\right] = \begin{pmatrix}\mathbb{E}\left[\mathbf{z}_n\mathbf{z}_n^\top\right] & \mathbb{E}\left[\mathbf{z}_n\right]\mathbf{y}_n^\top\\ \mathbf{y}_n\mathbb{E}\left[\mathbf{z}_n\right]^\top & \mathbf{y}_n\mathbf{y}_n^\top\end{pmatrix} \tag{5.183}$$

$$\mathbb{E}\left[\mathbf{z}_n\mathbf{z}_n^\top\right] = \mathbb{E}\left[\mathbf{z}_n\right]\mathbb{E}\left[\mathbf{z}_n\right]^\top + \mathbf{V}_n \tag{5.184}$$

Page: 148

Author: petercerno Subject: Comment on Text Date: 01.02.21, 15:26:37
E [z_n ; y_n]

Author: petercerno Subject: Cross-Out Date: 01.02.21, 15:27:02