1 Categorical MLE

The categorical distribution represents a discrete random variable that can take on one of K values. It is described by a parameter vector $\mu = (\mu_1, \mu_2, \cdots, \mu_K)$, where $\sum_{k=1}^{K} \mu_k = 1$ and each μ_k determines the probability that the random variable is in category k.
For a random variable that obeys the categorical distribution, we can represent it as a K-dimensional vector $x = (x_1, x_2, \cdots, x_k)$ with one component set to 1 and the remaining set to 0, depending on which category it belongs to. Note that the Bernoulli distribution is a special case of the categorical distribution, where $K = 2$.
Question 1: For a dataset $\mathcal{D}$ drawn from the categorical distribution, write the likelihood function $p(\mathcal{D} \mid \mu)$.

Question 2: Show that the log-likelihood function is concave and compute the maximum likelihood estimate for μ.

2 Gaussian MLE

Recall that the pdf for a univariate Gaussian is given by the following equation

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu^2)}{\sigma^2}}.$$

Question 1: Derive the likelihood function $p(\mathcal{D} \mid \theta)$ and compute the maximum likelihood estimates for μ and σ^2 using the log-likelihood function.

3 Gradient Descent

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function. The gradient of f, $\nabla f(x)$, is defined to be the direction of steepest ascent of the function - correspondingly, the negative gradient $-\nabla f(x)$ is the direction of steepest descent.
With this, we can derive a naive procedure for finding minima of a function by starting with an initial guess for our optimum and moving in the direction of the negative gradient

$$x_{t+1} \leftarrow x_t - \eta_t \nabla f(x_t)$$

where η_t is a step size parameter that controls how far in the direction of the negative gradient we move.
In the machine learning and optimization community, this procedure is called *gradient descent.*
Question 1: How can gradient descent be useful in a machine learning setting?

Question 2: Let $f(x) = \frac{1}{2}x^2$, $x_0 = 2$, and $\eta = 0.5$. Compute two steps of gradient descent on this function.

4 Stochastic Gradient Descent and Variants

In most machine learning applications, our metric of performance is a loss function $\mathcal{L}(h_\theta(x), y)$ that tells us how poorly our classifier performs on a single sample (x, y). For an entire dataset $\mathcal{D}$, we usually consider the empirical risk (defined to be the average loss over all the samples)

$$\mathcal{L}_{\mathcal{D}} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(h_\theta(x_i), y_i).$$

Question 1: Consider drawing a single sample (x_i, y_i) uniformly at random from your dataset and computing the gradient of the loss function evaluated at that sample. Let $\hat{g}$ denote this gradient vector. What is $\mathbb{E}[\hat{g}]$?

SGD and GD: With this in mind, we can define full-batch gradient descent and stochastic gradient descent (SGD). Full batch gradient descent proceeds in the same way as gradient descent defined on the previous slide.

Algorithm 1 Full-batch Gradient Descent

```
1: for $t$ in $[T]$ do
2:     for sample $x_i$ in dataset do
3:         Compute $\mathcal{L}(h_\theta(x_i), y_i)$ and $\nabla_\theta \mathcal{L}(h_\theta(x_i), y_i)$
4:     end for
5:     Set $\nabla f(\theta_t) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \mathcal{L}(h_\theta(x_i), y_i)$
6:     Perform gradient step $\theta_{t+1} \leftarrow \theta_t - \eta_t \nabla f(\theta_t)$
7: end for
```

Algorithm 2 SGD

```
1: for $t$ in $[T]$ do
2:     for sample $x_i$ in dataset do
3:         Compute $\mathcal{L}(h_\theta(x_i), y_i)$ and $\nabla_\theta \mathcal{L}(h_\theta(x_i), y_i)$
4:         Set $\hat{g}_t = \nabla_\theta \mathcal{L}(h_\theta(x_i), y_i)$
5:         Perform gradient step $\theta_{t+1} \leftarrow \theta_t - \eta_t \hat{g}_t$
6:     end for
7: end for
```

Question: What is the complexity of performing one step of gradient descent versus if we consider the cost of computing a single sample gradient to be $O(1)$? How might this inform our algorithm design choices when deciding between SGD and GD?

5 Variants of GD and SGD

Example 1 - Momentum: GD with momentum obeys the following update rule

$$\begin{aligned} v_t &\leftarrow \quad \beta v_{t-1} + \eta \nabla f(\theta_{t-1}) \\ \theta_t &\leftarrow \quad \theta_{t-1} - v_t. \end{aligned}$$

Why might momentum be useful, particularly in the context of stochastic gradient descent? (Hint: consider what happens as we vary β.)

Example 2 - AdaGrad Let $\hat{g}_{t,i}$ denote the i-th component of the stochastic gradient at time-step t. AdaGrad computes updates as follows:

$$\theta_{t,i} \leftarrow x_{t-1,i} - \frac{\eta}{\sqrt{\sum_{j=1}^{t-1} \hat{g}_{j,i} + \epsilon}} \hat{g}_{t-1,i}.$$

(Ignore ϵ as that's mostly used for numerical stability when performing division.) Why might performing this type of normalization be useful when doing SGD, especially with very high dimensional parameter vectors x?

Note: In practice, we typically use the Adam optimizer, which combines elements from momentum and AdaGrad and has been found to be very useful specifically when optimizing neural networks.