2016 Fall CPS 571/STA 561: Homework 5

Duke University

Due: November 30th

1 Gradient Computation For Recursive Logistic Regression by Backpropagation

You are expected to construct a neural network with architecture $784 - H - H - 1$, where H is the number of hidden nodes you choose. This neural network can be used for binary classification, such 0, 1 digits classification for MNIST. The model can be represented as

$$f(\mathbf{x}; \boldsymbol{\theta}) : \mathbb{R}^{784} \to [0, 1]$$

where $\boldsymbol{\theta} = \{\mathbf{W}_1 \in \mathbb{R}^{784 \times H}, \mathbf{b}_1 \in \mathbb{R}^H, \mathbf{W}_2 \in \mathbb{R}^{H \times H}, \mathbf{b}_2 \in \mathbb{R}^H, \mathbf{w}_3 \in \mathbb{R}^H, b_3 \in \mathbb{R}, \}$. Explicitly, $f(\mathbf{x})$ is

$$\begin{aligned} \mathbf{h}_1 &= \sigma\left(\mathbf{W}_1^\top \mathbf{x} + \mathbf{b}_1\right) \\ \mathbf{h}_2 &= \sigma\left(\mathbf{W}_2^\top \mathbf{h}_1 + \mathbf{b}_2\right) \\ f(\mathbf{x}) &= \sigma\left(\mathbf{w}_3^\top \mathbf{h}_2 + b_3\right) \end{aligned}$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$, and it is an element-wise operator, which means if $\mathbf{x} = (x^1, x^2, ..., x^d) \in \mathbb{R}^d$, we have $\sigma(\mathbf{x}) = (\sigma(x^1), \sigma(x^2), ..., \sigma(x^d))$.

The loss function for this model (or the negative log-likelihood) is

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{N} (y_i \log f(\mathbf{x}_i) + (1 - y_i) \log(1 - f(\mathbf{x}_i)))$$

(a) Derive the gradient of $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{W}_1}$ and $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{b}_1}$ by backpropagation.

(b) If there are $L - 1$ hidden layers, can you derive the general rule for computing $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{W}_1}$ and $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{b}_1}$?

2 EM for Coin Toss

Consider an experiment with coin A that has a probability θ_A of heads, and a coin B that has a probability θ_B of heads. We draw m samples as follows - for each sample, pick one of the coins at random, flip it n times, and record the number of heads and tails (that sum to n). If we recorded which coin we used for each sample, we have "complete" information and can estimate θ_A and θ_B in closed form. However, if we did not record the coin we used, we have "missing" data and the problem of estimating $\theta = [\theta_A, \theta_B]$ is harder to solve. One way to approach the problem is to ask - can we assign weights ω_i to each sample according to how likely it is to be generated from coin A or coin B.

(a) Use EM algorithm to derive the estimation.

(b) Implement the EM algorithm to estimate θ. In your data simulation, set $m = 5$, $n = 100$, $\theta_A = 0.8, \theta_B = 0.35$.

3 K-means for Cute Raccoon

We can use k-means clustering to build a rudimentary image compression scheme. In this problem, we are going to "compress" a cute raccoon (Well, that's not real – the staff for this course are always friendly). The idea is that for each pixel in the image, instead of storing the actual floating point value , we just store a nonnegative integer, which acts as an index to a small set of representative pixel values called a **codebook**. Storing the codebook along with the index for each pixel requires many fewer bits than storing the floating point values for each pixel, which means we can store and transmit images much more efficiently. The process of mapping the floating point pixel values to a small set of nonnegative integers is called **quantization**.

How can we come up with a codebook and quantization that allows us to approximately recreate the original image? k-means provides one solution. The k group means will serve as the codebook, and each pixel value is replaced with its group membership. More precisely, let $I \in \mathbb{R}^{m \times n}$ be an image, let $\mu_1, ..., \mu_k \in \mathbb{R}$ be the k group means obtained by running the k-means algorithm on the mn pixels of image I, and let $C_{i,j}$ be the group membership of pixel $I_{i,j}$. We can approximately recover the original image from the compressed version by the image $\hat{I}$, which is defined by

$$\hat{I}_{i,j} = \mu_{C_{i,j}} \tag{1}$$

(a) Original raccoon image (b) 2-means recovered image (c) 4-means recovered image

Figure 1: Quantization via k-means.

(a) Implement the k-means algorithm in the language of your choice. The **arguments** of your function are k and the input dataset. **Return** both the group assignment for each point in the dataset, as well as the mean μ_j for each group. You can initialize each mean by randomly selecting a point from the input data.

(b) For $k = 2$ and $k = 4$, perform this quantization technique on the image in the file `raccoon.png`. Plot the recovered image $\hat{I}$ in each case. How well do these images approximate the original? Your results should be comparable to those in Figure 1. If you'd like, you can try other k to see what raccoon you will get.