Assignment #3

Due: 23:55pm Friday 29 March 2019

Upload at: `https://dropbox.cs.princeton.edu/COS324_S2019/HW3`

Problem 1 (1pt)

Use the instructions above to put your name on the assignment when you compile this document.

Problem 2 (9pts)

Quadratic programs are optimization problems that have the following form:

$$z^\star = \arg\min_z \frac{1}{2} z^T \boldsymbol{P} z + \boldsymbol{q}^T z \qquad \text{such that } \boldsymbol{G}z \geq \boldsymbol{h} \text{ and } \boldsymbol{A}z = \boldsymbol{b}$$

To solve the primal "hard margin" case (that is, the one we discussed in class where the data are linearly separable) of the SVM with a quadratic program, what would we use for $\boldsymbol{P}$, $\boldsymbol{q}$, $\boldsymbol{G}$, $\boldsymbol{h}$, $\boldsymbol{A}$ and $\boldsymbol{b}$? What are these values for the dual version of the problem? [Hint: for the primal problem you'll need to make z include both the weights and the bias term.]

Problem 3 (15pts)

Prove that

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\{-||\boldsymbol{x} - \boldsymbol{x}'||_2^2\},$$

is a valid kernel, where $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^D$, using only the following properties about positive definite kernels:
If $K_1(\cdot,\cdot)$ and $K_2(\cdot,\cdot)$ are valid kernels, then the following are also valid kernels:

$$\begin{aligned}
K(\boldsymbol{x}, \boldsymbol{x}') &= c\, K_1(\boldsymbol{x}, \boldsymbol{x}') \quad \text{for } c > 0 \\
K(\boldsymbol{x}, \boldsymbol{x}') &= K_1(\boldsymbol{x}, \boldsymbol{x}') + K_2(\boldsymbol{x}, \boldsymbol{x}') \\
K(\boldsymbol{x}, \boldsymbol{x}') &= K_1(\boldsymbol{x}, \boldsymbol{x}')\, K_2(\boldsymbol{x}, \boldsymbol{x}') \\
K(\boldsymbol{x}, \boldsymbol{x}') &= \exp\{K_1(\boldsymbol{x}, \boldsymbol{x}')\} \\
K(\boldsymbol{x}, \boldsymbol{x}') &= f(\boldsymbol{x})\, K_1(\boldsymbol{x}, \boldsymbol{x}')\, f(\boldsymbol{x}') \quad \text{where } f \text{ is any function from } \mathbb{R}^D \text{ to } \mathbb{R}
\end{aligned}$$

Problem 4 (15pts)

One thing that might seem strange is that the multi-class logistic regression setup uses one function for each of the K classes but when $K = 2$ there is only one function. In fact, we can actually get a way with $K - 1$ functions in the multi-class case via (using one-hot coding):

$$\Pr(y_k = 1 \,|\, \boldsymbol{x}, \{\boldsymbol{w}_{k'}\}_{k'=1}^{K-1}) = \begin{cases} \dfrac{\exp\{\boldsymbol{w}_k^T \boldsymbol{x}\}}{1 + \sum_{k'=1}^{K-1} \exp\{\boldsymbol{w}_{k'}^T \boldsymbol{x}\}} & k < K \\ \dfrac{1}{1 + \sum_{k'=1}^{K-1} \exp\{\boldsymbol{w}_{k'}^T \boldsymbol{x}\}} & k = K \end{cases}.$$

A. Show why this reparameterization does not change the classifiers that can be represented.
B. Imagine that we have observed a datum of class K, i.e., $y_K = 1$, and with input $\boldsymbol{x}$. Derive the gradient of the log likelihood for this example with respect to one of the $\boldsymbol{w}_k$.

Problem 5 (20pts)

You're working on a regression data set where the data are non-negative integers, i.e., the data are $\{\boldsymbol{x}_n, y_n\}_{n=1}^N$ where $\boldsymbol{x}_n \in \mathbb{R}^D$ and $y_n \in \mathbb{N}_0$. It seems like it makes sense to model the labels with a Poisson distribution, but the Poisson's mean parameter has a complicated relationship with the features, so you'd like to use a neural network to parameterize the likelihood. You decide to construct a neural network with a single fully-connected hidden layer, and then linearly combine those outputs to parameterize the log of the Poisson mean. It doesn't seem like these data would work well with a simple sigmoid activation function for the hidden layer, so you decide to use the following nonlinear activation function for each of the J hidden units:

$$f(z) = \exp\{-z^2\}\,.$$

Putting these pieces together, in the first layer, your network takes an input vector $\boldsymbol{x} \in \mathbb{R}^D$, applies a first layer weight matrix $\boldsymbol{W} \in \mathbb{R}^{J \times D}$, adds bias $\boldsymbol{b} \in \mathbb{R}^J$, and then applies the above nonlinearity to each dimension. That is, your network computes $\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, which is a vector of length J, and then it applies $f(\cdot)$ above to each dimension to get a new vector we'll denote as $\boldsymbol{h} \in \mathbb{R}^J$. Call that vector $\boldsymbol{h}$ the "hidden unit activations". After computing $\boldsymbol{h}$, your network takes the hidden unit activations and computes their inner product with output weights $\boldsymbol{w}$, exponentiates the result, and then uses that positive real scalar as the mean parameter for a Poisson distribution. The Poisson has PMF:

$$\Pr(y \mid \lambda) = \frac{\lambda^y}{y!} e^{-\lambda}\,,$$

and here we're saying that $\lambda = \exp\{\boldsymbol{h}^T \boldsymbol{w}\}$ with $\boldsymbol{h}$ computed as above. You'd like to train $\boldsymbol{w}$, $\boldsymbol{W}$, and $\boldsymbol{b}$ via maximum likelihood by backpropagating through the log likelihood. The following questions assume you are looking at a single training example with inputs $\boldsymbol{x}_n \in \mathbb{R}^D$ and a non-negative integer label $y_n \in \mathbb{N}_0$.

A. For a fixed $\boldsymbol{h}$, what is the gradient of the log likelihood with respect to $\boldsymbol{w}$?
B. For a fixed $\boldsymbol{w}$, what is the gradient of the log likelihood with respect to $\boldsymbol{h}$?
C. If you fix $\boldsymbol{W}$, what is the Jacobian of $\boldsymbol{h}$ with respect to $\boldsymbol{b}$?
D. Given the result of questions B and C above, what is the gradient of the log likelihood with respect to $\boldsymbol{b}$?

Problem 6 (20pts)

Construct a "diamond-shaped" neural network with a scalar input, a scalar real-valued output, and one hidden layer with 15 hidden units. That is, take the scalar input, multiply it by a weight vector, add the biases, apply the element-wise nonlinearity, and then linearly combine these values to get a scalar output. Create three figures, one with hyperbolic tangent nonlinearity, one with rectified linear units ("relus"), and one with RBF nonlinearities (e^{-z^2}). For each of the three figures, plot ten random functions arising from independent normally distributed weights and biases. It is suggested to make the x-axis range from -5 to 5.

Problem 7 (20pts)

Download the `motorcycle.csv` file from the course website. These data are g-forces measured on a motorcycle helmet as a function of milliseconds after impact. Write code to backpropagate through the neural networks from the previous problem. Use gradient descent to fit the parameters to the motorcycle data using least squares for the hyperbolic tangent and RBF activation functions. Scaling the x axis down by a factor of 60 or so will make it easier to fit the neural networks. Your learning rate for full-batch training will probably need to be around 10^{-4}. Plot the data and the functions learned by the neural networks. Turn in your code. You don't need to try the relu activation function, but kudos to you if you get it to work.

Changelog

- 8 March 2019 – Initial version.