Assigned: 14 October

Homework #5

EE 541: Fall 2024

Due: Friday, 25 October at 22:00. Submission instructions will follow separately on brightspace.

1. **Backprop Initialization for Multiclass Classification**

   The softmax function $\mathbf{h}(\cdot)$ takes an M-dimensional input vector $\mathbf{s}$ and outputs an M-dimensional output vector $\mathbf{a}$ as

   $$\mathbf{a} = \mathbf{h}(\mathbf{s}) = \frac{1}{\sum_{m=1}^{M} e^{s_m}} \begin{bmatrix} e^{s_1} \\ e^{s_2} \\ \vdots \\ e^{s_M} \end{bmatrix}$$

   and the multiclass cross-entropy cost is given by

   $$C = -\sum_{i=1}^{n} y_i \ln a_i$$

   where $\mathbf{y}$ is a vector of *ground truth* labels. Define the error (vector) of the output layer as:

   $$\delta = \nabla_{\mathbf{s}} C = \dot{\mathbf{A}} \nabla_{\mathbf{a}} C$$

   where $\dot{\mathbf{A}}$ is the matrix of derivatives of softmax, given as

   $$\dot{\mathbf{A}} = \frac{d\mathbf{h}(\mathbf{s})}{d\mathbf{s}} = \begin{bmatrix} \frac{\partial p_1}{\partial s_1} & \cdots & \frac{\partial p_M}{\partial s_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial p_1}{\partial s_M} & \cdots & \frac{\partial p_M}{\partial s_M} \end{bmatrix}.$$

   (denominator convention with the left-handed chain rule.). Show that $\delta = \mathbf{a} - \mathbf{y}$ if $\mathbf{y}$ is one-hot.

2. Logistic regression

   The MNIST dataset of handwritten digits is one of the earliest and most used datasets to benchmark machine learning classifiers. Each datapoint contains 784 input features – the pixel values from a 28×28 image – and belongs to one of 10 output classes – represented by the numbers 0-9.

   This problem continues your logistic regression experiments from the previous Homework. Use only Python standard library modules, `numpy`, and `mathplotlib` for this problem.

   (a) Logistic "2" detector

   Previous HW.

(b) Softmax classification: gradient descent (GD)

In this part you will use soft-max to peform multi-class classification instead of distinct "one against all" detectors. The target **vector**

$$[\mathbf{Y}]_l = \begin{cases} 1 & \mathbf{x} \text{ is an } \text{“}l\text{”} \\ 0 & \text{else.} \end{cases}$$

for $l = 0, \ldots, K-1$. You can alternatively consider a scalar output Y equal to the value in $\{0, 1, \ldots, K-1\}$ corresponding to the class of input $\mathbf{x}$. Construct a logistic classifier that uses K seperate linear weight vectors $\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_{K-1}$. Compute estimated probabilities for each class given input $\mathbf{x}$ and select the class with the largest score among your K predictors:

$$P\left[Y = l | \mathbf{x}, \mathbf{w}\right] = \frac{\exp(\mathbf{w}_l^T \mathbf{x})}{\sum_{i=0}^{K} \exp(\mathbf{w}_i^T \mathbf{x})}$$

$$\hat{Y} = \arg\max_l P\left[Y = l | \mathbf{x}, \mathbf{w}\right].$$

Note that the probabilities sum to 1. Use log-loss and optimize with batch gradient descent. The (negative) likelihood function on an N sampling training set is:

$$L(w) = -\frac{1}{N} \sum_{i=1}^{N} \log P\left[Y = y^{(i)} | x^{(i)}, w\right]$$

where the sum is over the N points in our training set.

Submit answers to the following.

i. Compute (by-hand) the derivative of the log-likelihood of the soft-max function. Write the derivative in terms of conditional probabilities, the vector $\mathbf{x}$, and indicator functions (*i.e.,* do not write this expression in terms of exponentials). You need this gradient in subsequent parts of this problem.

ii. Implement batch gradient descent. What learning rate did you use?

iii. Plot log-loss (*i.e.,* learning curve) of the training set and test set on the same figure. On a separate figure plot the accuracy against iteration number of your model on the training set and test set. Plot each as a function of the iteration number.

iv. Compute the final loss and final accuracy for both your training set and test set.

(c) Softmax classification: stochastic gradient descent

In this part you will use stochastic gradient descent (SGD) in place of (deterministic) gradient descent above. Test your SGD implmentation using single-point updates and a mini-batch size of 100. You may need to adjust the learning rate to improve performance. You can either: modify the rate by hand or according to some decay scheme or you may choose a single learning

rate. You should get a final predictor comparable to that in the previous question.

Submit answers to the following.

i. Implement SGD with mini-batch size of 1 (*i.e.,* compute the gradient and update weights after each sample). Record the log-loss and accuracy of the training set and test set every 5,000 samples. Plot the sampled log-loss and accuracy values on the same (respective) figures against the batch number. Your plots should start at iteration 0 (*i.e.,* include initial log-loss and accuracy). Your curves should show performance comparable to batch gradient descent. How many iterations did it take to acheive comparable performance with batch gradient descent? How does this number depend on the learning rate? (or learning rate decay schedule if you have a non-constant learning rate).

ii. Compare (to batch gradient descent) the total computational complexity to reach a comparable accuracy on your training set. Note that each iteration of batch gradient descent costs an extra factor of N operations where N is the number data points.

iii. Implement SGD with mini-batch size of 100 (*i.e.,* compute the gradient and update weights with accumulated average after every 100 samples). Record the log-loss and accuracies as above (every 5,000 **samples** – not 5,000 batches) and create similar plots. Your curves should show performance comparable to batch gradient descent. How many iterations did it take to acheive comparable performance with batch gradient descent? How does this number depend on the learning rate? (or learning rate decay schedule if you have a non-constant learning rate).

iv. Compare the computational complexity to reach comparable perforamnce between the 100 sample mini-batch algorithm, the single-point mini-batch, and batch gradient descent.

Submit your trained weights to Autolab. Save your weights and bias to an hdf5 file. Use keys `W` and `b` for the weights and bias, respectively. `W` should be a 10×784 numpy array and `b` should be 10×1 – shape: `(10,)` – numpy array. The code to save the weights is the same as (a) – substituting `W` for `w`.

Note: you will **not** be scored on your models overall accuracy. But a low-score may indicate errors in training or poor optimization.