Practice Questions

CS4787 — Principles of Large-Scale Machine Learning Systems

Problem 1: Preconditioning and Adaptive Learning Rates.

Consider the linear regression problem defined by a dataset with examples $x \in \mathbb{R}^d$ and labels $y \in \mathbb{R}$, with model

$$h_w(x) = w^T x$$

and using the square loss

$$L(\hat{y}, y) = \frac{1}{2} \|\hat{y} - y\|^2 .$$

Suppose that $d = 2$ and the dataset contains the examples

$$x_1 = \begin{bmatrix} 1000 \\ 0 \end{bmatrix}, \quad x_2 = \begin{bmatrix} 0 \\ 4 \end{bmatrix}, \quad x_3 = \begin{bmatrix} 0 \\ 5 \end{bmatrix}, \quad x_4 = \begin{bmatrix} -1000 \\ 0 \end{bmatrix}, \quad x_5 = \begin{bmatrix} 0 \\ -2 \end{bmatrix},$$

but you don't know the labels y_i *a priori*.

(a). Roughly, what is the largest step size α you could use for SGD on this problem without it diverging? (An order-of-magnitude answer from a back-of-the-envelope calculation is fine.)

A: The second derivative matrix of this is $\frac{1}{5} \sum_{i=1}^5 x_i x_i^T = \begin{bmatrix} 400000 & 0 \\ 0 & 9 \end{bmatrix}$, and the largest eigenvalue of this is $L = 400000$. So I would expect the largest step size I could use to be something like $1/L = 0.0000025$.

(b). What is the condition number κ of this problem? What does this tell us about the convergence rate of gradient descent and SGD on this problem? Roughly how many steps do you expect they would take to converge? (Again, an order-of-magnitude answer from a back-of-the-envelope calculation is fine for this second question.)

A: The condition number is the ratio of the largest to smallest eigenvalues of the problem,

$$\kappa = \frac{400000}{9} \approx 44444$$

This suggests that some small multiple of 40000 steps would be needed for SGD or gradient descent to converge on this problem (give or take some factor that depends on how accurate we want our solutions to be and on how accurate our initial guess was).

(c). Choose a preconditioning matrix P such that preconditioned SGD

$$w_{t+1} = w_t - \alpha P \nabla f(w_t)$$

would perform better than baseline SGD on this task. Roughly how many steps do you expect your new preconditioned model would take to converge? (Again, an order-of-magnitude answer from a back-of-the-envelope calculation is fine for this second question.)

A: The natural preconditioning matrix to pick is

$$P = \begin{bmatrix} 400000 & 0 \\ 0 & 9 \end{bmatrix}^{-1} \approx \begin{bmatrix} 0.0000025 & 0 \\ 0 & 0.1111111 \end{bmatrix}$$

I would expect this to now take a small number of iterations, much closer to 1, because the condition number is much smaller now (again this is give or take some factor that depends on how accurate we want our solutions to be and how close our initial guess was).

(d). How well do you expect AdaGrad and Adam would perform on this problem? Why?

A: I expect they would perform quite well, because they would automatically learn to use a different step size in each coordinate, which would have the equivalent effect to the diagonal preconditioning matrix in 1(c).

(e). Do you expect that adding Polyak averaging to SGD could help accelerate learning for this problem? Why or why not?

A: I expect that adding Polyak averaging could help a little, but not nearly as much as these preconditioning or adaptive learning rate methods. Polyak averaging would help by making SGD get more accurate solutions by averaging out iterations w_t to produce a lower-variance solution.

Problem 2: Dimensionality Reduction and Sparsity.

(a). Describe the method of random projections. When is it useful? According to the J-L lemma, how many dimensions do I need for the space I am projecting into in order to ensure the existence of a projection that preserves pairwise distances for a dataset of size n up to an error tolerance of ϵ?

A: Random projections projects the d-dimensional features into a D-dimensional linear subspace ($D < d$). This is useful in large feature sets because using the Johnson-Lindenstrauss lemma, we can show that D can now dependent on n rather than d. To preserve pairwise distances:

$$D > 8\log(n)/\epsilon^2$$

for a dataset of size n and error tolerance of ϵ.

(b). When would I want to use an *autoencoder* instead of random projection or PCA? How does the overall compute time needed compare between dimensionality reduction with an autoencoder and dimensionality reduction with random projection?

A: An autoencoder would be used to encode nonlinear feature representations whereas random projection or PCA can only encode linear representations. It may also be able to do a better job at reducing dimensionality than the other methods. Since autoencoders feature a neural network for encoding/decoding, the overall compute time is significantly higher than that of random projection.

(c). Consider the matrix

$$A = \begin{bmatrix} -7 & 0 & 3 & 0 & 0 & 4 & -2 & -4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Express this matrix as a sparse matrix in **COO** format and **CSR** format. What are some of the advantages of each storage format?

A: See lecture 10 for more information about sparsity formats.

In **COO** format,

$$\begin{aligned} \texttt{row indexes:} &\quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \end{bmatrix} \\ \texttt{column indexes:} &\quad \begin{bmatrix} 0 & 2 & 5 & 6 & 7 & 7 & 5 & 3 \end{bmatrix} \\ \texttt{values:} &\quad \begin{bmatrix} -7 & 3 & 4 & -2 & -4 & 3 & -2 & -1 \end{bmatrix} \end{aligned}$$

In **CSR** format,

$$\begin{aligned} \texttt{row offsets:} &\quad \begin{bmatrix} 0 & 5 & 6 & 7 \end{bmatrix} \\ \texttt{column indexes:} &\quad \begin{bmatrix} 0 & 2 & 5 & 6 & 7 & 7 & 5 & 3 \end{bmatrix} \\ \texttt{values:} &\quad \begin{bmatrix} -7 & 3 & 4 & -2 & -4 & 3 & -2 & -1 \end{bmatrix} \end{aligned}$$

COO is a very general purpose sparsity format that stores only the values that are non-zero. **CSR** is a better format for row access, giving way to faster matrix operations.

Problem 3: Deep Learning.

(a). Write down the expression for (forward) inference using a deep feedforward fully connected neural network. (That is, if I have an already trained neural network, how do I use it to get a prediction given a training example x?)

A: Copying from the recurrence relation from lecture 11:

$$\forall l \in \{1, \ldots, \mathcal{L}\}, \quad a_l = W_l \cdot o_{l-1} + b_l$$
$$\forall l \in \{1, \ldots, \mathcal{L}\}, \quad o_l = \sigma_l(a_l)$$

The forward inference expression for a single training example x would be:

$$f = \sigma_{\mathcal{L}}(W_{\mathcal{L}} \cdot \sigma_{\mathcal{L}-1}(W_{\mathcal{L}-1} \cdots \sigma_2(W_2 \cdot \sigma_1(W_1 x + b_1) + b_2) \cdots + b_{\mathcal{L}-1}) + b_{\mathcal{L}})$$

(b). What is the computational cost of running inference on a deep neural network? Suppose that all the nonlinearities operate element-wise using the ReLU function

$$\mathsf{ReLU}(a) = \max(a, 0)$$

and the sizes of the layers are d_0 (where $x \in \mathbb{R}^{d_0}$), $d_1, d_2, \ldots, d_{\mathcal{L}} = 1$. How many numerical operations would computing this forward pass require? Which operation dominates: the matrix multiplies or the nonlinearities?

A: It would take $\mathcal{L}$ additions, $\mathcal{L}$ multiplications and $\mathcal{L}$ nonlinearities of ReLU to compute the forward pass. Neither would dominate as both operations are used $\mathcal{L}$ times.

(c). Now write down the expression/algorithm for backpropagation on this same network, using ReLU activation functions. (That is, how do I compute the gradient of this neural network with respect to the weights for a training example x?)

A: The backpropogation rule would be:

$$\frac{\partial f}{\partial a_l} = \mathsf{ReLU}'(a_l) \cdot \frac{\partial f}{\partial o_l} \qquad \frac{\partial f}{\partial o_{l-1}} = W_l^T \cdot \frac{\partial f}{\partial a_l}$$
$$\nabla_{W_l} f = \frac{\partial f}{\partial a_l} \cdot o_{l-1}^T \qquad \nabla_{b_l} f = \frac{\partial f}{\partial a_l}$$

(d). What is the computational cost of running backpropagation on a deep neural network? That is, how many numerical operations would computing backpropagation require? Which operation dominates: the matrix multiplies or the nonlinearities?

A: To update all parameters in the deep neural network, we would need $3\mathcal{L}$ multiplications, and $\mathcal{L}$ nonlinearities of ReLU$'$ operations, assuming that all a_l and o_l were computed and stored in the forward pass. In backpropagation, the matrix multiplies dominate.