CSC413 Miterm Review

February 12, 2020

Basics

Linear Models:

- understand the expressivity & limitation
- given the loss function, derive its gradient w.r.t. parameters
- given the gradient, characterize the stationary point

Nonlinearity:

- recognize commonly-used activation functions
- why is nonlinearity useful in representing certain functions?

Neural Network

Fully-connected Networks:

- definition of neuron; parameter count and computation cost
- construct MLP to approximate certain functions (XOR, sort, etc.)

Convolutional Neural Networks:

- definition of convolution filters; parameter count and computational cost; apply filter to matrices
- benefit of CNNs over MLPs on images

Back-propagation

Computation Graph:

- write computation graph for simple architectures (MLPs, word embedding models, etc.)
- apply the chain rule to derive the backprop equations
- computational cost: understand why certain operations can be efficiently implemented (i.e. Jacobian-vector product)

Optimization

Geometry of Optimization:

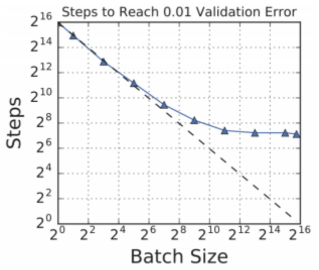
- definition of local / global minima; curvature
- benefit and implicit bias of normalization methods (e.g. 2nd order methods)

Gradient Descent:

- the gradient descent algorithm and its variants (SGD, batch GD, Adam, etc.)
- understand the role of hyperparameters (learning rate, batch size, momentum, etc.)

2019 Midterm Q4

4. **[2pts]** Recall the following plot of the number of stochastic gradient descent (SGD) iterations required to reach a given loss, as a function of the batch size:

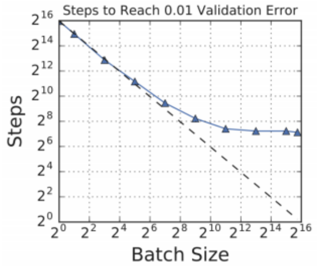


.

(a) **[1pt]** For small batch sizes, the number of iterations required to reach the target loss decreases as the batch size increases. Why is that?

2019 Midterm Q4

4. **[2pts]** Recall the following plot of the number of stochastic gradient descent (SGD) iterations required to reach a given loss, as a function of the batch size:

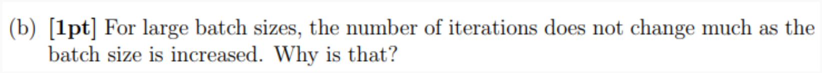


.

(a) **[1pt]** For small batch sizes, the number of iterations required to reach the target loss decreases as the batch size increases. Why is that?

Larger batch sizes reduces the variance in the gradient estimation of SGD. Hence, larger batch converges faster than smaller batch.

2019 Midterm Q4



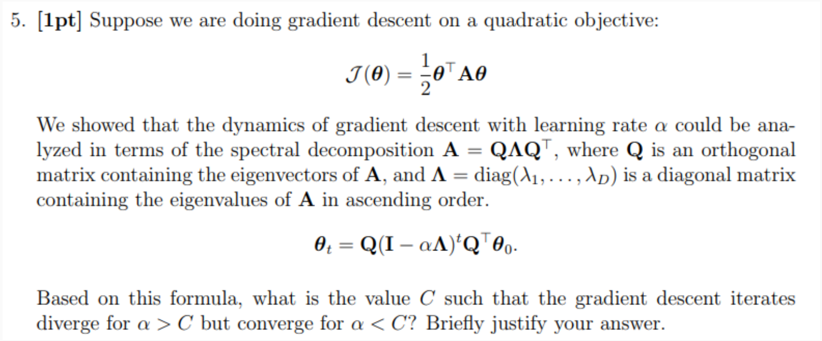
(b) **[1pt]** For large batch sizes, the number of iterations does not change much as the batch size is increased. Why is that?

2019 Midterm Q4

(b) **[1pt]** For large batch sizes, the number of iterations does not change much as the batch size is increased. Why is that?

As the batch size grows larger, SGD effectively becomes full batch gradient descent.

2019 Midterm Q5



5. **[1pt]** Suppose we are doing gradient descent on a quadratic objective:

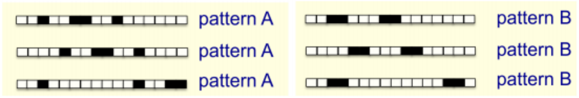
$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^\top \mathbf{A}\boldsymbol{\theta}$$

We showed that the dynamics of gradient descent with learning rate α could be analyzed in terms of the spectral decomposition $\mathbf{A} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top$, where $\mathbf{Q}$ is an orthogonal matrix containing the eigenvectors of $\mathbf{A}$, and $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_D)$ is a diagonal matrix containing the eigenvalues of $\mathbf{A}$ in ascending order.

$$\boldsymbol{\theta}_t = \mathbf{Q}(\mathbf{I} - \alpha\boldsymbol{\Lambda})^t\mathbf{Q}^\top\boldsymbol{\theta}_0.$$

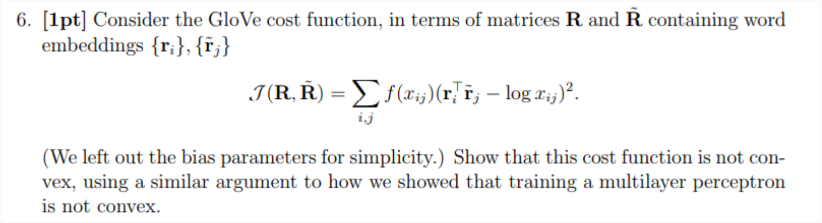
Based on this formula, what is the value C such that the gradient descent iterates diverge for $\alpha > C$ but converge for $\alpha < C$? Briefly justify your answer.

2. **[1pt]** Consider the following binary classiciation problem from Lecture 3, which we showed was impossible for a linear classifier to solve.



The training set consists of patterns A and B in all possible translations, with wrap-around. Consider a neural network that consists of a 1D convolution layer with a linear activation function, followed by a linear layer with a logistic output. Can such an architecture perfectly classify all of the training examples? Why or why not?

2019 Midterm Q6



6. [**1pt**] Consider the GloVe cost function, in terms of matrices $\mathbf{R}$ and $\tilde{\mathbf{R}}$ containing word embeddings $\{\mathbf{r}_i\}, \{\tilde{\mathbf{r}}_j\}$

$$\mathcal{J}(\mathbf{R}, \tilde{\mathbf{R}}) = \sum_{i,j} f(x_{ij})(\mathbf{r}_i^\top \tilde{\mathbf{r}}_j - \log x_{ij})^2.$$

(We left out the bias parameters for simplicity.) Show that this cost function is not convex, using a similar argument to how we showed that training a multilayer perceptron is not convex.

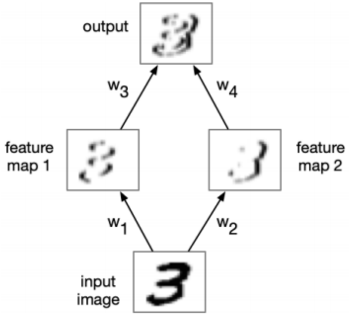
2019 Midterm Q8

8. **[2pts]** In this question, you will design a convolutional network to detect vertical boundaries in an image. The architecture of the network is as shown on the right.

The ReLU activation function is applied to the first convolution layer. The output layer uses the linear activation function.

For this question, you may assume either the standard definition of convolution (which flips and filters) or the version used in conv nets (which skips the filtering step). Conveniently, the same answer works either way.

In order to make the figure printable for the exam paper, we use white to denote 0 and darker values to denote larger (more positive) values.

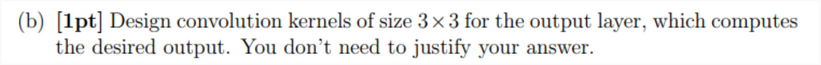


(a) **[1pt]** Design two convolution kernels for the first layer, of size 3×3. One of them should detect dark/light boundaries, and the other should detect light/dark boundaries. (It doesn't matter which is which.) You don't need to justify your answer.

2019 Midterm Q8

$$w_1 = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad w_2 = \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array}$$

2019 Midterm Q8



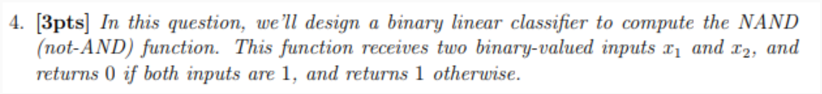
(b) **[1pt]** Design convolution kernels of size 3×3 for the output layer, which computes the desired output. You don't need to justify your answer.

2019 Midterm Q8

(b) **[1pt]** Design convolution kernels of size 3×3 for the output layer, which computes the desired output. You don't need to justify your answer.

$$w_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad w_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

2018 Midterm Q4 (Afternoon Section)



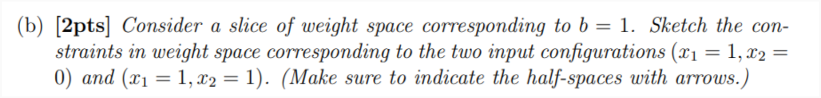
4. **[3pts]** *In this question, we'll design a binary linear classifier to compute the NAND (not-AND) function. This function receives two binary-valued inputs* x_1 *and* x_2*, and returns* 0 *if both inputs are* 1*, and returns* 1 *otherwise.*

2018 Midterm Q4 (Afternoon Section)

4. **[3pts]** *In this question, we'll design a binary linear classifier to compute the NAND (not-AND) function. This function receives two binary-valued inputs* x_1 *and* x_2*, and returns* 0 *if both inputs are* 1*, and returns* 1 *otherwise.*

(a) **[1pt]** *Give four constraints on the weights* w_1 *and* w_2 *and the bias* b*, i.e. one constraint for each of the 4 possible input configurations.*

2018 Midterm Q4 (Afternoon Section)



(b) **[2pts]** *Consider a slice of weight space corresponding to* $b = 1$. *Sketch the constraints in weight space corresponding to the two input configurations* $(x_1 = 1, x_2 = 0)$ *and* $(x_1 = 1, x_2 = 1)$. *(Make sure to indicate the half-spaces with arrows.)*