HOMEWORK SET 2

CSCI 5980/8980 Think Deep Learning (Fall 2020)

Due 11:59 pm, Oct 28 2020

Instruction Please typeset your homework in LaTeX and submit it as a single PDF file in Canvas. No late submission will be accepted. For each problem, your should acknowledge your collaborators if any. For problems containing multiple subproblems, there are often close logic connections between the subproblems. So always remember to build on previous ones, rather than work from scratch.

Notation We will use small letters (e.g., u) for scalars, small boldface letters (e.g., $\boldsymbol{a}$) for vectors, and capital boldface letters (e.g., $\boldsymbol{A}$) for matrices. For a matrix $\boldsymbol{A}$, $\boldsymbol{a}^i$ (supscripting) means its i-th row as a *row vector*, and $\boldsymbol{a}_j$ (subscripting) means the j-the column as a column vector, and a_{ij} means its (i, j)-th element. $\mathbb{R}$ is the set of real numbers. $\mathbb{R}^n$ is the space of n-dimensional real vectors, and similarly $\mathbb{R}^{m\times n}$ is the space of $m \times n$ real matrices. The dotted equal sign $\doteq$ means defining.

Problem 1 (Autoencoder, factorization, and PCA; gradient descent and back-tracking line search; 6/12) Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ be a collection of points in $\mathbb{R}^n$ and suppose they are zero-centered, i.e., $\sum_{i=1}^m \boldsymbol{x}_i = \boldsymbol{0}$. We write

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^\intercal \\ \vdots \\ \boldsymbol{x}_m^\intercal \end{bmatrix} \in \mathbb{R}^{m\times n},$$

i.e., stacking the data points row-wise into a data matrix. Recall that PCA extracts the first r (with $r \le n$) eigenvectors of $\boldsymbol{X}^\intercal\boldsymbol{X}$, collects them columnwise into a matrix $\boldsymbol{U} \in \mathbb{R}^{n\times r}$, and obtains a new representation of each data point $\boldsymbol{x}_i$ as $\boldsymbol{U}^\intercal\boldsymbol{x}_i \in \mathbb{R}^r$. Geometrically, PCA amounts to deriving the best rank-r linear subspace approximation to the point set $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$:

$$\min_{\substack{\boldsymbol{U}\in\mathbb{R}^{n\times r}:\, \boldsymbol{U}^\intercal\boldsymbol{U}=\boldsymbol{I} \\ \boldsymbol{Z}\in\mathbb{R}^{m\times r}}} \|\boldsymbol{X} - \boldsymbol{Z}\boldsymbol{U}^\intercal\|_F^2 .$$

So a crucial step in PCA is to compute the subspace basis $\boldsymbol{U}$. Let's now generate a synthetic point set as follows (please do NOT change the random seed and the dimensions),

```python
import numpy as np
from numpy.random import default_rng

# fix the random seed; don't modify
rng = default_rng(59808980)

n = 50
m = 200
r = 10

#  generate a random dataset
A = rng.standard_normal(size=(n, r))
B = rng.standard_normal(size=(m, r))
X = B @ A.T + 0.01 * rng.standard_normal(size=(m, n))

#  zero centering
X -= X.mean(axis=0,keepdims=True)
print(A[0, 0])
```

```
0.46481072686208924
```

and complete the following tasks–your $A[0,0]$ should be approximately 0.4648. **Please submit your code for this problem as a separate .ipynb file, not to be combined with the codes for other problems. No auto-differentiation is allowed for solving this problem.**

(a) Continue the above Python code to compute the basis for the best rank-10 subspace approximation to $\boldsymbol{X}$, i.e., a matrix $\boldsymbol{A}_1 \in \mathbb{R}^{n\times 10}$ that contains the first 10 PCA basis vectors. (0.5/12)

(b) A classic unsupervised learning technique in deep learning is the autoencoder (we'll cover it later in the course). The mathematical formulation specialized to our case is

$$\min_{\boldsymbol{A}\in\mathbb{R}^{n\times 10}} \; f(\boldsymbol{A}) \doteq \|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{A}\boldsymbol{A}^\intercal\|_F^2 .$$

(i) Derive the gradient of the objective, i.e., $\nabla f(\boldsymbol{A})$ (hint: it is much easier to use the Taylor expansion method rather than the chain rule method). (0.5/12)

(ii) Implement the gradient descent method to solve the optimization problem, with **backtracking line search** for the step size (1/12) and appropriate stopping criterion (say, by checking the gradient norm; 0.5/12).

(iii) Let's say the solution computed from the last step is $\boldsymbol{A}_2$. Now we want to compare the subspaces represented by $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$. We cannot directly do $\boldsymbol{A}_1 - \boldsymbol{A}_2$, as from linear algebra we know that $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ can span the same column/range space, but take very different forms. Instead, a reasonable metric here is the difference between the subspace projectors induced by them, i.e., $\left\|\boldsymbol{A}_1\boldsymbol{A}_1^\dagger - \boldsymbol{A}_2\boldsymbol{A}_2^\dagger\right\|_F$, where $\boldsymbol{A}^\dagger$ denotes the matrix pseudoinverse (https://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_inverse; in Numpy, you can call this function `numpy.linalg.pinv`). Report your result here. Is it close to 0 or not? (0.5/12)

(c) Consider another formulation, which is normally called **factorization**:

$$\min_{\boldsymbol{A}\in\mathbb{R}^{n\times 10},\boldsymbol{Z}\in\mathbb{R}^{m\times 10}} \; g(\boldsymbol{A},\boldsymbol{Z}) \doteq \|\boldsymbol{X} - \boldsymbol{Z}\boldsymbol{A}^\intercal\|_F^2 .$$

(i) Derive the gradient of the objective, i.e., $\nabla g(\boldsymbol{A},\boldsymbol{Z})$ (hint: it is much easier to use the Taylor expansion method rather than the chain rule method; also, one can derive the gradient for one block each time when there are multiple blocks of variables). (1/12)

(ii) Implement the gradient descent method to solve the optimization problem, with **backtracking line search** for the step size (1/12) and appropriate stopping criterion (say, by checking the gradient norm; 0.5/12)

(iii) Let's say the solution computed from the last step is $\boldsymbol{A}_3$. Please compute the subspace differences between $\boldsymbol{A}_1$, $\boldsymbol{A}_2$ and $\boldsymbol{A}_3$ and report your results here. Are they close to 0 or not? (0.5/12)

Problem 2 (Nonlinear least-squares and 2nd order methods; 4/12) Solving linear equation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, or equivalently the linear least-squares $\min_{\boldsymbol{x}} \; \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$, is classic. What about quadratic equations? Suppose we have $y_i = (\boldsymbol{a}_i^\intercal \boldsymbol{x})^2$ for $i = 1,\ldots,m$. Given y_i's and $\boldsymbol{a}_i$'s, we want to recover $\boldsymbol{x} \in \mathbb{R}^n$. It turns out all of sudden the problem becomes NP-hard in general.

Fortunately, the case $\boldsymbol{A}$ is random is qualitatively easier. Let's explore this a bit. *To make sure we can reproduce your results, please fix a random seed in Numpy similar to* ***Problem 1*** *to a number you like.*

Let's generate the data as follows: fix $n = 20$ and $m = 100$. Pick an $\boldsymbol{x} \neq \mathbf{0}$ you like (say random), and generate $\boldsymbol{a}_i$'s as iid standard normal, and then compute $y_1, \ldots, y_m$ accordingly.

(a) Consider a nonlinear least-squares formulation of the problem

$$\min_{\boldsymbol{x}} \ \frac{1}{4} \sum_{i=1}^{m} \left(y_i - (\boldsymbol{a}_i^\top \boldsymbol{x})^2 \right)^2 .$$

Derive the gradient and Hessian of the objective (hint: applying Taylor expansion method with 2nd order expansion might be easier than other means; 1/12) and implement the Newton's method. If the default step size 1 is not working (you can keep track of the objective value, and see if it is monotonically decreasing—include a plot of this in your report), try to implement the backtracking linear search to choose an appropriate step size. Does it solve your problem? The global minimum should be zero. (1/12)

(b) We did not cover it in the class, but Gauss-Newton method is a specialized method for solving nonlinear least-squares problems and can be considered as an approximate Newton's method. You can learn the method from https://en.wikipedia.org/wiki/Gauss%E2%80%93Newton_algorithm, or whatever sources you prefer. Implement the method and check if you find the global minimum. (2/12)

Problem 3 (MNIST classification with a 2-layer neural network; 2/12) In this problem, we perform simple digit recognition on the famous MNIST handwritten digit dataset. In case you haven't heard of it, check the website http://yann.lecun.com/exdb/mnist/. We will set up the problem in PyTorch, but you are free to choose TensorFlow as well. Here is a PyTorch example for setting up a whole classifier training pipeline https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#training-an-image-classifier. Go through and try to understand the example before answering the following questions.

(a) Load the MNIST dataset into workspace. For your information, similar to the CIFAR dataset, MNIST is a standard dataset in torchvision https://pytorch.org/docs/stable/torchvision/datasets.html. (0.5/12)

(b) Implement a two-layer neural network for the classification. For any input $\boldsymbol{x}$ (vector is a column by default), the network is $\sigma\left(\boldsymbol{x}^\top \boldsymbol{W}_1\right) \boldsymbol{W}_2$ (you can optionally add in biases also), and let's use $\sigma = \mathrm{ReLU}$ and the cross-entropy loss for this multi-class problem https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html.

- Implement gradient descent to minimize the loss. You can refer to this example https://pytorch.org/tutorials/beginner/pytorch_with_examples.html#pytorch-tensors, which is very close to what we want to do. **No auto differentiation, i.e., backward(), or PyTorch built-in optimizer allowed**. (1/12)
- Report your test accuracy. You can tune around the number of hidden nodes to optimize the performance. We expect to see at least 95% test accuracy. (0.5/12)