2 K-SVD

As you have seen in earlier homework problems, sparse representations are powerful. When we know the right dictionary of features within which our input is likely to have a sparse representations (possibly with some additional small non-sparse noise), we can use techniques like the LASSO or OMP (or even just rounding/thresholding) to recover the coefficients. As we have seen, this has a fundamentally better bias/variance trade-off.

But what if we don’t know the dictionary? How can the dictionary itself be found from the training data? This is known as dictionary learning. In this problem, we will introduce the K-SVD algorithm (Aharon, M., Elad, M. and Bruckstein, A., 2006. “k-SVD: An algorithm for designing overcomplete dictionaries for sparse representation.” IEEE Transactions on Signal Processing, 54(11), pp.4311-4322.) for unsupervised dictionary learning. (The naive perspective on directly supervised dictionary learning would reveal the dictionary to us, so that is not very interesting. However, it should be clear that the dictionary learning algorithm here can be adapted to deal with the case where we already know some dictionary elements to start with and want to learn more. It should also be clear that there are EM-based approaches to do dictionary learning in a more fully Bayesian perspective, as well as that various deep neural-network architectures are in effect trying to do dictionary learning at early layers to find the features that combine to best represent the signals being presented.) The K-SVD setting is where we would like to find a dictionary that enables good sparse fits to our data in the standard least-squares sense.

Mathematically, given an input matrix \( X \in \mathbb{R}^{n \times d} \), where \( n \) is the number of data points and \( d \) is the dimension of each data point, we want to solve the following optimization problem:

\[
\begin{align*}
\text{minimize} & \quad \|X - ZD\|_F^2 \\
\text{subject to} & \quad \|z_i^T\|_0 \leq s \text{ for all } i.
\end{align*}
\]

We explain the notation a bit more here:

- The matrix \( D \in \mathbb{R}^{K \times d} \) is called a dictionary or a codebook. Each row \( D_k \) represents a dictionary vector/element/codeword/atom.

- Each data sample \( x_i \) is presumed to be a combination of the dictionary vectors and the coefficients are given by \( z_i^T \), the \( i \)th row of \( Z \). See equation (2).

This homework is due **Thursday, May 3rd at 10pm.**
• We denote by $\|v\|_0$ the “zeroth-norm” of vector $v$, which is defined to be the number of nonzero entries in the vector $v$.

To summarize, we have $X \in \mathbb{R}^{n \times d}$, $Z \in \mathbb{R}^{n \times K}$, and $D \in \mathbb{R}^{K \times d}$ and that

$$X = \begin{bmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_n^\top \end{bmatrix}, \quad D = \begin{bmatrix} D_1^T \\ D_2^T \\ \vdots \\ D_K^T \end{bmatrix} \quad \text{and} \quad Z = \begin{bmatrix} z_1^T \\ z_2^T \\ \vdots \\ z_n^T \end{bmatrix} = [Z_1, Z_2, \ldots, Z_K]$$

with $D_k \in \mathbb{R}^d$, $z_i \in \mathbb{R}^K$ and $Z_k \in \mathbb{R}^n$.

And to summarize the goal: Our goal is to find a dictionary $D$ such that we can approximate the data samples $x_i$ by a sparse linear combination of atoms in $D$:

$$x_i^\top \approx \sum_{k=1}^{K} z_{ik} D_k^\top,$$  \hfill (2)

where the coefficients of linear combination are given in $Z$ (which also needs to be determined) and should have at most $s$ nonzero entries for each sample.

A greedy iterative algorithm called K-SVD is proposed for the above optimization problem. The algorithm optimizes the objective over $D$ and $Z$ in an alternating fashion. The pseudo code of the algorithm is given in Algorithm 1. It alternates between two stages: Sparse coding (Algorithm 2) and Update Codebook (Algorithm 4). The Sparse Coding procedure finds a sparse representation for each data sample with Algorithm 3 based on a given dictionary. The procedure for updating the dictionary updates each row of the dictionary with Algorithm 5 in a for loop, while modifying the effected representation coefficients correspondingly.

We have two goals in this problem: (1) to establish a relationship of K-SVD to the K-means algorithm you already know, and (2) to show that this K-SVD algorithm converges to a local minimum. (For this reason in practice, K-SVD is run with multiple random initial conditions to make sure
Algorithm 2: Sparse-coding

**Input:** Dictionary $\mathbf{D}$; Coefficient matrix $\mathbf{Z} \in \mathbb{R}^{n \times K}$; Data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$; Sparsity constraint $s$

**Output:** The coefficient matrix $\mathbf{Z} \in \mathbb{R}^{n \times K}$

**function** Sparse-coding $(\mathbf{D}, \mathbf{Z}, \mathbf{X}, s)$:

```
for $i = 1, \ldots, n$ do
    $z'_i = \text{Sparse-coding-single}(\mathbf{D}, x_i, s)$
    if $\| x_i^\top - z'_i^\top \mathbf{D} \|_2 > \| x_i^\top - (z'_i')^\top \mathbf{D} \|_2$ then
        $z_i \leftarrow z'_i$
    end
end
return $\mathbf{Z} = \begin{bmatrix} z_1^\top \\
                        \vdots \\
                        z_n^\top \end{bmatrix}$
```

end function

Algorithm 3: Sparse-coding-single

**Input:** Dictionary $\mathbf{D}$; Data sample $\mathbf{x} \in \mathbb{R}^d$; Sparsity constraint $s$

**Output:** A coefficient vector $\mathbf{z} \in \mathbb{R}^K$

**function** Sparse-coding-single $(\mathbf{D}, \mathbf{x}, s)$:

```
Initialize $\mathbf{z} \leftarrow 0 \in \mathbb{R}^k$
Initialize the basis $\mathcal{B}_0 = \{\}$
for $j = 1, \ldots, s$ do
    Find the index of the best dictionary vector $\mathbf{D}_{k(j)} \in \mathbb{R}^d$ by solving:
    $$\beta^{(j), k(j)} = \arg\min_{\beta_k \in \mathbb{R}^k, k \in [K]} \| \mathbf{x} - \sum_{\mathbf{D}_\ell \in \mathcal{B}_{j-1} \cup \{\mathbf{D}_{k(j)}\}} \beta_\ell \mathbf{D}_\ell \|_2^2$$
    Update $\mathcal{B}_j \leftarrow \mathcal{B}_{j-1} \cup \{\mathbf{D}_{k(j)}\}$
end
for $\mathbf{D}_\ell \in \mathcal{B}_s$ do
    $z_\ell \leftarrow \beta^{(s)}_\ell$
end
return $\mathbf{z}$
```
end function

HW14, ©UCB CS 189, Spring 2018. All Rights Reserved. This may not be publicly shared without explicit permission.
Algorithm 4: Update-dictionary

**Input:** Dictionary $D$; Coefficient matrix $Z \in \mathbb{R}^{n \times K}$; Data matrix $X \in \mathbb{R}^{n \times d}$

**Output:** Dictionary $D \in \mathbb{R}^{K \times d}$

**function** Update-dictionary ($D, Z, X$):

for $k = 1, \ldots, K$ do

Update $k$-th row $D_k^\top \leftarrow \text{Update-dictionary-single}(D, Z, X, k)$

end

return $D = \begin{bmatrix}
D_1^\top \\
\vdots \\
D_K^\top
\end{bmatrix}$

end function

Algorithm 1: K-SVD

**Input:** Data matrix $X \in \mathbb{R}^{n \times d}$; Number of atoms $K$; Sparsity constraint $s$

**Output:** A dictionary $D \in \mathbb{R}^{K \times d}$, and the coefficient matrix $Z \in \mathbb{R}^{n \times K}$

**function** K-SVD ($X, K, s$):

Initialize $D \leftarrow$ randomly chosen $K$ rows of the dataset $X$ (without replacement)

Initialize $Z \leftarrow \text{Sparse-coding}(D, 0, X, s)$

while not converged do

$Z \leftarrow \text{Sparse-coding}(D, Z, X, s)$

Update-dictionary ($D, Z, X$)

end

return $D, Z$

end function

(a) (Relationship to K-means) Recall the set-up of K-means: Given a data matrix $X$, K-means algorithm partitions the data into $K$ disjoint groups $\pi = \{\pi_1, \pi_2, \ldots, \pi_K\}$. Each sample $x_i$ is contained in exactly one partition $\pi_j$. Recall that the K-means algorithm tries to solve the following optimization problem:

$$\text{minimize}_{\pi, \mu} \sum_{k=1}^{K} \sum_{i \in \pi_k} \|x_i - \mu_k\|_2^2. \tag{3}$$

Let us connect the K-means optimization problem (3) with that given by equation (1). Let $\mu$ denote a $K \times d$ matrix and $\tilde{Z}$ denote an $n \times K$ matrix which are given by

$$\mu = \begin{bmatrix}
\mu_1^\top \\
\vdots \\
\mu_K^\top
\end{bmatrix} \quad \text{and} \quad \tilde{Z} = \begin{bmatrix}
\tilde{z}_1^\top \\
\vdots \\
\tilde{z}_n^\top
\end{bmatrix}.$$
Algorithm 5: Update-dictionary-single

**Input:** Dictionary $D$, Coefficient matrix $Z \in \mathbb{R}^{n \times K}$; Data matrix $X \in \mathbb{R}^{n \times d}$; Index $k$

**Output:** Return a row vector $\tilde{d} \in \mathbb{R}^{1 \times d}$

**function** Update-dictionary-single($D, Z, X, k$):

Compute the error matrix $E_k \in \mathbb{R}^{n \times d}$ that represent how well things are represented without the dictionary vector $D_k^\top$:

$$E_k = X - \sum_{j \neq k} Z_j D_j^\top$$

Find the indices of data samples whose sparse representation uses the dictionary vector $D_k^\top$.
In other words, find the indices corresponding to non-zero entries in the coefficient vector $z_k$:

$$\omega_k = \{i | i \in [n], Z_{k,i} \neq 0\}.$$

and let us denote $\omega_k = \{i_1, \ldots, i_{|\omega_k|}\}$.
Find the matrix $E_k^{\omega_k} \in \mathbb{R}^{|\omega_k| \times d}$ by choosing those rows of $E_k$ whose index belongs to the set $\omega_k$,
i.e., matrix $E_k^{\omega_k}$ is computed by picking the rows with indices $i_1, \ldots, i_{|\omega_k|}$.
Compute the SVD of $E_k^{\omega_k}$:

$$E_k^{\omega_k} = UV^\top$$

where $U \in \mathbb{R}^{|\omega_k| \times |\omega_k|}$, $A \in \mathbb{R}^{|\omega_k| \times d}$ and $V \in \mathbb{R}^{d \times d}$. We assume $A$ is a diagonal matrix with non-increasing diagonal entries ($A_{11}$ is its largest entry). Let $U_1 \in \mathbb{R}^{|\omega_k|}$ and $V_1 \in \mathbb{R}^{d}$ denote the first column of the matrices $U$ and $V$ respectively.
Update $D_k \leftarrow V_1$
Update only the non-zero entries of the vector $z_k^\top$:

```
for $j = 1, \ldots, |\omega_k|$ do
    $z_{k,i_j} = U_1,_{i_j} A_{11}
```

end

return $D_k^\top$

**end function**
Show that the objective function given in the problem (3) is equivalent to

\[
\minimize_{\hat{Z} \in \mathbb{R}^{n \times K}, \mu \in \mathbb{R}^{K \times d}} \| X - \hat{Z} \mu \|^2_F
\]
subject to \( \hat{z}_i \in \{0, 1\}^K \) and \( \| \hat{z}_i \|_0 = 1 \) for all \( i = 1, \ldots, n. \)

(4)

Conclude that K-means is equivalent to K-SVD with \( s = 1 \) with an additional constraint of forcing the coefficient entries in the matrix \( \hat{Z} \) to take values in the set \( \{0, 1\} \).

Solution: We have

\[
\| X - \hat{Z} \mu \|^2_F = \sum_{i=1}^{n} \| x_i^T - \hat{z}_i^T \mu \|^2_2 = \sum_{i=1}^{n} \| x_i - \mu^T \hat{z}_i \|^2_2.
\]

(5)

For each \( i \), assume \( x_i \) lies in the \( k(i) \)th class, then \( \hat{z}_{i,k(i)} = 1 \) and the other terms in \( \hat{z}_i \) are zero. Thus \( \mu^T \hat{z}_i = \mu_{k(i)} \). That means, for each \( i \), if \( x_i \) lies in the \( k \)th class, we have

\[
\| x_i - \mu^T \hat{z}_i \|^2_2 = \| x_i - \mu_k \|^2_2.
\]

Partitioning the data into \( k \) classes, we have

\[
\sum_{i=1}^{n} \| x_i - \mu^T \hat{z}_i \|^2_2 = \sum_{k=1}^{K} \sum_{i \in \pi_k} \| x_i - \mu_k \|^2_2.
\]

Therefore, we have

\[
\sum_{k=1}^{K} \sum_{i \in \pi_k} \| x_i - \mu_k \|^2_2 = \| X - \hat{Z} \mu \|^2_F,
\]

and thus the two optimization problems are equivalent.

K-means can be viewed as iteratively looking for a dictionary of means so that a 1-sparse representation of the vectors where we are forced to use the coefficient 1 for a dictionary element if we include it.

(b) Show that at each step \( j = 1, 2, \ldots, s \) in Algorithm 3 for finding a sparse solution, the min value of the objective \( \min_{\beta \in \mathbb{R}^J} \min_{k \in [K]} \| x - \sum_{\ell \in B_{j-1} \cup \{D_k\}} \beta_\ell D_\ell \|^2_2 \) is non-increasing.

The following hint may or may not be useful for you but is worth thinking through. \textit{Hint: Can you identify this algorithm as OMP?}

Solution: This algorithm is exactly doing orthogonal matching pursuit. At each step, we take the previous set of dictionary elements chosen and see how much the residual would shrink if we added another candidate element. The best candidate is added to set, and then we iterate. We know that the residuals get better (smaller norm) as OMP proceeds. The same will hold here.

In Algorithm 3 we do

\[
\hat{\beta}^{(j)}, k^{(j)} = \arg \min_{\beta \in \mathbb{R}^J} \min_{k \in [K]} \| x - \sum_{D_\ell \in B_{j-1} \cup \{D_k\}} \beta_\ell D_\ell \|^2_2
\]
at step $j$. Because we can always set the coefficient in front of the newly selected variable to be 0 and keep the coefficients in front of the previously selected variables unchanged, we conclude that

$$\min_{\beta \in \mathbb{R}} \min_{k \in [K]} \left\| x - \sum_{D_k \in B_{j-1} \cup \{D_k\}} \beta_k D_k \right\|^2_2 \leq \left\| x - \sum_{D_k \in B_{j-1}} \beta_k^{(j-1)} D_k \right\|^2_2$$

where

$$\beta^{(j-1)} = \arg\min_{\beta \in \mathbb{R}^{j-1}} \left\| x - \sum_{D_k \in B_{j-1}} \beta_k^{(j-1)} D_k \right\|^2_2.$$

is the previous best update. Thus the new residual error is generally smaller than the previous residual error.

(c) (Sparse coding decreases the objective) Conclude from the above part that Algorithm 2 cannot increase the value of the objective function $f$. (In practice, people use straight matching pursuit — Algorithm 6 — to replace Algorithm 3 which is much more efficient, but gets comparable performance in most practical settings where there isn’t a huge dynamic range of coefficients and the target sparsity $s$ is low. But showing that matching pursuit usually works is omitted here.)

**Algorithm 6: Matching-pursuit**

**Input:** Dictionary $D$; Data sample $x \in \mathbb{R}^d$; Sparsity constraint $s$

**Output:** A coefficient vector $z \in \mathbb{R}^K$

**function** Sparse-coding-single-in-practice ($D, x, s$):

1. Initialize $z \leftarrow 0 \in \mathbb{R}^K$
2. Initialize the basis $B_0 = \{\}$
3. Initialize the residue $r_0 = x$
4. for $j = 1, \ldots, s$ do
   1. Find the best dictionary vector $D_{k(j)}^\top \in \mathbb{R}^d$ by solving:
      $$k(j) = \arg\max_{k \in [K]} \frac{|r_{j-1}^\top D_k|}{\|D_k\|^2_2} = \arg\min_{k \in [K]} \min_{\beta_k \in \mathbb{R}} \left\| r_{j-1} - \beta_k D_k \right\|^2_2$$
   2. Update the coefficient $z_{k(j)} \leftarrow \frac{|r_{j-1}^\top D_k|}{\|D_k\|^2_2}$
   3. Update the residue $r_j \leftarrow r_{j-1} - z_{k(j)} D_{k(j)}$
5. end
6. return $z$.

**Solution:** Algorithm 2 uses a for loop to loop through the data points.

And for every such point, we update the $z_i$ to $z_i'$ only if the new proposed coefficients $z_i'$ decrease the objective value — bring us closer to the data point. If not, the objective remains unchanged since the representation is not updated.
As a result, we can conclude that the Algorithm 2 does not increase the objective value.

(d) Show that the single-atom dictionary update given by Algorithm 5 does not increase the objective function while preserving the sparsity constraint. (You may or may not find the following hint useful.)

**Hint: Recall the Eckart-Young theorem.**

**Solution:** What this step does is to look at all of the data points that use a certain dictionary element, look at the matrix of residuals if we omit this one dictionary element, and then replace the dictionary element with the hypothesized best rank-1 approximation to this matrix of residuals (obtained via the SVD by means of the largest singular value). Because it is the best rank-1 estimate, we have to do no worse in the Frobenius norm on those vectors than if we had kept the original dictionary element. So, there is a collection of data points whose representation is unchanged (those which did not use that dictionary element) and there are others who now are either strictly better represented or no worse represented on average. So the overall average representation quality cannot get worse. This is the spirit of why this algorithm works.

We write the objective as
\[
\|X - ZD\|_F^2 = \|X - \sum_{j=1}^{K} Z_j D_j^T\|_F^2.
\]
\[
= \|\left(X - \sum_{j \neq k} Z_j D_j^T\right) - Z_k D_k^T\|_F^2.
\]
\[
= \|E_k - Z_k D_k^T\|_F^2.
\]

Recall that \(\omega_k = \{i | i \in [n], Z_{k,i} \neq 0\}\). Let \(\omega_k^c = [n] \setminus \omega_k\) denote the set of indices for which \(Z_{k,i} = 0\). Separating the sum above into two sets based on whether the indices in \(Z_k\) are zero or not,
\[
\|E_k - Z_k D_k^T\|_F^2 = \|E_k^{\omega_k} - Z_k^{\omega_k} D_k^T\|_F^2 + \|E_k^{\omega_k^c} - Z_k^{\omega_k^c} D_k^T\|_F^2.
\]
where we have used the fact that \(Z_{k, \omega_k^c}\) is a zero-vector.

Recall the SVD notation: \(E_k^{\omega_k} = U \Lambda V\). If we update \(D_k\) to be the first column \(V_1\) of the matrix \(V\) and update the nonzero element of \(Z_k\) to be the first column \(U_1\) of the matrix \(U\) multiplied by the largest singular value \(\Lambda_{11}\) of the matrix \(E_k^{\omega_k}\), we have that \(Z_k^{\omega_k} D_k^T = \Lambda_{11} U_1 V_1^T\). Recalling the Eckart-Young theorem, we have that
\[
\|E_k^{\omega_k} - \Lambda_{11} U_1 V_1^T\|_F \leq \|E_k^{\omega_k} - ab^T\|_F,
\]
for any vector \(a, b\), and hence we also have that for previous vectors \(Z_k^{\omega_k}, D_k^T\) we have
\[
\|E_k^{\omega_k} - \Lambda_{11} U_1 V_1^T\|_F \leq E_k^{\omega_k} - Z_k^{\omega_k} D_k^T\|_F.
\]
As a result, updating \(Z_k^{\omega_k} \leftarrow \Lambda_{11} U_1\) and \(D_k \leftarrow V_1\) decreases the objective. This is precisely the update performed in Algorithm 5 and our claim follows.
(e) (Updating dictionary decreases the objective) **Conclude that the iterations of the K-SVD algorithm put together cannot increase the objective function and hence the objective function must converge.**

**Solution:** Algorithm uses a for loop to go through the atoms one at a time, and each iteration does not increase the objective value by the previous part. Thus Algorithm cannot increase the objective value.

Putting everything together, each step of the K-SVD algorithm always leaves alone or strictly improves (decreases) the objective function. This objective is non-negative by construction. A non-increasing sequence of non-negative numbers must converge.

(f) (Implementation, BONUS) Alas! Here we have a bad news. Due to time and resource constraints, we were unable to provide a good and debugged implementation part to this problem. You are free to wander around and play with the starter code and construct simulated datasets (or find real world data sets) where K-SVD recovers the underlying dictionary and/or provides a good reconstruction of the data using sparse linear combinations. No submission is required but you are encouraged to share your empirical findings on the piazza post associated with this problem. Feel free to share the datasets/results that you find. Extra credit will be given to those who help put together a nice exploration part of this problem. Given that the original K-SVD paper has over six thousand citations, we are confident that it should be possible to do this, and fun too.

**HINT:** For those who have taken EE16A in a semester where Massive-IoT-CDMA was used to motivate OMP, can you come up with an example where an eavesdropper listens in without knowledge of the wireless device codes but after lurking for a while and hearing small numbers of devices talk simultaneously can now figure out what all the individual devices’ codes were? Alternatively, how about imaging lots of very sparse unknown images but not knowing what the imaging masks were. Can you learn the masks from lots of unknown sparse images being observed using the unknown random masks? Can you replicate the seminal work of Olshausen and Field from 1996 that showed that dictionary learning can help explain mammalian vision? See the Proceedings of the IEEE survey article on dictionary learning 10.1109/JPROC.2010.2040551 for more potential references.

**Solution:** Give yourself full credit if you played around and got anywhere. Be sure to remember to submit any real substantive progress here for extra credit.

3 Dropout in Neural Network and Linear Regression

Dropout is an algorithm to “prevent over-fitting” that is used while training lot of modern neural network implementations. This question will introduce how dropout works and why we can understand dropout as method to do regularization. Consider the simplest neural network – linear regression, which tries to find a vector $w$ to minimize

$$\ell(w) = \|y - Xw\|_2^2 = \sum_{i=1}^{n} (y_i - x_i^\top w)^2.$$  \hfill (7)
Dropout is usually used with stochastic gradient descent. Instead of doing a stochastic gradient (with mini-batch-size 1) step using

$$w_{t+1} = w_t - \alpha \nabla_w (y_i - x_i^\top w)^2,$$

we instead apply the formula

$$w_{t+1} = w_t - \alpha \nabla_w \left[ \left( y_i - \frac{1}{p} (x_i \circ m) \right)^\top w \right]^2. \quad (8)$$

Here $\alpha$ is the learning rate, $p$ is a user-specified dropout retention rate from the $(0, 1)$ interval, $i \sim U(1, n)$ is an index drawn from the uniform distribution, $\circ$ is the element-wise multiplication operator, and $m$ is a random vector where $m_i \sim \text{Bernoulli}(p)$ iid. In other words, we will randomly mask each feature of a sample to be zero with a probability $1 - p$ during SGD steps.

While this problem only deals with dropout on linear regression, SGD steps with dropout seen here can easily generalize to any layers of neural networks that contain trainable parameters. We can randomly mask each element of the output vector from the previous layer to be zero with a probability $1 - p$ to regularize the training of the current layer. Dropout is commonly used with fully-connected layers and recurrent neural networks as they often contain a large number of parameters, making them prone to overfitting by nature.

(a) For simplicity, let us say that you are only solving a linear regression problem with only two features, i.e.,

$$X = [x^{(1)} \ x^{(2)}].$$

And for concreteness, suppose that we are using $p = \frac{2}{3}$ as the dropout retention rate.

**Find a dataset $X'$ and $y'$ so that the normal SGD steps on the dataset $X'$ and $y'$ is effectively the same as the dropout SGD steps on the original $X$ and $y$.** Hint: You can augment the dataset and make it effectively be a linear regression problem by making multiple copies of data points. You can define

$$X' = \begin{bmatrix} \alpha_1 x^{(1)} & \alpha_1 x^{(2)} \\ \alpha_2 x^{(1)} & 0 \\ 0 & \alpha_3 x^{(2)} \\ 0 & 0 \end{bmatrix},$$

and

$$y' = \begin{bmatrix} \beta_1 y \\ \beta_2 y \\ \beta_3 y \\ \beta_4 y \end{bmatrix},$$

and **find the expression for** $\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3,$ and $\beta_4$, along with how many copies you want to include of each “row” above in your augmented data set. Verify that the SGD steps are indeed the same by showing that the probability of making each kind of update to the weights is the same.

**Solution:**

HW14, ©UCB CS 189, Spring 2018. All Rights Reserved. This may not be publicly shared without explicit permission.
**Method 1** To make the gradient the same, we can assign $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \frac{1}{p} = \frac{3}{2}$, and $\beta_1 = \beta_2 = \beta_3 = \beta_4 = 1$.

Notice that the four possible assignments of random variable $m$ have the same probabilities

\[
\Pr[m = (1, 1)] = \frac{4}{9}, \quad \Pr[m = (0, 1)] = \frac{2}{9}, \quad \Pr[m = (1, 0)] = \frac{2}{9}, \quad \Pr[m = (0, 0)] = \frac{1}{9}.
\]

Therefore, we need to duplicate 4 times of the first rows of $X'$ and $y'$, 2 times of the second and the third rows, and 1 times of the fourth rows in the augmented dataset.

**Method 2** Alternatively, you can use the weighted least square so you do need to not duplicate any samples. In this case, you have $\alpha_1 = \frac{12}{7}, \alpha_2 = \frac{6}{7}, \alpha_3 = \frac{6}{7}, \alpha_4 = \frac{3}{2}, \beta_1 = 4, \beta_2 = 2, \beta_3 = 2, \text{and } \beta_4 = 1$.

(b) Leveraging what you have learned from the previous part, **explain why** the SGD steps with dropout for linear regression is the same as doing standard SGD without dropout for the following modified loss function

\[
\ell'(w) = \mathbb{E}_{\mathcal{M}} \left[ \left\| y - \frac{1}{p} (X \odot \mathcal{M}) w \right\|^2 \right],
\]

where $\odot$ is an element-wise matrix multiplication operator and random matrix $\mathcal{M}_{ij} \sim \text{Bernoulli}(p)$ iid.

Hint: You can “unroll” the expectation here so that

\[
\ell'(w) = \frac{1}{n} \sum_{i=1}^{n} \sum_{m} f(x_i \odot m, y_i; w)
\]

and then show that SGD step above is as same as taking a gradient step gradient $f(\cdot)$ with a random sample. Computing $x_i \odot m_i$ can be viewed as augmenting the data like we did in the previous part.

**Solution:** Let

\[
f(x, y; w) = \left( y - \frac{1}{p} x^\top w \right)^2.
\]

Plugging this equation into Equation (12) yields Equation (11). Applying a gradient step on it with random selected $i$ and $m$ yields the SGD update with dropout in Equation (8).
(c) To see the fact that dropout is a way of doing regularization, prove that

$$\arg \min_w \ell'(w) = \arg \min_w \|y - Xw\|_2^2 + \frac{1 - p}{p} \|\Gamma w\|_2^2,$$  \hspace{1cm} (13)

where $\Gamma = \text{diag}(X^T X)^{1/2}$ — i.e. $\Gamma$ is a diagonal matrix with just the Euclidean norms of the feature vectors along the diagonal. This is saying that applying dropout on the linear regression problem is equivalent to doing the ridge regression with a special diagonal Tikhonov regularization matrix. For the effect of dropout on more complex models such as logistic regression or convolution neural networks, it is hard to obtain a closed-form solution. But the idea dropout is equivalent to some kind of regularization still applies.

Hint: If random variables $X, Y \sim \text{Bernoulli}(p)$, we have $E[X^2] = p$ and $E[XY] = p^2$.

Solution: We have

$$\ell'(w) = \sum_{i=1}^n E_m \left[ \left( y_i - \frac{1}{p} (x_i \odot m)^\top w \right)^2 \right]$$

$$= \sum_{i=1}^n E_m \left[ y_i^2 - \frac{2}{p} y_i (x_i \odot m)^\top w + \frac{1}{p^2} [(x_i \odot m)^\top w]^2 \right]$$

$$= \sum_{i=1}^n E_m \left[ y_i^2 - 2y_i x_i^\top w + (x_i^\top w)^2 - (x_i^\top w)^2 + \frac{1}{p^2} [(x_i \odot m)^\top w]^2 \right]$$

$$= \|y - Xw\|_2^2 + \sum_{i=1}^n E_m \left[ -(x_i^\top w)^2 + \frac{1}{p^2} [(x_i \odot m)^\top w]^2 \right]$$

$$= \|y - Xw\|_2^2 + \sum_{i=1}^n E_m \left[ - \left( \sum_{j=1}^d x_{ij}w_j \right)^2 + \frac{1}{p^2} \left( \sum_{j=1}^d x_{ij}w_jm_j \right)^2 \right]$$

$$= \|y - Xw\|_2^2 + \sum_{i=1}^n E_m \left[ - \sum_{j_1=1}^d \sum_{j_2=1}^d x_{ij_1}w_{j_1}x_{ij_2}w_{j_2} + \frac{1}{p^2} \sum_{j_1=1}^d \sum_{j_2=1}^d x_{ij_1}w_{j_1}m_{j_1}x_{ij_2}w_{j_2}m_{j_2} \right].$$

Notice that when $j_1 \neq j_2$, we have that the expectation of the last term is equal to

$$E_m \left[ \frac{1}{p^2} \sum_{j_1=1}^d \sum_{j_2=1}^d x_{ij_1}w_{j_1}m_{j_1}x_{ij_2}w_{j_2}m_{j_2} \right]$$

$$= \frac{1}{p^2} \sum_{j_1=1}^d \sum_{j_2=1}^d x_{ij_1}w_{j_1}x_{ij_2}w_{j_2} E_m [m_{j_1}m_{j_2}]$$

$$= \frac{1}{p^2} \sum_{j_1=1}^d \sum_{j_2=1}^d x_{ij_1}w_{j_1}x_{ij_2}w_{j_2} \left[ \mathbb{I}[j_1 \neq j_2]E_m [m_{j_1}m_{j_2}] + \mathbb{I}[j_1 = j_2]E_m [m_{j_1}^2] \right]$$
\[
\frac{1}{p^2} \sum_{j_1=1}^{d} \sum_{j_2=1}^{d} x_{ij_1}w_{j_1}x_{ij_2}w_{j_2} \left[ I[j_1 \neq j_2]p^2 + I[j_1 = j_2]p \right]
\]
\[
= \sum_{j_1=1}^{d} \sum_{j_2=1}^{d} x_{ij_1}w_{j_1}x_{ij_2}w_{j_2} \left[ I[j_1 \neq j_2]1 + I[j_1 = j_2] \frac{1}{p} \right]
\]

You can see that when \(j_1 \neq j_2\), term 2 will cancel out term 1. Plug this back into \(\ell'\), we have

\[
\ell' (w) = \|y - Xw\|_2^2 + \sum_{i=1}^{n} E_m \left[ - \sum_{j_1=1}^{d} \sum_{j_2=1}^{d} x_{ij_1}w_{j_1}x_{ij_2}w_{j_2} + \frac{1}{p^2} \sum_{j_1=1}^{d} \sum_{j_1=1}^{d} x_{ij_1}w_{j_1}m_jx_{ij_2}w_{j_2}m_{j_2} \right]
\]
\[
= \|y - Xw\|_2^2 + \sum_{i=1}^{n} \left[ \left( \frac{1}{p} - 1 \right) \sum_{j=1}^{d} x_{ij}w_jx_{ij}w_j \right]
\]
\[
= \|y - Xw\|_2^2 + \frac{1-p}{p} \|\Gamma w\|_2^2.
\]

(d) If you normalize the columns of \(X\) (i.e. scale the features) so that \(\|x^{(i)}\|_2^2 = 1\) for each column \(x^{(i)}\), what is the equivalent Tikhonov regularization matrix if you use dropout? Explain why (or in what context) dropout’s induced implicit Tikhonov regularization might be better than the identity matrix that vanilla ridge regression uses when we are working with unnormalized columns of \(X\).

Solution: If we normalize the columns of \(X\), \(\Gamma = I\) will be the identity matrix, which means that linear regression with dropout is equivalent to ridge regression. When the features in \(X\) are not normalized, the corresponding term in the Tikhonov regularization matrix will be larger if the scale of the corresponding feature is bigger.

When we do not have any reason to put a lot of importance on the units used to represent the individual features, this is probably what we want because it will match the magnitude of \(x^{(i)}w_i\) and \(\Gamma_iw_i\). In other word, it will act like doing ridge regression as though each feature was normalized even if we do not explicitly normalize them. This explains the attractiveness of dropout in algorithmic scenarios like within neural nets where normalization of features might not be natural.

(e) Run the code in Part A. Explain what the code is doing and the meaning of the plot.

Solution: The code in Part A is an empirical experiment showing the equivalence of dropout and ridge for linear regression. The plot shows the solution \(w^*\) for a 2D problem for both linear regression with dropout and ridge regression.

(f) The code in Part B compares dropout with bagged linear regression. Read the code and explain the differences and similarities between bagged linear regression and linear regression with dropout (implemented using ridge regression in the code).

Solution: Similarities: Both bagged linear regression and linear regression with dropout apply regularization to the problem.
Differences: Bagged linear regression trains multiple instances of \( w \) and then averages them to regularize \( w \) while dropout and ridge regression do the regularization in a joint way. The arguments above tell us that dropout is effectively tying the weights together across the different bags for bagged linear regression.

4 Your Own Question

Write your own question, and provide a thorough solution.

Writing your own problems is a very important way to really learn the material. The famous “Bloom’s Taxonomy” that lists the levels of learning is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top-level. We rarely ask you any HW questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself. (e.g. make yourself flashcards) But we don’t want the same to be true about the highest level.

As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams.

Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don’t have to achieve this every week. But unless you try every week, it probably won’t happen ever.