CS221 Problem Workout

Week 2

1) [CA session] Problem 1: Least-Squares Linear Regression

In last week's module we studied the linear regression algorithm, which solves a regression problem using a linear predictor via optimizing the objective

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{train}}} (\mathbf{w}\cdot\phi(\mathbf{x}) - y)^2. \tag{1}$$

The training loss was minimized via gradient descent, which works iteratively to decrease the training loss. As mentioned in the module, we can actually solve for the optimal weights $\mathbf{w}^\star$ in closed-form. In this problem we will derive the *normal equations* used to solve for this estimator.

Solution We let $n = |\mathcal{D}_{\text{train}}|$ so that we can write Equation 1 as

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}\cdot\phi(\mathbf{x}_i) - y_i)^2. \tag{2}$$

.

What this shows us is that the training loss takes n real numbers, squares each one, and then adds them all up. But this is exactly the squared Euclidean norm of an n-dimensional vector! In mathematical notation, we can write

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{n}\left\|\begin{bmatrix}\mathbf{w}\cdot\phi(\mathbf{x}_1) - y_1\\ \vdots \\ \mathbf{w}\cdot\phi(\mathbf{x}_n) - y_n\end{bmatrix}\right\|_2^2. \tag{3}$$

This might look a bit cluttered, but we'll introduce some additional notation to clean things up. First, we define the n-dimensional vector

$$\mathbf{y} = \begin{bmatrix}y_1\\ \vdots \\ y_n\end{bmatrix}. \tag{4}$$

In other words, we take each of the n target variables and put them into a column vector. Next, we define the *matrix*

$$X = \begin{bmatrix} \phi(\mathbf{x}_1)^\top \\ \vdots \\ \phi(\mathbf{x}_n)^\top \end{bmatrix}. \tag{5}$$

Here we take each of the feature vectors $\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)$, transpose them into row vectors, and then stack them into an $n \times d$ matrix. With this X and $\mathbf{y}$, we can succinctly write the training loss as

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{n}\|X\mathbf{w} - \mathbf{y}\|_2^2. \tag{6}$$

This looks much better! From this point we can compute the gradient of $\text{TrainLoss}(\mathbf{w})$, set it equal to zero, and solve for the minimizer $\mathbf{w}^\star$. If you're comfortable with vector/matrix calculus then you can probably take it from here. Otherwise, we will actually rework the above expression for the training loss to get it into a form that is more amenable to taking gradients.

We will ignore the $\frac{1}{n}$ term for now and just focus on $\|X\mathbf{w} - \mathbf{y}\|_2^2$ Recall that we can write this squared norm as

$$\|X\mathbf{w} - \mathbf{y}\|_2^2 = (X\mathbf{w} - \mathbf{y})^\top (X\mathbf{w} - \mathbf{y}). \tag{7}$$

To expand (7), we recall four properties of matrix transposes:

$$\begin{aligned} (A + B)^\top &= A^\top + B^\top \\ (cA)^\top &= cA^\top \\ (AC)^\top &= C^\top A^\top \\ (c)^\top &= c \end{aligned}$$

for appropriately sized matrices A, B, C and any scalar c. Then (7) becomes

$$\begin{aligned} (X\mathbf{w} - \mathbf{y})^\top (X\mathbf{w} - \mathbf{y}) &= (\mathbf{w}^\top X^\top - \mathbf{y}^\top)(X\mathbf{w} - \mathbf{y}) && (8) \\ &= \mathbf{w}^\top X^\top X\mathbf{w} - \mathbf{w}^\top X^\top \mathbf{y} - \mathbf{y}^\top X\mathbf{w} + \mathbf{y}^\top \mathbf{y}. && (9) \end{aligned}$$

As it turns out, the two middle terms on the right-hand side of (9) are equal, so we can combine them. To see why this is true, we first observe that both terms are scalars, and hence they are equal to their transposes. In particular,

$$\mathbf{w}^\top X^\top \mathbf{y} = \mathbf{w}^\top (X^\top \mathbf{y}) = (\mathbf{w}^\top (X^\top \mathbf{y}))^\top = (X^\top \mathbf{y})^\top \mathbf{w}, \tag{10}$$

where the first equality follows from the associativity of matrix multiplication, and the last equality follows from the product rule for matrix transposes. In addition, we also have $\mathbf{y}^\top X = (X^\top \mathbf{y})^\top$, so (9) becomes

$$\mathbf{w}^\top X^\top X \mathbf{w} - 2(X^\top \mathbf{y})^\top \mathbf{w} + \mathbf{y}^\top \mathbf{y}. \tag{11}$$

Phew! From here, we can take the gradient with respect to $\mathbf{w}$. We need two gradient identities that you can prove if you're bored on a rainy day:

$$\nabla_{\mathbf{w}}(\mathbf{w}^\top A \mathbf{w}) = A\mathbf{w} + A^\top \mathbf{w} \tag{12}$$

$$\nabla_{\mathbf{w}}(\mathbf{a}^\top \mathbf{w}) = \mathbf{a} \tag{13}$$

for any square matrix $A \in \mathbb{R}^{d \times d}$ and any vector $\mathbf{a} \in \mathbb{R}^d$. In particular, notice that if A is symmetric, then Equation 12 simplifies to $2A\mathbf{w}$. Since $X^\top X$ is symmetric (check this!), the gradient of TrainLoss with respect to $\mathbf{w}$ is (notice we reintroduce the $\frac{1}{n}$ term)

$$\frac{1}{n}(2X^\top X \mathbf{w} - 2X^\top \mathbf{y}). \tag{14}$$

Here, the $\mathbf{y}^\top \mathbf{y}$ term has a gradient of zero with respect to $\mathbf{w}$. Setting (14) equal to 0 gives us the famous *normal equations*:

$$X^\top X \mathbf{w} = X^\top \mathbf{y}. \tag{15}$$

From here, there are technically two possibilities. If $X^\top X$ is invertible (which it usually is), then we have the unique minimizer

$$\mathbf{w}^\star = (X^\top X)^{-1} X^\top \mathbf{y}, \tag{16}$$

and we are done! If $X^\top X$ is not invertible, then things are a bit trickier. We can still solve for a minimizer of the training loss, but this minimizer will not be unique.

2) [CA session] Problem 2: Non-linear features

Consider the following two training datasets of (x, y) pairs:

- $\mathcal{D}_1 = \{(-1, +1), (0, -1), (1, +1)\}$.
- $\mathcal{D}_2 = \{(-1, -1), (0, +1), (1, -1)\}$.

Observe that neither dataset is linearly separable if we use $\phi(x) = x$, so let's fix that.

Define a two-dimensional feature function $\phi(x)$ such that:

- There exists a weight vector $\mathbf{w}_1$ that classifies $\mathcal{D}_1$ perfectly (meaning that $\mathbf{w}_1 \cdot \phi(x) > 0$ if x is labeled $+1$ and $\mathbf{w}_1 \cdot \phi(x) < 0$ if x is labeled -1); and
- There exists a weight vector $\mathbf{w}_2$ that classifies $\mathcal{D}_2$ perfectly.

Note that the weight vectors can be different for the two datasets, but the features $\phi(x)$ must be the same.

Solution One option is $\phi(x) = [1, x^2]$, and using $\mathbf{w}_1 = [-1, 2]$ and $\mathbf{w}_2 = [1, -2]$.

Then in $\mathcal{D}_1$:

- For $x = -1$, $\mathbf{w}_1 \cdot \phi(x) = [-1, 2] \cdot [1, 1] = 1 > 0$
- For $x = 0$, $\mathbf{w}_1 \cdot \phi(x) = [-1, 2] \cdot [1, 0] = -1 < 0$
- For $x = 1$, $\mathbf{w}_1 \cdot \phi(x) = [-1, 2] \cdot [1, 1] = 1 > 0$

In $\mathcal{D}_2$:

- For $x = -1$, $\mathbf{w}_2 \cdot \phi(x) = [1, -2] \cdot [1, 1] = -1 < 0$
- For $x = 0$, $\mathbf{w}_2 \cdot \phi(x) = [1, -2] \cdot [1, 0] = 1 > 0$
- For $x = 1$, $\mathbf{w}_2 \cdot \phi(x) = [1, -2] \cdot [1, 1] = -1 < 0$

Note that there are many options that work, so long as -1 and 1 are separated from 0.

Some additional food for thought: Is every dataset linearly separable in some feature space? In other words, given pairs $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, can we find a feature extractor ϕ such that we can perfectly classify $(\phi(\mathbf{x}_1), y_1), \ldots, (\phi(x_n), y_n)$ for some linear model $\mathbf{w}$? If so, is this a good feature extractor to use?

Solution In theory, yes we can. If we assume that our inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are distinct, then we can construct a feature map $\phi : x_i \mapsto y_i$ for $i = 1, \ldots, n$. By setting $\mathbf{w}^\star = [1]$, it's clear that

$$y_i \mathbf{w}^\star \cdot \phi(x_i) = y_i * y_i = 1 > 0, \quad i = 1, \ldots, n, \tag{17}$$

so $\mathbf{w}^\star$ correctly classifies all the points in the dataset.

Hopefully, it's clear that this is a poor choice of feature map. For one, this feature extractor is undefined for any points outside of the training set! But even more broadly, this process is not at all *generalizeable*. We are essentially just memorizing our dataset instead of learning patterns and structures within the data that will allow us to accurately predict new points in the future. While minimizing training loss is an important part of the machine learning process (the aforementioned procedure gives you zero training loss!), it does not guarantee you good performance in the future.

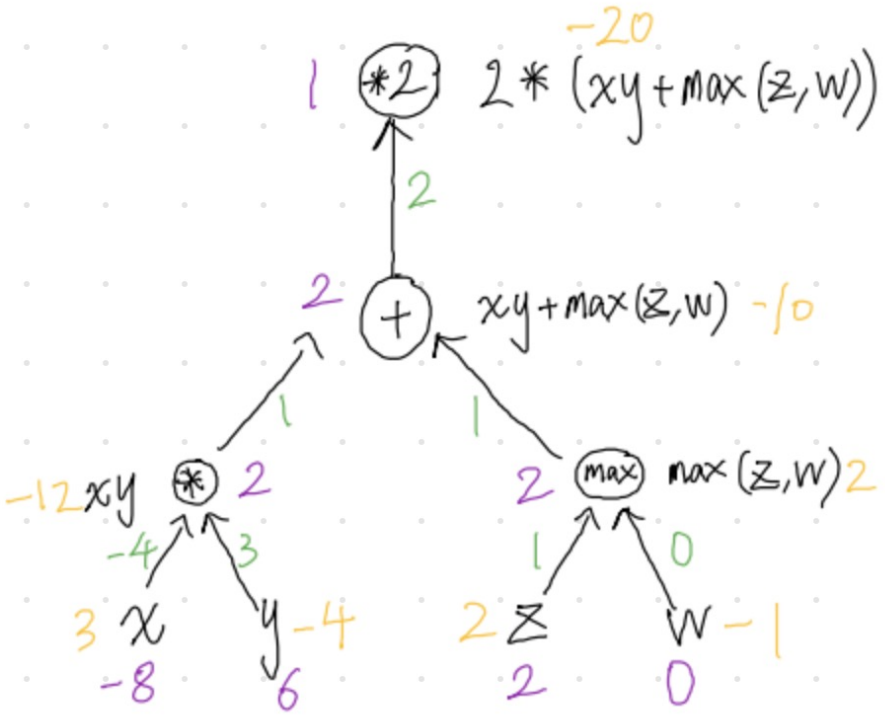
3) [CA session] Problem 3: Backpropagation

Consider the following function

$$\text{Loss}(x, y, z, w) = 2(xy + \max\{w, z\})$$

Run the backpropagation algorithm to compute the four gradients (each with respect to one of the individual variables) at $x = 3$, $y = -4$, $z = 2$ and $w = -1$. Use the following nodes: addition, multiplication, max, multiplication by a constant.

Solution When calculating the gradients, we run backpropagation from the root node to the leaves nodes. As shown on the computation graph below, the purple values are the gradients of Loss with respect to each node.



4) **[breakout, optional] Problem 4: Non-linear decision boundaries**

Suppose we are performing classification where the input points are of the form $(x_1, x_2) \in \mathbb{R}^2$. We can choose any subset of the following set of features:

$$\mathcal{F} = \left\{ x_1^2, x_2^2, x_1 x_2, x_1, x_2, \frac{1}{x_1}, \frac{1}{x_2}, 1, \mathbf{1}[x_1 \geq 0], \mathbf{1}[x_2 \geq 0] \right\} \tag{18}$$

For each subset of features $F \subseteq \mathcal{F}$, let $D(F)$ be the set of all decision boundaries corresponding to linear classifiers that use features F.

For each of the following sets of decision boundaries E, provide the minimal F such that $D(F) \supseteq E$. If no such F exists, write 'none'.

- E is all lines [CA hint]:

  _____ (19)

- E is all circles centered at the origin:

  _____ (20)

- E is all circles:

  _____ (21)

- E is all axis-aligned rectangles:

  _____ (22)

- E is all axis-aligned rectangles whose lower-right corner is at $(0, 0)$:

  _____ (23)

Solution

- Lines: $x_1, x_2, 1$ $(ax_1 + bx_2 + c = 0)$
- Circles centered at the origin: $x_1^2, x_2^2, 1$ $(x_1^2 + x_2^2 = r^2)$
- Circles centered anywhere in the plane: $x_1^2, x_2^2, x_1, x_2, 1$ $((x_1 - a)^2 + (x_2 - b)^2 = r^2)$
- Axis aligned rectangles: not possible (need features of the form $\mathbf{1}[x_1 \leq a]$)
- Axis aligned rectangles with lower right corner at (0, 0): not possible

5) [breakout, optional] Problem 5: K-means

Consider doing ordinary K-means clustering with $K = 2$ clusters on the following set of 3 one-dimensional points:

$$\{-2, 0, 10\}. \tag{24}$$

Recall that K-means can get stuck in local optima. Describe the precise conditions on the initialization $\mu_1 \in \mathbb{R}$ and $\mu_2 \in \mathbb{R}$ such that running K-means will yield the global optimum of the objective function. Notes:

- Assume that $\mu_1 < \mu_2$.
- Assume that if in step 1 of K-means, no points are assigned to some cluster j, then in step 2, that centroid μ_j is set to ∞.
- Hint: try running K-means from various initializations μ_1, μ_2 to get some intuition; for example, if we initialize $\mu_1 = 1$ and $\mu_2 = 9$, then we converge to $\mu_1 = -1$ and $\mu_2 = 10$.

Solution The objective is minimized for $\mu_1 = -1$ and $\mu_2 = 10$. First, note that if all three points end up in one cluster, K-means definitely fails to recover the global optimum. Therefore, -2 must be assigned to the first cluster, and 10 must be assigned to the second cluster. 0 can be assigned to either: If 0 is assigned to cluster 1, then we're done. If it is assigned to cluster 2, then we have $\mu_1 = -2, \mu_2 = 5$; in the next iteration, 0 will be assigned to cluster 1 since its closer. Therefore, the condition on the initialization written formally is $|-2-\mu_1| < |-2-\mu_2|$ and $|10-\mu_1| > |10-\mu_2|$.