CS221 Problem Workout Solutions

Week 2

1) Problem 1: 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

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

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.

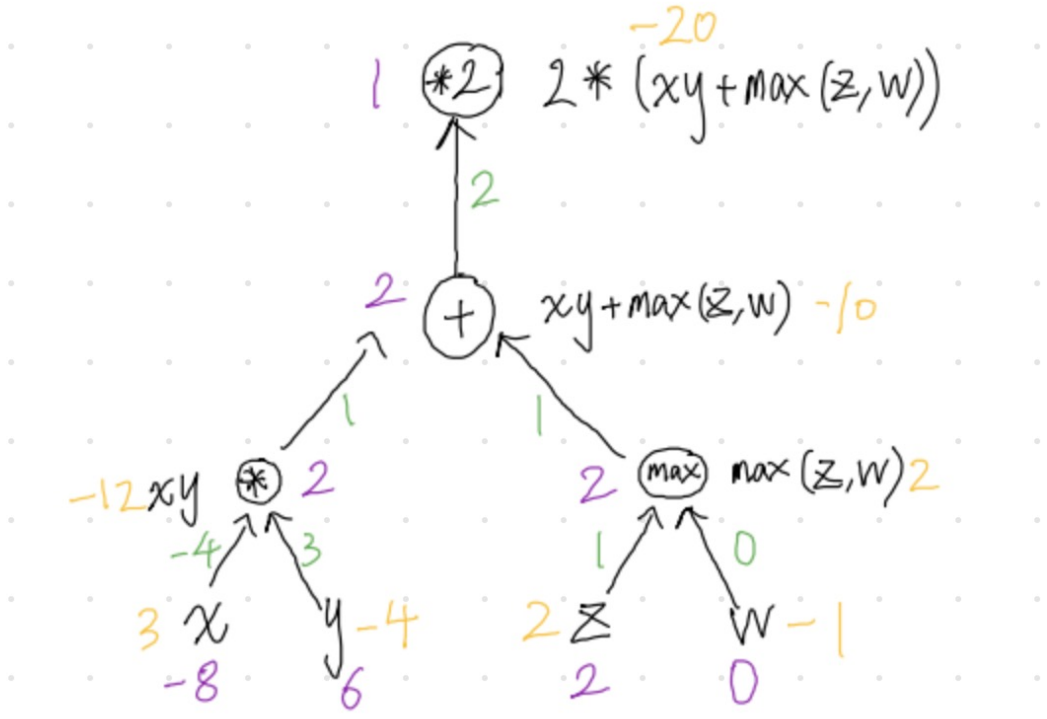
2) Problem 2: Backpropagation

Consider the following function

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

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.



3) **Problem 3: K-means**

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

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

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 $|10-\mu_1| > |10-\mu_2|$.

4) [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:

$$\mathcal{F} = \left\{ x_1^2, x_2^2, x_1x_2, x_1, x_2, \frac{1}{x_1}, \frac{1}{x_2}, 1, \mathbf{1}[x_1 \geq 0], \mathbf{1}[x_2 \geq 0] \right\} \tag{3}$$

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]:

  ______________________________ (4)

- E is all circles centered at the origin:

  ______________________________ (5)

- E is all circles:

  ______________________________ (6)

- E is all axis-aligned rectangles:

  ______________________________ (7)

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

  ______________________________ (8)

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

1. Movie Genre(s) (*20 points*)

You are interested in classifying the genre(s) of a movie given its plot summary. For simplicity, assume there are only three genres in the universe: action, romance, and comedy. You decide that you will model this as three binary classification problems, one for each genre.

For each plot summary x, you represent the corresponding movie M's genre(s) with $y \in \{0,1\}^3$, where

$$\begin{aligned} y_1 &= \mathbf{1}[M \text{ is an action movie}], \\ y_2 &= \mathbf{1}[M \text{ is a romance movie}], \\ y_3 &= \mathbf{1}[M \text{ is a comedy movie}]. \end{aligned}$$

a. (*12 points*) **Sharing is Caring**

You decide to use a feature extractor ϕ that maps each word in the English vocabulary to the number of occurrences of that word in the plot summary. Assume our English vocabulary D contains only the top 10,000 words in English (by frequency). Any out-of-vocabulary word is ignored.

Recall that the logistic function $\sigma(z) = (1 + e^{-z})^{-1}$ takes in a real number and outputs a probability in $(0, 1)$. After some research, you learned that the logistic function is often used with the *logistic loss* function for binary classification. Given a label $y \in \{0, 1\}$ and predicted probability $p \in [0, 1]$,

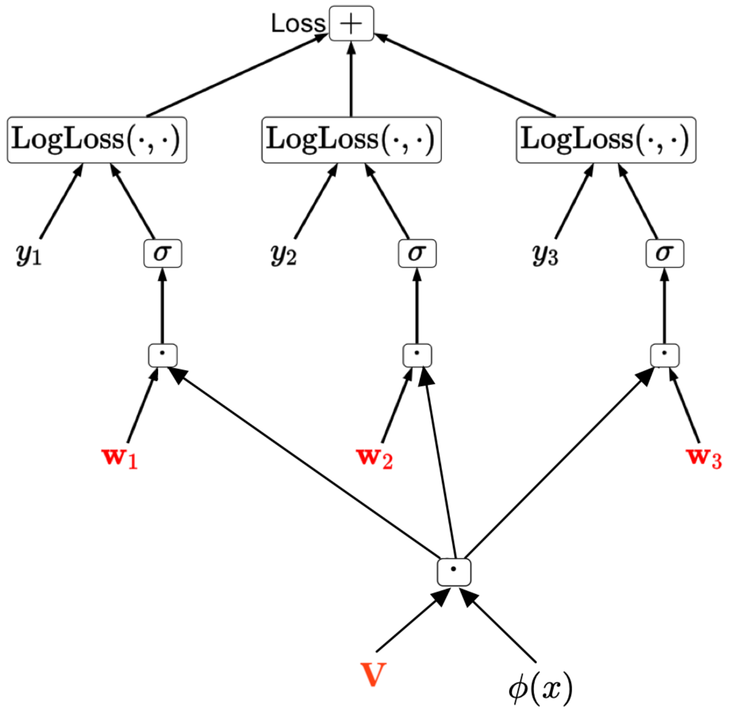
$$\text{LogLoss}(y, p) = -\big(y \log p + (1 - y) \log(1 - p)\big).$$

You decide to use these shiny new tools to solve your problem.

(i) [1 point] One way to solve your problem is to build three binary classifiers. For a plot summary x and genre $k \in \{1, 2, 3\}$, the predicted probability that $y_k = 1$ is $p_k = \sigma(\mathbf{w}_k \cdot \phi(x))$. How many parameters will there be in the three classifiers combined? No justification required.

Solution $30,000$. Each classifier has $10,000$ parameters, so there are $3 \times 10,000 = 30,000$ parameters.

(ii) [6 points] You've heard that having too many parameters can lead to bad generalization and want to find a way to reduce it. Since the three binary classification tasks are all about predicting movie genre and are thus related in nature, why not let the three classifiers share parameters? You come up with the following computation graph:



where $\mathbf{V}$ is a trainable weight matrix shared among all three classifiers such that $\mathbf{V}\phi(x) \in \mathbb{R}^2$ for all x.

Now, for any $k \in \{1, 2, 3\}$, the predicted probability that $y_k = 1$ is $p_k = \sigma(\mathbf{w}_k \cdot \mathbf{V}\phi(x))$.

- In this setup, how many parameters are there in the three classifiers combined? No justification required.

Solution 20,006. Since $\mathbf{V}\phi(x) \in \mathbb{R}^2$, $\mathbf{V}$ must be a $2 \times |D|$ matrix, which has $2 \times 10,000$ parameters. For the dimensionality to match, w_1, w_2, w_3 are all 2-dimensional vectors. So there are 20,006 parameters in total.

- The pointwise loss function $\text{Loss}(x, y, \mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3)$ is defined as follows:

$$\begin{aligned}\text{Loss}(x, y, \mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3) &= \sum_{k=1}^{3} \text{LogLoss}(y_k, p_k) \\ &= \sum_{k=1}^{3} -\Big(y_k \log \sigma(\mathbf{w}_k \cdot \mathbf{V}\phi(x)) + (1 - y_k) \log\big(1 - \sigma(\mathbf{w}_k \cdot \mathbf{V}\phi(x))\big)\Big).\end{aligned}$$

Derive the gradient of the pointwise loss with respect to $\mathbf{w}_1$, i.e., $\nabla_{\mathbf{w}_1}\text{Loss}(x, y, \mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3)$. You can use p_1 in your final expression. Show your work.
Hint: Feel free to use $\sigma'(z) = \sigma(z)\big(1 - \sigma(z)\big)$ directly without showing the intermediate steps.

Solution

$$\nabla_{\mathbf{w}_1}\text{Loss}(x, y, \mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3) = (p_1 - y_1)\mathbf{V}\phi(x)$$

Derivation:

$$\begin{aligned}
&\nabla_{\mathbf{w_1}}\text{Loss}(x, y, \mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3) && (1)\\
=&\nabla_{\mathbf{w_1}}\ \text{LogLoss}(y_1, p_1) \\
=&\nabla_{\mathbf{w_1}} - \Big(y_1 \log \sigma(\mathbf{w}_1 \cdot \mathbf{V}\phi(x)) + (1 - y_1) \log\big(1 - \sigma(\mathbf{w}_1 \cdot \mathbf{V}\phi(x))\big)\Big) && (2)\\
=&-\Big(y_1 \frac{1}{p_1}\sigma'\big(\mathbf{w}_1 \cdot \mathbf{V}\phi(x)\big)\nabla_{\mathbf{w_1}}(\mathbf{w}_1 \cdot \mathbf{V}\phi(x)) \\
&+ (1 - y_1)\frac{1}{1 - p_1}\Big(-\sigma'\big(\mathbf{w}_1 \cdot \mathbf{V}\phi(x)\big)\Big)\nabla_{\mathbf{w_1}}(\mathbf{w}_1 \cdot \mathbf{V}\phi(x))\Big) && (3)\\
=&-\Big(y_1 \frac{p_1(1 - p_1)}{p_1}\nabla_{\mathbf{w_1}}(\mathbf{w}_1 \cdot \mathbf{V}\phi(x)) - (1 - y_1)\frac{p_1(1 - p_1)}{1 - p_1}\nabla_{\mathbf{w_1}}(\mathbf{w}_1 \cdot \mathbf{V}\phi(x))\Big) && (4)\\
=&-\big(y_1(1 - p_1)\mathbf{V}\phi(x) - (1 - y_1)p_1\mathbf{V}\phi(x)\big) && (5)\\
=&-\mathbf{V}\phi(x)(y_1 - p_1) = (p_1 - y_1)\mathbf{V}\phi(x) && (6)
\end{aligned}$$

(iii) [5 points] You plan to use regular gradient descent to train your classifiers. Suppose you have a dataset of N points $(x^{(i)}, y^{(i)})$. You define the overall training loss as follows:

$$\text{TrainLoss}(\mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3) = \frac{1}{N}\sum_{i=1}^{N} \text{Loss}(x^{(i)}, y^{(i)}, \mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3).$$

Below is your gradient descent algorithm:

Algorithm 1: Gradient Descent

```
1: Randomly shuffle the training data
2: Initialize $\mathbf{w}_1 = \mathbf{w}_2 = \mathbf{w}_3 = \mathbf{0}$, initialize $\mathbf{V}$ with random non-zero matrix
3: for $t = 1, 2, \ldots, T$ do
4:     // calculate gradients
5:     for $k = 1, 2, 3$ do
6:         $\mathbf{v}_k \leftarrow \nabla_{\mathbf{w}_k}\text{TrainLoss}(\mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3)$ // get gradient w.r.t $\mathbf{w}_k$
7:     end for
8:     $\mathbf{U} \leftarrow \nabla_{\mathbf{V}}\text{TrainLoss}(\mathbf{V}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3)$ // get gradient w.r.t $\mathbf{V}$
9:
10:    // update weights
11:    for $k = 1, 2, 3$ do
12:        $\mathbf{w}_k \leftarrow \mathbf{w}_k - \eta\mathbf{v}_k$ // update $\mathbf{w}_k$
13:    end for
14:    $\mathbf{V} \leftarrow \mathbf{V} - \eta\mathbf{U}$ // update $\mathbf{V}$
15: end for
```

Consider a training dataset $\mathcal{D}$ with two datapoints $(x^{(1)}, y^{(1)})$ and $(x^{(2)}, y^{(2)})$. You run gradient descent on $\mathcal{D}$ and initialize $\mathbf{w}_1 = \mathbf{w}_2 = \mathbf{w}_3 = \mathbf{0}$ per Line 2 in the algorithm described above. Upon initializing $\mathbf{V}$ randomly, we see that

$$\begin{aligned} \mathbf{V}\phi(x^{(1)}) &= [5, -2]^T, \quad y^{(1)} = [0, 1, 1]^T \\ \text{and } \mathbf{V}\phi(x^{(2)}) &= [-1, 2]^T, \quad y^{(2)} = [1, 0, 1]^T. \end{aligned}$$

You run gradient descent on this dataset with $\eta = 0.1$. What is the value of $\mathbf{w}_1$ at the end of one iteration? Show your work.

Note: We are expecting the value of $\mathbf{w}_1$ only, **not** $\mathbf{w}_2$ or $\mathbf{w}_3$.

Solution For all k,

$$\begin{aligned} \mathbf{v}_k = \nabla_{\mathbf{w}_k}\text{TrainLoss}(\mathbf{V}_0, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3) &= \frac{1}{2}\sum_{i=1}^{2} \nabla_{\mathbf{w}_k} Loss(x^{(i)}, y^{(i)}, \mathbf{V}_0, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3) \\ &= \frac{1}{2}\big((p_k^{(1)} - y_k^{(1)})\mathbf{V}_0\phi(x^{(1)}) + (p_k^{(2)} - y_k^{(2)})\mathbf{V}_0\phi(x^{(2)})\big). \end{aligned}$$

Note that for all k, since $\mathbf{w}_k = \mathbf{0}$, $p_k^{(1)} = p_k^{(2)} = \sigma(\mathbf{0} \cdot \mathbf{V}_0\phi(x)) = \sigma(0) = (1+e^0)^{-1} = 0.5$. And since $\eta = 0.1$, after the first iteration, $\mathbf{w}_k = \mathbf{0} - \eta\mathbf{v}_k = -0.1\mathbf{v}_k$.

We see that

$$\mathbf{v}_1 = \frac{1}{2}\big((0.5 - 0)[5, -2]^T + (0.5 - 1)[-1, 2]^T\big) = [1.5, -1]^T.$$

Therefore,

$$\mathbf{w}_1 = [-0.15, 0.1]^T.$$

Misses that the teaching staff saw include:

- Not averaging the gradient update across the dataset (i.e., dropping the $\frac{1}{N}$ term)
- Calculating gradient but forgetting to update
- Mixing up the subscript used in $\mathbf{w}_1$ with the superscripts used in the datapoints

b. (*8 points*) **Less is More**

You tried running gradient descent, but since $\mathbf{V}$ is a large matrix, your computer does not have enough memory to store the gradient with respect to $\mathbf{V}$, so you couldn't train your model properly.

Luckily, your friend Alice has worked on a similar problem (e.g., classifying fiction genres given synopsis) and trained a model with the same architecture as yours. You ask Alice to share with you the weights of her *trained* classifier, which includes a weight matrix $\mathbf{V}_0$ that has exactly the same shape as $\mathbf{V}$.

Since Alice's problem is similar to yours, you believe the solution should be similar as well. Hence, you decide to initialize $\mathbf{V}$ with $\mathbf{V}_0$ and never update it in order to get around the memory constraint. In other words, **you run gradient descent only on $\mathbf{w}_1$, $\mathbf{w}_2$, and $\mathbf{w}_3$.**

You tell Alice about your new gradient descent algorithm, and the idea to make a new memory-efficient classifier that shares $\mathbf{V}_0$.

Alice responds: "Actually, what you have here isn't a new kind of classifier. It's actually three classifiers with a shared feature extractor." Though shocked at first, after some thought, you agree with her as well.

(i) [2 points] Show that Alice is right by filling in the blank below:

> Given a plot summary x, for each genre $k \in \{1, 2, 3\}$,
> the predicted probability that $y_k = 1$ is $p_k = \sigma(\mathbf{w}_k \cdot \tau(x))$,
> where $\tau(x) =$ ____________ is the shared feature extractor.

No justification required.

Solution $\tau(x) = \mathbf{V}_0\phi(x)$.

(ii) [1 point] In this setup, how many parameters are there in the three classifiers combined? No justification required.

Solution 6. The only parameters are $\mathbf{w}_1$, $\mathbf{w}_2$, and $\mathbf{w}_3$, which are all 2-dimensional.

(iii) [5 points] Between the following:

(A) three classifiers that share $\phi(x)$ as feature extractor (i.e., your idea in **a**(i))

(B) three classifiers that share $\tau(x)$ as feature extractor

Which setup has higher approximation error? Which one has higher estimation error? Justify your answer with 1-2 sentences.

Solution (B) has higher approximation error. (A) has higher estimation error. This is because (A) has a much larger hypothesis class. Each of the three weight vectors in (A) has much higher dimensionality than each of the three weight vectors in (B). More specifically, as we saw in 1a(i), each weight vector in (A) is 10,000-dimensional; as we saw in 1b(ii), each weight vector in (B) is 2-dimensional.

A commonly seen incorrect justification relates estimation/approximation error to the accuracy/inaccuracy of having a pre-trained weight matrix rather than the number of parameters.