Benjamin Ye
CS/CNS/EE 156a: Learning Systems (Fall 2023)
November 6, 2023

Homework 6

Problem	Answer
1	[b]
2	[a]
3	[d]
4	[e]
5	[d]
6	[b]
7	[c]
8	[d]
9	[a]
10	[e]

Overfitting and Deterministic Noise

1. Deterministic noise depends on $\mathcal{H}$, as some models approximate f better than others. Assume that $\mathcal{H}' \subset \mathcal{H}$ and that f is fixed. In general (but not necessarily in all cases), if we use $\mathcal{H}'$ instead of $\mathcal{H}$, how does deterministic noise behave?

   Answer: [b] In general, deterministic noise will increase.

   In general, there are fewer hypotheses in $\mathcal{H}'$ than $\mathcal{H}$ because $\mathcal{H}'$ is a subset of $\mathcal{H}$. As such, we expect that $\mathcal{H}'$ will model less of the target function f correctly in the input space $\mathcal{X}$. Therefore, the deterministic noise (or the bias) should increase since it is higher for smaller models.

Regularization with Weight Decay

In the following problems, use the data provided in the files

http://work.caltech.edu/data/in.dta

http://work.caltech.edu/data/out.dta

as a training and test set, respectively. Each line of the files corresponds to a two-dimensional input $\mathbf{x} = (x_1, x_2)$, so that $\mathcal{X} = \mathbb{R}^2$, followed by the corresponding label from $\mathcal{Y} = \{-1, 1\}$. We are going to apply linear regression with a non-linear transformation for classification. The nonlinear transformation is given by

$$\Phi(x_1, x_2) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2, |x_1 - x_2|, |x_1 + x_2|)$$

Recall that the classification error is defined as the fraction of misclassified points.

2. Run linear regression on the training set after performing the non-linear transformation. What values are closest (in Euclidean distance) to the in-sample and out-of-sample classification errors, respectively?

   Answer: [a] 0.03, 0.08

3. Now add weight decay to linear regression, that is, add the term $(\lambda/N)\sum_{i=0}^{7} w_i^2$ to the squared in-sample error, using $\lambda = 10^k$. What are the closest values to the in-sample and out-of-sample classification errors, respectively, for $k = -3$? Recall that the solution for linear regression with weight decay was derived in class.

   Answer: [d] 0.03, 0.08

4. Now, use $k = 3$. What are the closest values to the new in-sample and out-of-sample classification errors, respectively?

   Answer: [e] 0.4, 0.4

5. What value of k, among the following choices, achieves the smallest out-of-sample classification error?

   Answer: [d] -1

6. What value is closest to the minimum out-of-sample classification error achieved by varying k (limiting k to integer values)?

   Answer: [b] 0.06

(The sample program output, discussion, and Python 3 source code are on the following pages. ⟶)

Sample program output

```
[Homework 6 Problem 2]
For the linear regression model without regularization, the in-sample and out-of-
sample errors are 0.02857 and 0.08400, respectively.

[Homework 6 Problems 3–6]
   k  in-sample error  out-of-sample error
-5.0         0.028571                0.084
-4.0         0.028571                0.084
-3.0         0.028571                0.080
-2.0         0.028571                0.084
-1.0         0.028571                0.056
 0.0         0.000000                0.092
 1.0         0.057143                0.124
 2.0         0.200000                0.228
 3.0         0.371429                0.436
 4.0         0.428571                0.452
 5.0         0.428571                0.456
 6.0         0.428571                0.456
```

For Problem 6, the out-of-sample classification error stops changing for $k \leq -5$ and $k \geq 6$.

For $k \ll -5$, we get overfitting as $\lambda \to 0$, while for $k \gg 6$, we get underfitting as $\lambda \to \infty$. The expected E_{out} for the weight decay regularization should have a minimum in the range $-5 \leq k \leq 6$, which is exactly what we see for $k = -1$.

Python 3 source code

```python
from pathlib import Path

import numpy as np
import pandas as pd
import requests

DATA_DIR = (Path(__file__).resolve().parents[2] / "data").resolve()

class LinearRegression:
    def __init__(
            self, *, vf=None, regularization=None, transform=None, noise=None,
            rng=None, seed=None, **kwargs):
        self.rng = np.random.default_rng(seed) if rng is None else rng
        self.set_parameters(vf=vf, regularization=regularization,
                            transform=transform, noise=noise, **kwargs)

    def get_error(self, x, y):
        if self.transform:
            x = self.transform(x)
        if self.noise:
            N = x.shape[0]
            index = self.rng.choice(N, round(self.noise[0] * N), False)
            y[index] = self.noise[1](y[index])
        if self.vf is not None and self.w is not None:
            return self.vf(self.w, x, y)
    def set_parameters(
            self, *, vf=None, regularization=None, transform=None, noise=None,
            update=False, **kwargs):
        self._reg_params = {}
        self.w = None
        if update:
            self.noise = noise or self.noise
            self.regularization = regularization or self.regularization
            if self.regularization == "weight_decay" \
                    and "weight_decay_lambda" in kwargs:
                self._reg_params["lambda"] = kwargs["weight_decay_lambda"]
            self.transform = transform or self.transform
            self.vf = vf or self.vf
        else:
            self.noise = noise
            self.regularization = regularization
            if regularization == "weight_decay":
                self._reg_params["lambda"] = kwargs["weight_decay_lambda"]
            self.transform = transform
            self.vf = vf

    def train(self, x, y):
        if self.transform:
            x = self.transform(x)
        if self.noise:
            N = x.shape[0]
            index = self.rng.choice(N, round(self.noise[0] * N), False)
            y[index] = self.noise[1](y[index])
        if self.regularization is None:
            self.w = np.linalg.pinv(x) @ y
        elif self.regularization == "weight_decay":
            self.w = np.linalg.inv(
                x.T @ x
                + self._reg_params["lambda"] * np.eye(x.shape[1], dtype=float)
            ) @ x.T @ y
        if self.vf is not None:
            return self.vf(self.w, x, y)

def validate_binary(w, x, y):
    return np.count_nonzero(np.sign(x @ w) != y, axis=0) / x.shape[0]
```

```python
if __name__ == "__main__":
    rng = np.random.default_rng()

    DATA_DIR.mkdir(exist_ok=True)
    data = {"train": "in.dta", "test": "out.dta"}
    for dataset, file in data.items():
        if not (DATA_DIR / file).exists():
            r = requests.get(f"http://work.caltech.edu/data/{file}")
            with open(DATA_DIR / file, "wb") as f:
                f.write(r.content)
        data[dataset] = np.loadtxt(DATA_DIR / file)

    transform = lambda x: np.hstack((
        np.ones((len(x), 1), dtype=float),
        x,
        x[:, :1] ** 2,
        x[:, 1:] ** 2,
        np.prod(x, axis=1, keepdims=True),
        np.abs(x[:, :1] - x[:, 1:]),
        np.abs(x[:, :1] + x[:, 1:])
    ))
    reg = LinearRegression(vf=validate_binary, transform=transform, rng=rng)
    E_in = reg.train(data["train"][:, :-1], data["train"][:, -1])
    E_out = reg.get_error(data["test"][:, :-1], data["test"][:, -1])
    print("\n[Homework 6 Problem 2]\n",
          "For the linear regression model without regularization, the "
          f"in-sample and out-of-sample errors are {E_in:.5f} and "
          f"{E_out:.5f}, respectively.", sep="")

    df = pd.DataFrame(columns=["k", "in-sample error", "out-of-sample error"])
    for k in np.arange(-5, 7):
        reg.set_parameters(regularization="weight_decay",
                           weight_decay_lambda=10.0 ** k, update=True)
        E_in = reg.train(data["train"][:, :-1], data["train"][:, -1])
        df.loc[len(df)] = (k, E_in, reg.get_error(data["test"][:, :-1],
                                                  data["test"][:, -1]))
    print(f"\n[Homework 6 Problems 3–6]\n{df.to_string(index=False)}")
```

Regularization for Polynomials

Polynomial models can be viewed as linear models in a space $\mathcal{Z}$, under a nonlinear transform $\Phi : \mathcal{X} \to \mathcal{Z}$. Here, Φ transforms the scalar x into a vector $\mathbf{z}$ of Legendre polynomials, $\mathbf{z} = \left(1, L_1(x), L_2(x), \ldots, L_Q(x)\right)$. Our hypothesis set will be expressed as a linear combination of these polynomials,

$$\mathcal{H}_Q = \left\{ h \mid h(x) = \mathbf{w}^\top \mathbf{z} = \sum_{q=0}^{Q} w_q L_q(x) \right\}$$

where $L_0(x) = 1$.

7. Consider the following hypothesis set defined by the constraint:

$$\mathcal{H}(Q, C, Q_o) = \left\{ h \mid h(x) = \mathbf{w}^\top \mathbf{z} \in \mathcal{H}_Q;\ w_q = C \text{ for } q \geq Q_o \right\}$$

Which of the following statements is correct?

Answer: [c] $\boldsymbol{\mathcal{H}(10, 0, 3) \cap \mathcal{H}(10, 0, 4) = \mathcal{H}_2}$

From the definition, the hypothesis set $\mathcal{H}_Q$ contains all Legendre polynomials of order Q and below. With the constraint, the set can lose higher-order polynomials if $C = 0$ and $Q_o < Q$.

For [a], $\mathcal{H}(10, 0, 3)$ has polynomials of order 2 and lower and is equivalent to $\mathcal{H}_2$, and $\mathcal{H}(10, 0, 4)$ has polynomials of order 3 and lower and is equivalent to $\mathcal{H}_3$. Therefore, their union is $\mathcal{H}(10, 0, 3) \cup \mathcal{H}(10, 0, 4) = \mathcal{H}_2 \cup \mathcal{H}_3 = \mathcal{H}_3$, not $\mathcal{H}_4$. This statement is false.

For [b], $\mathcal{H}(10, 1, 3)$ is $\mathcal{H}_2$ plus higher-order polynomials with powers $3 \leq q \leq 10$, while $\mathcal{H}(10, 1, 4)$ is $\mathcal{H}_3$ plus higher-order polynomials with powers $4 \leq q \leq 10$. Therefore, their union is some combination of $\mathcal{H}_3$ and higher-order terms, not just $\mathcal{H}_3$. This statement is false.

For [c], $\mathcal{H}(10, 0, 3)$ is $\mathcal{H}_2$ and $\mathcal{H}(10, 0, 4)$ is $\mathcal{H}_3$, as established earlier. Their intersection is simply $\mathcal{H}(10, 0, 3) \cap \mathcal{H}(10, 0, 4) = \mathcal{H}_2 \cap \mathcal{H}_3 = \mathcal{H}_2$. This statement is true (and the answer).

For [d], the intersection of $\mathcal{H}(10, 1, 3)$ and $\mathcal{H}(10, 1, 4)$ is $\mathcal{H}_2$ plus higher-order polynomials with powers $4 \leq q \leq 10$, not just $\mathcal{H}_1$. This statement is false.

Neural Networks

8. A fully connected neural network has $L = 2; d^{(0)} = 5, d^{(1)} = 3, d^{(2)} = 1$. If only products of the form $w_{ij}^{(l)} x_i^{(l-1)}$, $w_{ij}^{(l)} \delta_j^{(l)}$, and $x_i^{(l-1)} \delta_j^{(l)}$ count as operations (even for $x_0^{(l-1)} = 1$), without counting anything else, which of the following is the closest to the total number of operations in a single iteration of backpropagation (using SGD on one data point)?

   Answer: [d] 45

   The first step is the forward propagation algorithm, in which the inputs and outputs of a hidden or output layer l are related via

$$\mathbf{x}^{(l)} = \begin{bmatrix} 1 \\ \theta\left(\mathbf{s}^{(l)}\right) \end{bmatrix}$$

   where θ is a transformation function and $\mathbf{s}^{(l)}$ is a vector of incoming signals $s_i^{(l)}$ given by

$$s_j^{(l)} = \sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}$$

   By including the bias nodes, the number of times $w_{ij}^{(l)} x_i^{(l-1)}$ is evaluated across the L layers is

$$Q_{\text{fp}} = \sum_{l=1}^{L} d^{(l)}\left(d^{(l-1)} + 1\right) = \left(d^{(0)} + 1\right)d^{(1)} + \left(d^{(1)} + 1\right)d^{(2)} = 6 \times 3 + 4 \times 1 = 22$$

   Then, in the backpropagation algorithm, the SGD weights are updated using

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \eta \frac{\partial \mathrm{e}}{\partial w_{ij}^{(l)}} = w_{ij}^{(l)} - \eta x_i^{(l-1)} \delta_j^{(l)}$$

   where $\delta_j^{(l)}$ is the sensitivity of the error e with respect to the input signal $s_j^{(l)}$. As the gradient is evaluated with respect to each of the weights (of which there are 22 since each connection between nodes is a weight), there are $Q_{\text{SGD}} = 22$ multiplications of $x_i^{(l-1)} \delta_j^{(l)}$ to be performed.

   The sensitivity $\delta_j^{(l)}$ can be obtained by running a modified version of the neural network backwards, and is given by

$$\delta_j^{(l)} = \theta'\left(s_j^{(l)}\right) \sum_{k=1}^{d^{(l+1)}} w_{jk}^{(l+1)} \delta_k^{(l+1)}$$

With $\delta_j^{(L)} = 2\left(x_j^{(L)} - y\right)\theta'\left(s_j^{(L)}\right)$, which does not count as an operation for this problem, there are

$$Q_{\text{bp}} = \sum_{l=1}^{L-1} d^{(l)} d^{(l+1)} = d^{(1)} d^{(2)} = 3$$

$w_{ij}^{(l)} \delta_j^{(l)}$ operations to be carried out.

Putting everything together, we have a total of

$$Q = Q_{\text{fp}} + Q_{\text{SGD}} + Q_{\text{bp}} = 22 + 22 + 3 = 47$$

multiplicative operations in a single iteration of backpropagation.

Let us call every "node" in a neural network a unit, whether that unit is an input variable or a neuron in one of the layers. Consider a neural network that has 10 input units (the constant $x_0^{(0)}$ is counted here as a unit), one output unit, and 36 hidden units (each $x_0^{(l)}$ is also counted as a unit). The hidden units can be arranged in any number of layers $l = 1, \ldots, L - 1$, and each layer is fully connected to the layer above it.

9. What is the minimum possible number of weights that such a network can have?

Answer: [a] 46

To minimize the number of weights, the network should have as few connections as possible. As such, the neural network should have 18 hidden layers with two nodes (including the bias node) each.

The number of dimensions in each layer is

$$\mathbf{d} = [9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$$

where the input layer is $d^{(0)} = 9$ (first value) and the output layer is $d^{(L)} = 1$ (last value).

The total number of weights is

$$Q = \sum_{l=1}^{L} d^{(l)}\left(d^{(l-1)} + 1\right) = 1 \times 10 + 17 \times (1 \times 2) + 1 \times 2 = 46$$

10. What is the maximum possible number of weights that such a network can have?

Answer: [e] 510

To maximize the number of weights, the network should have as many connections as possible. Based on the logic in Problem 9, the neural network should have few hidden layers with many nodes.

For a single hidden layer with dimension $d^{(1)} = 35$ (or 36 nodes), the total number of weights is

$$Q^{(1)} = \sum_{l=1}^{L} d^{(l)}\left(d^{(l-1)} + 1\right) = d^{(1)}\left(d^{(0)} + 1\right) + d^{(L)}\left(d^{(1)} + 1\right) = 35 \times 10 + 1 \times 36 = 386$$

For a neural network with two hidden layers, the total number of weights is

$$\begin{aligned} Q^{(2)} = \sum_{l=1}^{L} d^{(l)}\left(d^{(l-1)} + 1\right) &= d^{(1)}\left(d^{(0)} + 1\right) + d^{(2)}\left(d^{(1)} + 1\right) + d^{(L)}\left(d^{(2)} + 1\right) \\ &= 10d^{(1)} + d^{(1)}d^{(2)} + 2d^{(2)} + 1 \end{aligned}$$

The only constraint is that $d^{(1)} + d^{(2)} + 2 = 36$ such that there are a total of 36 nodes. Substituting $d^{(2)} = 34 - d^{(1)}$ into Q,

$$Q^{(2)} = 10d^{(1)} + d^{(1)}\left(34 - d^{(1)}\right) + 2\left(34 - d^{(1)}\right) + 1 = -\left(d^{(1)}\right)^2 + 42d^{(1)} + 69$$

To maximize Q, we set its derivative with respect to $d^{(1)}$ to 0 and solve for $d^{(1)}$:

$$\frac{\partial Q^{(2)}}{\partial d^{(1)}} = -2d^{(1)} + 42 = 0 \longrightarrow d^{(1)} = 21$$

Therefore, the first hidden layer should have dimension $d^{(1)} = 21$ (or 22 nodes) and the second hidden layer should have dimension $d^{(2)} = 13$ (or 14 nodes). In this scenario, the total number of weights is

$$Q^{(2)} = -21^2 + 42 \times 21 + 69 = 510$$

As [e] 510 is the largest choice, it must be the answer. However, to be sure, we also check a neural network with three hidden layers:

$$d^{(1)} + d^{(2)} + d^{(3)} + 3 = 36 \longrightarrow d^{(3)} = 33 - d^{(1)} - d^{(2)}$$

$$\begin{aligned} Q^{(3)} = 10d^{(1)} + d^{(1)}d^{(2)} + d^{(2)} + d^{(2)}d^{(3)} + 2d^{(3)} + 1 \\ &= 10d^{(1)} + d^{(1)}d^{(2)} + d^{(2)} + d^{(2)}\left(33 - d^{(1)} - d^{(2)}\right) + 2\left(33 - d^{(1)} - d^{(2)}\right) + 1 \\ &= 8d^{(1)} + 32d^{(2)} - \left(d^{(2)}\right)^2 + 67 \end{aligned}$$

With the domains $1 \le d^{(1)} \le 31$ and $1 \le d^{(2)} \le 32 - d^{(1)}$, Wolfram Alpha reports $\max\left(Q^{(3)}\right) = 467$ for $d^{(1)} = 20$, $d^{(2)} = 12$, and $d^{(3)} = 1$. Indeed, the neural network with two hidden layers with 22 and 14 nodes, respectively, has the maximum number of weights.