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 , as some models approximate better than others. Assume that and that is fixed. In general (but not necessarily in all cases), if we use instead of , how does deterministic noise behave?

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

In general, there are fewer hypotheses in than because is a subset of . As such, we expect that will model less of the target function correctly in the input space . 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 , so that , followed by the corresponding label from . We are going to apply linear regression with a non-linear transformation for classification. The nonlinear transformation is given by

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

1. 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**

1. Now add weight decay to linear regression, that is, add the term to the squared in-sample error, using . What are the closest values to the in-sample and out-of-sample classification errors, respectively, for ? Recall that the solution for linear regression with weight decay was derived in class.

**Answer: [d] 0.03, 0.08**

1. Now, use . What are the closest values to the new in-sample and out-of-sample classification errors, respectively?

**Answer: [e] 0.4, 0.4**

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

**Answer: [d]**

1. What value is closest to the minimum out-of-sample classification error achieved by varying (limiting 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 and .

For , we get overfitting as , while for , we get underfitting as . The expected for the weight decay regularization should have a minimum in the range , which is exactly what we see for .

**Python 3 source code**

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]

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 , under a nonlinear transform . Here, transforms the scalar into a vector of Legendre polynomials, . Our hypothesis set will be expressed as a linear combination of these polynomials,

where .

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

Which of the following statements is correct?

**Answer: [c]**

From the definition, the hypothesis set contains all Legendre polynomials of order and below. With the constraint, the set can lose higher-order polynomials if and .

For [a], has polynomials of order 2 and lower and is equivalent to , and has polynomials of order 3 and lower and is equivalent to . Therefore, their union is , not . This statement is false.

For [b], is plus higher-order polynomials with powers , while is plus higher-order polynomials with powers . Therefore, their union is some combination of and higher-order terms, not just . This statement is false.

For [c], is and is , as established earlier. Their intersection is simply . This statement is true (and the answer).

For [d], the intersection of and is plus higher-order polynomials with powers , not just . This statement is false.

**Neural Networks**

1. A fully connected neural network has ; , , . If only products of the form , , and count as operations (even for ), 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 are related via

where is a transformation function and is a vector of incoming signals given by

By including the bias nodes, the number of times is evaluated across the layers is

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

where is the sensitivity of the error with respect to the input signal . 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 multiplications of to be performed.

The sensitivity can be obtained by running a modified version of the neural network backwards, and is given by

With , which does not count as an operation for this problem, there are

operations to be carried out.

Putting everything together, we have a total of

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 is counted here as a unit), one output unit, and 36 hidden units (each is also counted as a unit). The hidden units can be arranged in any number of layers , and each layer is fully connected to the layer above it.

1. 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

where the input layer is (first value) and the output layer is (last value).

The total number of weights is

1. 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 (or 36 nodes), the total number of weights is

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

The only constraint is that such that there are a total of 36 nodes. Substituting into ,

To maximize , we set its derivative with respect to to 0 and solve for :

Therefore, the first hidden layer should have dimension (or 22 nodes) and the second hidden layer should have dimension (or 14 nodes). In this scenario, the total number of weights is

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:

With the domains and , Wolfram Alpha reports for , , and . Indeed, the neural network with two hidden layers with 22 and 14 nodes, respectively, has the maximum number of weights.