COM S 474/574 Spring 2020
Introduction to Machine Learning

Homework 5

1 Directions:

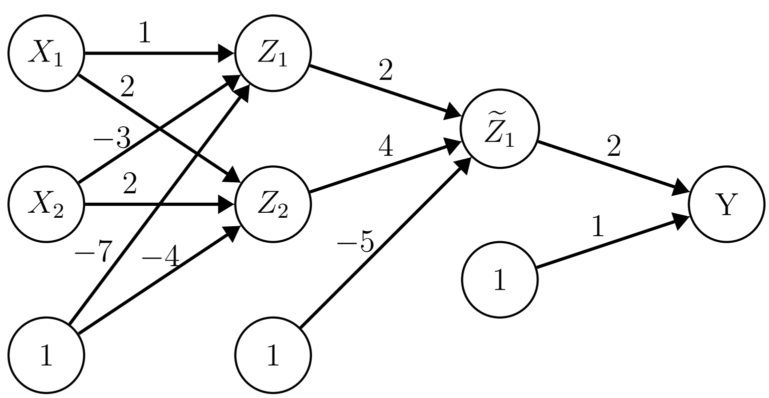
- **Due: Thursday April 16, 2020 at 10pm.** Late submissions will be accepted for 24 hours after that time, with a 15% penalty.
- Upload the homework to Canvas as a pdf file. Answers to problems 1-3 can be handwritten, but writing must be neat and the scan should be high-quality image. Other responses should be typed or computer generated.
- Any non-administrative questions must be asked in office hours or (if a brief response is sufficient) Piazza.

2 Problems

Problem 1. [10 points] Suppose we have trained the neural network displayed below. Each of the hidden nodes uses the ReLU function

$$g(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

as its activation function. For the input $\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \end{bmatrix}$, what would the prediction $\widehat{Y}$ be? Show your calculations.



Problem 2. [10 points] How many edges (and thus coefficients) are there in a network with three input nodes for features X_1, X_2, and X_3, one output node for Y, and the hidden layers described below? Assume there are no bias nodes (nodes labeled "1" that send a constant value to the next layer).

(a). one layer of 12 nodes

(b). 2 layers of 6 nodes each

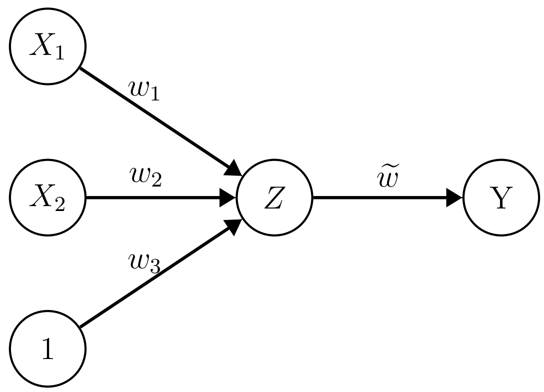
(c). 6 layers of 2 nodes each

(d). 12 layers of one node each

Problem 3. [30 points] Consider the network shown below. The input features X_1 and X_2 are both binary, taking values in $\{0, 1\}$. The single hidden node Z uses the step function

$$g(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

as its activation function.



For each of the data-sets below,

- specify a set of coefficient values that would result in the neural network achieving 100% accuracy.
- make a scatter-plot of the data, writing the Y value next to the data point, and draw the decision boundary (corresponding to the coefficients you specified), labeling which side is predicted as 0 and which side is predicted as 1.
- if there is no set of coefficients that can achieve 100% accuracy, use the scatter-plot to explain why it is impossible.

A.

X_1	X_2	Y
0	0	0
1	0	1
0	1	1
1	1	1

B.

X_1	X_2	Y
0	0	1
1	0	0
0	1	1
1	1	0

C.

X_1	X_2	Y
0	0	0
1	0	1
0	1	1
1	1	0

Remark: For this homework, we will use Scikit-learn's implementation of neural networks. Scikit-learn's implementation should be fine for small scale projects, though for large scale projects you will want to use other libraries. One popular choice is Pytorch `https://pytorch.org/tutorials/` or Keras `https://keras.io/` with Tensorflow `https://www.tensorflow.org/tutorials`.

We will examine the performance of different networks. If you are using Python, you can call

```
from sklearn.neural_network import MLPClassifier
```

and then

```
clf=MLPClassifier(...)
```

using the following arguments

- `hidden_layer_sizes=(5,2)` // this example input would create two hidden layers with 5 and 2 nodes nodes respectively.
- `activation='relu'` // this is the non-linear function each hidden node applies to its input. We will use rectified-linear, though you can experiment with how others perform.
- `solver='sgd'` // this data set is small enough we do not need to worry about using mini-batches or avoiding second order methods (eg estimate second derivatives). However, we will still set up the classifier to do mini-batch gradient descent

  With 'sgd' selected, it automatically uses momentum (by default a version known as `nesterovs_momentum=True` with `momentum=0.9`) this gives the updates "inertia" like a ball rolling down a hill
- `learning_rate='adaptive'` // it will decrease automatically when it senses accuracy does not change much.
- `alpha=0.001` // This is parameter for regularization to penalize large coefficients: $\min\ Loss + \alpha \sum |w|^2$
- `learning_rate_init=0.001` this is the η parameter that controls how big step sizes are
- `max_iter=200`
- Unfortunately, dropout is not implemented in sklearn

Remark: For some settings, the training process may not converge within `max_iter` iterations. You may get convergence warnings, and that is ok. You can suppress warnings with

```
import warnings
from sklearn.exceptions import ConvergenceWarning
warnings.simplefilter("ignore", ConvergenceWarning)
```

Problem 4. [30 points] This problem uses the same data sets as homework 3. You should be able to re-use code from homework 3 with minor modifications.

A. Make plots of testing and training accuracies as a function of the number of nodes in the hidden layer (use a single hidden layer), with the number of hidden nodes ranging from 1 to 10. For each number of hidden nodes, use the best `alpha` and `learn_rate_init` input parameters from the sets
   ```
   alpharange = np.logspace(-6,0,5)
   learnrateinitrange = np.logspace(-3,-1,3)
   ```
   Report the best number of hidden nodes and its testing accuracy.

B. For the best number of hidden nodes you found above, make a scatter plot of the training data like in homework 3, with the neural network classification function plotted using colors. Use the best input parameters `alpha` and `learn_rate_init` you found previously.

C. In about 2-5 sentences, comment on how the decision boundaries and accuracies are similar or different than the methods used in HW 3 and HW 4.

D. Make a 3d plot of the training accuracy of a small neural network as a function of two coefficients. A template for generating the 3d image will be available on canvas. After training a neural network, you can modify the coefficients using
   ```
   clf.coefs_[i][j][k] = ...
   ```
   which is the coefficient of the edge from the jth node in the ith layer to the kth node in the $(i+1)$th layer. For three different pairs of edges, make 3d pictures and examine them. For each pair, report which edges they correspond to and what the training accuracy landscape looks like. For instance,

   - are there areas of steep changes?
   - are there flat areas?
   - are there local maxima where the solver could potentially get stuck at?

   You do not need to submit 2d snapshots.

Problem 5. [20 points] This problem uses the same digits data set as homework 4. You should be able to re-use code from homework 4, with only a few modifications.

Standardize the data. In Python, you can use

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

Fit the scaler to the Xtrain data, then transform both the Xtrain and Xtest data:

```
scaler.fit(Xtrain)
Xtrain = scaler.transform(Xtrain)
Xtest = scaler.transform(Xtest)
```

A. Make plots of testing and training accuracies as a function of the number of nodes in the hidden layer (use a single hidden layer), with the number of hidden nodes ranging

```
noderange = range(5,100,5)
```

For each number of hidden nodes, use the best input parameters from the sets

```
alpharange = np.logspace(-6,0,4)
learnrate = np.logspace(-2,-0.5,4)
```

Report the best number of hidden nodes and its testing accuracy.
Report how this compares to results from HW 4.

B. Make plots of the training and testing accuracy as a function of the number of iterations, from iteration 1 to 50, for that best number of hidden nodes. Use the corresponding `learn_rate_init`, but set `alpha=0.0`. Also, instead of calling the `clf.fit(...)` function, use

```
clf.partial_fit(Xtrain, ytrain,np.unique(ytrain))
```

within a `for` loop. That function will update the coefficients each time it is called.
Report at around which iteration does the training accuracy curve seem to flatten out.
At what iteration does the testing accuracy seem to flatten out?