Lab week 8: MDPs

Name: Kassi Winter

Micro-Blackjack

In micro-blackjack, you repeatedly draw a card (with replacement) that is equally likely to be a 2, 3, or 4. You can either Draw or Stop if the total score of the cards you have drawn is less than 6. If your total score is 6 or higher, the game ends, and you receive a utility of 0. When you Stop, your utility is equal to your total score (up to 5), and the game ends. When you Draw, you receive no utility. There is no discount (= 1). Let's formulate this problem as an MDP with the following states: 0; 2; 3; 4; 5 and a Done state, for when the game ends.

1. What is the transition function and the reward function for this MDP?

   T(s, Draw, s') =
   → s is 2 and s' is Done OR s - s' is either {2, 3,4} = 1/3
   → s is 3 and s' is Done = 2/3
   → s is either {4, 5} and s' is Done = 1
   T(s, Stop, Done) = 1

   R(s, Stop, Done) = s for s <= 5
   R(s, a, s') = 0 otherwise

2. Fill in the following table of value iteration values for the first 4 iterations.

States	0	2	3	4	5
V_0	0	0	0	0	0
V_1	0	2	3	4	5
V_2	3	3	3	4	5
V_3	10/3	3	3	4	5
V_0	10/3	3	3	4	5

3. You should have noticed that value iteration converged above. What is the optimal policy for the MDP?

States	0	2	3	4	5

π*	Draw	Draw	Stop	Stop	Stop

Top of the Hill

In this assignment, you will solve for the optimal policy and values of a small MDP, given by the following story. You will solve using both value iteration and policy iteration, which you are allowed to do in any way you like (by hand or computer), but you should show all your work (including your source code or whatever else you use).

A little autonomous rover on Mars depends on its solar panels for energy. Its goal is to collect as much energy as possible. It is close to a small hill; it collects the most energy when it is at the top of the hill, but it has a tendency to roll off the hill, and it takes energy to get back up the hill.

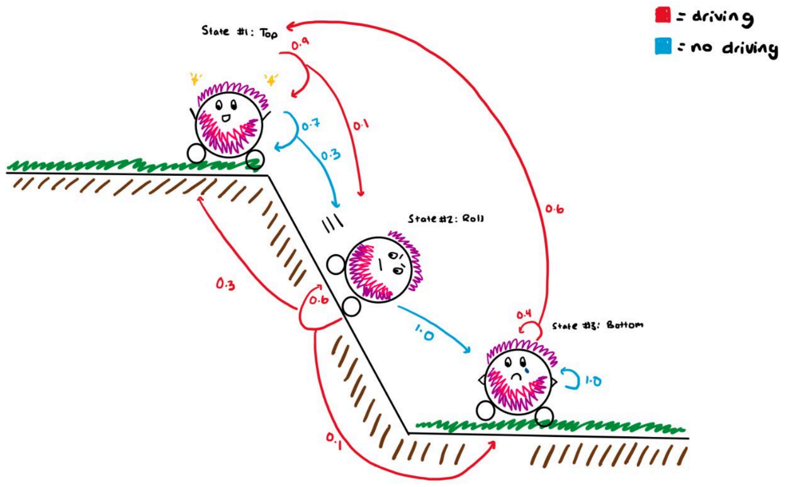
Specifically, the rover can be in one of three states: on **top of the hill**, **rolling down the hill,** or at the **bottom of the hill**. There are two actions that it can take in each state: **drive or don't drive**. Driving always costs 1 unit of energy. **When it is at the top of the hill, it collects 3 units of energy**; in the other two states, it collects 1 unit of energy. For example, if the rover is at the top of the hill and is driving (to stay on top of the hill), its reward is $3 - 1 = 2$.

If the rover is at the top of the hill and drives, then it is still at the top of the hill in the next period with probability 0.9 and rolling down in the next period with probability 0.1. If it is at the top of the hill and does not drive, these probabilities are 0.7 and 0.3, respectively.

If it is rolling down and drives, then with probability 0.3 it is at the top of the hill in the next period, with probability 0.6 it is still rolling down in the next period, and with probability 0.1 it is at the bottom in the next period. If it is rolling down and does not drive, then with probability 1.0 it is at the bottom in the next period.

Finally, if it is at the bottom of the hill and drives, then in the next period it is at the top of the hill with probability 0.6, and at the bottom with probability 0.4. If it does not drive, then with probability 1.0 it is at the bottom in the next period.

4. Draw the MDP graphically, similarly to the race car example from class.



5. Using a discount factor of 0.8, solve the MDP using value iteration (until the values have become reasonably stable). You should start with the values set to zero. You should show both the optimal policy and the optimal values.

NOTE:

- V(s) = max_a [R(s,a) + γ * Σ_s' T(s,a,s') * V(s')]
- s is the current state
- a is the action
- R(s,a) is the reward for taking action a in state s
- γ is the discount factor
- T(s,a,s') is the transition probability of going to state s' after taking action a in state s
- V(s') is the value of the next state s'

VARIABLES:

- Transition from Top to Top, Driving: 0.9
- Transition from Top to Top, Not Driving: 0.7
- Transition from Top to Rolling, Driving: 0.1
- Transition from Top to Rolling, Not Driving: 0.3

- Transition from Rolling to Top, Driving: 0.3
- Transition from Rolling to Rolling, Driving: 0.6
- Transition from Rolling to Bottom, Driving: 0.1
- Transition from Rolling to Bottom, Not Driving: 1.0

- Transition from Bottom to Top, Driving: 0.6
- Transition from Bottom to Bottom, Driving: 0.4
- Transition from Bottom to Bottom, Not Driving: 1.0

- Reward for Top, Driving: 2
- Reward for Top, Not Driving: 3
- Reward for Rolling, Driving: 0
- Reward for Rolling, Not Driving: 1
- Reward for Bottom, Driving: 0
- Reward for Bottom, Not Driving: 1

FORMULAS:

- V(Top) → max [R(Top, Drive) + 0.8 * (0.9*V(Top) + 0.1*V(Rolling)), R(Top, Don't Drive) + 0.8 * (0.7*V(Top) + 0.3*V(Rolling))]
- V(Rolling) → max [R(Rolling, Drive) + 0.8 * (0.3*V(Top) + 0.6*V(Rolling) + 0.1*V(Bottom)), R(Rolling, Don't Drive) + 0.8 * (1.0*V(Bottom))]
- V(Bottom) → max [R(Bottom, Drive) + 0.8 * (0.6*V(Top) + 0.4*V(Bottom)), R(Bottom, Don't Drive) + 0.8 * (1.0*V(Bottom))]

Iteration #1

V(Top) → max [2 + 0.8 * (0 + 0), 3 + 0.8 * (0 + 0)] = max [2, 3] = 3

V(Rolling) → max [0 + 0.8 * (0 + 0 + 0), 1 + 0.8 * (0)] = max [0, 1] = 1

V(Bottom) → max [0 + 0.8 * (0 + 0), 1 + 0.8 * (0)] = max [0, 1] = 1

Iteration #2

V(Top) → max [2 + 0.8 * ((0.9 * 3) + (0.1 * 1)) , 3 + 0.8 * ((0.7 * 3) + (0.3 * 1))]

max [4.24, 4.92] = 4.92

V(Rolling) → max [0 + 0.8 * ((0.3 * 3) + (0.6 * 1) + (0.1 * 1)) , 1 + 0.8 * (1.0 * 1)]

max [1.28, 1.8] = 1.8

V(Bottom) → max [0 + 0.8 * ((0.6 * 3) + (0.4 * 1)), 1 + 0.8 * (1.0 * 1)]

max [1.76, 1.8] = 1.8

Iteration #3

V(Top) → max [2 + 0.8 * ((0.9 * 4.92) + (0.1 * 1.8)) , 3 + 0.8 * ((0.7 * 4.92) + (0.3 * 1.8))]

max [5.68, 6.18] = 6.18

V(Rolling) → max [0 + 0.8 * ((0.3 * 4.92) + (0.6 * 1.8) + (0.1 * 1.8)) , 1 + 0.8 * (1.0 * 1.8)]

max [2.19, 2.44] = 2.44

V(Bottom) → max [0 + 0.8 * ((0.6 * 4.92) + (0.4 * 1.8)), 1 + 0.8 * (1.0 * 1.8)]

max [2.93, 2.44] = 2.93

Iteration #4

V(Top) → max [2 + 0.8 * ((0.9 * 6.18) + (0.1 * 2.44)) , 3 + 0.8 * ((0.7 * 6.18) + (0.3 * 2.44))]

max [6.64, 7.04] = 7.04

V(Rolling) → max [0 + 0.8 * ((0.3 * 6.18) + (0.6 * 2.44) + (0.1 * 2.93) , 1 + 0.8 * (1.0 * 2.93)]

max [2.88, 3.34] = 3.34

V(Bottom) → max [0 + 0.8 * ((0.6 * 6.18) + (0.4 * 2.93)), 1 + 0.8 * (1.0 * 2.93)] max [3.90, 3.34] = 3.90

Iteration #5

V(Top) → max [2 + 0.8 * ((0.9 * 7.04) + (0.1 * 3.34)) , 3 + 0.8 * ((0.7 * 7.04) + (0.3 * 3.34))]

max [7.34, 7.74] = 7.74

V(Rolling) → max [0 + 0.8 * ((0.3 * 7.04) + (0.6 * 3.34) + (0.1 * 3.90)) , 1 + 0.8 * (1.0 * 3.90)]

max [3.60, 4.12] = 4.12

V(Bottom) → max [0 + 0.8 * ((0.6 * 7.04) + (0.4 * 3.90)), 1 + 0.8 * (1.0 * 3.90)]

max [4.62, 4.12] = 4.62

Iteration #6

V(Top) → max [2 + 0.8 * ((0.9 * 7.74) + (0.1 * 4.12)) , 3 + 0.8 * ((0.7 * 7.74) + (0.3 * 4.12))]

max [7.9, 8.32] = 8.32

V(Rolling) → max [0 + 0.8 * ((0.3 * 7.74) + (0.6 * 4.12) + (0.1 * 4.62)) , 1 + 0.8 * (1.0 * 4.62)]

max [4.2, 4.7] = 4.7

V(Bottom) → max [0 + 0.8 * ((0.6 * 7.74) + (0.4 * 4.62)), 1 + 0.8 * (1.0 * 4.62)]

max [5.2, 4.7] = 5.2

And so on with iteration (I gave up writing them all out lol) till hitting stability …

V(Top) → **10.6**

V(Rolling) → **7.0**

V(Bottom) → **7.5**

… SO –

The Optimal Move for V(Top) is 'Don't Drive', π(Top) = Don't Drive

The Optimal Move for V(Rolling) is 'Don't Drive', π(Rolling) = Don't Drive

The Optimal Move for V(Bottom) is 'Drive', π(Bottom) = Drive

6. Using a discount factor of 0.8, solve the MDP using policy iteration (until you have complete convergence). You should start with the policy that never drives. Again, you should show both the optimal policy and the optimal values (and of course they should be the same as in 2.).

For my code I created these variables as stated above:

```
rewards = {
        (TOP, DRIVE): 2,
        (TOP, NO_DRIVE): 3,
        etc.
}

probabilities = {
        (TOP, DRIVE): {TOP: 0.9, ROLL: 0.1},
        (TOP, NO_DRIVE): {TOP: 0.7, ROLL: 0.3},
        etc.
}

etc..
```

Then I created these functions to recursively calculate the policies and their improvements:

```
def iterate_policies(states, actions, rewards, probabilities, G):
        current_policy = {s: 'NO_DRIVE' for s in states}
        stable = False
        while not stability:
                stability = True
                value = evaluate_policy(current_policy, states, actions, rewards, probabilities, gamma)
                new_policy = improve_policy(value, states, actions, rewards, probabilities, gamma)
                 if new_policy != current_policy:
                        stability = False
                        current_policy = new_policy
         return value, policy
```

```
def evaluate_policy(policy, states, actions, rewards, probabilities, gamma:
    values = {s: 0 for s in states}
    while True:
        max_change = 0
        min_threshold = 1e-6
        for s in states:
            v = values[s]
            p = policy[s]
            values[s] = rewards[(s, p)] + gamma * sum(probabilities[(s, p)][s’] * values[s’]
for s’ in P[(s, p)])
            delta = max(max_change, abs(v - values[s]))
        if max_change < min_threshold:
            break
    return values



def improve_policy(values, states, actions, rewards, probabilities, gamma):
    new_policy = {}
        for s in states:
            state_actions = {}
            for a in actions:
                state_actions[a] = rewards[(s, a)] + gamma * sum(probabilities[(s, a)][s’] *
values[s’] for s’ in P[(s, a)])
            new_policy[s] = max(actions, key=action_values.get)
    return new_policy
```

In order to get:

V(Top) → **10.6**

V(Rolling) → **7.0**

V(Bottom) → **7.5**

The Optimal Move for V(Top) is ‘Don’t Drive’, π(Top) = Don’t Drive

The Optimal Move for V(Rolling) is ‘Don’t Drive’, π(Rolling) = Don’t Drive

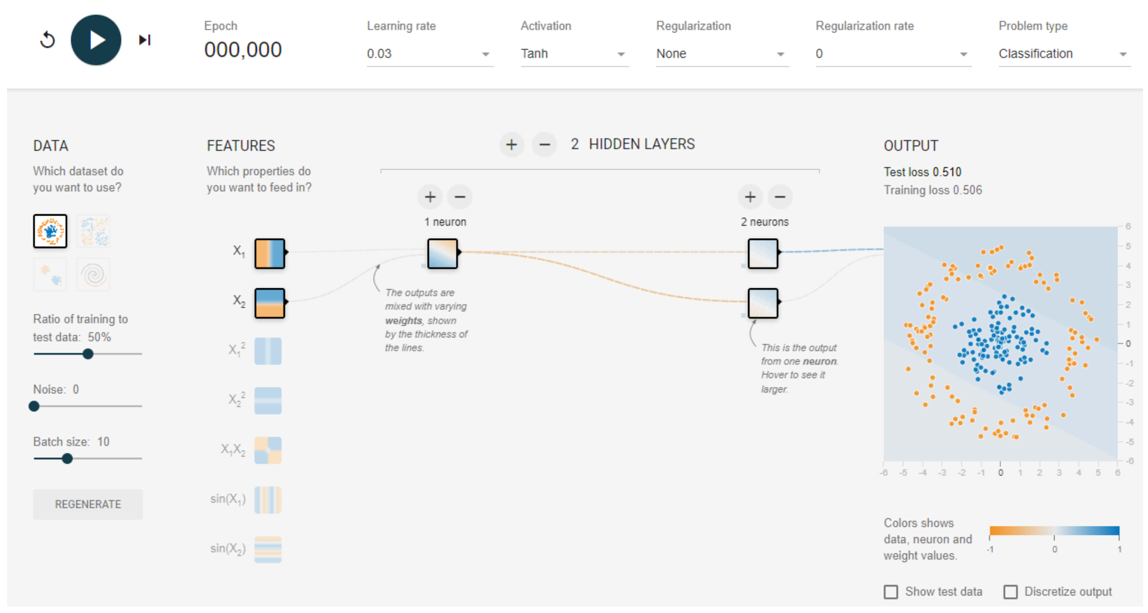
The Optimal Move for V(Bottom) is ‘Drive’, π(Bottom) = Drive

Neural Networks as Universal Approximators

Getting Started: Watch the video at https://cs50.harvard.edu/ai/2023/weeks/5/ from 39:10 to 45:50

At the same time link to https://playground.tensorflow.org/ and follow along to get a good sense of how the simulation works.

In the following problems you will experiment with designing neural networks in the Tensorflow Playground. For each of the following questions, you will need to submit a screenshot including the network configuration, test loss, and your parameter settings (including the DATA column on the left). For each part of this question, you will be asked to classify a different dataset (available in the DATA column). **Figure1** is a sample screenshot similar to what you are going to submit in your writeup to Canvas.



Epoch 000,000
Learning rate 0.03
Activation Tanh
Regularization None
Regularization rate 0
Problem type Classification

DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10

FEATURES
Which properties do you want to feed in?

2 HIDDEN LAYERS
1 neuron
2 neurons

The outputs are mixed with varying *weights*, shown by the thickness of the lines.

This is the output from one *neuron*. Hover to see it larger.

OUTPUT
Test loss 0.510
Training loss 0.506

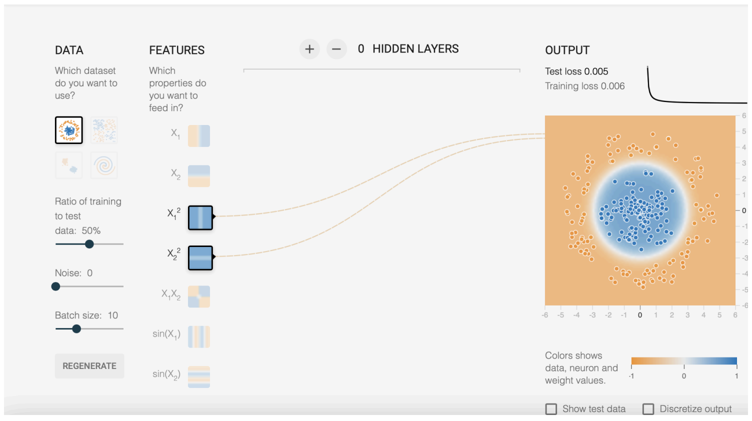
Colors shows data, neuron and weight values.

You can change the network structure (number of layers and number of neurons in each layer),and adjust the following hyperparameters: learning rate, activation, regularization type and regularization rate. Please leave other parameters fixed: leave ratio of training to test data to 50%, leave Noise at 0, and Batch size at 10. After you configure your network and parameter settings, you will use the "play" button to train the network and check whether the test loss can drop below the 0.1 threshold.

7. **Circle dataset**: Can you obtain a test loss of less than 0.1 using only the raw input features? i.e.,using only X_1 and X_2 as features? Submit a screenshot of your network results as described above (2 layers, with 1 and two neurons respectively.)

   Now suppose you use features X_1^2 and X_2^2, instead of raw features X1 and X2. Your objective now is to obtain a test loss of less than 0.1 in as few neurons as possible. How many neurons would you need to classify the dataset? Justify your answer.

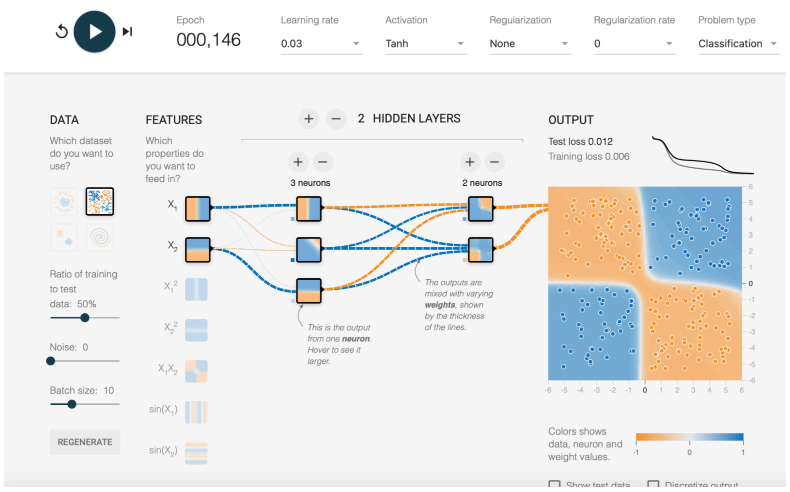
In this case, you would not need any neurons or hidden layers in order to classify the dataset. X12 and X22 use the x and y axis of the grid to draw a line boundary over where the blue points are clustered horizontally and vertically. Due to the blue cluster located directly in the center, the model is able to easily classify the blue versus the orange.


DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
0 HIDDEN LAYERS
OUTPUT
Test loss 0.005
Training loss 0.006
Colors shows data, neuron and weight values.
Show test data
Discretize output

8. **Exclusive-Or dataset**: Consider the following method to train your neural network model with raw features X_1 and X_2. Set the activation function to be tanh, use a **learning rate of 0.03** and don't use any regularization. Use a neural network with two hidden layers with the first hidden layer having 3 neurons and the second layer having 2 neurons. Do you obtain a test loss of less than 0.1? Now change the learning rate to 0.3 and report the final test loss. Repeat the same for learning rate =3. How does the test loss change as you increase the learning rate? Briefly explain why it changes that way.

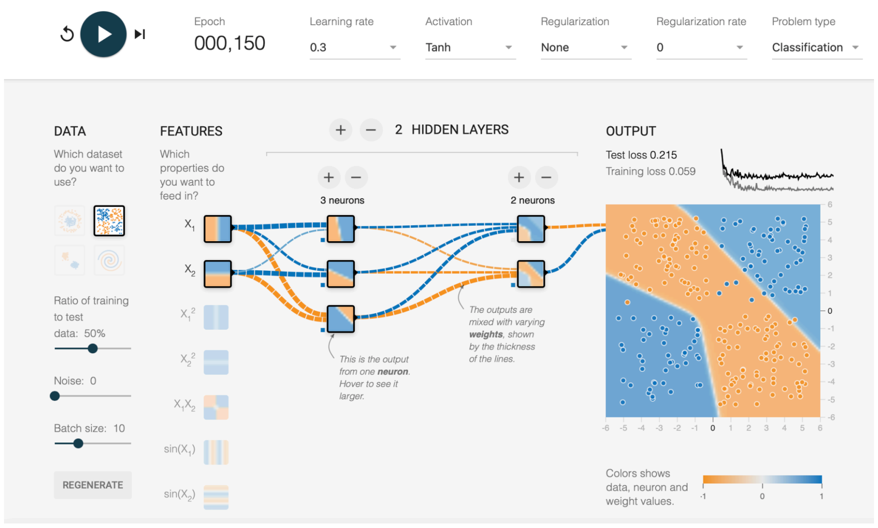
   Find another network architecture that also gives you a test loss of less than 0.1 on this dataset. Submit a screenshot of this architecture.

A. Use a neural network with two hidden layers with the first hidden layer having 3 neurons and the second layer having 2 neurons, you do obtain a test loss of less than 0.1.


Epoch 000,146 | Learning rate 0.03 | Activation Tanh | Regularization None | Regularization rate 0 | Problem type Classification

DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
2 HIDDEN LAYERS
3 neurons
2 neurons
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.012
Training loss 0.006
Colors shows data, neuron and weight values.
Show test data
Discretize output

B. By changing the learning rate to 0.3, the final test loss was 0.215



Epoch 000,150

Learning rate 0.3

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

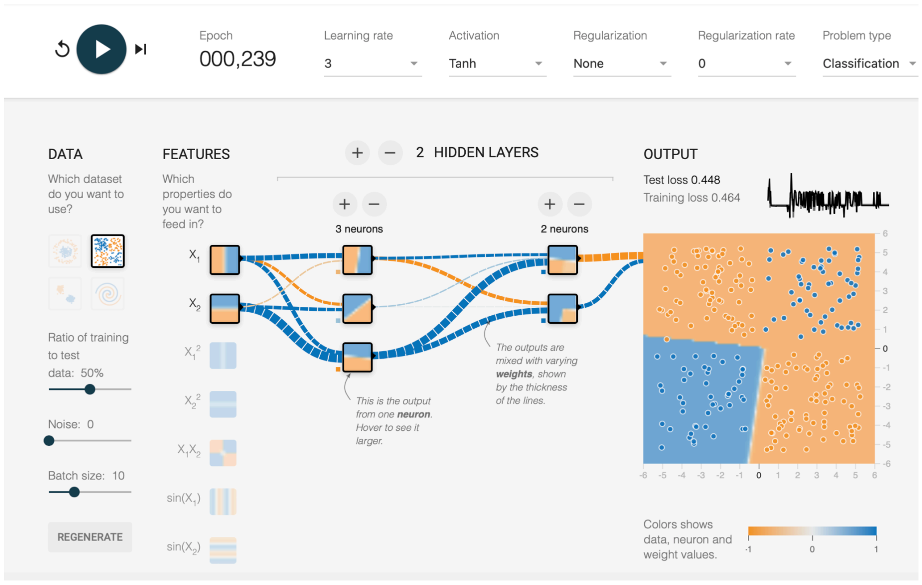
2 HIDDEN LAYERS: 3 neurons, 2 neurons

Test loss 0.215

Training loss 0.059

DATA
Which dataset do you want to use?
REGENERATE
FEATURES
Which properties do you want to feed in?
OUTPUT
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
Colors shows data, neuron and weight values.

C. By changing the learning rate to 3.0, the final test loss was increased to 0.448



Epoch 000,239

Learning rate 3

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

2 HIDDEN LAYERS: 3 neurons, 2 neurons

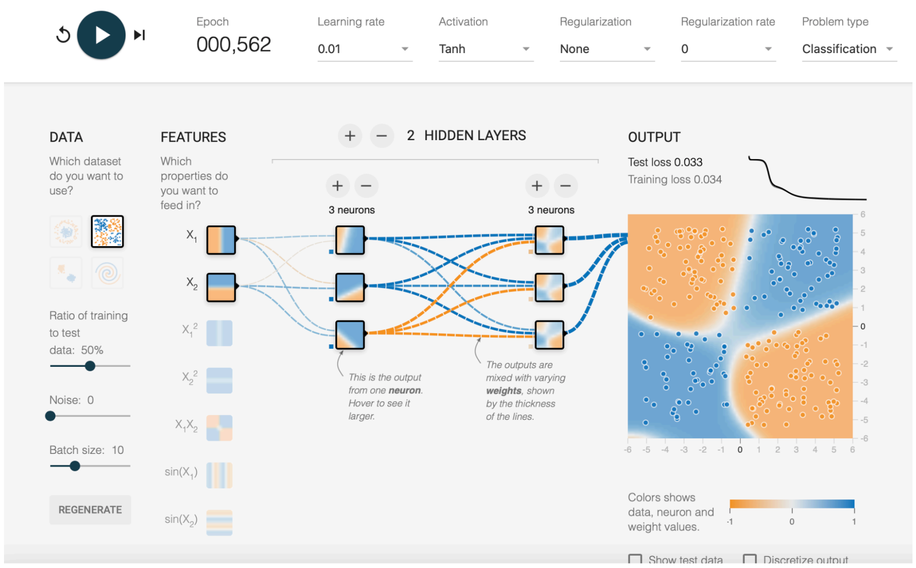
Test loss 0.448

Training loss 0.464

DATA
Which dataset do you want to use?
REGENERATE
FEATURES
Which properties do you want to feed in?
OUTPUT
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
Colors shows data, neuron and weight values.

Therefore, as you increase the learning rate to a too high amount, the test loss will increase because it will begin to put more weight on some features rather than others, resulting in faulty classification.

D. Find another network architecture that also gives you a test loss of less than 0.1 on this dataset by changing the learning rate to 0.01 and adding a third neuron onto the second hidden layer.



Epoch 000,562 · Learning rate 0.01 · Activation Tanh · Regularization None · Regularization rate 0 · Problem type Classification

Ratio of training to test data: 50% · Noise: 0 · Batch size: 10

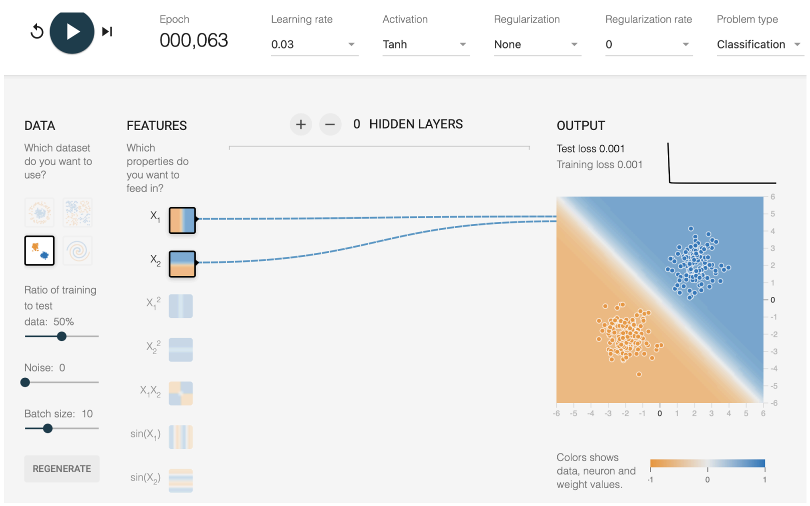
2 HIDDEN LAYERS: 3 neurons, 3 neurons

Test loss 0.033
Training loss 0.034

DATA
Which dataset do you want to use?
REGENERATE
FEATURES
Which properties do you want to feed in?
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Colors shows data, neuron and weight values.
Show test data
Discretize output

9. **Gaussian data set**: Can you obtain a test loss of less than 0.1 using only the raw inputs? i.e., using only X_1 and X_2? Try to use as few neurons as possible. Report the number of neurons used in your model and justify your answer.
Now suppose you use features X_1^2, X_2^2 instead of X_1,X_2. Can you still obtain a test loss of less than 0.1? Justify your answer.

A. Yes, I can obtain a test loss of less than 0.1 using only the raw inputs with zero hidden layers or neurons.



Epoch 000,063 · Learning rate 0.03 · Activation Tanh · Regularization None · Regularization rate 0 · Problem type Classification

Ratio of training to test data: 50% · Noise: 0 · Batch size: 10

0 HIDDEN LAYERS

Test loss 0.001
Training loss 0.001

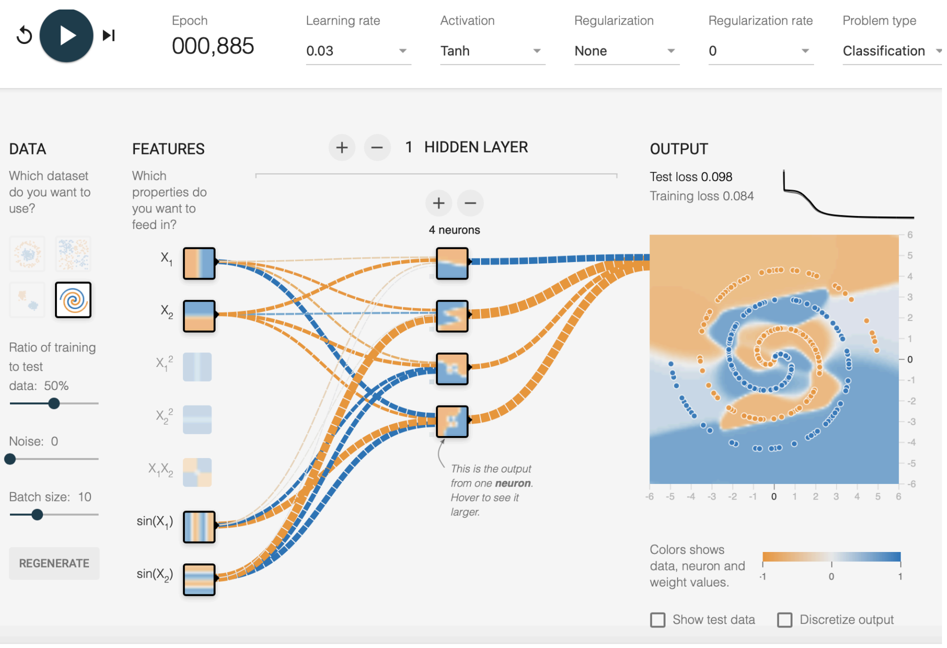
DATA
Which dataset do you want to use?
REGENERATE
FEATURES
Which properties do you want to feed in?
OUTPUT
Colors shows data, neuron and weight values.

B. No, I cannot obtain a test loss of less than 0.1 because of the way X12 and X22 split across the x and y axis.

10. **Spiral data set**: Spiral data set: Can you obtain test loss of less than 0.1 using only the raw inputs? i.e., using only X_1 and X_2? Submit your screenshot.
Now you are allowed to use any/all of the 7 different input features ($X_1,\ X_2,\ X_1^2,\ X_2^2,\ X_1X_2,\ \sin(X_1)$, and $\sin(X_2)$). Can you obtain a test loss of less than 0.1 for the Spiral data set (Figure 5) with fewer neurons?

C. Can I obtain a test loss of less than 0.1 using only the raw inputs? i.e., using only X_1 and X_2? The answer is very much no, I cannot. And I am extremely frustrated about it.

D. Obtain a test loss of less than 0.1 for the spiral data by using X1, X2, sin(X1) and sin(X2) features with 1 hidden layer containing 4 neurons.


Epoch 000,885
Learning rate 0.03
Activation Tanh
Regularization None
Regularization rate 0
Problem type Classification

DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE

FEATURES
Which properties do you want to feed in?

1 HIDDEN LAYER
4 neurons
This is the output from one **neuron**. Hover to see it larger.

OUTPUT
Test loss 0.098
Training loss 0.084
Colors shows data, neuron and weight values.
Show test data
Discretize output