Lab 2

Welcome to the second lab. The first exercise is a simple extension of the last lab, where we adapt the network to multiclass classification and we write generic facilities for the forward pass. The second exercise will put your geometric understanding of neural networks to the test. The third exercise contains some theoretical results about symmetries in the weight space to prompt some thinking on your part.

Exercise 1

This is a simple continuation of the second exercise of the previous lab, and you are allowed to re-use parts the solution you found. As a reminder, suppose you have five input points, $\mathbf{x}_1 = |0, 0|^T$, $\mathbf{x}_2 = |1, 0|^T$, $\mathbf{x}_3 = |0, -1|^T$, $\mathbf{x}_4 = |-1, 0|^T$ and $\mathbf{x}_5 = |0, 1|^T$, and the corresponding classes are $y_1 = 1$ and $y_2 = y_3 = y_4 = y_5 = 0$.

1. Add a second neuron to the output layer, and make it compute $1 - y$
   - How do the weights/biases change?
   - What should you use as activation function for the last layer?
   - And what loss function?
   - Pen and paper!
2. Write the code to perform the forward pass for a neural network with a generic number of layers, and test it with the weights and biases you found previously.

```
xs = matrix(c(
  0, 0,
  1, 0,
  0, -1,
  -1, 0,
  0, 1
), nrow = 5, ncol = 2, byrow = TRUE)

ys = matrix(c(
  1, 0,
  0, 1,
  0, 1,
  0, 1,
  0, 1
), nrow = 5, ncol = 2, byrow = TRUE)

ws1 = matrix(
  # TODO fill in the weights for the first layer
)

bs1 = c(
  # TODO fill in the biases for the first layer
)

ws2 = matrix(
  # TODO fill in the weights for the output layer
)

bs2 = c(
  # TODO fill in the biases for the output layer
)
```

```
relu = function(x) {
  ifelse(x > 0, x, 0)
}

softmax = function(predictions) {
  # TODO compute the softmax activation
}

predictions = (
  # TODO compute the predictions of the network
)

predictions
```

Compare with the desired outputs to make sure it is correct.

```
categorical_crossentropy = function(y_true, y_predicted) {
  # TODO compute the loss between the predictions and labels
}

loss = categorical_crossentropy(ys, predictions)

loss
```

As the network is giving the correct outputs, the loss should be low.

We can now create a generic function to perform the forward pass:

```
# This function represents a dense layer with given weights and biases.
dense = function(weights, bias) function(x) {
  # TODO compute the output of a dense layer.
}

forward_pass = function(layers, x) {
  # TODO use a for loop to perform the forward pass, and return the result.
}


forward_pass(c(
  dense(ws1, bs1),
  relu,
  dense(ws2, bs2),
  softmax
), xs)
```

Make sure this matches the output you had earlier.

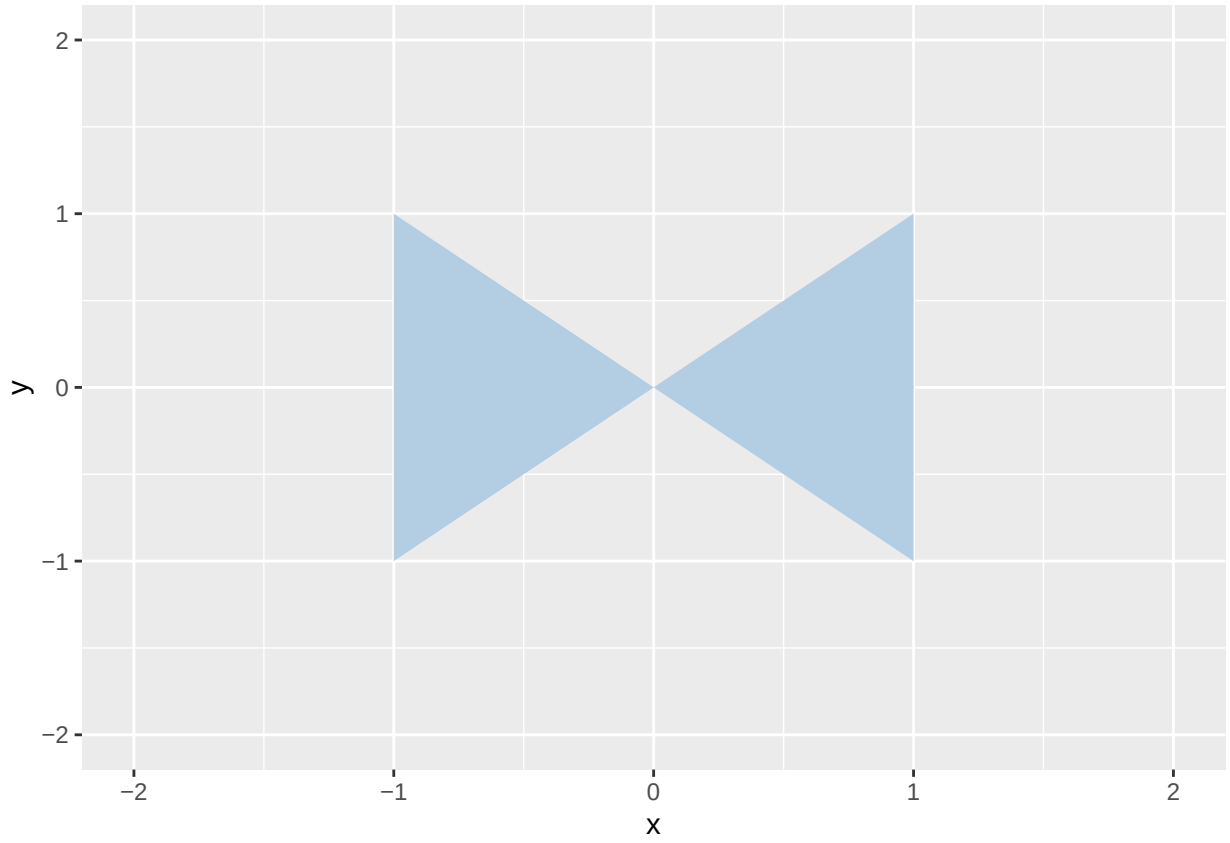
Exercise 2

Create a feed-forward neural network that can correctly classify the blue region in the plot below as $y = 1$, and the points outside it as $y = 0$.

- Explain why it cannot be done with a single hidden layer, but it is possible with two.
- Use four units for the first hidden layer, and two for the second hidden layer.
- Use the sigmoid activation function.
- Use the code you wrote previously to perform the forward pass.

```r
library(ggplot2)

df = data.frame(x=c(-1, -1, 0, 1, 1, 0, -1), y=c(1, -1, 0, -1, 1, 0, 1))
ggplot(df, aes(x=x, y=y)) +
  geom_polygon(fill='#b3cde3') +
  xlim(-2, 2) +
  ylim(-2, 2)
```




```r
sigmoid = function(x) {
  1 / (1 + exp(-x))
}


network_params = c(
  dense(
    weights = matrix(
      # TODO fill in the first weight matrix
    ), bias = c(
      # TODO fill in the first bias vector
    )
  ),
  sigmoid,
  dense(
    weights = matrix(
      # TODO fill in the second weight matrix
    ), bias = c(
```

```
      # TODO fill in the second bias vector
    )
  ),
  sigmoid,
  dense(
    weights = matrix(
      # TODO fill in the third weight matrix
    ), bias = c(
      # TODO fill in the third bias vector
    )
  ),
  sigmoid
)

# We evaluate the predictions of the network on a grid of points
halfres = 30
data = as.matrix(expand.grid(-halfres:halfres, -halfres:halfres)) / halfres * 2
predictions = forward_pass(network_params, data)

df = data.frame(x=data[,1], y=data[,2], c=predictions[,1])
ggplot(df, aes(x=x, y=y, col=c)) +
  geom_point(size=3) +
  scale_colour_gradient2(midpoint = 0.5)
```

Exercise 3

Consider a neuron with incoming weights $\mathbf{w} = w_1, \ldots, w_n$ and bias b. This neuron is connected to the i-th neuron of the next layer with the weight v_i, and the bias of the latter neuron is c_i. We want to replace $\mathbf{w}$, v_i, b and c_i with new parameters $\mathbf{w}'$, v_i', b' and c_i' so that the output of the network is unchanged for all inputs. At least one of the new parameters must be different, but some are allowed to equal the old ones.

1. Suppose that τ is the hyperbolic tangent. Show that the network computes the same function if we let $\mathbf{w}' = -\mathbf{w}$, $v_i' = -v_i$, $b' = -b$ and $c_i' = c_i$.
2. Now suppose that τ is the logistic sigmoid function. How should you set $\mathbf{w}'$, v_i', b' and c_i'?
3. Can you find other ways of modifying the parameters of a neural network without altering its output? Equivalently, given a neural network computing a certain function, how can you find a different network that computes the same function?
   - You do not have to provide a formal answer, but you can do so if you wish.