Exercise 1. Decision Trees and Ensemble Learning

1. (i) Given the binary classification problem from Homework 1, where we use a linear decision boundary not perpendicular to either the x1 or x2 axis, we would not be able to use a shallow decision tree with a small depth, because you can't sufficiently approximate the slope using a small number of perpendicular splits of the data. With only a small numbrer of splits, we have a zig zagging boundary.

1. (ii) if we rotated the data, such that the linear split between the data was either horizontal or vertical, then we could use a single split decision tree to recover the decision boundary.

2.

- Random forests select to split along some random subset (typically $\sqrt{n}$ rounded down) of the features to generate each tree using random sampling with replacement.
- Gradient Boosting modifies the labels using residuals of previous learners to train stronger learner.
- AdaBoost uses a higher probability to pick samples that are missclassified by previous trees. It modifies the weights of the datapoints in order to higher weight the data that performs poorly.

3. (i) Switching to

```
X["random_cat"] = rng.randint(4, size=X.shape[0])
```

Changing 3 to 4 increases the feature importance from below 0.05 to above 0.05. Our train and test accuracies stay roughly the same.

3. (ii) When features are correlated, you get the same information from both features. This means that when we permute one feature, it has little effect on the model's performance due to the existence of that information in a coorelated feature, this make it seem like the permuation imporance is low.

Exercise 2. Training Neural Networks

1. The sigmoid function captures the outcomes of binary classification as a probability, we are projecting the neural network outputs onto a sigmoid curve, between 0 and 1, which is easily interpreble as the probability of being in the positive class. Tanh is between 1 and -1, and probability can't be negative. ReLU has a slope of 1 on the right side, so quickly becomes greater than 1, and probability of a class can't be greater than 1.

2.

$$\frac{\partial \mathcal{L}}{\partial W_j^{(2)}} = \frac{\partial}{\partial W_j^{(2)}} C(f(g^{(2)}(h_2))$$

$$= \frac{\partial C}{\partial f} \frac{\partial f}{\partial g^{(2)}} \frac{\partial g^{(2)}}{\partial W_j^{(2)}}$$

$$= (o - y) f'(z) h_j$$

$$\frac{\partial \mathcal{L}}{\partial W_j^{(2)}} = (o - y) f'(z) h_j$$

3. The network might have a change of very small change in f'(z), which will kill all layers/paths before it, because the change of the cost function is multiplied by every other layer during backpropogation, making all gradients 0. Basically, vanishing gradients.

4. a. The log likelihood maximizes probability of some outcome, so inverting it allows us to minimize in order to do our optimization.

4. b. The range of the cross entropy log function is $[0, \infty)$. If the two are different (0 and 1), the result is infinity. If they are the same, then the result is 0.

5. $\frac{\partial \mathcal{L}}{\partial W_j^{(2)}} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial g^{(2)}}\frac{\partial g^{(2)}}{\partial W_j^{(2)}}$

$$\frac{\partial L}{\partial f} = (-\tfrac{y}{o} + \tfrac{1-y}{1-o})$$

$$\frac{\partial \mathcal{L}}{\partial W_j^{(2)}} = (-\tfrac{y}{o} + \tfrac{1-y}{1-o})f'(z)h_j = (-\tfrac{y}{o} + \tfrac{1-y}{1-o})o(1-o)h_j$$

$$= (-y(1-o) + o(1-y))h_j = (-y + yo + o - yo)h_j$$

$$= (o-y)h_j$$

6. So there clearly isn't a problem anymore with vanishing gradients. We don't have any gradients which can go to zero, even if we're somewhat certain, we don't pollute our other paths due to high certainty in one path.

7. We use linear activation for regression because it wont have a zero gradient. We can use squared error, as we're just looking to find the distance error in our prediction.

Exercise 3. Multilayer Perceptron

1. $\frac{\partial \mathcal{L}}{\partial W_{21}^{(1)}} = \frac{\partial C}{\partial f}\frac{\partial f}{\partial g^{(2)}}\frac{\partial g^{(2)}}{\partial f_1^{(1)}}\frac{\partial f_1^{(1)}}{\partial g_1^{(1)}}\frac{\partial g_1^{(1)}}{\partial W_{21}^{(1)}}$

$$\frac{\partial C}{\partial f} = \frac{\partial}{\partial f}(-ylog(o) - (1-y)log(1-o)) = (-\tfrac{y}{o} + \tfrac{1-y}{1-o})$$

$$\frac{\partial \mathcal{L}}{\partial W_{21}^{(1)}} = (-\tfrac{y}{o} + \tfrac{1-y}{1-o})f'(z)W_1^{(2)}f'(g_1^{(1)})x_2$$

$$= (-\frac{y}{o} + \frac{1-y}{1-o})o(1-o)W_1^{(2)} f(g_1^{(1)}(x))(1 - f(g_1^{(1)}(x)))x_2$$

$$= (o-y)W_1^{(2)} f(g_1^{(1)}(x))(1 - f(g_1^{(1)}(x)))x_2$$

2. If those conditions are true, that means we are certain about our answer within those paths. So, we the associated bias term must be zero, $\frac{\partial \mathcal{L}}{\partial b_2^{(1)}} = 0$ and the weights feeding in would also likely not change either. There isn't enough room behind h_2^1 to update it, so weights $\frac{\partial \mathcal{L}}{\partial W_{12}^{(1)}} = 0, \frac{\partial \mathcal{L}}{\partial W_{22}^{(1)}} = 0$ as they are killed off by the two weight gradients in the second layer feeding into them vanishing.

3. This isn't necessarily true if only one weight gradient is 0, because then there is a path to update each weight and the bias mentioned above, and so their gradients wont vanish.