1 Training

1. There are two data sets, one with 961 binarized features corresponding to structural information of the data sets, and one with 135 binarized features corresponding to content

2. Potentially, we can train the data without the use of a Sigmoid function (and just using sign). There is a paper here: https://arxiv.org/pdf/1602.02830.pdf . which describes training BNN's, and they mention techniques for training networks that aren't binarized but that do use the sign function as an activation function.

3. The plan now is to train the network using a traditional sigmoid in Tensor Flow and then to try to switch the sigmoid to sign and redefine the gradient of sign using the straight-forward estimator (gradient is 1 if close enough to 0, and 0 otherwise). Tensor Flow is chosen over SKLearn because it seems that SKLearn does not give great access to the inner structures of the models, and doing something like implementing a custom activation function is unsupported. (Additionally, Tensor Flow seems much more powerful and popular, so I'd like to learn it.)

2 Adjusting the Neural network

1. With no coding, try to apply $v_i = \hat{\tau}(\{i\})$.

2. Idea for finding the bias: If we want our neuron to be unbiased by the transformation, we should have that

   $$\mathbb{E}_{x\sim\mathbb{F}_2^n}[\tau(x)] = \mathbb{E}_{x\sim\mathbb{F}_2^n}[\tau'(x)]$$

   While this expectation may be hard to calculate due to the sign, we can certainly try to make the inside of the signs match. Then, because this is what Welch did with his two sample t-test, we'll try to make sure that at the very least the expectation and variance match each other. For the inside function of a neuron, generalized as $w^T x - \theta$, we have that

   $$\mathbb{E}_{x\sim\mathbb{F}_2^n}[w^T x - \theta] = \mathbb{E}_{x\sim\mathbb{F}_2^n}[w^T x] - \theta.$$

   Then, by waving my hands (or induction), we have that $\mathbb{E}_{x\sim\mathbb{F}_2^n}[w^T x] = 0$ since every point has a buddy that is the additive inverse, so we have

   $$\mathbb{E}_{x\sim\mathbb{F}_2^n}[w^T x - \theta] = -\theta.$$

   Next, we will find the variance, $\mathbb{E}_{x\sim\mathbb{F}_2^n}[(w^T x - \theta - (-\theta))^2]$. We have

$$\begin{aligned}
\mathbb{E}_{x\sim\mathbb{F}_2^n}[(w^Tx-\theta-(-\theta))^2] &= \mathbb{E}_{x\sim\mathbb{F}_2^n}[(w^Tx)^2] \\
&= \mathbb{E}_{x\sim\mathbb{F}_2^n}\left[\left(\sum_{i=1}^n w_ix_i\right)^2\right] \\
&= \mathbb{E}_{x\sim\mathbb{F}_2^n}\left[\sum_{i=1}^n\sum_{j=1}^n w_iw_jx_ix_j\right] \\
&= \sum_{i=1}^n\sum_{j=1}^n \mathbb{E}_{x\sim\mathbb{F}_2^n}[w_iw_jx_ix_j].
\end{aligned}$$

But, if $i \neq j$,

$$\mathbb{E}_{x\sim\mathbb{F}_2^n}[w_iw_jx_ix_j] = \frac{1}{4}w_iw_j\left((1)(1)+(-1)(1)+(1)(-1)+(-1)(-1)\right) = 0,$$

so, we restrict ourselves to the terms where $i = j$, so we have

$$= \sum_{i=1}^n \mathbb{E}_{x\sim\mathbb{F}_2^n}[w_i^2x_i^2].$$

However, since $x_i \in \{-1, 1\}$, $x_i^2 = 1$, so we have

$$\begin{aligned}
&= \sum_{i=1}^n \mathbb{E}_{x\sim\mathbb{F}_2^n}[w_i^2] \\
&= \|w\|_2^2.
\end{aligned}$$

Thus, in order to make our two neurons have the same expectation and variance, which seems like a reasonable way to make sure they behave somewhat similarly, we make θ the same for both and scale v so that $\|v\|_2^2 = \|w\|_2^2$. Specifically for sign functions, if you assume that the distribution of w^Tx given $x \sim \mathbb{F}_2^n$ is approximately normally distributed (this is just my intuition; it may not be true), then we would expect that the proportion of points that are classified as $+1$ or -1 are very close across the two neurons. Hopefully, this would also allow the two neurons to behave similarly across many other families of activation functions.

3. Testing with coding. Definitely want to try parity coding. I'm also interested in trying out other linear codes that are not just parity coding, but finding these may be difficult. I think just seeing how much better a parity code performs could itself be interesting.

3 Attacks

We want to take the adjustment of the Neural network to see how robust it is to noise. Vorobeychik lays out three main classes of attacks. Much his section on deep learning focusses instead on images, but it seems like we could use the same sorts of attacks. Each attack is seen as taking initially some x_0, then adding η such that $x_0 + \eta$ is misclassified. A general theme is either trying to minimize $\|\eta\|_p$ for some p, or restricting attacks to $\|\eta\|_p < \epsilon$ for some exogenous ϵ.

3.1 l_0

l_0 attacks aim to minimize $\|\eta\|_0$. Since $\|\eta\|_0$ is the number of elements of η that are non-zero, and since we are dealing with binary data and an element of η must be -2 or 2 (representing a flip from -1 to 1), then we must have that $\|\eta\|_1 = 2\,\|\eta\|_0$, so really this l_0 case is the same as the l_1 case. Since you solved for average robustness in the l_1 case, and since that case is distinct, this l_0/l_1 case could be worthwhile.

3.2 l_2

Vorobeychik specifically mentions l_2 as another of the cases, although in our cases, it just represents robustness using the l_p norm where $p > 1$. This could then prove to be an interesting case to analyze, since this is distinct from the l_0/l_1 case. However, while it would be interesting to see how robust the model is to these sorts of attacks after the transformation using the l_2 norm, it's unclear what meaning the l_2 norm has for binary input, and it seems that the l_1 norm is far more natural.

3.3 l_∞

Since $\|\eta\|_\infty$ measures the maximum change in an element of η, and since this can only be 2 if $\eta \neq 0$, this case is uninteresting.

3.4 Attack Algorithms

https://arxiv.org/abs/1511.07528 (algorithm 1). This paper, alluded to by Vorobeychik describes a method for computing an attack that minimizes the l_0 norm. The algorithm is essentially just the greedy algorithm you might expect.

For l_2 methods, some sort of gradient descent seems typical. However, this presents a challenge because, if we use the sign function, the model will not have a well-defined derivative. This could potentially be mended using the straight-through estimator, but testing methods for attack seems to stray from the goal of this project. The other option is to *not* use a sign activation function, and instead see how well the ”robustness transformation” works with a differentiable activation function. I'm inclined to do the later, since all the attacks Vorobeychik alludes to involve a gradient, but even if we do use a differentiable activation function, the gradient doesn't really make sense with binary input data.

Thus, since the l_0 attack algorithm is simple and seems to make more sense in the context of binary inputs, I'm more inclined to explore l_0 attacks and robustness to them, although l_2 attacks could make for a worthy sequel.

4 Calculation

Many of the things we'll be calculating would require a summation of 2^n terms. However, since they are expected values, we can compute them randomly. Since we are calculating the expectation of a boolean function, there's probably some inequality that applies here (I can look this up).

5 Some things I'd like to try

Not directly related to the project

1. Calculating all the expected values using actual data instead of random points

6 Approximate Road Map

1. Learn TensorFlow and train a model using a typical sigmoid function (2 weeks)
2. Then, implement a custom activation function and train the model using sign and the straight forward estimator. This roadmap assumes I will be able to do this. (1 week)
3. Implement the transformation (1/2 week)
4. Implement the adversary algorithm (the one mentioned above that aims to minimize the l_0 norm) and have it modify training data to test on the transformed model. Additionally, test the transformed model for accuracy across unchanged data (1 1/2 week).
5. Implement and test parity coding (1 week)
6. Try other methods of coding, maybe just randomly generated (1 week)