**Tensorflow Playground Activity**

**From Google ML Crash Course**

## **A First Neural Network (Data: Exclusive OR)**

**A picture containing graphical user interface

Description automatically generated**

In this exercise, we will train our first little neural net. Neural nets will give us a way to learn nonlinear models without the use of explicit feature crosses.

**Task 1:** The model as given above combines our two input features into a single neuron. Will this model learn any nonlinearities? Run it to confirm your guess.

**Answer:** Because the Activation is set to Linear, no nonlinearities can be learned by this model. We argue the model underfits the data since the loss is so significant.

**Task 2:** Try increasing the number of neurons in the hidden layer from 1 to 2, and also try changing from a Linear activation to a nonlinear activation like ReLU. Can you create a model that can learn nonlinearities? Can it model the data effectively?

**Answer:** Nonlinear models may be learned using the nonlinear activation function. A single hidden layer with two neurons, on the other hand, will not be able to represent all the nonlinearities in this data set, and will have a significant loss even if there is no noise: it will still underfit the data. Because these exercises are nondeterministic, some runs will fail to develop an effective model, while others will perform admirably.

**Task 3:** Try increasing the number of neurons in the hidden layer from 2 to 3, using a nonlinear activation like ReLU. Can it model the data effectively? How does model quality vary from run to run?

**Answer:** The nondeterministic nature of Playground is evident in this practice. In the absence of noise, a single hidden layer with three neurons is adequate to describe the data set, however not all runs will converge on an appropriate model. Due to the fact that the XOR function can be expressed as a combination of three half-planes, three neurons sufficient. This is demonstrated by the neuron images, which depict the output of individual neurons. In a decent model with 3 neurons with ReLU activation, there will be 1 image with an almost vertical line detecting X1 being positive (or negative; the sign may be inverted), 1 image with an almost horizontal line detecting the sign of X2, and 1 image with a diagonal line detecting their interaction. Nonetheless, not every run will yield a satisfactory model. Some runs will only yield a model with two neurons, and in these instances, duplicate neurons will be produced.

**Task 4:** Continue experimenting by adding or removing hidden layers and neurons per layer. Also feel free to change learning rates, regularization, and other learning settings. What is the *smallest* number of neurons and layers you can use that gives test loss of 0.177 or lower?

Does increasing the model size improve the fit, or how quickly it converges? Does this change how often it converges to a good model? For example, try the following architecture:

* First hidden layer with 3 neurons.
* Second hidden layer with 3 neurons.
* Third hidden layer with 2 neurons.

Graphical user interface, diagram

Description automatically generated

**Answer:**

A single hidden layer with three neurons can describe the data, but because there is no redundancy, it will lose a neuron on many runs and fail to learn a satisfactory model. Because a single layer with more than three neurons contains greater redundancy, it is more likely to converge to a suitable model.

As we've shown, a single hidden layer with only two neurons is unable to adequately model the data. You can see that all the objects in the output layer can only be forms made up of the lines from those two nodes if you test it. In this scenario, a deeper network may describe the data set better than the first hidden layer alone: by merging neurons in the first layer, individual neurons in the second layer can model more complicated forms, such as the upper-right quadrant. While adding that second hidden layer can still represent the data set better than the first hidden layer alone, it may make more sense to increase the number of nodes in the first layer to allow more lines to be included in the kit from which the second layer constructs its forms.

However, no matter how deep the model is, a model with only one neuron in the first hidden layer cannot learn a decent model. This is because the first layer's output only fluctuates in one dimension (typically a diagonal line), which is insufficient to adequately simulate this data set. Later layers, no matter how complicated, will not be able to compensate for this; information in the input data has been irreversibly lost. What if, instead of attempting to create a tiny network, we used several layers with many neurons to solve an issue like this? As we've seen, the first layer will be able to experiment with a variety of line slopes. And the second layer will be able to collect them into a variety of various forms, with even more shapes appearing as the layers progress. You've created enough space for the model to start easily overfitting on the noise in the training set, allowing these complex shapes to match the foibles of the training data rather than the generalized ground truth, by allowing the model to consider so many different shapes through so many different hidden neurons. Larger models, for example, may have complex bounds to match the precise data points. In extreme instances, a big model might learn an island around a single source of noise, a process known as data memorizing. You'll see that by making the model so massive, it typically performs worse than the smaller model with just enough neurons to solve the problem.

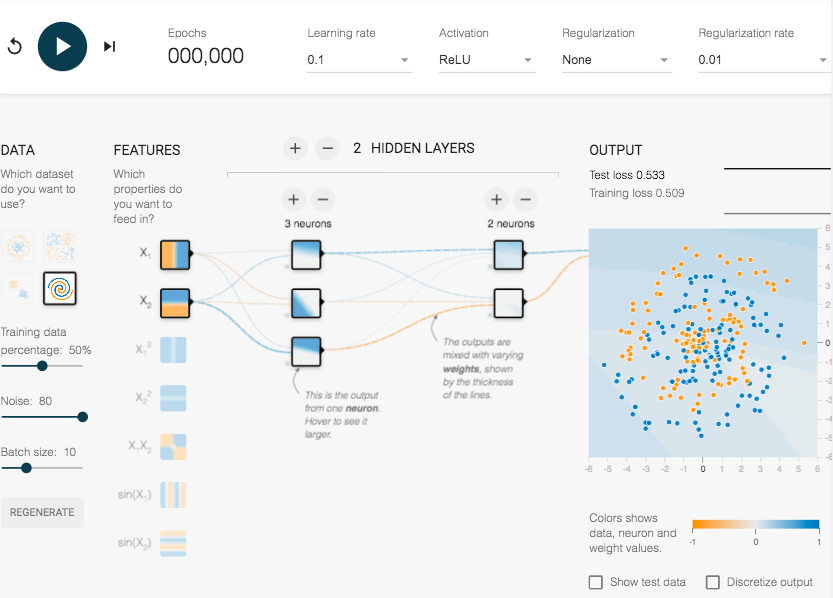
## **Neural Net Initialization**

This exercise uses the XOR data again but looks at the repeatability of training Neural Nets and the importance of initialization.

**Task 1:** Run the model as given above four or five times. Before each trial, hit the **Reset the network** button to get a new random initialization. (The **Reset the network** button is the circular reset arrow just to the left of the Play button.) Let each trial run for at least 500 steps to ensure convergence. What shape does each model output converge to? What does this say about the role of initialization?

**Answer:** On each iteration, the learnt model took on a new form. From lowest to highest, the converged test loss ranged by about 2X.

**Task 2:** Try making the model slightly more complex by adding a layer and a couple of extra nodes. Repeat the trials from Task 1. Does this add any additional stability to the results?



**Answer:** The addition of the layer and more nodes resulted in more consistent results. The final model was nearly identical on each iteration. Additionally, the converged test loss had reduced variation across runs.

## **Neural Net Spiral**

This data set is a noisy spiral. Obviously, a linear model will fail here, but even manually defined feature crosses may be hard to construct.

**Task 1:** Train the best model you can, using just X1 and X2. Feel free to add or remove layers and neurons, change learning settings like learning rate, regularization rate, and batch size. What is the best test loss you can get? How smooth is the model output surface?

**Answer:** Including many layers and countless neurons in each layer. There are too many nodes in the network. This model is sluggish because adjusting all the weights in the graph is computationally costly. With so many layers and nodes in the graph, it will be tedious to count the iterations, and the model will be difficult to comprehend. If we can lower the number of nodes, we can obtain a model that is quicker and more logical, and we can also alter the parameters to improve test loss. Reducing the learning rate so that there is less back and forth movement in the loss curve. With less nodes and a slower pace of learning. It is evident that the model is learning more quickly. The iterations are proceeding considerably more rapidly, but the test loss remains comparable. We can see that the loss curves are still fluctuating, and that the model's output is not as smooth as the spirals we are attempting to match.

**Task 2:** Even with Neural Nets, some amount of feature engineering is often needed to achieve best performance. Try adding in additional cross-product features or other transformations like sin(X1) and sin(X2). Do you get a better model? Is the model output surface any smoother?

**Answer:** If we have noisy data, we must also perform feature engineering to improve the test loss. We have turned up the volume. Add sine to the squares of x1 and x2. One of the most noticeable effects of adding more complex features is the increased complexity of the model's first layer's characteristics. The things that neurons are learning are far more complicated than merely lions on varying slopes. By adding more elements, it is evident that we are learning these curves far faster. Nonetheless, it is evident that this is a highly complicated curve and that our network has too many nodes, which is our first problem. Eliminate a portion of these layers and neurons. By decreasing the number of layers and neurons inside each layer. I also decrease the learning rate. This yields a significantly smoother loss curve and a lot better fit to the data, even though the data are still noisy. The features that we've implemented have been quite beneficial. And by lowering model complexity, the amount of model layers, we are far better at avoiding overfitting the data. By modifying a few the other parameters in my activation function, such as learning rate and regularization rate, I am able to obtain a much better test loss, a much smoother fitting curve, and a much faster convergence with a model including only one hidden layer and five neurons.