VisualizingNNs

June 2, 2020

1 Visualizing Neural Networks

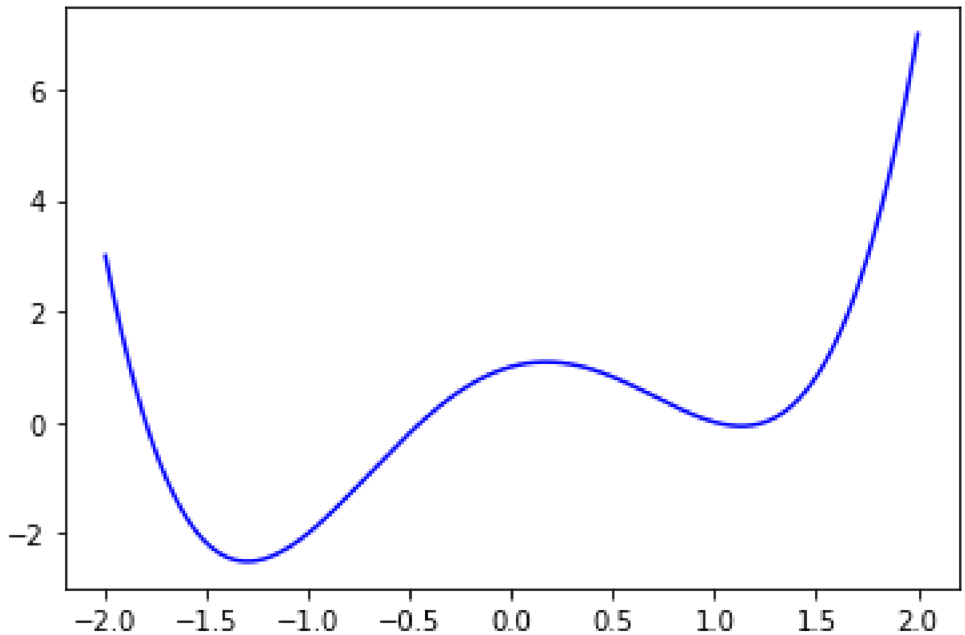
Alex Marshall In this notebook I create a function to animate the learning of a neural network performing either a polynomial line fit or a ring classification. This was inspired by Chris Olah's blog post Neural Networks, Manifolds, and Topology and Scott Rome's tutorial Visualizing the Learning of a Neural Network Geometrically. The code in this notebook was modified from Scott Rome's.

[8]:
```
import numpy as np
import pandas as pd

from scipy import special

import matplotlib.pyplot as plt
import matplotlib.animation as animation


from IPython.display import HTML

import keras
from keras import metrics
from keras import backend as K
from keras.models import Sequential, Model
from keras.layers import Dense, Activation, Input, Lambda
%matplotlib inline
```

1.0.1 Create Datasets to fit

[3]:
```
#Polynomial

plt.clf()

# Training data is in [-2,2]
x_train=4*np.random.rand(4000)-2
y_train=x_train**4-3*x_train**2+x_train+1
```

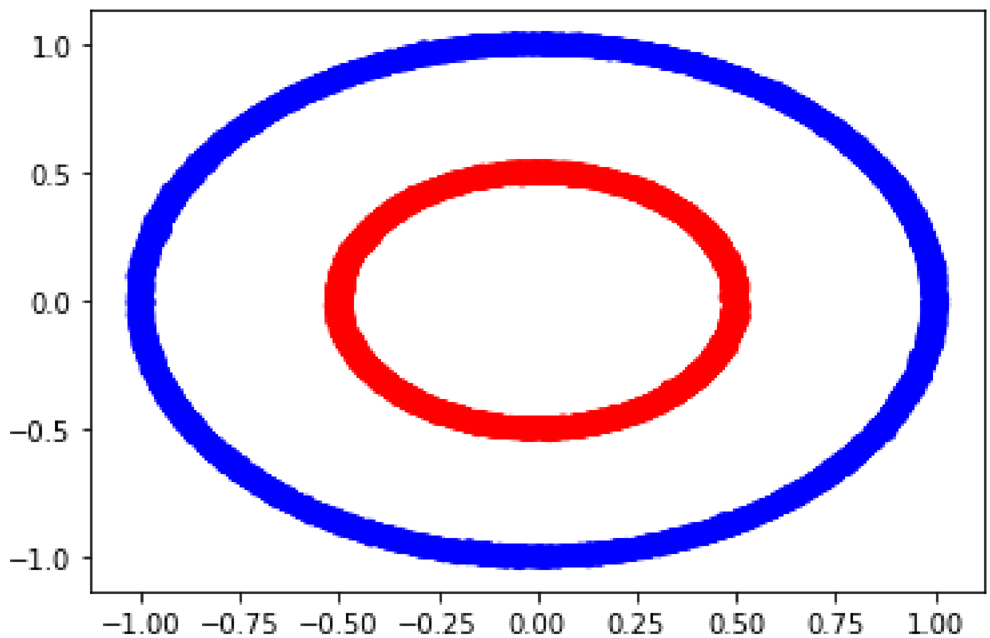
```
# target distribution
x_test=np.linspace(-2,2,400)
y_test=x_test**4-3*x_test**2+x_test+1

trainDF=pd.DataFrame({'x':x_train, 'y':y_train})
testDF=pd.DataFrame({'x':x_test, 'y':y_test})

# Show target distribution
plt.plot(testDF['x'],testDF['y'],color='b')
plt.show()
```



[4]:
```
# Rings for classification
# using code from the Rome tutorial
plt.clf()

#create inner circle - radius 0.5
n=20000
t = np.linspace(0,2,n) #points along axis
x = np.sin(np.pi*t) + np.random.normal(0,.005,n) #length 2, magnitude 2
y = np.cos(np.pi*t) + np.random.normal(0,.005,n)
label = np.ones(n)  #label 1

tdf = pd.DataFrame({'label' : label, 'x' : x, 'y' : y})

#create outer circle - radius 1
t = np.linspace(0,2,n)
```

```
x = .5*np.sin(np.pi*t) + np.random.normal(0,.005,n) #length 2, magnitude 1
y = .5*np.cos(np.pi*t) + np.random.normal(0,.005,n)
label = 0.*np.ones(n) #label 0

df = pd.concat([tdf, pd.DataFrame({'label' : label, 'x' : x, 'y' : y})]).
 ↪reset_index()

plt.scatter(df['x'],df['y'], color=df['label'].apply(lambda l :  'b' if l > .5␣
 ↪else 'r'))
```

[4]: <matplotlib.collections.PathCollection at 0x1d4ae5c2cf8>



1.0.2 Functions for Animation

Default is the fitter animation to animate the output of a neural network attempting to fit a 2D line. The classifier animation will animate the two nodes of the last hidden layer of the network. The idea is to see how the network represents the space that the input data is in as it tries to classify it.

[5]:
```
"""
NN Layer Animator
This is a generalized version of the code from the Scott Rome tutorial
"""
```

```
def get_decision_boundary(model):
    """ Function to return the x-y coodinates (-1,1) of the decision boundary␣
↪given a (classification) model.
        assumes the 2nd to last layer is a 2 hidden unit layer with a bias term
        and sigmoid activation on the last layer."""
    a = model.layers[-1].get_weights()[0][0][0] #weight for 1st node as scalar␣
↪in last layer
    b = model.layers[-1].get_weights()[0][1][0] #weight for 2nd node as scalar␣
↪in last layer
    c = model.layers[-1].get_weights()[1][0] #bias as scalar for last layer
    decision_x = np.linspace(-1,1,100)
    decision_y = (special.logit(.5)-c-a*decision_x)/b #isolate y in␣
↪sigmoid(ax+by+c)=.5
    return decision_x, decision_y

def animate_model(model, n_frames=100, Type='Fitter'): #add arg to animate 1 or␣
↪2 nodes
    """ Function to animate a model's first n_frames epochs of training. """

    plt.clf()
    if Type=='Fitter':
        lyr=-1
    elif Type=='Classifier':
        lyr=-2


    #define function for 1-node output layer of NN
    f = K.function(inputs = model.inputs, outputs = [model.layers[lyr].output])
    # define fig, ax as subplots
    fig, ax = plt.subplots()

    if Type=='Classifier':
        #creates x and y pts of a grid 0.1 units apart to rep input space
        grids = [np.column_stack((np.linspace(-1,1, 100), k*np.ones(100)/10.))␣
↪for k in range(-10,11)] +\
                [np.column_stack((k*np.ones(100)/10.,np.linspace(-1,1, 100)))␣
↪for k in range(-10,11) ]
        #pass model into decision boundary funtion to get x and y arrays for␣
↪decision boundary
        decision_x, decision_y = get_decision_boundary(model)
        #returns two list of indexes. One where label=1 (blue), one where␣
↪label=0 (red)
        indb = df.index[df['label']==1].tolist()
        indr = df.index[df['label']==0].tolist()
        grid_lines=[]
```

```
        orig_vals = f(inputs=[df[['x','y']].values])[0] #[0] for formatting
        line, = ax.plot(decision_x,decision_y,color='black') #plot decision␣
↪boundary assuming node 1 is x and node 2 is y
        lineb, = ax.plot(orig_vals[indb,0], orig_vals[indb,1], marker='.',␣
↪color='b') #plot blue ring at hidden 1
        liner, = ax.plot(orig_vals[indr,0], orig_vals[indr,1], marker='.',␣
↪color='r') # plot red ring at hidden 1
        #loop through grids and plot
        for grid in grids:
            vals = np.array(grid)
            l, = ax.plot(vals[:,0],vals[:,1], color='grey', alpha=.5)
            grid_lines.append(l)
        all_lines = tuple([line, lineb, liner, *grid_lines]) # * takes out of␣
↪list form so that values can be part of tuple not list in tuple

        def animate(i):
            model.fit(df[['x','y']].values, df[['label']].values, epochs=1,␣
↪batch_size=32, verbose=0)
            line.set_data(*get_decision_boundary(model))
            vals = f(inputs = [df[['x','y']].values])[0] #keras backend funct
            lineb.set_data(vals[indb,0], vals[indb,1])
            liner.set_data(vals[indr,0], vals[indr,1])

            for k in range(len(grid_lines)):
                ln = grid_lines[k]
                grid = grids[k]
                vals = f(inputs = [np.array(grid)])[0] #keras backend funct
                ln.set_data(vals[:,0],vals[:,1])

            return all_lines

        def init():
            line.set_ydata(np.ma.array(decision_x, mask=True))
            lineb.set_data(orig_vals[indb,0],orig_vals[indb,1])
            liner.set_data(orig_vals[indr,0],orig_vals[indr,1])
            for k in range(len(grid_lines)):
                ln = grid_lines[k]
                grid = grids[k]
                vals = f(inputs = [np.array(grid)])[0]
                ln.set_data(vals[:,0],vals[:,1])
            return all_lines

    elif Type=='Fitter':
        orig_vals = f(inputs=[testDF[['x']].values])[0] #[0] for formatting
        target, = ax.plot(testDF['x'], testDF['y'], marker='.', color='b')␣
↪#plot orig function
```

```
        line, = ax.plot(testDF['x'], orig_vals, marker='.', color='r') #plot␣
↪orig function
        all_lines = tuple([target,line])

        def animate(i):
            model.fit(trainDF[['x']].values, trainDF[['y']].values, epochs=1,␣
↪batch_size=32, verbose=0)
            vals = f(inputs = [testDF[['x']].values])[0] #keras backend funct
            line.set_data(testDF[['x']].values,vals)

            return all_lines

        def init():
            line.set_data(testDF['x'],orig_vals)
            return all_lines

    return animation.FuncAnimation(fig, animate, np.arange(1, n_frames),␣
↪init_func=init,
                                   interval=100, blit=True)
```

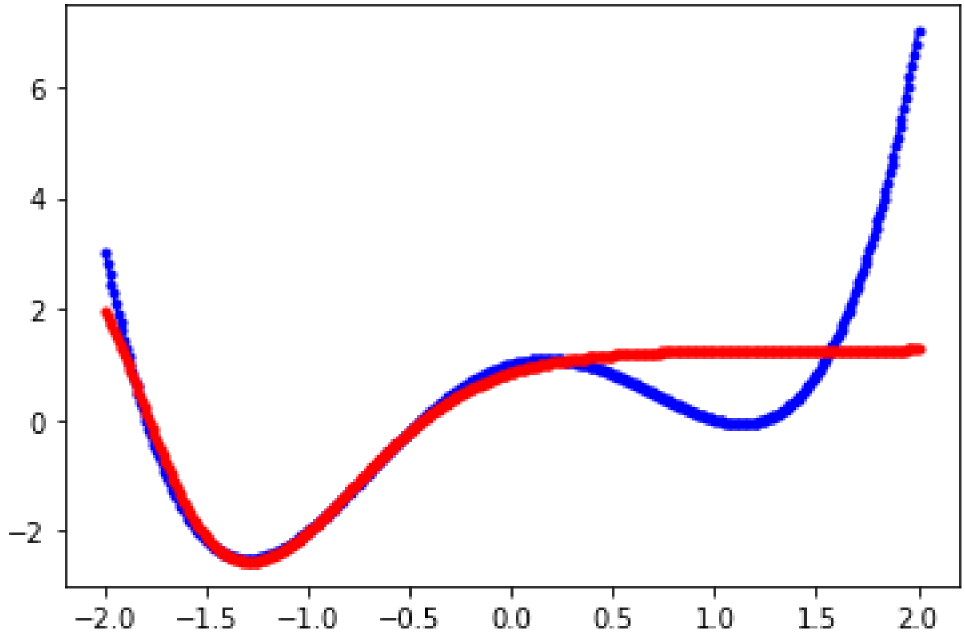
1.0.3 Architectures for polynomial fit problem

Start with a shallow and narrow network.

[10]:
```
#1-2-1, hyptan, sgd lr 0.01

model = Sequential() #linear stack of layers
model.add(Dense(2, activation='tanh', input_dim=1)) #2 node fully connected␣
 ↪layer takes 1 inputs
model.add(Dense(1, activation='linear')) #fully connected single node output␣
 ↪layer
sgd = keras.optimizers.SGD(lr=0.01) #stoch gradient desc optimization with␣
 ↪learning rate of .01
model.compile(optimizer=sgd,loss='mse')

anim = animate_model(model);
HTML(anim.to_html5_video())
```

[10]:
```
<IPython.core.display.HTML object>
```

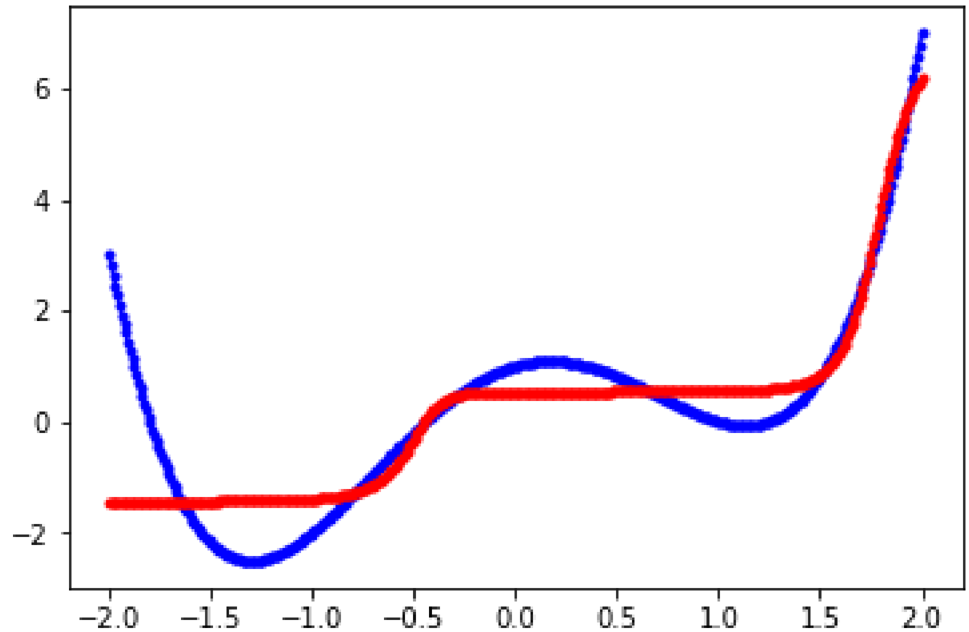
```
<Figure size 432x288 with 0 Axes>
```



Try adding more depth.

[32]:
```
#1-2-2-2-1, hyptan, sgd lr 0.01
model = Sequential() #linear stack of layers
model.add(Dense(2, activation='tanh', input_dim=1))
model.add(Dense(2, activation='tanh'))
model.add(Dense(2, activation='tanh'))
model.add(Dense(1, activation='linear'))
sgd = keras.optimizers.SGD(lr=0.01)
model.compile(optimizer=sgd,loss='mse')

anim = animate_model(model);
HTML(anim.to_html5_video())
```

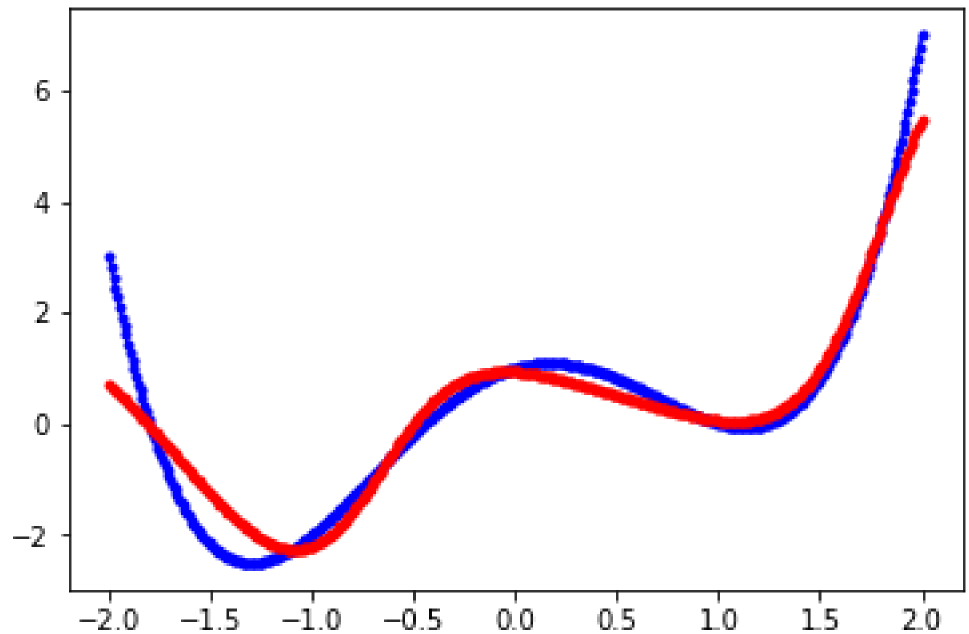
[32]:
```
<IPython.core.display.HTML object>

<Figure size 432x288 with 0 Axes>
```



Now with more width.

[33]:
```
#1-6-1, hyptan, sgd lr 0.01
model = Sequential() #linear stack of layers
model.add(Dense(6, activation='tanh', input_dim=1))
model.add(Dense(1, activation='linear'))
sgd = keras.optimizers.SGD(lr=0.01)
model.compile(optimizer=sgd,loss='mse')

anim = animate_model(model);
HTML(anim.to_html5_video())
```

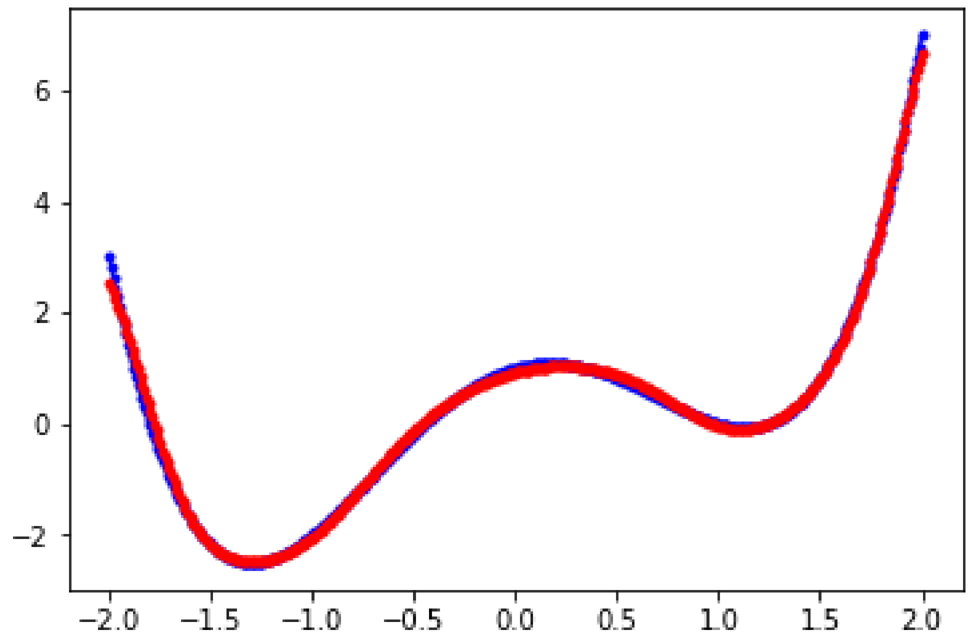
[33]:
```
<IPython.core.display.HTML object>

<Figure size 432x288 with 0 Axes>
```



Now something in between.

[37]:
```
#1-3-3-1, hyptan, sgd lr 0.01
model = Sequential() #linear stack of layers
model.add(Dense(3, activation='tanh', input_dim=1))
model.add(Dense(3, activation='tanh'))
model.add(Dense(1, activation='linear'))
sgd = keras.optimizers.SGD(lr=0.01)
model.compile(optimizer=sgd,loss='mse')

anim = animate_model(model);
HTML(anim.to_html5_video())
```

[37]:
```
<IPython.core.display.HTML object>

<Figure size 432x288 with 0 Axes>
```

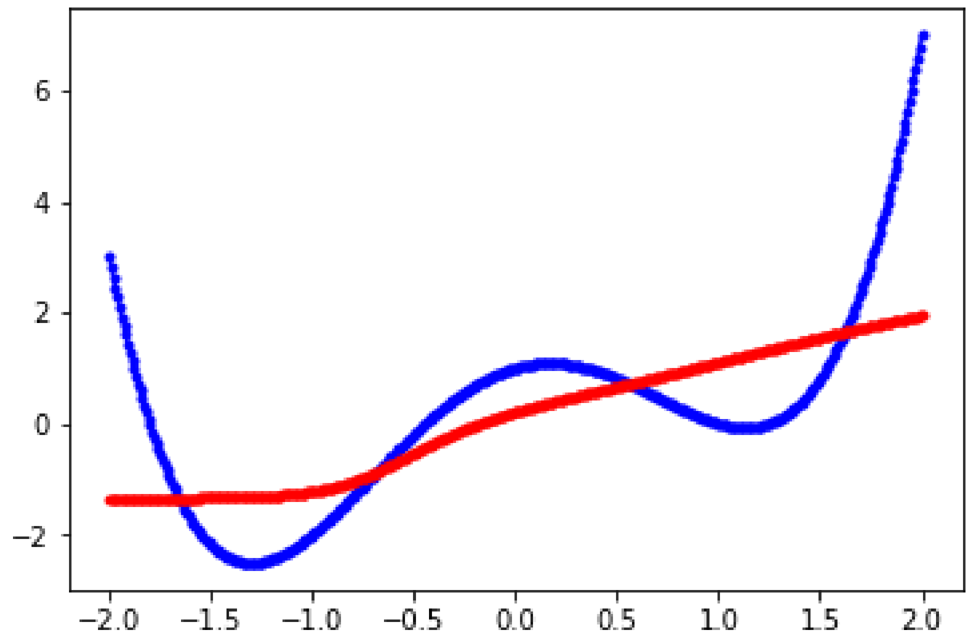


And with a chokepoint.

[39]:
```
#1-3-1-3-1, hyptan, sgd lr 0.01
model = Sequential()
model.add(Dense(3, activation='tanh', input_dim=1))
model.add(Dense(1, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(1, activation='linear'))
sgd = keras.optimizers.SGD(lr=0.01)
model.compile(optimizer=sgd,loss='mse')

anim = animate_model(model);
HTML(anim.to_html5_video())
```

[39]:
```
<IPython.core.display.HTML object>
```

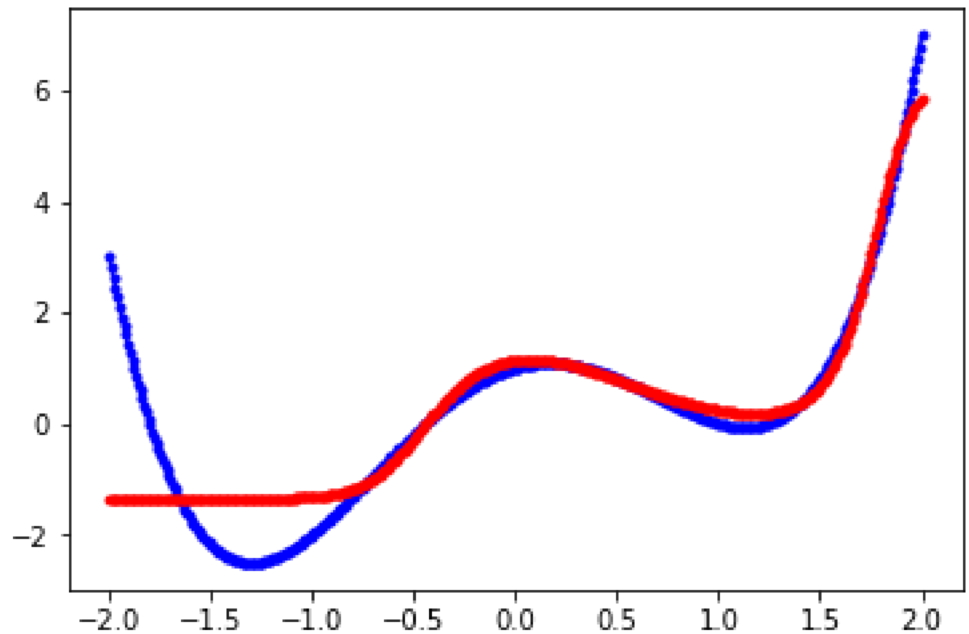
```
<Figure size 432x288 with 0 Axes>
```



[38]:
```
#1-3-3-1-3-3-1, hyptan, sgd lr 0.01
model = Sequential()
model.add(Dense(3, activation='tanh', input_dim=1))
model.add(Dense(3, activation='tanh'))
model.add(Dense(1, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(1, activation='linear'))
sgd = keras.optimizers.SGD(lr=0.01)
model.compile(optimizer=sgd,loss='mse')

anim = animate_model(model);
HTML(anim.to_html5_video())
```

[38]: <IPython.core.display.HTML object>

<Figure size 432x288 with 0 Axes>



This is a visualization of the results from different models attempting to fit points in a polynomial line. I start with a shallow and narrow network with one hidden layer of two nodes with hyperbolic tangent activation functions. The result here resembles spliced hyperbolic tangents. It's clear this model is lacking capacity. Using three hidden layers allows the model to find its final state faster but doesn't fit the line any better. I was expecting better performance with the added depth. Depth allows composition with the previous layer which I expected would be able to match the polynomial shape better.

Adding width was more effective. It took many runs to get there but the 1-6-1 model produces a function similar to the data. An approach with both some depth and some breadth works best.

I was also interested in the results of adding a chokepoint to the models. The 1-3-1-3-1 was results were very poor. It just looks like a hyperbolic tangent function and is probably a worse fit than the 1-2-1 model. 1-3-3-1-3-3-1 is a little better but still misses the lower x values of the function.

1.0.4 Architectures for Classifier problem

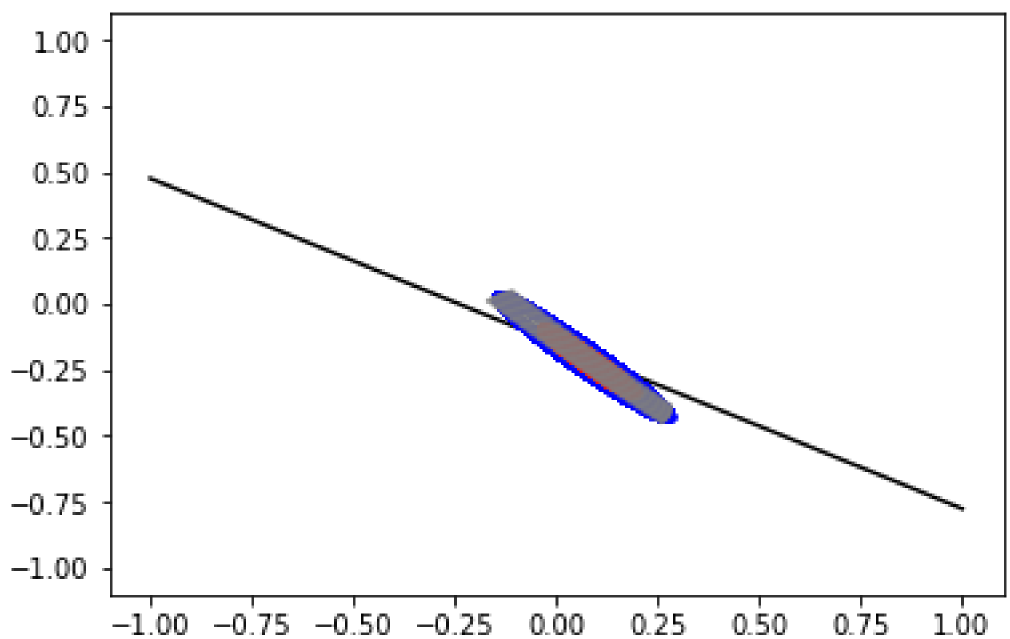
Start with only a single node in the first hidden layer.

[29]:
```
#1-1-2-1, hyptan, sgd lr 0.001
model = Sequential() #linear stack of layers
model.add(Dense(1, activation='tanh', input_dim=2)) #2 node fully connected␣
↪layer takes 2 inputs
model.add(Dense(2, activation='tanh')) #fully connected 2 node hidden layer
```

```
model.add(Dense(1, activation='sigmoid')) #fully connected single node output␣
 ↪layer
sgd = keras.optimizers.SGD(lr=0.001) #stoch gradient desc optimization
model.compile(optimizer=sgd,
              loss='mse',
              metrics=['accuracy']) #mean square error loss

anim = animate_model(model,Type='Classifier');
HTML(anim.to_html5_video())
```

[29]: <IPython.core.display.HTML object>

<Figure size 432x288 with 0 Axes>



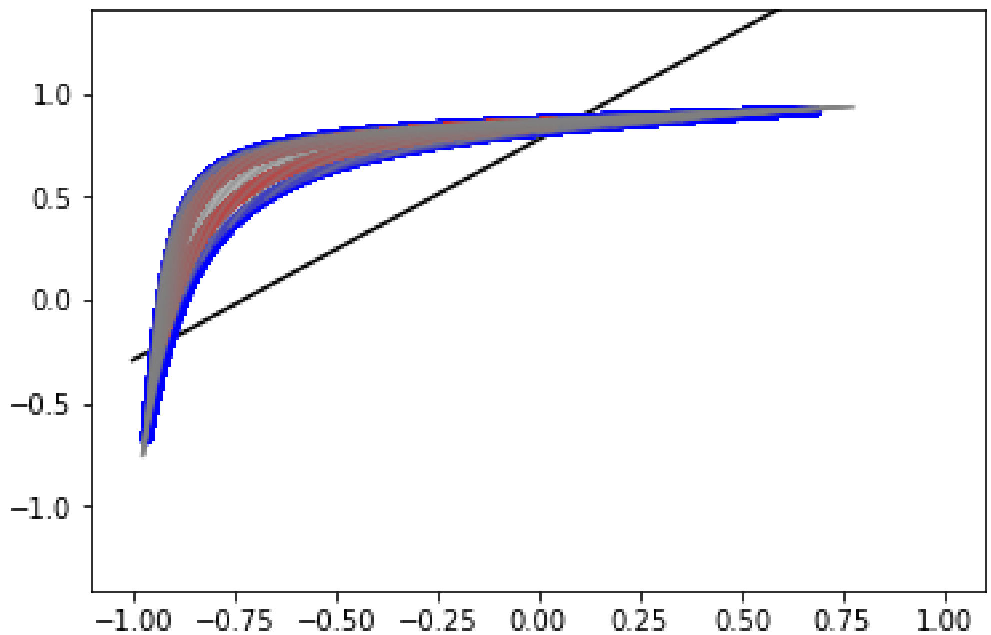
Increase to 2 nodes.

[21]:
```
#1-2-2-1, hyptan, sgd lr 0.001
model = Sequential() #linear stack of layers
model.add(Dense(2, activation='tanh', input_dim=2)) #2 node fully connected␣
 ↪layer takes 2 inputs
model.add(Dense(2, activation='tanh')) #fully connected 2 node hidden layer
model.add(Dense(1, activation='sigmoid')) #fully connected single node output␣
 ↪layer
sgd = keras.optimizers.SGD(lr=0.001) #stoch gradient desc optimization
model.compile(optimizer=sgd,
```

```
              loss='mse',
              metrics=['accuracy']) #mean square error loss

anim = animate_model(model,Type='Classifier');
HTML(anim.to_html5_video())
```

[21]: <IPython.core.display.HTML object>

<Figure size 432x288 with 0 Axes>



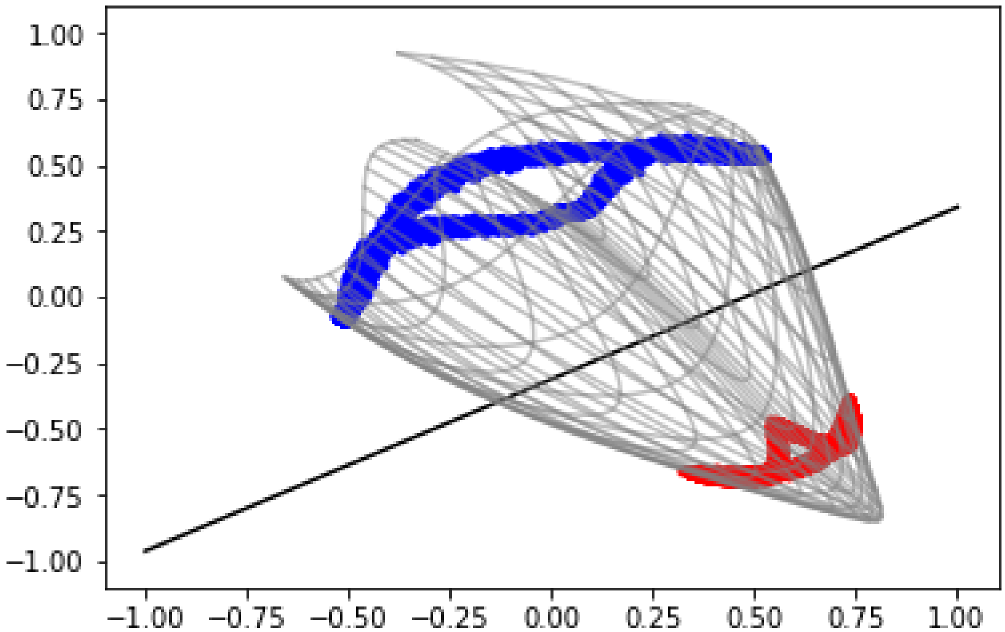
Add a third node to the first hidden layer.

[27]:
```
#1-3-2-1, hyptan, sgd lr 0.001
model = Sequential() #linear stack of layers
model.add(Dense(3, activation='tanh', input_dim=2)) #3 node fully connected␣
 ↪layer takes 2 inputs
model.add(Dense(2, activation='tanh')) #fully connected 2 node hidden layer
model.add(Dense(1, activation='sigmoid')) #fully connected single node output␣
 ↪layer
sgd = keras.optimizers.SGD(lr=0.001) #stoch gradient desc optimization
model.compile(optimizer=sgd,
              loss='mse',
              metrics=['accuracy']) #mean square error loss

anim = animate_model(model,Type='Classifier');
```

```
HTML(anim.to_html5_video())
```

[27]: <IPython.core.display.HTML object>

<Figure size 432x288 with 0 Axes>

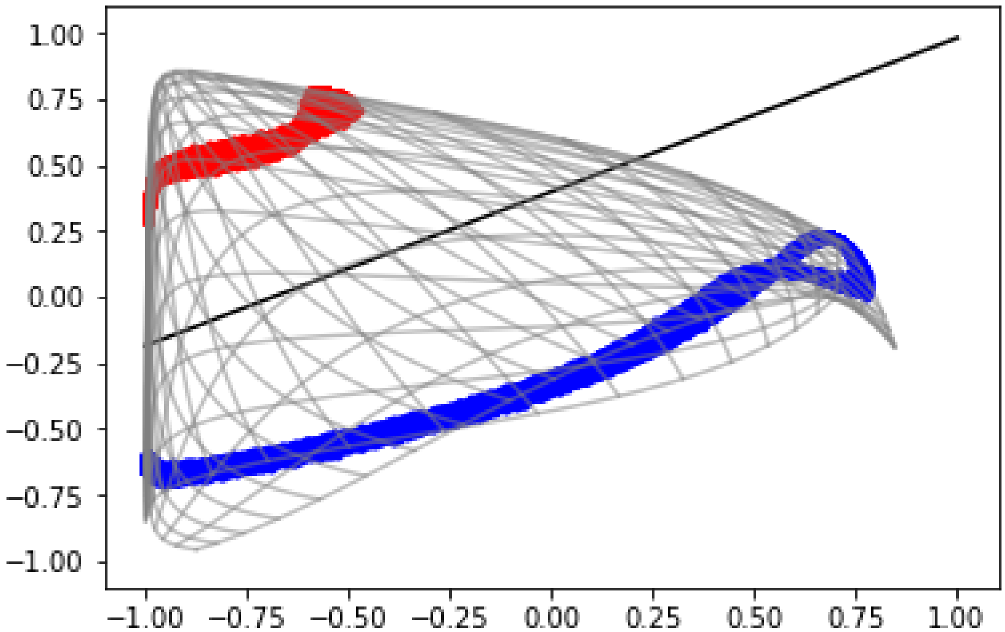


Finally try with 4 nodes.

[26]:
```
#1-4-2-1, hyptan, sgd lr 0.001
model = Sequential() #linear stack of layers
model.add(Dense(4, activation='tanh', input_dim=2)) #4 node fully connected␣
 ↪layer takes 2 inputs
model.add(Dense(2, activation='tanh')) #fully connected 2 node hidden layer
model.add(Dense(1, activation='sigmoid')) #fully connected single node output␣
 ↪layer
sgd = keras.optimizers.SGD(lr=0.001) #stoch gradient desc optimization
model.compile(optimizer=sgd,
              loss='mse',
              metrics=['accuracy']) #mean square error loss

anim = animate_model(model,Type='Classifier');
HTML(anim.to_html5_video())
```

[26]: <IPython.core.display.HTML object>

```
<Figure size 432x288 with 0 Axes>
```



The goal of this excercise is to see how space is transformed by the network for classification. I use a the 2 nodes in the layer before the sigmoid classification output as components of a cartesian plane. They start with the network's initialized values and we can see how these change over the training period as the network attempts to separate the red and blue points.

With only a single node between the input and output, the network cannot accomplish very much. With two nodes, we can see the space stretched but it's impossible to separate concentric circles in two dimensions and this model can't fully separate the red and blue points. Broadening the network to three nodes in the middle gives it the ability to separate the red and blue rings.

1.0.5 Conclusion

There are so many possibile neural network architectures. Being able to experiment with visualizations from simple examples helps to gain a better understanding of what happens inside the models.