Idea behind Neural Networks

1. Obtain some structured data (always a good idea 😊).
2. Use some subset of that data as training
3. Feed each training example through the network
   1. Calculate the error for each training example
   2. Update the weights for each neuron to minimize the error using Gradient Descent (Back Propagation)
   3. Feed in the data again until you reach the desired % error or trials run out
   4. If you reached % error or trials stop and go to the next training input
      1. Else (Back Propagation)
An example

What is this data?

Each synapse has a weight.

Each hidden layer neuron has an activation function.
Forward Propagation

1. Assign random weights to the synapses
2. Feed in the training data
3. Calculate the hidden layers neurons from the inputs and the weights using an activation function
4. Calculate the output from the hidden layer neurons and the output weights
5. Calculate the error from what is expected
Activation Function

\[ \phi(z) = \frac{1}{1 + e^{-z}} \]
Activation Functions

- **Sigmoid**

- **Hyperbolic tangent**

- **ReLU**

- **Leaky ReLU**
Forward propagation

weights are random initially → sigmoid

\[ E = \frac{1}{2}(z_e - z_a)^2 = \frac{1}{2}(z_e - S(r_3))^2 = \frac{1}{2}(0 - S(h_0 \cdot w_4 + h_1 \cdot w_5))^2 \]

\[ r_0 = 0.4 \cdot 1 + 0.3 \cdot 1 = 0.7 \]
\[ h_0 = \frac{1}{1 + e^{-0.7}} = 0.67 \]
\[ r_1 = 0.6 \cdot 1 + 0.8 \cdot 1 = 1.4 \]
\[ h_1 = \frac{1}{1 + e^{-1.4}} = 0.8 \]
\[ r_3 = 0.67 \cdot 1 + 0.8 \cdot 0.7 = 0.83 \]
\[ z_e = 0.65 \]
\[ z_a = 0 \]
\[ E = \frac{1}{2}(0 - 0.65)^2 = 0.21 \]
Back propagation

We need to adjust the weights to minimize the error

\[
E = \left( \frac{Z_e - Z_a}{2} \right)^2 = \left( Z_e - S(r_3) \right)^2 = \left( 0 - S(h_0 \cdot w_4 + h_1 \cdot w_5) \right)^2
\]

\[
E = \left( \frac{-S(h_0 \cdot w_4 + h_1 \cdot w_5)}{2} \right)^2, \text{ so how do we adjust } w_4? \frac{\partial E}{\partial w_4}
\]

\[
\frac{\partial E}{\partial w_4} = \frac{\partial E}{\partial S} \cdot \frac{\partial S}{\partial r_3} \cdot \frac{\partial r_3}{\partial w_4} \frac{\partial E}{\partial S} = -S(r_3), \frac{\partial S}{\partial r_3} = S(r_3) \cdot (1 - S(r_3))
\]

\[
\frac{\partial r_3}{\partial w_4} = h_0 \quad \frac{\partial E}{\partial w_4} = -S(r_3) \cdot S(r_3) \cdot (1 - S(r_3)) \cdot h_0 = -h_0
\]
Back propagation

Where do these errors come from?

How do we adjust the weights coming into the hidden layer?

new weights from final error
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense

# the four different states of the XOR gate
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")

# the four expected results in the same order
target_data = np.array([[0],[1],[1],[0]], "float32")

#use sequential vs functional since we’re feedforward
model = Sequential()
#Dense is used for single input data 0,1,1,0 for each neuron
model.add(Dense(16, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

#this is the building of the net
model.compile(loss='mean_squared_error',
              optimizer='adam',
              metrics=['binary_accuracy'])

#now optimize
model.fit(training_data, target_data, epochs=500, verbose=0)

print(model.predict(training_data).round())
Derivative of the sigmoid

Let's denote the sigmoid function as $\sigma(x) = \frac{1}{1 + e^{-x}}$.

The derivative of the sigmoid is $\frac{d}{dx} \sigma(x) = \sigma(x) (1 - \sigma(x))$.

Here's a detailed derivation:

$$\frac{d}{dx} \sigma(x) = \frac{d}{dx} \left[ \frac{1}{1 + e^{-x}} \right]$$

$$= -\frac{d}{dx} \left[ (1 + e^{-x})^{-1} \right]$$

$$= -\frac{1}{(1 + e^{-x})^2} (-e^{-x})$$

$$= \frac{e^{-x}}{(1 + e^{-x})^2}$$

$$= \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}}$$

$$= \frac{1}{1 + e^{-x}} \cdot \frac{1 - e^{-x}}{1 + e^{-x}}$$

$$= \sigma(x) \cdot (1 - \sigma(x))$$