CS682 Computer Vision
Quiz 7 Write Up

Mithilaesh Jayakumar
G01206238

1 Neural Network Properties

1. Prove that if we have a linear activation function, then the number of hidden layers has no effect on the actual (fully-connected) neural network.

If a linear activation function is used, the derivative of the cost function is a constant with respect to input, so the value of input (to neurons) does not affect the updating of weights. This means that we can not figure out which weights are most effective in creating a good result and therefore we are forced to change all weights equally. In general, weights are updated by the formula *W_new = W_old - Learn_rate * D_loss*. This means that the new weight is equal to the old weight minus the derivative of the cost function. If the activation function is a linear function, then its derivative w.r.t input is a constant, and the input values have no direct effect on the weight update. For example, we intend to update the weights of last layer neurons using backpropagation. We need to calculate the gradient of the weight function w.r.t weight. With chain rule we have,

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial h}\frac{\partial h}{\partial x}\frac{\partial x}{\partial w} = (h - y)grad(F)h^2$$

h and y are (estimated) neuron output and actual output value, respectively. And x is the input of neurons. grad (f) is derived from the input w.r.t activation function. The value calculated above (by a factor) is subtracted from the current weight and a new weight is obtained. We can now compare these two types of activation functions more clearly. If the activating function is a linear function, such as F(x) = 2 * x then,

$$\frac{\partial J}{\partial w} = (h - y) * 1 * h^2$$

the new weight will be,

$$w_{new} = w_{old} - \alpha * (h - y) * 1 * h^2$$

As you can see, all the weights are updated equally and it does not matter what the input value is. But if we use a non-linear activation function like Tanh(x) then,

$$\frac{\partial J}{\partial w} = (h - y) * (1 - tanh(x)^2) * h^2$$

becomes,

$$w_{new} = w_{old} - \alpha * (h - y) * (1 - tanh(x)^2) * h^2$$

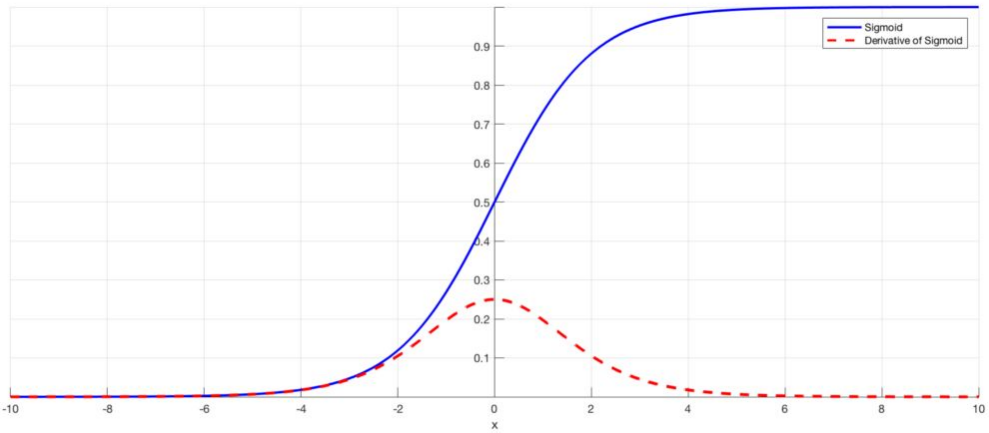
and now we can see the direct effect of input in updating weights. Different input values make different weights changes.

2. Is this property (independence of the number of hidden layers) also true of convolutional neural networks with a linear activation function but that include max pooling operations? Explain in 1 or 2 sentences why or why not.

No this property is not true for convolutional neural networks with a linear activation function but that include max pooling operations. When max pooling is combined with the linear activation it becomes ReLU. This function is partly linear and nonlinear and allows the network to converge very quickly. Although it looks like a linear function, ReLU has a derivative function and allows for backpropagation. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.This means that the neurons will only be deactivated if the output of the linear transformation is less than 0.

4.How does the gradient of the sigmoid activation function behave as the absolute value of x increases? Can you think of any problems this behavior may create for the gradient descent algorithm, when the sigmoid is used as the activation function for many layers?

As more layers using sigmoid activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train. The sigmoid function, squishes a large input space into a small input space between 0 and 1. Therefore, a large change in the input of the sigmoid function will cause a small change in the output. Hence, the derivative becomes small. Consider the example below. When the inputs of the sigmoid function becomes larger or smaller (when |x| becomes bigger), the derivative becomes close to zero.



For a shallow network with only a few layers that use these activations, this isn't a big problem. However, when n hidden layers use an activation like the sigmoid function, n small derivatives are multiplied together. Thus, the gradient decreases exponentially as we propagate down to the initial layers. A small gradient means that the weights and biases of the initial layers will not be updated effectively with each training session. Since these initial layers are often crucial to recognizing the core elements of the input data, it can lead to overall inaccuracy of the whole network.

3. Derive the gradient of the sigmoid function with respect to x and show that it can be written as a function of sigmoid function itself.

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

Take derivative with respect to Input x,

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

Using the quotient rule,

$$\frac{d}{dx}\sigma(x) = \frac{(1+e^{-x})(0) - (1)(-e^{-x})}{(1+e^{-x})^2}$$

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

Add and subtract 1 in numerator,

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

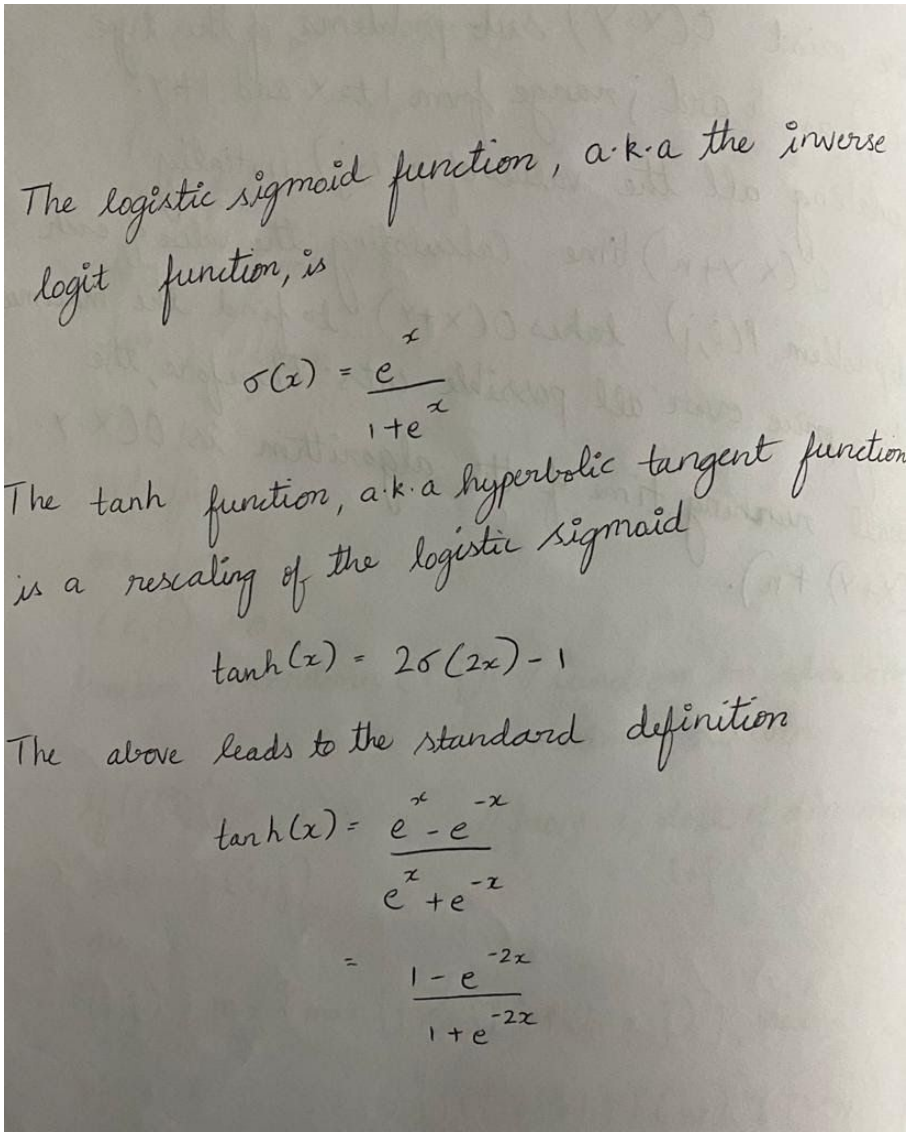
$$= \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2}$$

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

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

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x))$$

5. Often, the sigmoid function is replaced with the hyperbolic tangent function. Show that this function and its gradient can be written in terms of sigma. (Note: your answer should not contain any exponentials.)



The logistic sigmoid function, a.k.a the inverse logit function, is

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

The tanh function, a.k.a hyperbolic tangent function is a rescaling of the logistic sigmoid

$$\tanh(x) = 2\sigma(2x) - 1$$

The above leads to the standard definition

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

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

6. What are the output ranges of the sigmoid and the hyperbolic tangent functions? When would we prefer to use each of these functions?

The Sigmoid Function curve looks like a S-shape. The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice. The function is differentiable. That means, we can find the slope of the sigmoid curve at any two points. The function is monotonic but the function's derivative is not. The logistic sigmoid function can cause a neural network to get stuck at the training time. The softmax function is a more generalized logistic activation function which is used for multiclass classification.

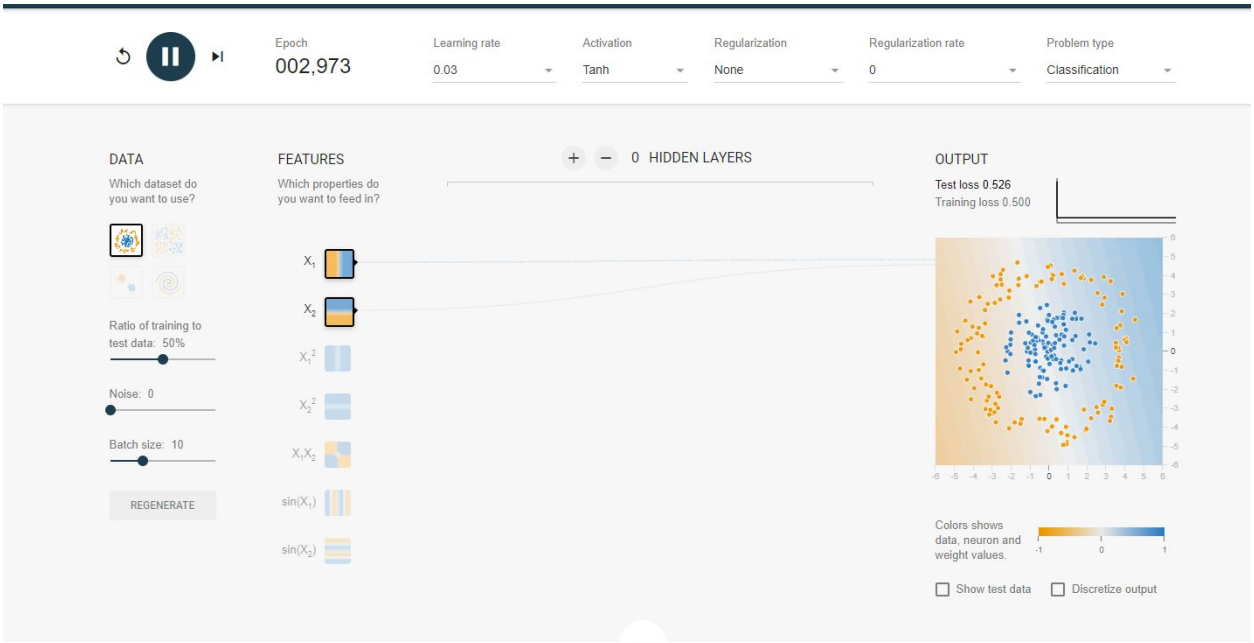
tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped). The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph. The function is differentiable. The function is monotonic while its derivative is not monotonic.The tanh function is mainly used for classification between two classes.

Both tanh and logistic sigmoid activation functions are used in feed-forward nets.

2 Neural Net (TensorFlow) Playground

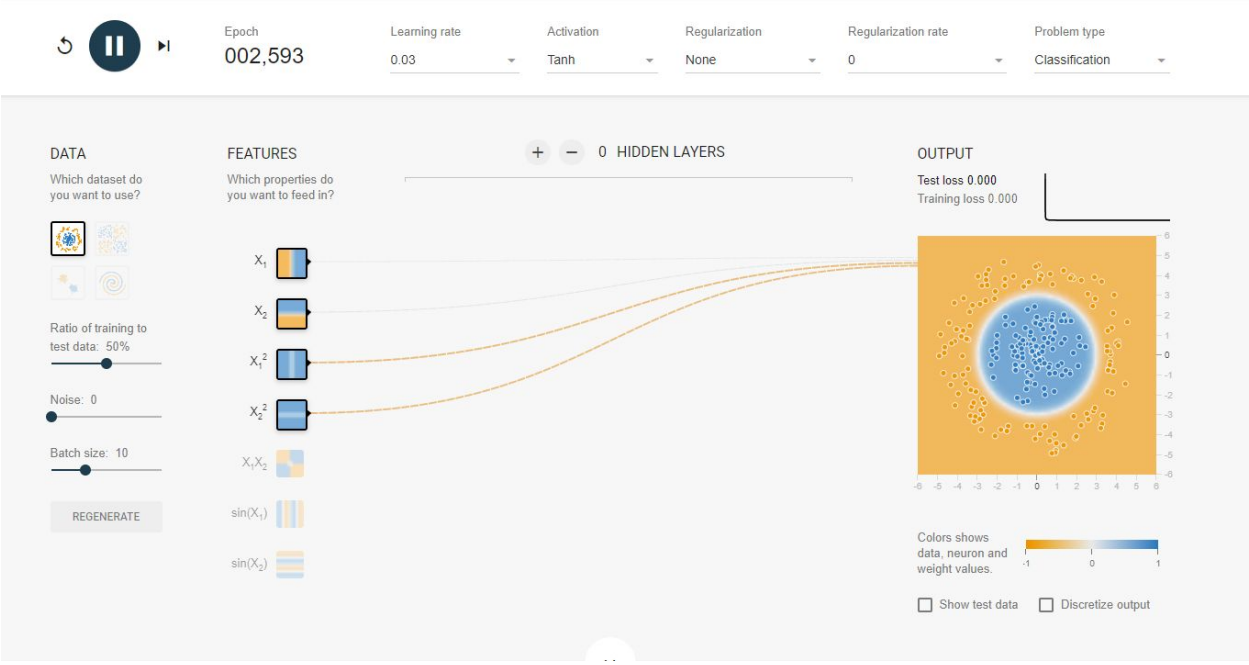
1. To begin, we will try training on the same data (called Circle in the GUI element on the top left), but with no hidden layers, which means that this problem is a simple (linear) logistic regression objective. Use the '-' button at the top to remove all the hidden layers and try retraining. How is the performance on the "circular" dataset?

The model does not perform well on the circular dataset.



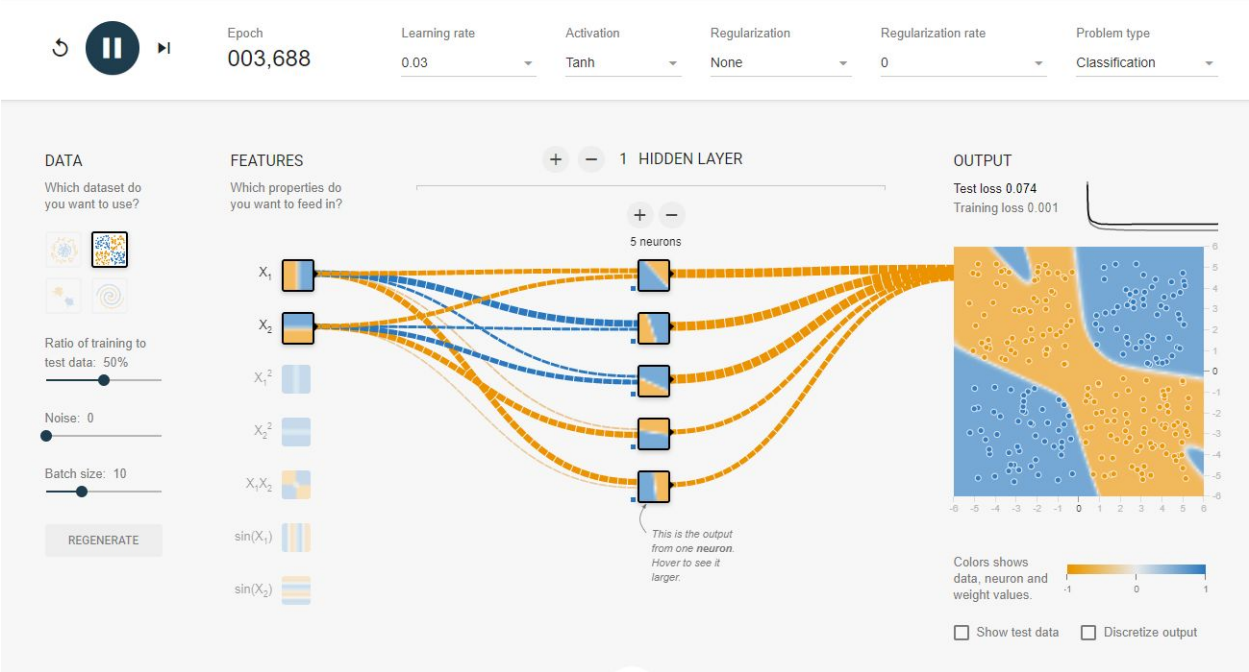
2. To make this training work, we will need to do some feature engineering, a nonlinear preprocessing step that allows for nonlinear functions of the data to contribute to the learning performance. Try enabling the next two features (by clicking on them): x^2_1 and x^2_2 . Retrain and comment on the performance: how does the system do? Include a screenshot of your Playground. (Your screenshot should contain the entire window, like Figure 1, and not just the output).

The model performs very well in classifying the data.



3. Now, let's change the data and see what we can do with a few hidden layers. Switch to using the Exclusive Or data (the one with + and - examples at alternating corners) and limit yourself to using only the two linear features: x_1 and x_2. Add a few hidden layers and play around with the settings, including the number of neurons. Can you get nearly perfect performance on the test data? Include a screenshot of your final configuration, showing the performance of your trained model.

Yes we are able to obtain nearly perfect performance on the test data by using the below configuration.



4. Finally, switch to using the spiral data and try to recover good (perfect is not required) performance. Feel free to tune any of the parameters you'd like (including enabling the non-linear features). Include a screenshot of your trained system.