Notes on Data Science and Physics

Dowling Wong

June 2023

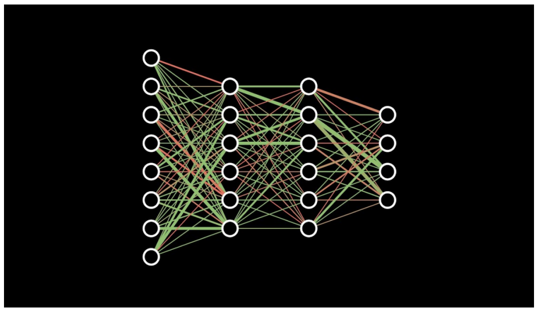
1 Prelude

This document has been written as Dowling's learning notes for 2023 summer research and **MIT Data Science and Physics**. The course **MIT Data Science and Physics** has been taught by Phil in 2023 spring.

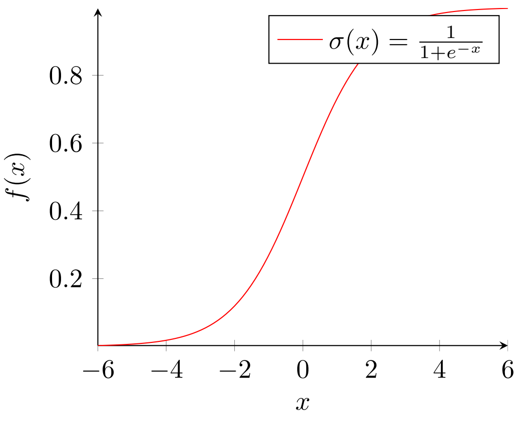
2 Concepts

2.1 Neural network

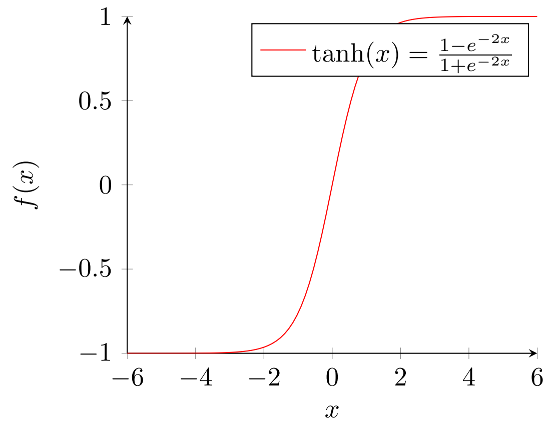
The neural network is used to recognize certain patterns. For example, consider a picture with a resolution 128x128=16384 pixels, after grey scale picture, each pixel has a grey scale value between 0 and 1(pure black=1, pure white=0), so the sample layer has a sample of 16384 floats, named neuron means a unit holds a value. Between result and sample layers are the hidden layers, stored some specific geometric patterns in the picture. To move from sample to hidden layers, we need a connection that gives weight neurons of the previous layer, and add them up to produce the value for new layer of neurons, the weighted sum could be at any place on the axis.



For each neuron in the new layer, we want them to be normalized between 0 and 1, to achieve this, we introduce the sigmoid function, a common function for activation with a value between 0 and 1 is usually the Sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$



Similar to the Sigmoid, tanh is also a function for activation.



$$\omega_n = \Sigma\sigma(x)(a_i x_i - 10)$$

So the activation here is actually how positive the number in the new neuron cell is. To active meaningfully, we need to set a bias for inactivity, let's say the bias=10. The bias tells how high the neuron need to have, to be active. In the new layer, each neuron has a bias. To continue, assume we have 2 hidden layers of 16 neurons, and the final result has 10 neurons for 10 numbers from 0 to 9. By calculating, then we have total of 16384x16+16x16+16x10=262,560 weights and 16+16+10=36 bias. So the term learning simply just means finding the gigantic number of right weights and biases. A point need to mention here, all works and layers here is to get the neuron with largest number, the most most possible result.

Denote the numbers in neurons as $a_i^{(j)}$, in which (j) denotes layer number and i represents the number of that neuron in the (j) th layer.

$$a_0^{(1)} = \sigma(\omega_{0,0}a_0^{(0)} + \omega_{0,1}a_1^{(0)} + ... + \omega_{0,n}a_n^{(0)})$$

This is actually a vector equation:

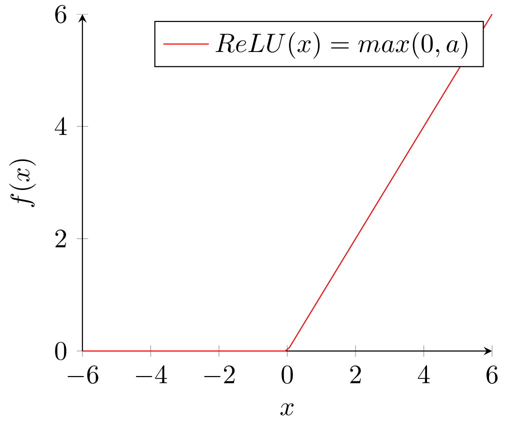
$$a^{(1)} = \sigma(\begin{bmatrix} \omega_{0,0} & \omega_{0,1} & ... & \omega_{0,n} \\ \omega_{1,0} & \omega_{1,1} & ... & ... \\ \vdots & \vdots & \vdots & \vdots \\ ... & ... & ... & \omega_{n,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}) = \sigma(Wa^{(0)} - b)$$

As corresponding code:

```python
class Network(object):
    def __init__(self, *args, **kwargs):
        #this code is written in Python, here has initial weights and biases

    def feedforward(self, a):
        for b, w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a) + b)
        return a
```

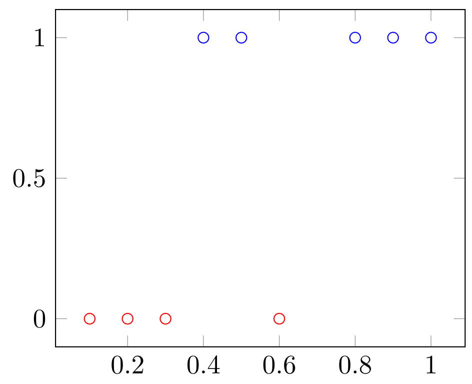
One last thing to mention for the neural network is the activation function of Sigmoid is considered old school nowadays, possibly another activation function might be more widely used: ReLU(x)=max(0, a). Easier for computation and rectified linear units. Split at threshold x=0, left is inactive, and right is active.



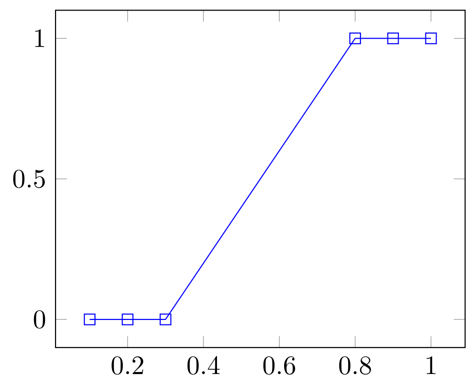
2.2 ROC and AUC

Receiver Operating Characteristic (ROC) metric to evaluate the quality of multiclass classifiers.
ROC curves typically feature true positive rate (TPR) on the Y axis, and false positive rate (FPR) on the X axis. This means that the top left corner of the plot is the "ideal" point - a FPR of zero, and a TPR of one. This is not very realistic, but it does mean that a larger area under the curve (AUC) is usually better. The "steepness" of ROC curves is also important, since it is ideal to maximize the TPR while minimizing the FPR.

1. **Confusion matrix** A confusion matrix shows which predicted classes are often confused for the other classes. The vertical axis (y) represents the true labels and the horizontal axis (x) represents the predicted labels. When the true label and predicted label are the same, the highest values occur down the diagonal extending from the upper left to the lower right. The other values, outside the diagonal, represent incorrect predictions. For example, in the confusion matrix below, the value in row 2, column 1 shows how often the predicted value A occurred when it should have been B. Suppose we have a set of data, that has been defined into binary categories. Plot the 2D graph of category again data, we have a plot like this:



Then doing logistic regression, the y-axis read as the probability of being recognized as 0 or 1.



2. **True Positive**: Your test (neural network) correctly identified that the samples are positive. It usually writes as TPR, true positive rate.
3. **False Positive**F: Your test (neural network) indicated that the samples are positive; however, it is not.

2.3 Classifiers

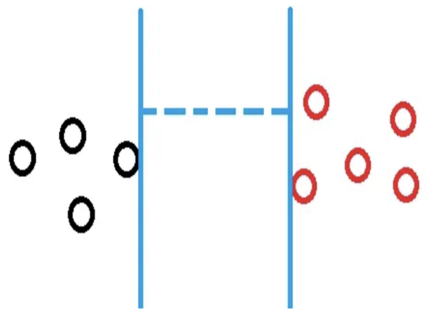
2.3.1 Logistic Regression

Logistic Regression (aka logit, MaxEnt) Classifier. In the multi-class case, the training algorithm uses the one-vs-rest (OvR) scheme if the '$multi_class$' option is set to 'ovr', and uses the cross-entropy loss if the '$multi_class$' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

2.3.2 SVC

C-Support Vector Classification. Support Vector Machine (SVM) is a supervised learning algorithm that can be used both for classification or regression. It works for both linear and non-linear calculations of the boundaries, therefore being useful for a handful of problems. A Support Vector Machine is an algorithm that classifies the data points in two categories. Once it is done for all the points, the algorithm starts to trace some lines at the edge of the separation between the two classes, with the objective of maximizing the distance between them. The place where it finds the largest distance is where the best separation line is.
For linear separated datasets, the algorithm works very well. For the data not linear, sklearn also has implementations as
1. Linear

```
SVC(kernel='linear')
```



The linear kernel is pretty straightforward. The SVM will create the lines just like the figure previously presented.

2. Polynomial

```
SVC(kernel='poly', degree=3, coef0=1, C=5)
```

The poly option is for a polynomial kernel. If you look at the shapes of polynomials, you will see that as we increase the degree of it, the more the line creates new curves and becomes more irregular. So, for a model underfitting, it might be a good idea to increase the polynomial degree, making the decision boundary to go around more points. The C is the regularization hyperparameter and the coef0 balances how the model is influenced by high or low degrees of polynomials.

3. RBF

```
SVC(kernel='rbf', gamma=4, C=100)
```

This kernel rbf is for Gaussian Radial Basis Function. It creates Gaussian distributions to calculate in which one the points will be better fit in to determine how the points will be classified. The hyperparameter gamma makes the Gaussian curves more narrow (high gamma values, more bias) or wide (low gamma values, smoother boundary). So, if we increase gamma, our decision boundary is more irregular. If your model is underfitting, try increasing that number. If it's overfitting, reduce the gamma. C is the regularization number. It works on similar fashion as the gamma argument.

4. Sigmoid

```
SVC(kernel='sigmoid', gamma=2)
```

The kernel sigmoid uses a logic similar to the Logistic Regression, where the probabilities up to 50% go to one class and over that number, it goes to the opposite class. You can use the gamma hyperparameter to regularize. **5. Precomputed**
this last kernel is for a more advanced/ customized case, where you create your own kernel to run with the model.

Appendix A Code