

Noize net — WORK IN PROGRESS

Liam Watson

October 14, 2021

1 Abstract

Complete abstract at the end

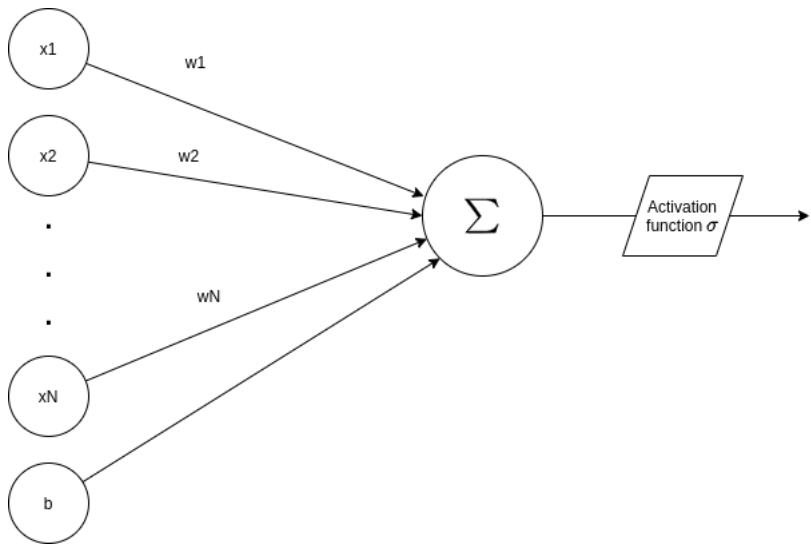
2 Intorduction to Neural networks

Before we procede to the more advanced recurant neural network let us begin with an abbreviated coveradge of neaural networks and the convepts underpinning them.

2.1 Perceptrons

The simplest unit of a neural network is, as the name suggests a neuron or more commonly a peceptron. In it's simplest form a peceptron recieves an input, performs some calculation and produces output. Given these inputs and

Figure 1: Peceptron showing N input variables, N weights and a bias

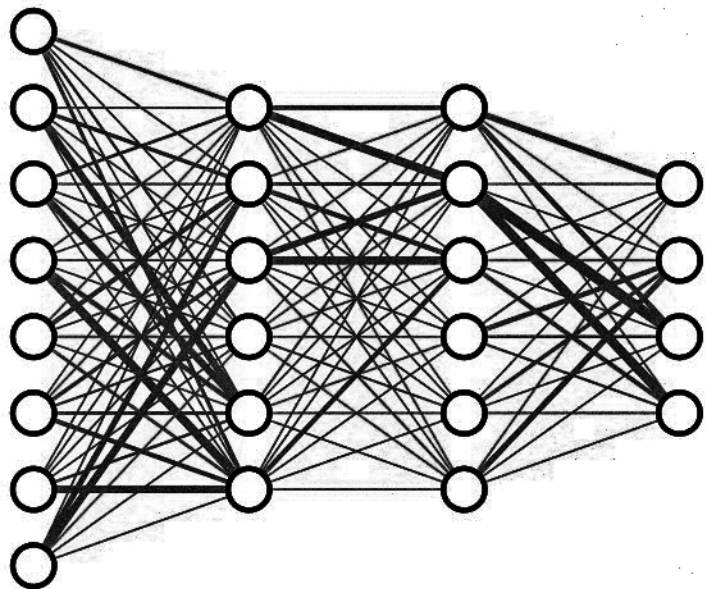


bias we can adjust weights and bais to satisfy our desired output after summation and activation function (2.4). More rigorously: Given some input $\{x_i\} \forall i \in \mathbb{Z}^+$ predict some $y = \sigma(W_i x_i)$ where σ is some activation function. NOTE PECEPTRON VS ARTIFICIAL NEURON

2.2 Multi layer peceptrons

Peceptrons, however, are not powerful in this form, rather we only begin to see the utility when we start connecting them together in a mesh much like neurons in the brain. In figure two we can see a depiction of this with weights represented as line thickness (Figure 2).

Figure 2: Image of a simple neural network archutecture with 8 inputs two hidden layers and four output neurons



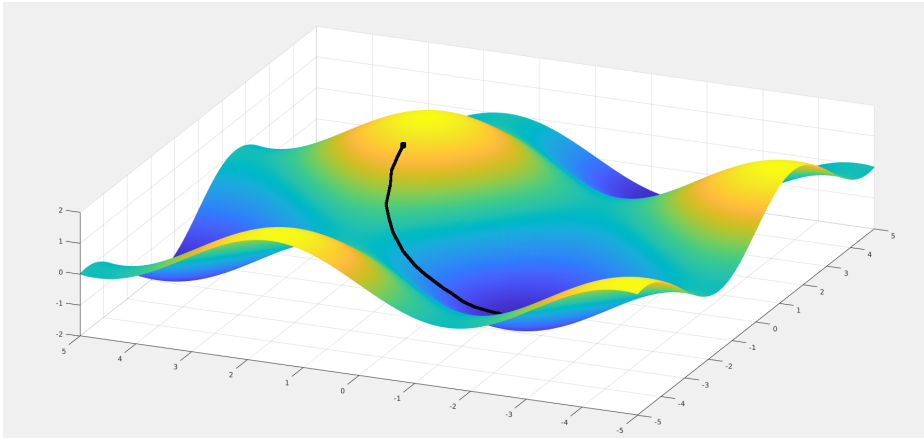
Sorce

2.3 Gradient decent

Now we need a way to determine the peceptron weights, for this we use Gradient decent. There are many adaptions of gardient descent that aim to optimize its computational performance or over come some issue with converging to a poory optimised solution such as Fast gradient methods or momentum adapted gradient descent.

The aim of gradient descent is to iteratively optimize the peceptron weights to converge on a local minimum by taking steps in the direction of steepest descent, after many itterations we will find that the networks weights are well optimised for some goal. However we may find that a local minimum is not sufficient for our purposes and as such may need to employ some hyperparameter tuning such as changing the the step size we take or adding momentum in the hopes that we converge to a more optimal solution.

Figure 3: A plot of $f(x, y) = \sin(x) + \sin(y)$ with a path showing gradient decent steps



2.4 Activation functions

Now we will discuss, much like with a biological neuron, how will a neuron decide to acivate or not. There are many functions that are used in the literature but here we give a quick overview of the most common functions and their uses. The purpose of an activation function is to format the output of a peceptron, we begin with the most elementary of these functions (excluding the identity function defined as $f(x) = x$).

1. The binary step function Definition:

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

The binary step function is primarily used for true, false classification where the result is not a probability but a certainty. Beyond this this activation function has limited use in modern neural networks, however it should be noted that it is very computationally efficient.

2. Rectified Linear Unit(ReLU) Definition:

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$

The ReLU Function finds much use despite its simplicity mostly due to its computational efficiency when compared to the more complex activation functions. ReLU reduces the input domain to only non-negative numbers which can be useful in cases where one wishes to disregard such values.

3. Sigmoid Definition:

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$

4. Softmax Definition: CHANGE THIS

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$

2.5 Feed forward

NB: NEED TO WRITE MUCH MORE HERE During the feed forward stage of a network we will recieve input's at the input neurons and travel through all the network layer calculating sum's and activations until the algorithm reaches the output layer.

2.6 Error functions

Once we have output from the model, we need a metric for the error between the output and ground truth, an error function. There are many error functions that appear in the litterature, however, their use is often highly application dependent. In the case of Noize net we are dealing with simple two dimensional time series data and as such the relevant error functions are elementary:

1. Mean Square Error (MSE)

This error function finds the average square difference between the predicted value and ground truth, defined as

$$MSE = \frac{\sum_{i=0}^N (y_i - y'_i)^2}{N}$$

Where N is the number of output values, y_i is the ground truth value and y'_i is the predicted value.

This loss function is favorable because of its simplicity and computational efficiency. One should note that MSE can "amplify" large errors and squash small errors due to the square and notice that the direction of the error is also ignored.

2. Mean absolute error

If one would not like to square the error in order to better capture small errors one can use the MAE function which shares many similar properties with the MSE function but more accurately depicts the difference between a prediction and the ground truth.

$$MAE = \frac{\sum_{i=0}^N |y_i - y'_i|}{N}$$

3. Mean Bias error

If the application requires a signed error function the MBE error function could be applicable. However, one should note that positive and negative values may cancel each other out leading to unpredictable results in practice.

$$MBE = \frac{\sum_{i=0}^N (y_i - y'_i)}{N}$$

2.7 Back propagation

Back propagation is the learning step for a model. Once we have completed the feed forward step and calculated the error we need to travel back through the network and adjust the weights and biases in order to optimize the model. For this we will use gradient descent. NB: We need to add the maths here

3 Introduction to Recurrent Neural Networks

When using neural networks for time series data prediction, some semblance of memory is required for successive predictions. Unfortunately standard multi-layer perceptron and convolutional neural networks tend to lose this information quickly as they train due to the vanishing gradient problem. RNNs seek to resolve this by constructing hand crafted compositions of so called "gates" that can incorporate prior information for successive predictions.

3.1 RNN concepts

3.2 LSTM

4 Noise net

4.1 Data and preprocessing

The data used for training, validation and prediction is the free music ?

4.2 Architecture

4.3 Implementation

4.4 Results

4.5 Conclusion

References

@inproceedingsfma challenge, title = Learning to Recognize Musical Genre from Audio, subtitle = Challenge Overview, author = Defferrard, Michaël and Mohanty, Sharada P. and Carroll, Sean F. and Salathé, Marcel, book-title = The 2018 Web Conference Companion, year = 2018, publisher = ACM Press, isbn = 9781450356404, doi = 10.1145/3184558.3192310, archiveprefix = arXiv, eprint = 1803.05337, url = <https://arxiv.org/abs/1803.05337>,