INFO 7375 - Neural Networks & AI

Homework to Chapter - 6

Submitted By:

Abdul Haseeb Khan
NUID: 002844724
khan.abdulh@northeastern.edu

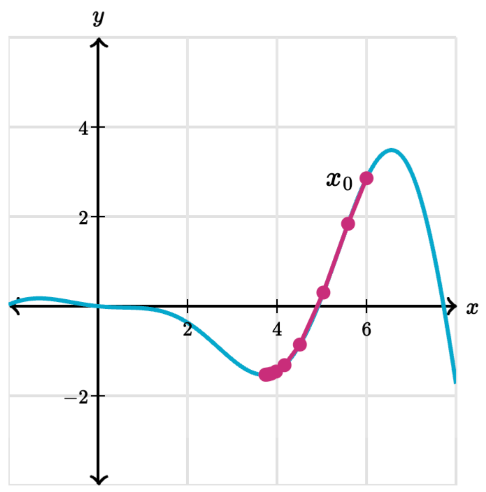
Why are multilayer (deep) neural networks needed?

Multilayer neural networks are needed because neural networks need multiple layers in order to learn complex representations of data, with each layer extracting increasingly abstract features that capture intricate patterns and relationships in the input. To simulate a deep network with a single hidden layer it needs to be exponentially large, which makes depth a practical necessity.

What is the structure of weight matrix (how many rows and columns)?

The Weight Matrix is denoted by $W^{[s]}$ where 's' is the receiving layer. The size of the matrix is $N_s \times N_{s-1}$, where N_s is the number of neurons in the receiving layer, and N_{s-1} is the number of neurons in the sending layer.

Describe the gradient descent method.



Gradient descent is a robust and widely used optimization technique in which the objective function is minimized by iteratively adjusting the optimization parameters in proportion to the gradients, or derivatives, of the objective function with respect to those parameters until the optimization criteria are met. For gradient descent to be effective, the objective function should be analytically differentiable with respect to the optimization parameters. In the context of neural networks and logistic regression, the loss function, which measures the difference between the calculated output and the target output, is minimized by varying the weights and bias. The process involves calculating the gradients of the loss function with respect to the weights and bias, and then updating these parameters in the direction that reduces the loss. This iterative process continues until the loss function reaches a minimum, ideally the global minimum, although the objective function may have multiple local minima that can prevent the process from finding the actual minimum.

Describe in detail forward propagation and backpropagation for deep neural networks

Forward propagation computes layer-by-layer linear combinations and activations from the input up to the output, while backpropagation computes gradients layer-by-layer from the output back to the input to update weights and biases using the learning rate r.

Forward Propogation

Forward propagation is the process where data flows from the input through all the layers to produce an output. The network consists of an input layer, multiple hidden layers numbered from [1] to [L-1], and an output layer [L].
During forward propagation, information travels sequentially through each layer. Starting with the input data X, each layer receives activations from the previous layer, applies weights and biases to these inputs, and then passes the result through an activation function to produce outputs for the next layer. This continues until the final output layer produces the network's prediction A.
This illustrate this with a visual showing how an image of a face gets processed through multiple layers, where early layers detect simple features like edges and lines, middle layers detect more complex patterns, and later layers recognize complete faces. This hierarchical feature extraction is fundamental to how deep networks learn.

Input X is not a neuron layer, but for the recursion process convenience it may be referred to as layer , and the first hidden layer uses $Z^{[1]} = W^{[1]}X + E_{N_1} b^{[1]} E_M^T$ followed by $A^{[1]} = f^{[1]}(Z^{[1]})$

For subsequent hidden layers s from 2 to L-1, the formula becomes $Z^{[s]} = W^{[s]}A^{[s-1]} + E_{N_s} b^{[s]} E_M^T$, with $A^{[s]} = f^{[s]}(Z^{[s]})$. The output layer computation is $Z^{[L]} = W^{[L]}A^{[L-1]} + b^{[L]} E_M^T$, resulting in the final output $A^{[L]} = f^{[L]}(Z^{[L]})$

The matrices involved include W^[s] as the weight matrix of size Ns x Ns-1, where Ns is the number of neurons in layer s and Ns-1 is the number of neurons in the previous layer

Backpropagation

Backpropagation is the training mechanism that adjusts the network's parameters (weights W and biases b) to minimize the difference between predicted outputs and target outputs Y.

The process works by calculating how much each parameter contributed to the overall error and then adjusting parameters accordingly.
The key insight is that errors propagate backward from the output layer through the network. Starting at the output layer where we can directly compare predictions to targets, the algorithm determines how wrong the prediction was. It then traces this error backward through each layer, determining how much each neuron in the previous layers contributed to that error.

The Training Process

Training involves repeatedly alternating between forward and backward passes. The forward pass computes predictions given current parameters, while the backward pass computes how to adjust parameters to improve predictions. Parameters are updated using a learning rate r that controls how much to adjust them in each iteration. This continues until a stopping criteria is met.