deeplearning.ai

One hidden layer Neural Network

Gradient descent for neural networks

Gradient descent for neural networks

Params $\rightarrow$ $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$

$W^{[1]} \rightarrow (n^{[1]}, n^{[2]})$, $b^{[1]} \rightarrow (n^{[1]}, 1)$, $W^{[2]} \rightarrow (n^{[2]}, n^{[1]})$, $b^{[2]} \rightarrow (n^{[2]}, 1)$

for 1 Hidden layer

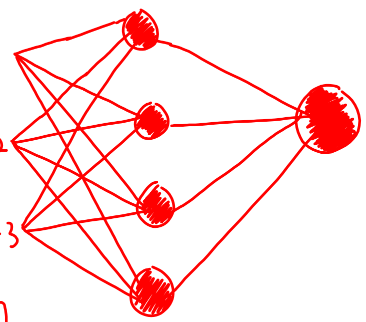
If $n_x = n^{[0]} \rightarrow$ # nodes at 0^{th} layer

$n^{[1]} \rightarrow$ # nodes in 1^{st} hidden layer

$n^{[2]} \rightarrow$ # nodes in O/p layer, ie, 1 in our example

Cost func: $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}, y)$

$\hat{y} \rightarrow a^{[2]}$



$n^{[0]} = 3$

$n^{[1]} = 4$

$n^{[2]} = 1$

Gradient Descent

Repeat {

compute prediction $\hat{y}^{(i)}$ for $i \rightarrow 1$ to m

$dW^{[1]} = \frac{\partial J}{\partial W^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$, ... $W^{[2]}$, $b^{[2]}$

$W^{[1]} = W^{[1]} - \alpha \cdot dW^{[1]}$

$W^{[2]} = W^{[2]} - \alpha \cdot dW^{[2]}$

$b^{[1]}$, $b^{[2]}$

}

Andrew Ng

Formulas for computing derivatives

Fwd prop:

$Z^{[1]} = W^{[1]}X + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

$Z^{[2]} = W^{[2]} \cdot A^{[1]} + b^{[2]}$

$A^{[2]} = g^{[2]}(Z^{[2]})$

Assumption - we do bin classification for logistic Regression on final layer $\therefore dZ^{[2]} = A^{[2]} - Y$

Back prop

$Y = [y^{(1)}, y^{(2)}, \dots y^{(m)}]$

$dZ^{[2]} = A^{[2]} - Y$

$dW^{[2]} = \frac{1}{m} dZ^{[2]} \cdot A^{[1]T}$

$db^{[2]} = \frac{1}{m}$ np.sum($dZ^{[2]}$, axis = 1, keepdims = True)

↳ horizontal

↳ removes the "Rank Array"

$dZ^{[1]} = W^{[2]T} \cdot dZ^{[2]} * \underbrace{g^{[1]'}}(Z^{[1]})$

↳ derivative of Activation func (g')

element wise product

dim Analysis

$(n^{[1]}, m) * (n^{[1]}, m)$

$dW^{[1]} = \frac{1}{m} dZ^{[1]} \cdot X^T$

$db^{[1]} = \frac{1}{m}$ np.sum($dZ^{[1]}$, axis = 1, keepdims = True)

↳ $db^{[1]}$ is a $(n^{[1]}, 1)$ vector, so you want to retain that shape & not convert it to a Rank Array