Shallow Neural Network

Sidharth Baskaran

July 2021

Overview

- Superscript $[l]$ refers to l-th layer
- Right-to-left propagation allows for computing derivative at each step

Neural Network Representation

- Hidden layers are in between output and input layers
  - True values not observed, only I/O
- Notation → $a^{[0]} = x$, $a^{[1]}$ represents the activation unit **vectors** of dimension being number of nodes in layer
  - Also contain $w^{[l]}$ and $b^{[l]}$
- Counting layers → **input** layer not counted, so indexed by 0

Computing Neural Network Output

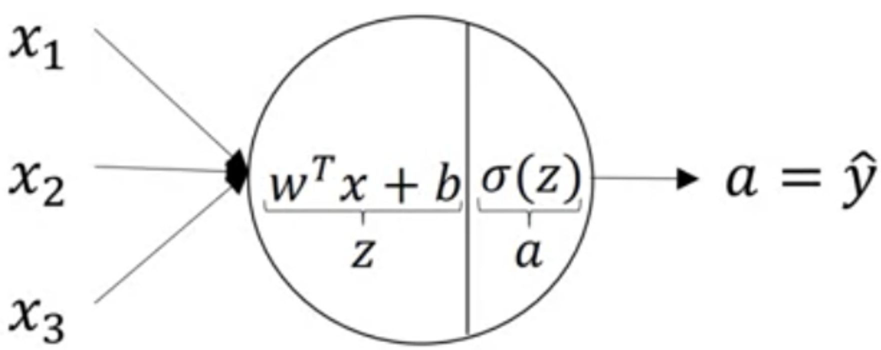

Figure 1: Neural net node

- Single node takes in all elements of a feature
- Notation → $a_i^{(l)}$ is the ith node of the lth layer
- Want to vectorize $z_i^{[l]} = w_i^{[l]T} x + b_i^{[l]}, a_i^{[l]} = \sigma(z_i^{[l]})$
- $w_i^{[l]T}$ is a row-vector, so results in a matrix when vectorized, with the row vectors stacked. Then just multiply by x, i.e. vector

$$z^{[l]} = W^{[l]} x^{[l]} + b^{[l]}$$

- Then, $a^{[l]} = \sigma(z^{[l]})$

Vectorizing across multiple examples

$$x^{(i)} \to a^{[l](i)} = \hat{y}^{(i)}$$

For $i \in [1, m]$, this accounts for m training examples.

$$\begin{aligned} &\text{for } i = 1 \text{ to } m: \\ &z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]} \\ &a^{[1](i)} = \sigma\left(z^{[1](i)}\right) \\ &z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]} \\ &a^{[2](i)} = \sigma\left(z^{[2](i)}\right) \end{aligned}$$

Vectorize:

$$Z^{[l]} = W^{[l]}X + b^{[l]} A^{[l]} = \sigma(Z^{[l]})$$

Closer look:

$$z^{[l]} = \left[z^{[l](1)} \; z^{[l]2)} \; \dots \; z^{[l](m)}\right]$$

Activation Functions

- $\tanh(z)$ is better than $\sigma(z)$ and has range $[-1, 1]$
  - Formula $\to a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
  - Represent of $g^{[l]}(z^{[l]})$

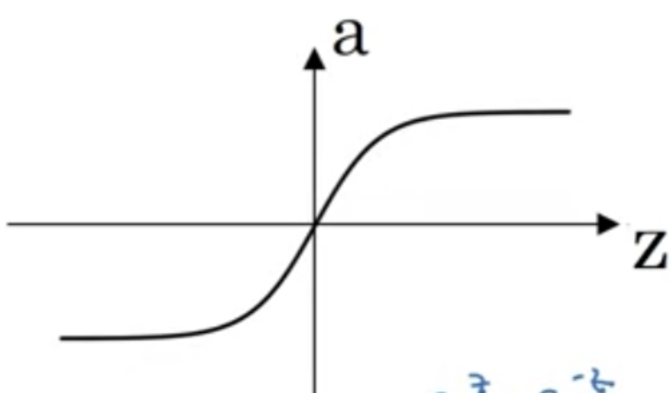

Figure 2: tanh

- Formula for ReLU function is $a = \max(0, z) \implies \frac{d}{dz}a = \max(0, 1)$, is a good default choice
  - Is faster than tanh and σ
    - Less effect of slope approaching 0, since it is rectified
- Only use sigmoid for binary, always use tanh, but ReLU is most commonly used, or the leaky ReLU ($a = \max(0.001z, z)$)

Reason for nonlinear activation functions

- Say $g(z) = z$, so an identity activation function
- A linear function of input is the output, which results in no point for hidden layers
  - The output layer can use a linear function to give output $\in \mathbb{R}$

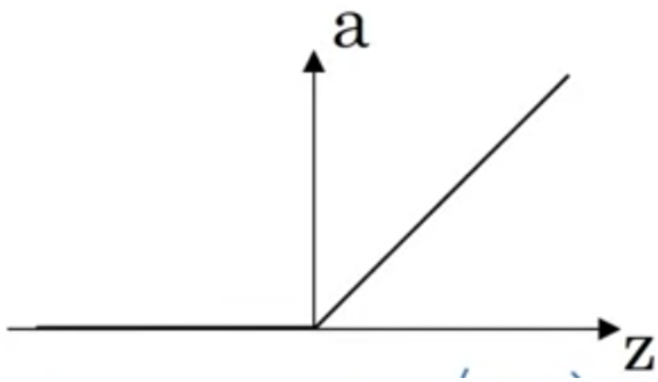

Figure 3: ReLU

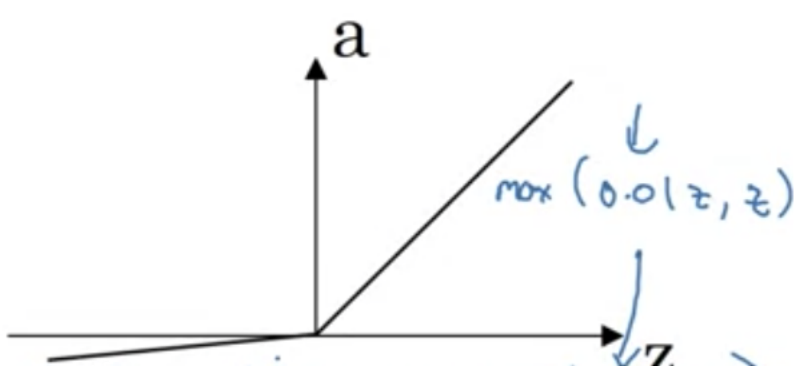

Figure 4: Leakly ReLU

Derivatives of Activation Functions

- Sigmoid function
  - $g'(z) = \frac{d}{dz}\frac{1}{1+e^{-z}}(1 - \frac{1}{1+e^{-z}}) = g(z)(1 - g(z))$
- Tanh function
  - $g'(z) = \frac{d}{dz}g(z) = 1 - (\tanh(z))^2$
- ReLU function
  - $g(z) = \max(0, z)$
  - $g'(z) = \begin{cases} 0, z < 0 \\ 1, z \geq 0 \\ \text{DNE}, z = 0 \end{cases}$
- Leaky ReLU
  - $g(z) = \max(0.01z, z)$
  - $g'(z) = \begin{cases} 0.01, z < 0 \\ 1, z > 0 \\ \text{DNE}, z = 0 \end{cases}$

Gradient Descent for Neural Networks

- Parameters - $W^{[1]}, (n^{[1]}, n^{[0]})$, $b^{[1]}, (n^{[1]}, 1)$, and then $W^{[2]}, b^{[2]}$
  - $n_x = n^{[0]}, n^{[1]}, n^{[2]} = 1$
- Cost function if $J(\text{params}) = \frac{1}{m}\sum_{i=1}^{n}\mathcal{L}(\hat{y}, y)$
- In each iteration (for $i \in [1, m]$)
  - Compute $\hat{y}^{(i)}$ predictions
  - Compute $dW^{[i]}, db^{[i]}$
  - Then update $w^{[i]}, b^{[i]}$ with α
- Formulas for partial derivatives
  - $dZ^{[2]} = A^{[2]} - Y$, where Y is row vector of ground truth values
  - $dW^{[2]} = \frac{1}{m}dZ^{[2]}A^{[1]T}$
  - $db^{[2]} = \frac{1}{m}$np.sum($dZ^{[2]}$,axis=1,keepdims=true), prevents a rank-1 array creation
  - $dZ^{[1]} = W^{[2]T}dZ^{[2]}. * g^{[1]'}(Z^{[1]})$, where both are $(n^{[1]}, m)$
  - $dW^{[1]} = \frac{1}{m}dZ^{[1]}X^T$
  - $db^{[1]} = \frac{1}{m}$np.sum($dZ^{[1]}$,axis=1,keepdims=true)
  - Backpropagation explanation

Random Initialization

- Should not initialize weights (W) to zero
  - Hidden units will compute same function
- Example
  - $W^{[1]}$ =`np.random.randn((2,2))*0.01`
  - $b^{[1]}$ =`np.zeros((2,1))`
- Large values in W to start makes activation function saturated to high values, slows down learning