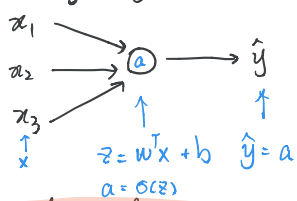
We have seen how to compute one layer of NN in lesson 1.2. Here we will show how to compute a shallow NN with more layers, but before that let's review the structure again.

One layer of neural network structure (logistic Regression)

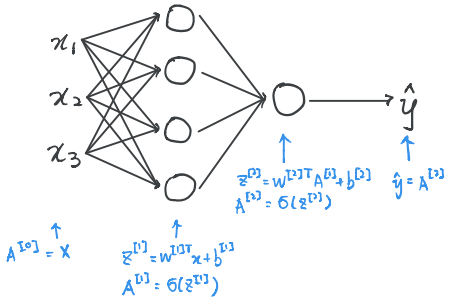


w, b: $z = w^T x + b \rightarrow a = \sigma(z) \rightarrow J(a, y)$

update $w := w - \alpha dw$ $b := b - \alpha db$

matrix size:
- X: $3 \times m$
- w: 3×1
- b: scalar
- z, a, y: $1 \times m$

Two layers of neural network structure (Shallow NN)



$w^{[1]}, w^{[2]}, b^{[1]}, b^{[2]}$: $z^{[1]} = w^{[1]T} x + b^{[1]} \rightarrow a^{[1]} = \sigma(z^{[1]}) \rightarrow z^{[2]} = w^{[2]T} a^{[1]} + b^{[2]} \rightarrow a^{[2]} = \sigma(z^{[2]}) \rightarrow J(a^{[2]}, y)$

update $w^{[1]} := w^{[1]} - \alpha dw^{[1]}$ $b^{[1]} := b^{[1]} - \alpha db^{[1]}$

$w^{[2]} := w^{[2]} - \alpha dw^{[2]}$ $b^{[2]} := b^{[2]} - \alpha db^{[2]}$

* can be 4×3 or 1×4 depending on how $z^{[1]}$ and $z^{[2]}$ are defined. Here $z^{[1]} = w^{[1]T} x + b^{[1]}$ and $z^{[2]} = w^{[2]T} a^{[1]} + b^{[2]}$.

matrix size:
- X: $3 \times m$
- $w^{[1]}$: 3×4 *
- $b^{[1]}$: 4×1
- $z^{[1]}, a^{[1]}$: $4 \times m$ (first layer)

- $w^{[2]}$: 4×1 *
- $b^{[2]}$: scalar
- $z^{[2]}, a^{[2]}, \hat{y}$: $1 \times m$ (output layer)

So to vectorize these, we have:

$$w^{[1]} = \begin{bmatrix} | & | & | & | \\ w_1^{[1]} & w_2^{[1]} & w_3^{[1]} & w_4^{[1]} \\ | & | & | & | \end{bmatrix}_{3 \times 4}$$

[superscript] associates with which layer

subscript associates with which node in the layer.

e.g. $w_1^{[1]}$ represents the weight parameters for the first node of the first layer, so it has to be 3×1 to multiply x for each node ($z_1^{[1]} \dots z_4^{[1]}$). So $w^{[1]}$ is 3×4 since there are 4 nodes in the next layer.

recall (superscript) associates with which example

$$b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}_{4 \times 1}$$

So

$$z^{[1]} = w^{[1]T} x + b^{[1]} = \begin{bmatrix} — & w_1^{[1]T} & — \\ & \vdots & \\ — & w_4^{[1]T} & — \end{bmatrix}_{4 \times 3} \begin{bmatrix} | & & | \\ x^{(1)} & \dots & x^{(m)} \\ | & & | \end{bmatrix}_{3 \times m} + \begin{bmatrix} b_1^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}_{4 \times 1} = \begin{bmatrix} w_1^{[1]} x^{(1)} + b_1^{[1]} & \dots & w_1^{[1]} x^{(m)} + b_1^{[1]} \\ \vdots & & \vdots \\ w_4^{[1]} x^{(1)} + b_4^{[1]} & \dots & w_4^{[1]} x^{(m)} + b_4^{[1]} \end{bmatrix}_{4 \times m}$$

$$= \begin{bmatrix} | & & | \\ z^{[1](1)} & \dots & z^{[1](m)} \\ | & & | \end{bmatrix}_{4 \times m} = \begin{bmatrix} — & z_1^{[1]} & — \\ & \vdots & \\ — & z_4^{[1]} & — \end{bmatrix}_{4 \times m}$$

↔ different examples ↕ different nodes

$$A^{[1]} = \begin{bmatrix} | & & | \\ a^{[1](1)} & \dots & a^{[1](m)} \\ | & & | \end{bmatrix} = \begin{bmatrix} — & a_1^{[1]} & — \\ & \vdots & \\ — & a_4^{[1]} & — \end{bmatrix}_{4 \times m} = g^{[1]}(z^{[1]})_{4 \times m}$$

By convention, we use $g^{[i]}$ to denote activation function for the ith layer.

for the second layer:

$$w^{[2]} = \begin{bmatrix} w_1^{[2]} \\ \vdots \\ w_4^{[2]} \end{bmatrix}_{4 \times 1}$$

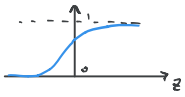
$$z^{[2]} = w^{[2]T} A^{[1]} + b^{[2]} = \begin{bmatrix} w_1^{[2]} & \dots & w_4^{[2]} \end{bmatrix}_{1 \times 4} \begin{bmatrix} | & & | \\ a^{[1](1)} & \dots & a^{[1](m)} \\ | & & | \end{bmatrix}_{4 \times m} + b^{[2]} = \begin{bmatrix} — & w^{[2]} a^{[1]} + b^{[2]} & — \end{bmatrix}_{1 \times m}$$

$$A^{[2]} = g^{[2]}(z^{[2]})_{1 \times m} = \hat{Y} = \begin{bmatrix} \hat{y}^{(1)} & \dots & \hat{y}^{(m)} \end{bmatrix}_{1 \times m}$$

Choosing Activation functions

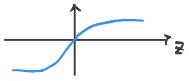
So far we've only used sigmoid function σ, but almost always there are other better choices for activation functions. The following will compare different activation functions.

Sigmoid $\frac{1}{1+e^{-z}}$



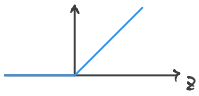
- almost never use, except for output layer for binary classification.
- slow learning rate as the slope is close to 0.

Tanh $\frac{e^z - e^{-z}}{e^z + e^{-z}}$



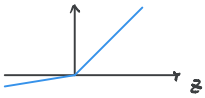
- almost always superior to sigmoid
- has the effect of centering data to 0 as the mean is 0
- still slow learning rate for the same reason as sigmoid's

Relu $\max(0, z)$



- default activation function for most NNs
- fast learning rate as the slope $\neq 1$ (for $z>0$)
- slope $=0$ for $z \leq 0$ but in practise z rarely becomes negative.

leaky Relu $\max(0.01z, z)$ ← can be other constant



- not as oftenly used as Relu
- slope $= 0.01$ for $z \leq 0$

The reason we use non-linear functions as activation functions is because if we use linear function (or identity activation function), a NN is the same as linear regression.

Prove: suppose we have a 2-layers NN, let $a^{[1]} = z^{[1]}$, $a^{[2]} = z^{[2]}$.

Then $\hat{y} = a^{[2]} = z^{[2]} = w^{[2]T}a^{[1]} + b^{[2]} = w^{[2]T}(w^{[1]T}x + b^{[1]}) + b^{[2]} = \underbrace{w^{[2]T}w^{[1]T}}_{=w'}x + \underbrace{w^{[2]T}b^{[1]} + b^{[2]}}_{=b'}$

$= w'x + b'$

Differentiation of Activation Functions

To update w and b by gradient descent, we need to first find out $dw^{[1]}$, $dw^{[2]}$, $db^{[1]}$ and $db^{[2]}$ given our choice of activation functions and thus we need to know $g'(z)$ first.

Sigmoid $g(z) = \frac{1}{1+e^{-z}}$ $\quad g'(z) = g(z)(1-g(z))$

Relu $g(z) = \max(0, z)$ $\quad g'(z) = \begin{cases} 0 & \text{if } z<0 \\ 1 & \text{if } z \geq 0 \end{cases}$

not exactly right mathematically, but it works just fine in practise

Tanh $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ $\quad g'(z) = 1-(g(z))^2$

leaky Relu $g(z) = \max(0.01z, z)$ $\quad g'(z) = \begin{cases} 0.01 & \text{if } z<0 \\ 1 & \text{if } z \geq 0 \end{cases}$

Shallow NN

To generalise our model, suppose we have $n_x = n^{[0]}$ input features, $n^{[1]}$ hidden layer features, $n^{[2]} = 1$ output layer feature and m training examples. Then:

X: $n_x \times m$ $\quad w^{[1]}$: $n_x \times n^{[1]}$ $\quad b^{[1]}$: $n^{[1]} \times 1$ $\quad w^{[2]}$: $n^{[1]} \times n^{[2]}$ $\quad b^{[2]}$: $n^{[2]} \times 1$ $\quad z^{[1]}, A^{[1]}$: $n^{[1]} \times m$ $\quad z^{[2]}, A^{[2]}$: $n^{[2]} \times m$

First thing we need to do is to initialise w and b.

We need to initialise $w^{[1]}$ and $w^{[2]}$ randomly and initialise b to zeros, because initialising w to zero will let all hidden units compute the same activation function, but initializing b to zero will not have this problem, often known as symmetry breaking.

So:

$w^{[1]} = np.random.randn(n, n^{[1]}) \cdot 0.01$ → smaller value of w allows for smaller values of z and therefore larger slope of activation function, i.e. faster learning rate.

$b^{[1]} = np.zeros((n,1))$

$w^{[2]} = np.random.randn(n^{[1]}, 1) \cdot 0.01$

$b^{[2]} = 0$

For forward propagation:

- X : $n_x \times m$ $\quad w^{[1]}$: $n_x \times n^{[1]}$ $\quad b^{[1]}$: $n^{[1]} \times 1$ $\quad w^{[2]}$: $n^{[1]} \times 1$ $\quad b^{[2]}$: 1×1

  $z^{[1]} = np.dot(w^{[1]}.T, x) + b^{[1]} = w^{[1]T} x + b^{[1]}$ $\quad A^{[1]} = g^{[1]}(z^{[1]}) \Rightarrow$ $z^{[1]}, A^{[1]}$: $n^{[1]} \times m$

  $z^{[2]} = np.dot(w^{[2]}.T, A^{[1]}) + b^{[2]} = w^{[2]T} A^{[1]} + b^{[2]}$ $\quad A^{[2]} = g^{[2]}(z^{[2]}) \Rightarrow$ $z^{[2]}, A^{[2]}$: $1 \times m$

  $\hat{Y} = A^{[2]}$

- the cost function $J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(a^{[2]}, y) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log a^{[2](i)} + (1-y^{(i)}) \log(1 - a^{[2](i)}) \right]$ $g^{[2](i)}(1 - g^{[2](i)}(z^{[2](i)}))$ for binary classification

  $= -\frac{1}{m} \cdot \left(np.dot(Y, (\log A^{[2]}).T) + np.dot(1-Y, (\log(1-A^{[2]})).T) \right)$ or $\begin{cases} logprobs = np.multiply(np.log(A^{[2]}), Y) \\ -\frac{1}{m} np.sum(logprobs) \end{cases}$

For back propagation:

- $dz^{[2]} = A^{[2]} - Y$: $1 \times m$
- $dw^{[1]} = \frac{1}{m} A^{[1]} (dz^{[2]})^T = \frac{1}{m} np.dot(A^{[1]}, dz^{[2]}.T)$: $n^{[1]} \times 1$
- $db^{[2]} = \frac{1}{m} np.sum(dz^{[2]}, axis=1, keepdims=True)$: 1×1 (sum up row)
- $dz^{[1]} = \underbrace{np.dot(w^{[2]}, dz^{[2]})}_{n^{[1]} \times m} \cdot \underbrace{g^{[1]'}(z^{[1]})}_{n^{[1]} \times m}$: $n^{[1]} \times m$ (element-wise multiplication)
- $dw^{[1]} = \underbrace{\frac{1}{m} X}_{n_x \times m} \underbrace{(dz^{[1]})^T}_{m \times n^{[1]}} = \frac{1}{m} np.dot(x, dz^{[1]}.T)$: $n_x \times n^{[1]}$
- $db^{[1]} = \frac{1}{m} np.sum(dz^{[1]}, axis=1, keepdims=True)$: $n^{[1]} \times 1$

The following will show the prove for back propagation formula we used above:

- $dz^{[2](i)} = \frac{\partial L^{(i)}}{\partial z^{[2](i)}} = \frac{\partial L^{(i)}}{\partial a^{[2](i)}} \frac{\partial a^{[2](i)}}{\partial z^{[2](i)}} = \frac{\partial}{\partial a^{[2](i)}} - \left[y^{(i)} \log a^{[2](i)} + (1-y^{(i)}) \log(1-a^{[2](i)}) \right] g^{[2](i)'}(z^{[2](i)})$ for binary classification

  $= -\left(\frac{y^{(i)}}{a^{[2](i)}} - \frac{1-y^{(i)}}{1-a^{[2](i)}} \right) \underbrace{(a^{[2](i)}(1-a^{[2](i)}))}_{\text{from sigmoid function differentiation.}} = -y^{(i)}(1-a^{[2](i)}) + (1-y^{(i)}) a^{[2](i)}$

  $= a^{[2](i)} - y^{(i)}$ $_{1 \times 1}$

  $dz^{[2]} = \frac{\partial}{\partial z^{[2]}} \sum_{i=1}^{m} L^{(i)} = \left[\frac{\partial}{\partial z^{[2](1)}} L^{(1)} \quad \cdots \quad \frac{\partial}{\partial z^{[2](m)}} L^{(m)} \right]_{1 \times m} = \left[a^{[2](1)} - y^{(1)} \quad \cdots \quad a^{[2](m)} - y^{(m)} \right]_{1 \times m}$

  $= \underbrace{(A^{[2]} - Y)_{1 \times m}}$

- $dw^{[2](i)} = \frac{\partial L^{(i)}}{\partial w^{[2]}} = \frac{\partial L^{(i)}}{\partial z^{[2](i)}} \frac{\partial z^{[2](i)}}{\partial w^{[2]}} = dz^{[2](i)} \frac{\partial}{\partial w^{[2]}} w^{[2]T} a^{[1](i)} + b^{[2]} = dz^{[2](i)} \begin{bmatrix} \frac{\partial}{\partial w_1^{[2]}} w_1^{[2]} a_1^{[1](i)} \\ \vdots \\ \frac{\partial}{\partial w_4^{[2]}} w_4^{[2]} a_4^{[1](i)} \end{bmatrix}_{n^{[1]} \times 1} = dz^{[2](i)} a^{[1](i)}_{n^{[1]} \times 1}$

  $dw^{[2]} = \frac{\partial J}{\partial w^{[2]}} = \frac{\partial}{\partial w^{[2]}} \frac{1}{m} \sum_{i=1}^{m} L^{(i)} = \frac{1}{m} \sum_{i=1}^{m} dz^{[2](i)} a^{[1](i)}_{n^{[1]} \times 1} = \underbrace{\frac{1}{m} A^{[1]} (dz^{[2]})^T}_{n^{[1]} \times 1}$

- $db^{[2](i)} = \frac{\partial L^{(i)}}{\partial b^{[2]}} = \frac{\partial L^{(i)}}{\partial z^{[2](i)}} \frac{\partial z^{[2](i)}}{\partial b^{[2]}} = dz^{[2](i)} \frac{\partial}{\partial b^{[2]}} w^{[2]T} a^{[1](i)} + b^{[2]} = dz^{[2](i)}$

  $db^{[2]} = \frac{\partial J}{\partial b^{[2]}} = \frac{\partial}{\partial b^{[2]}} \frac{1}{m} \sum_{i=1}^{m} L^{(i)} = \frac{1}{m} \sum_{i=1}^{m} dz^{[2](i)} = \underbrace{\frac{1}{m} dz^{[2]} \text{ summed up horizontally}}$.

- $dz^{[1](i)} = \frac{\partial L^{(i)}}{\partial z^{[1](i)}} = \frac{\partial L^{(i)}}{\partial z^{[2](i)}} \frac{\partial z^{[2](i)}}{\partial a^{[1](i)}} \frac{\partial a^{[1](i)}}{\partial z^{[1](i)}} = dz^{[2](i)} \frac{\partial}{\partial a^{[1](i)}} w^{[2]T} a^{[1](i)} + b^{[2]} \frac{\partial}{\partial z^{[1](i)}} g^{[1]}(z^{[1](i)}) = dz^{[2](i)} w^{[2]} g^{[1]'}(z^{[1](i)})$

  $dz^{[1]} = \frac{\partial}{\partial z^{[1]}} \sum_{i=1}^{m} L^{(i)} = \left[\frac{\partial}{\partial z^{[1](1)}} L^{(1)} \quad \cdots \quad \frac{\partial}{\partial z^{[1](m)}} L^{(m)} \right]_{n^{[1]} \times m} = \underbrace{\left[dz^{[2](1)} \underbrace{w^{[2]}}_{1 \times n^{[1]}} \underbrace{g^{[1]'}(z^{[1](1)})}_{1 \times 1} \quad \cdots \quad dz^{[2](m)} w^{[2]} g^{[1]'}(z^{[1](m)}) \right]_{n^{[1]} \times m}}$

  $= \underbrace{w^{[2]} dz^{[2]} * g^{[1]'}(z^{[1]})}_{n^{[1]} \times m}$ (element-wise multiplication)

- $dw^{[l](i)} = \frac{\partial \mathcal{L}^{(i)}}{\partial w^{[l]}} = \frac{\partial \mathcal{L}^{(i)}}{\partial z^{[l](i)}} \frac{\partial z^{[l](i)}}{\partial w^{[l]}} = dz^{[l](i)} \frac{\partial}{\partial w^{[l]}} w^{[l]} x^{(i)} + b^{[l]} = dz^{[l](i)} \frac{\partial}{\partial \begin{bmatrix} w_{11}^{[l]} & \cdots & w_{1n^{[l-1]}}^{[l]} \\ \vdots & \ddots & \vdots \\ w_{n^{[l]}1}^{[l]} & \cdots & w_{n^{[l]}n^{[l-1]}}^{[l]} \end{bmatrix}} w^{[l]} x^{(i)} = dz^{[l](i)} \begin{bmatrix} x_1^{(i)} & \cdots & x_{n^{[l-1]}}^{(i)} \\ \vdots & & \vdots \\ x_1^{(i)} & \cdots & x_{n^{[l-1]}}^{(i)} \end{bmatrix}$

  where $w^{[l]} x^{(i)} = \begin{bmatrix} w_{11}^{[l]} x_1^{(i)} + \cdots + w_{1n}^{[l]} x_n^{(i)} \\ \vdots \\ w_{n^{[l]}1}^{[l]} x_1^{(i)} + \cdots + w_{n^{[l]}n}^{[l]} x_n^{(i)} \end{bmatrix}_{n^{[l]} \times 1}$ $\Rightarrow dw^{[l](i)} = dz^{[l](i)} \begin{bmatrix} x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} = dz^{[l](i)} x^{(i)}$

  $dw^{[l]} = \frac{\partial \mathcal{L}}{\partial w^{[l]}} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}^{(i)}}{\partial w^{[l]}} = \frac{1}{m} \sum_{i=1}^{m} dz^{[l](i)} x^{(i)} = \underline{\frac{1}{m} \times (dz^{[l]})^{T}}$

- $db^{[l](i)} = \frac{\partial \mathcal{L}^{(i)}}{\partial b^{[l]}} = \frac{\partial \mathcal{L}^{(i)}}{\partial z^{[l](i)}} \frac{\partial z^{[l](i)}}{\partial b^{[l]}} = dz^{[l](i)}$

  $db^{[l]} = \frac{\partial \mathcal{L}}{\partial b^{[l]}} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}^{(i)}}{\partial b^{[l]}} = \frac{1}{m} \sum_{i=1}^{m} dz^{[l](i)} = \underline{\frac{1}{m} dz^{[l]} \text{ summed up horizontally}}.$