Deep Learning

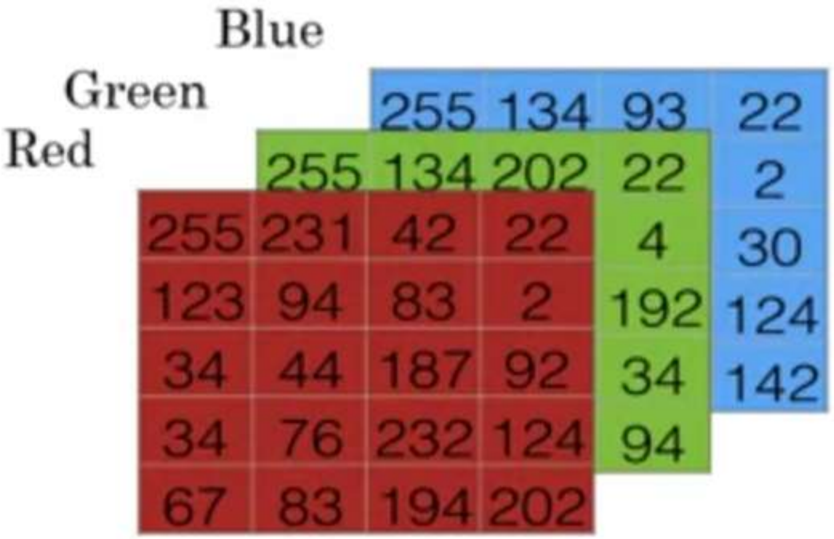
Computer Science and Engineering, IIIT Dharwad

Arun Chauhan

Binary Classification

$\longrightarrow$ 1 (cat) vs 0 (non cat)

Binary Classification



$$X = \begin{bmatrix} 255 \\ 231 \\ \cdot \\ 255 \\ 134 \\ \cdot \\ 255 \\ 134 \\ \cdot \end{bmatrix}$$

n= n_x = 64 X 64 X 3 = 12288

Binary Classification: Objective

- $f(x) = y$
- Sample: (x, y) $x \in \mathbb{R}^{n_x}$, $y \in \{0,1\}$
- m_{train} : training examples
  $\{ (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)}) \}$
- m_{test} : test examples

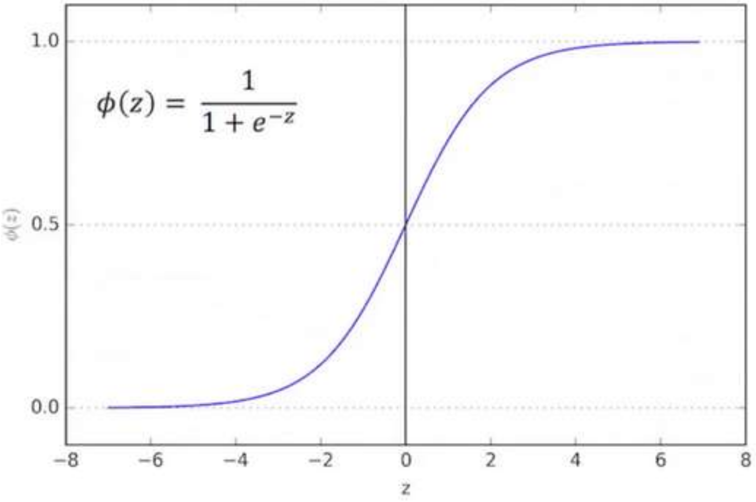
$$X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \ldots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix} \quad ; X \in \mathbb{R}^{n_x \times m}$$

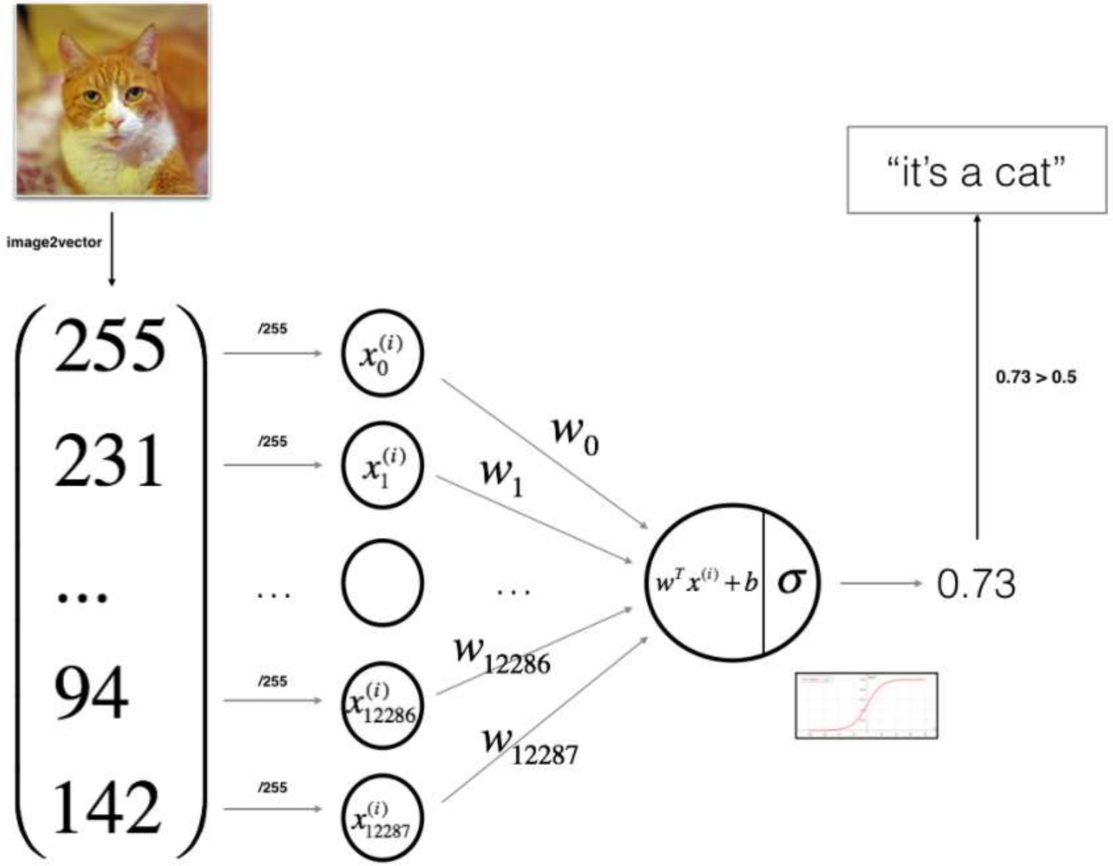
$$Y = [y^{(1)} \quad y^{(2)} \quad \ldots \quad y^{(m)}] \quad ; Y \in \mathbb{R}^{1 \times m}$$

Logistic Regression

- Given x, we want $\hat{y} = P\,(y = 1 \mid x)$; $\quad 0 \le \hat{y} \le 1$
- $x \in \mathbb{R}^{n_x}$
- Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$
- Output:

$$\hat{y} = \sigma(w^T x + b)$$





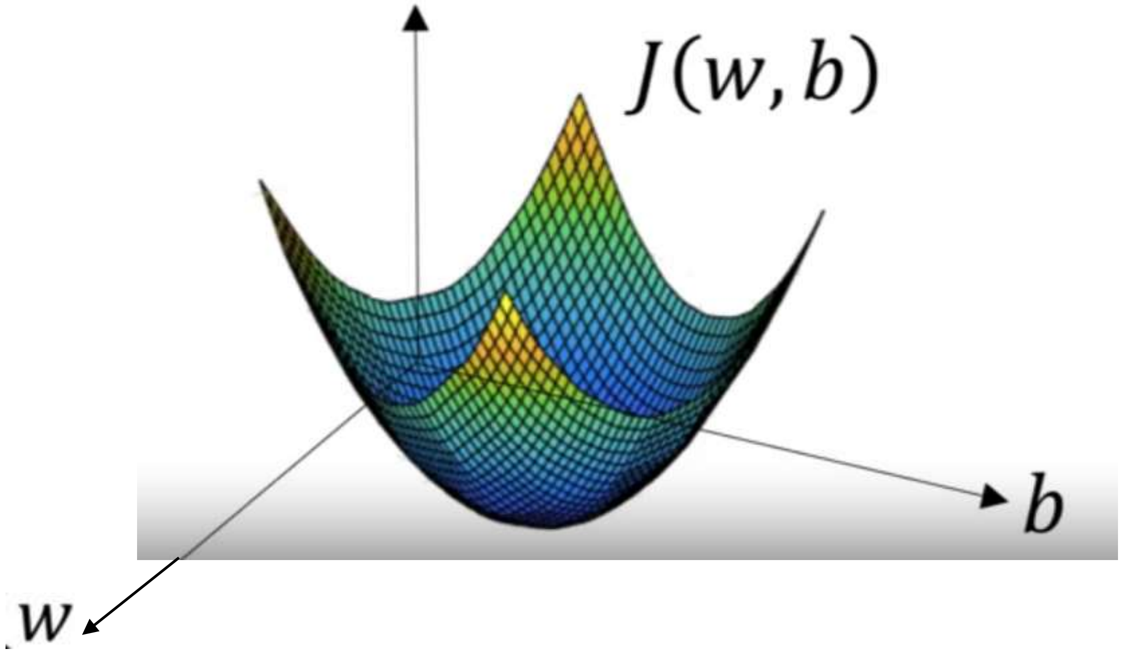
Logistic Regression: Cost Function

$$\hat{y} = \sigma(w^T x + b), \;\; \sigma(z) = \frac{1}{1+e^{-z}}$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

Gradient Decent

- . Want to find w, b that minimize $J(w, b)$



$$w_1 = w_1 - \alpha \, dw_1$$
$$w_2 = w_2 - \alpha \, dw_2$$

....

$$w_n = w_n - \alpha \, dw_n$$

Vectorizing Logistic Regression: Forward Pass

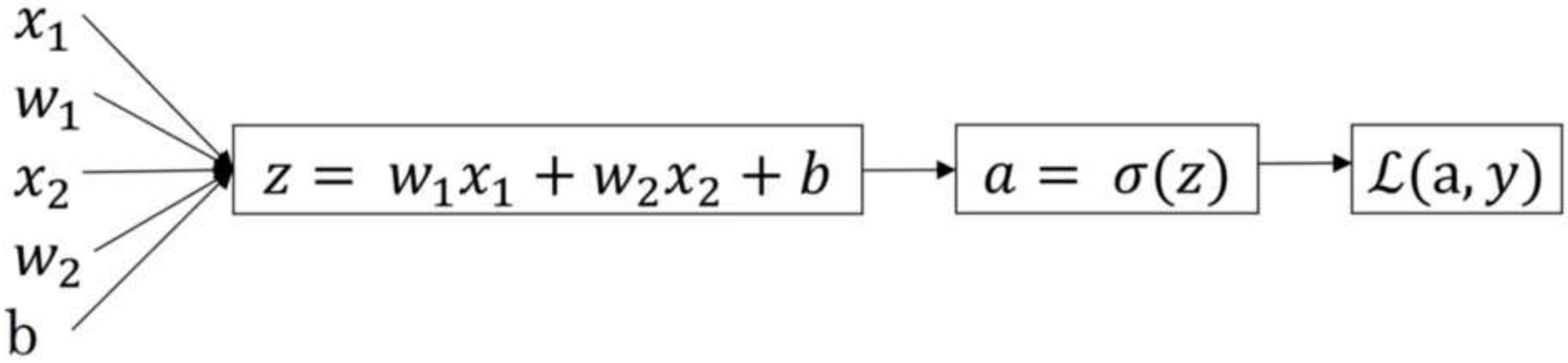
- $z^{(1)} = w^T x^{(1)} + b$ $\quad$ $z^{(2)} = w^T x^{(2)} + b$ $\quad$ $z^{(3)} = w^T x^{(3)} + b$

  $a^{(1)} = \sigma(z^{(1)})$ $\quad$ $a^{(2)} = \sigma(z^{(2)})$ $\quad$ $a^{(3)} = \sigma(z^{(3)})$

- $X \in \mathbb{R}^{n_x \times m}$

- $w^T X + [b\ b \ldots b]_{1 \times m} = [z^{(1)}\ z^{(2)} \ldots z^{(m)}] = Z$
- $A = \sigma(Z) = ???$

Computational Graph for Logistic Regression



Gradient decent for Logistic Regression

- $z = w^T x + b$

  $\hat{y} = a = \sigma(z)$

  $\mathcal{L}(a, y) = -(y \log(a) + (1-y)\log(1-a))$

$$\frac{dL}{da} = -\frac{y}{a} + \frac{1-y}{1-a} = \frac{-y + ya + a - ya}{a(1-a)} = \frac{a-y}{a(1-a)}$$

$$\frac{da}{dz} = a(1-a)$$

$$\frac{dL}{dz} = \frac{dL}{da} \text{X} \frac{da}{dz} = a - y$$

$$\frac{dz}{dw} = \boldsymbol{x}$$

$$\frac{dz}{db} = 1$$

$d\boldsymbol{w}$

$$\frac{1}{m}\left[\left(a^{(1)} - y^{(1)}\right)\boldsymbol{x}^{(1)} + \left(a^{(2)} - y^{(2)}\right)\boldsymbol{x}^{(2)} + \cdots + \left(a^{(m)} - y^{(m)}\right)\boldsymbol{x}^{(m)}\right]$$

$$\underbrace{\begin{bmatrix} | & | & & | \\ \boldsymbol{x}^{(1)} & \boldsymbol{x}^{(2)} & \cdots & \boldsymbol{x}^{(m)} \\ | & | & & | \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} a^{(1)} - y^{(1)} \\ a^{(2)} - y^{(2)} \\ \cdot \\ \cdot \\ \cdot \\ a^{(m)} - y^{(m)} \end{bmatrix}}_{\mathrm{dZ}^{\mathrm{T}}}$$

$$\frac{\mathbf{X}(\mathbf{A} - \mathbf{Y})^{\mathrm{T}}}{\mathrm{m}}$$

$$d\boldsymbol{w} = \frac{\mathbf{X}\mathrm{dZ}^{\mathrm{T}}}{m}$$

db

$$\frac{\left(a^{(1)} - y^{(1)}\right) + \left(a^{(2)} - y^{(2)}\right) + \cdots + \left(a^{(m)} - y^{(m)}\right)}{m}$$

$db = \dfrac{1}{m}$ ∗np.sum ((**A**−**Y**))

Vectorizing LR's Gradient Computation

for iter:

$$Z = w^T X + b$$
$$A = \sigma(Z)$$
$$dZ = A - Y$$
$$dw = \frac{1}{m} X\, dZ^T$$
$$db = \frac{1}{m}\, np.sum(dZ)$$
$$w = w - \alpha(dw)$$
$$b = b - \alpha(db)$$

Code and Implementation for Logistic Regression

- Each image is of size: (64, 64, 3)
- Train_set shape: (***m***, 64, 64, 3)

```
# Reshape the training and test examples
train_set_x_flatten = train_set_x_orig.reshape(train_set_x_orig.shape[0], -1).T
```

- Train_set flatten shape: (12288, ***m***)
- Train_set shape: (1, ***m***)

```
train_set_x = train_set_x_flatten / 255.
test_set_x = test_set_x_flatten / 255.
```

- w = np.zeros(shape=(dim, 1))
- b = 0

Code and Implementation for Logistic Regression

Forward Pass:

```
A = sigmoid( np.dot(w.T, X) + b)
cost = (- 1 / m) * np.sum(Y * np.log(A) + (1 - Y) * (np.log(1 - A)))
```

Backward Pass:

```
dw = (1 / m) * np.dot(X, (A - Y).T)
db = (1 / m) * np.sum(A - Y)
```

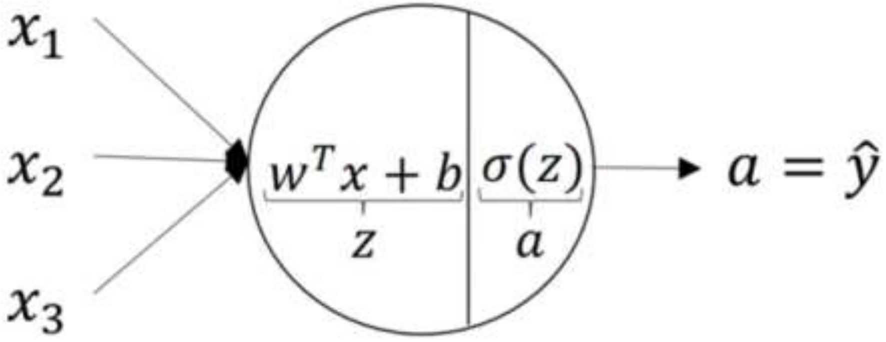
Optimize (Batch Gradient):

```
w = w - learning_rate * dw
b = b - learning_rate * db
```

Predict:

```
A = sigmoid( np.dot(w.T, X) + b)
Y_prediction = 1 if A > 0.5 else 0
```

Notations

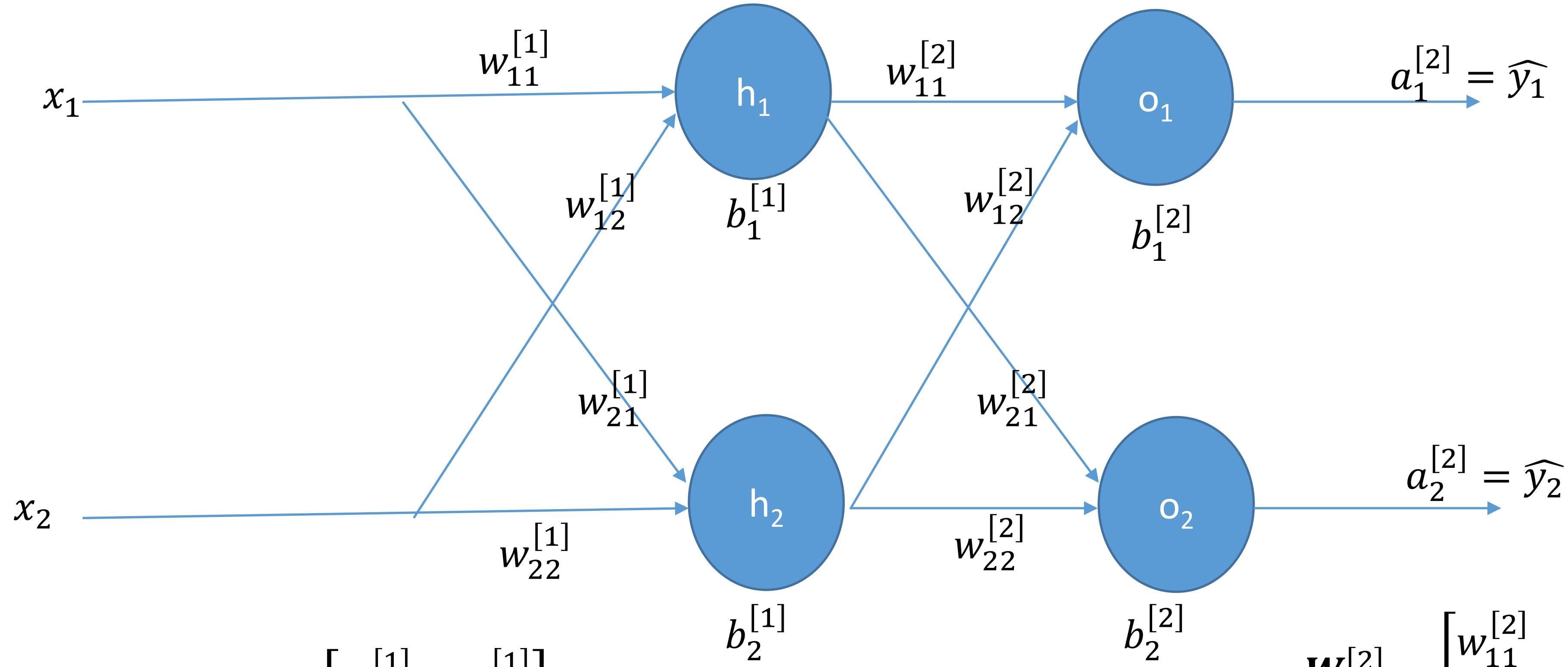


$z = w^T x + b$

$a = \sigma(z)$

Weight Matrix: ***W***

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$



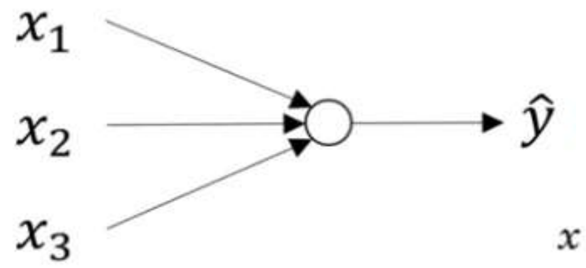
$$\boldsymbol{W}^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \end{bmatrix}$$

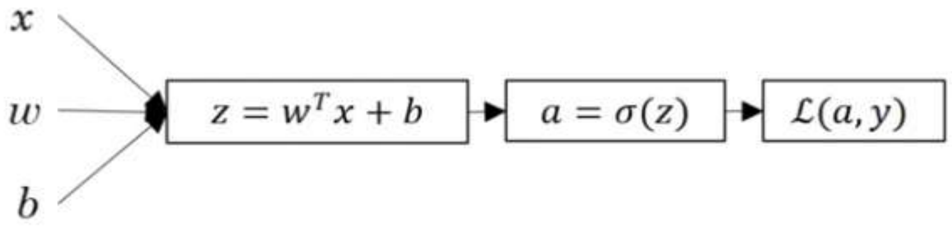
$$\boldsymbol{b}^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \end{bmatrix}$$

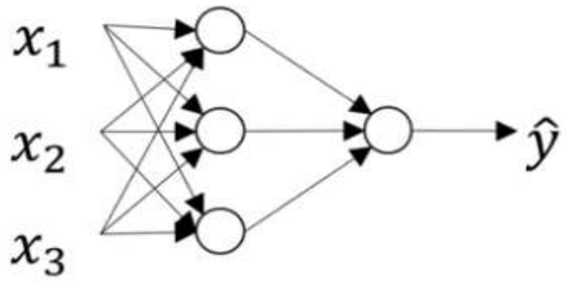
$$\boldsymbol{W}^{[2]} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} \\ w_{21}^{[2]} & w_{22}^{[2]} \end{bmatrix}$$

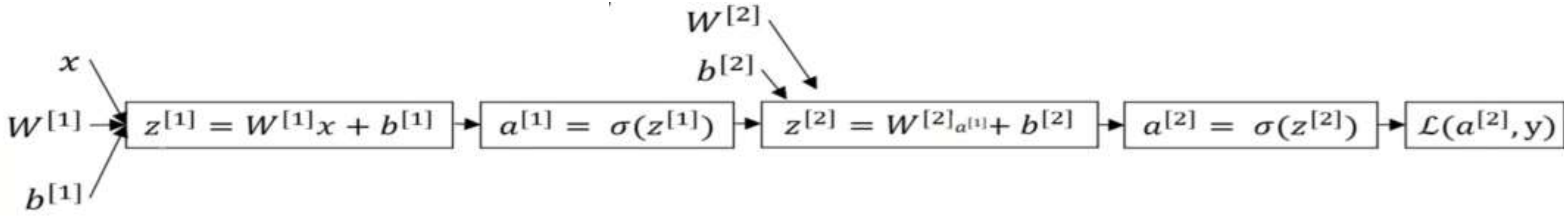
$$\boldsymbol{b}^{[2]} = \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \end{bmatrix}$$

Neural Network









Vectorizing across multiple examples

$Z^{[1]} = W^{[1]}X + b^{[1]}$
$A^{[1]} = \sigma(Z^{[1]})$
$Z^{[2]} = W^{[2]}X + b^{[2]}$
$A^{[2]} = \sigma(Z^{[2]})$

$$Z^{[1]} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ z^{[1](1)} & z^{[1](2)} & \dots & z^{[1](m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}$$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ a^{[1](1)} & a^{[1](2)} & \dots & a^{[1](m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}$$

Gradient decent for neural network

Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$

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

Gradient decent:

Repeat{

Compute prediction: $(\hat{y}^{(i)}, i = 1 \ldots m)$,

`

$dW^{[1]} = \frac{dJ}{dW^{[1]}}$, $db^{[1]} = \frac{dJ}{db^{[1]}}$, $dW^{[2]} = \frac{dJ}{dW^{[2]}}$, $db^{[2]} = \frac{dJ}{db^{[2]}}$

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

$b^{[1]} = b^{[1]} - \alpha\, db^{[1]}$

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

$b^{[2]} = b^{[2]} - \alpha\, db^{[2]}$

}

Matrix Calculus

$$\mathbf{y} = \psi(\mathbf{x})$$
$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \mathbf{A}$$

$$\alpha = \mathbf{y}^T \mathbf{A}\mathbf{x}$$

$$\frac{\partial \alpha}{\partial \mathbf{x}} = \mathbf{y}^T \mathbf{A} \qquad \frac{\partial \alpha}{\partial \mathbf{y}} = \mathbf{x}^T \mathbf{A}^T$$

$$\alpha = \mathbf{x}^T \mathbf{A}\mathbf{x}$$

$$\frac{\partial \alpha}{\partial \mathbf{x}} = \mathbf{x}^T \left(\mathbf{A} + \mathbf{A}^T\right)$$

If **A** is symmetric ?

https://explained.ai/matrix-calculus/
http://cs231n.stanford.edu/slides/2018/cs231n_2018_ds02.pdf
https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/

Formulas:
Computing derivative for Vectorized Input

Forward propagation:

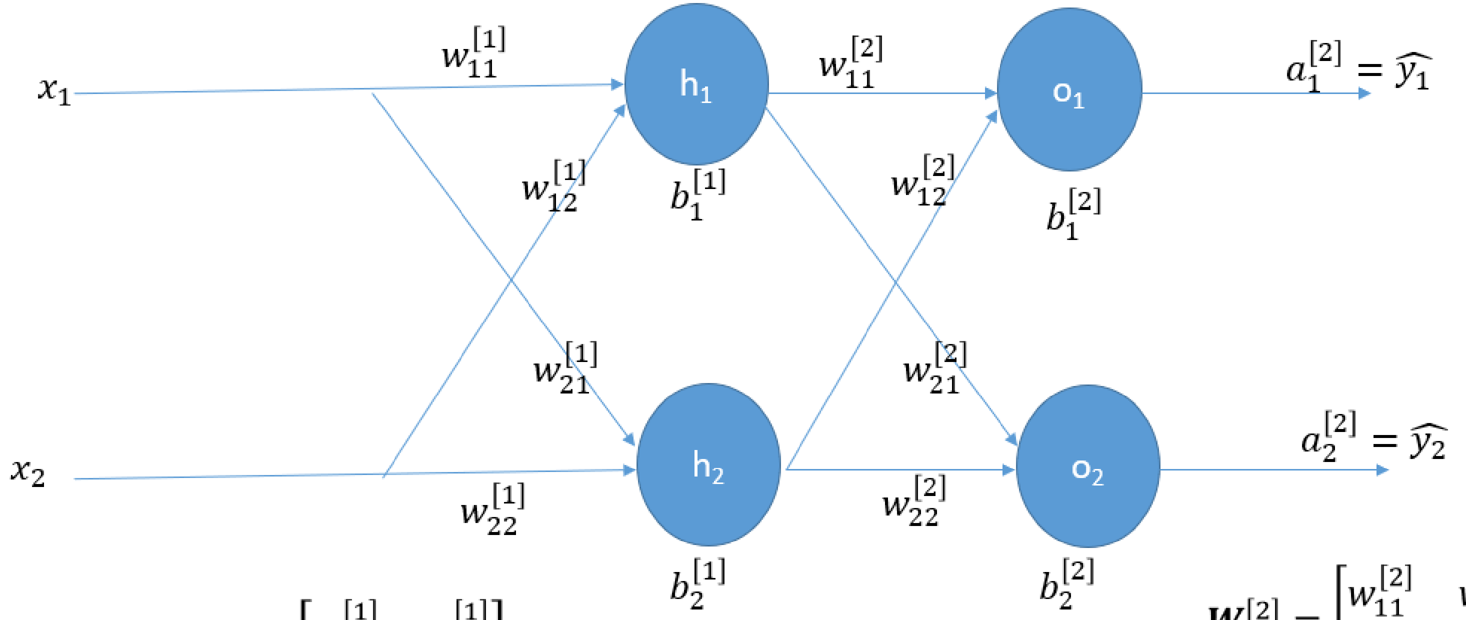
$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

Back propagation:

$$dZ^{[2]} = A^{[2]} - Y$$
$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$
$$db^{[2]} = \frac{1}{m}\ \mathrm{np.sum}(\ dZ^{[2]}, axis = 1, \mathrm{keepdims} = True)$$
$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^{T}$$
$$db^{[1]} = \frac{1}{m}\ \mathrm{np.sum}(\ dZ^{[1]}, axis = 1, \mathrm{keepdims} = True)$$

Dimensionality of W, b matrices???

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$



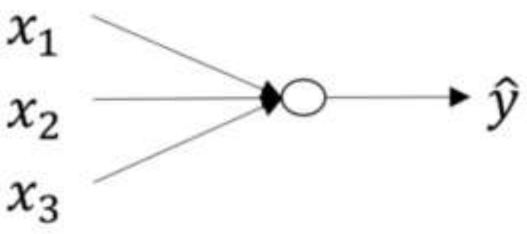
$$\boldsymbol{W}^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \end{bmatrix}$$

$$\boldsymbol{b}^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \end{bmatrix}$$

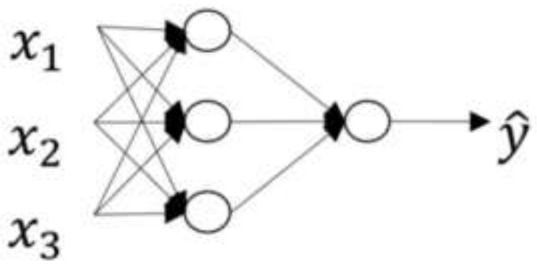
$$\boldsymbol{W}^{[2]} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} \\ w_{21}^{[2]} & w_{22}^{[2]} \end{bmatrix}$$

$$\boldsymbol{b}^{[2]} = \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \end{bmatrix}$$

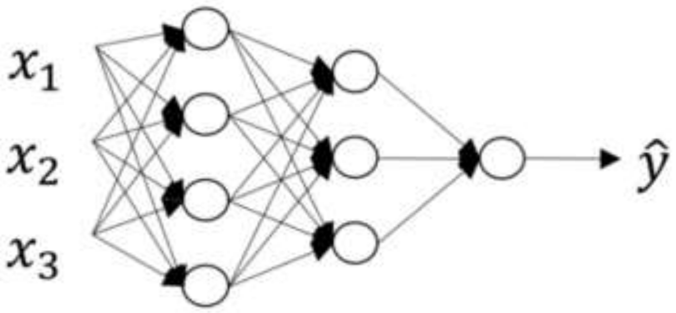
What is a deep neural network?



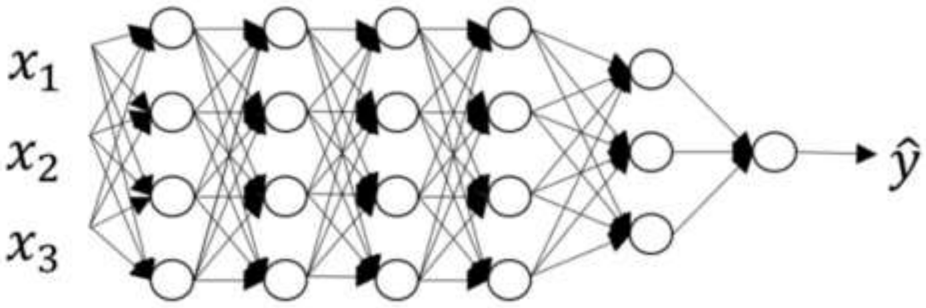
logistic regression



1 hidden layer



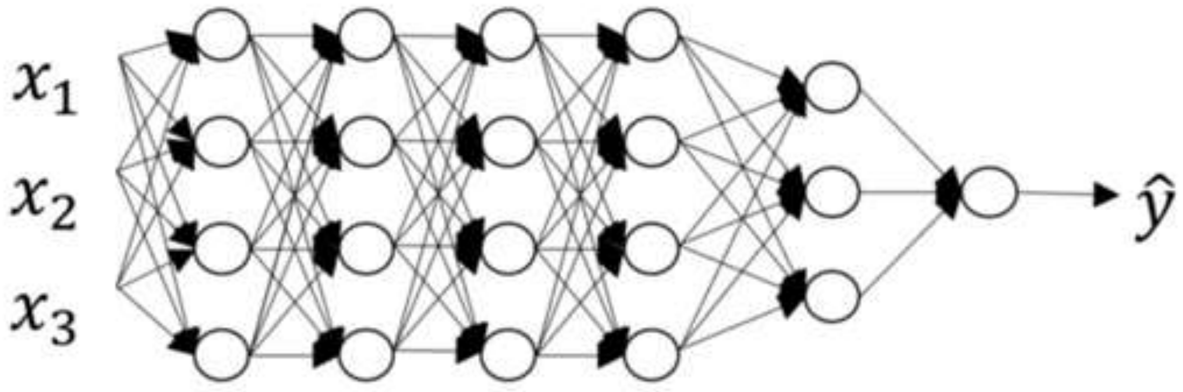
2 hidden layers



5 hidden layers

Deep neural network notation

l :	number of layers
$n^{[l]}$:	number of units in layer l
$a^{[l]}$:	activations in layer l
$z^{[l]}$:	logit in lyer l
$W^{[l]}$:	weight for $z^{[l]}$
$b^{[l]}$:	bias
$n^{[0]}=n_x$	Input dimension



Gradient decent for Logistic Regression

$$z = w^T x + b$$
$$\hat{y} = a = \sigma(z)$$
$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

$$\frac{dL}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$= \frac{-y+ya+a-ya}{a(1-a)} = \frac{a-y}{a(1-a)}$$

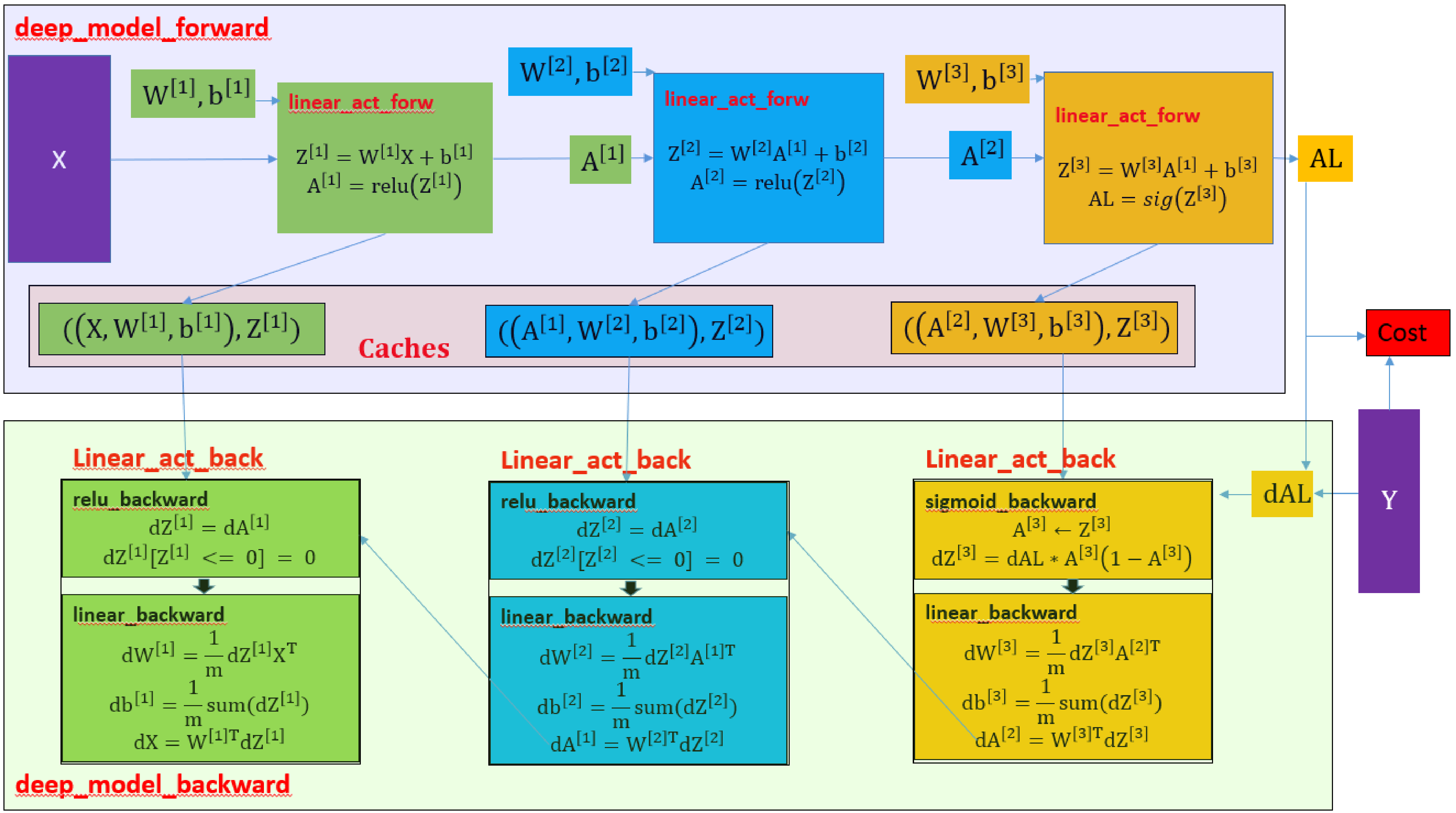
$$\frac{da}{dz} = a(1-a)$$

$$\frac{dL}{dz} = \frac{dL}{da} \text{X} \frac{da}{dz} = a - y$$

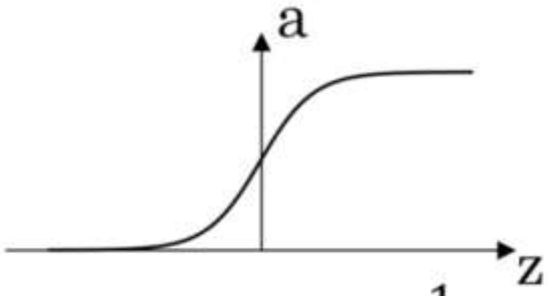
$$\frac{dz}{dw} = \boldsymbol{x}$$

$$\frac{dz}{db} = 1$$

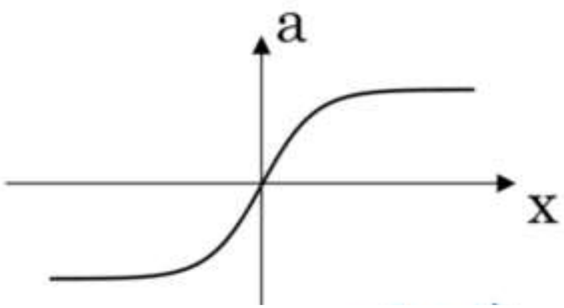
Deep neural network architecture

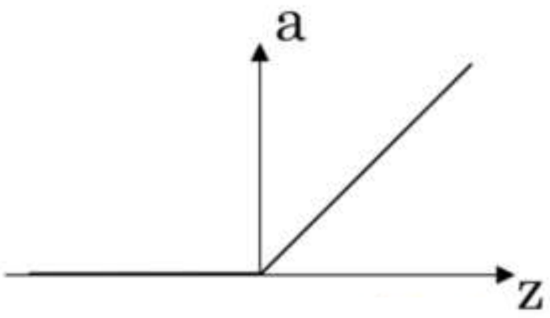


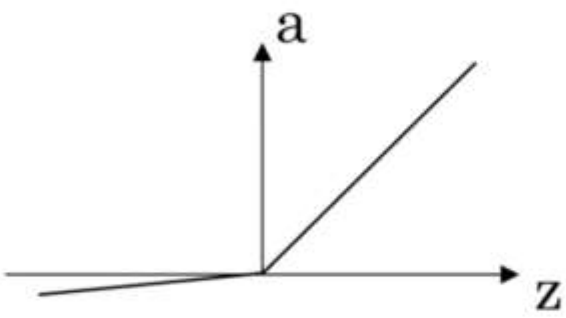
Pros and cons of activation functions



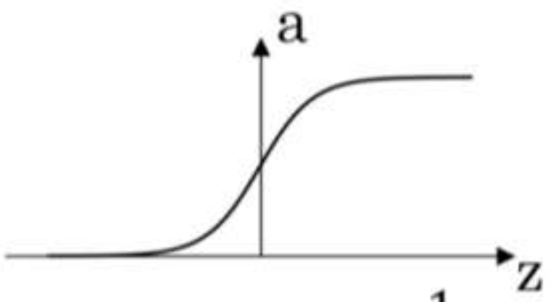
sigmoid: $a = \dfrac{1}{1 + e^{-z}}$



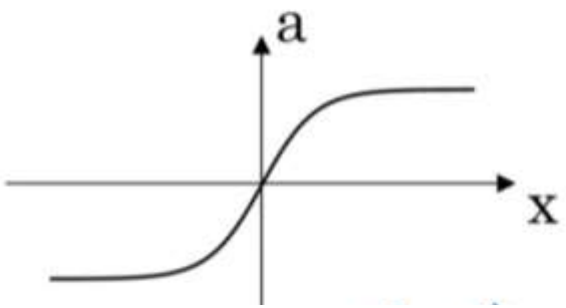


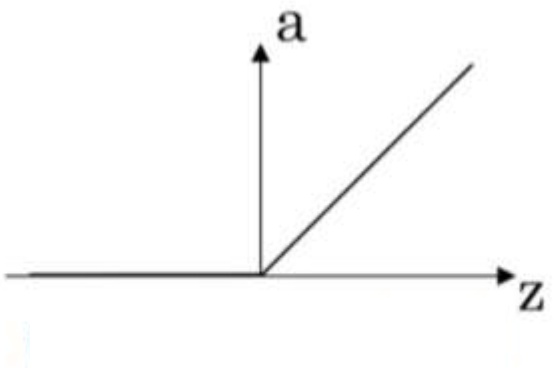


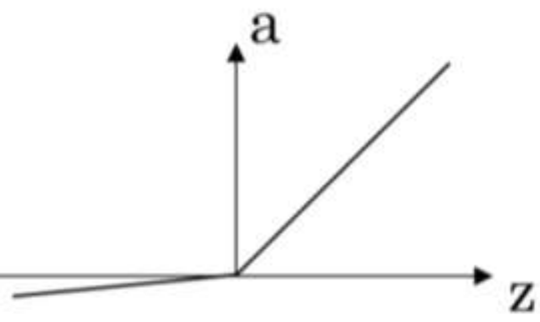
Derivative of activation functions?



sigmoid: $a = \dfrac{1}{1 + e^{-z}}$







Output Units

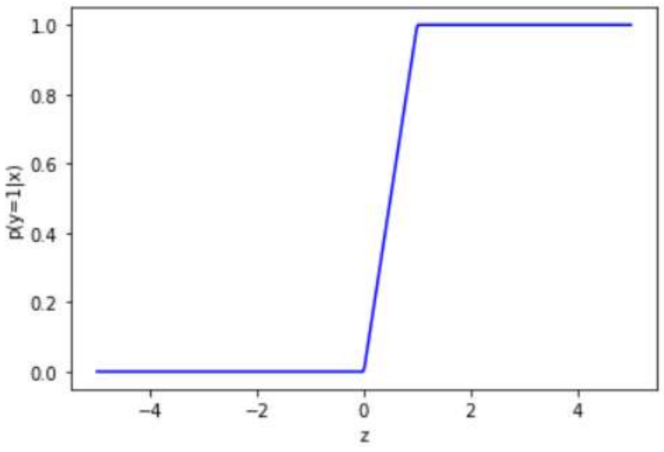
- Linear Units for Gaussian Output Distributions

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I}).$$

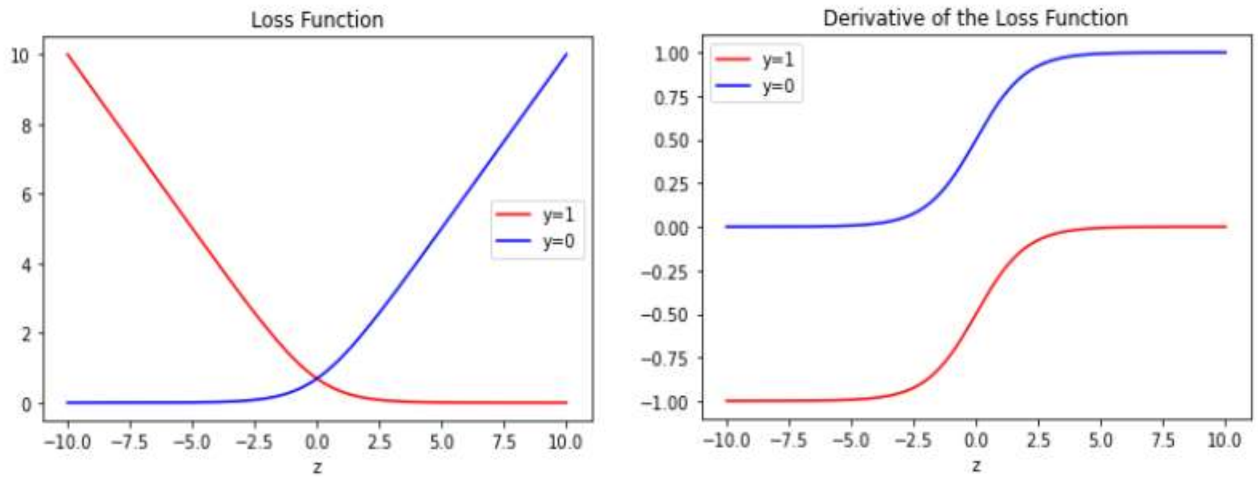
Maximizing the log-likelihood is then equivalent to minimizing the mean squared error.

- Sigmoid Units for Bernoulli Output Distributions

$$P(y=1|x) = \max\{0, \min\{1, \theta^T h + b\}\}$$



Sigmoid Units



Output Units

- Softmax Units for Multinoulli Output Distributions

$$\mathrm{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}.$$

END