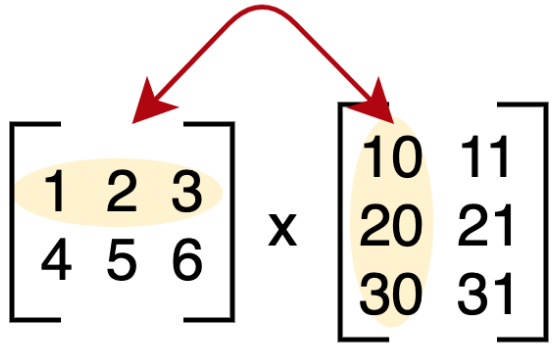
Neural Networks

Matrix Multiplication

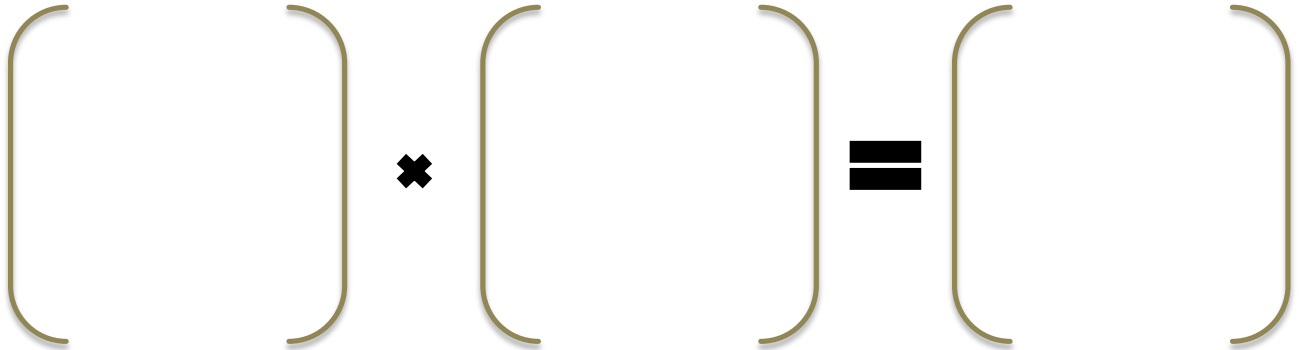


$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 10 & 11 \\ 20 & 21 \\ 30 & 31 \end{bmatrix}$$

$$= \begin{bmatrix} 1x10 + 2x20 + 3x30 & 1x11 + 2x21 + 3x31 \\ 4x10 + 5x20 + 6x30 & 4x11 + 5x21 + 6x31 \end{bmatrix}$$

$$= \begin{bmatrix} 10+40+90 & 11+42+93 \\ 40+100+180 & 44+105+186 \end{bmatrix} = \begin{bmatrix} 140 & 146 \\ 320 & 335 \end{bmatrix}$$

Matrix dimensions



X × M M × Y X × Y

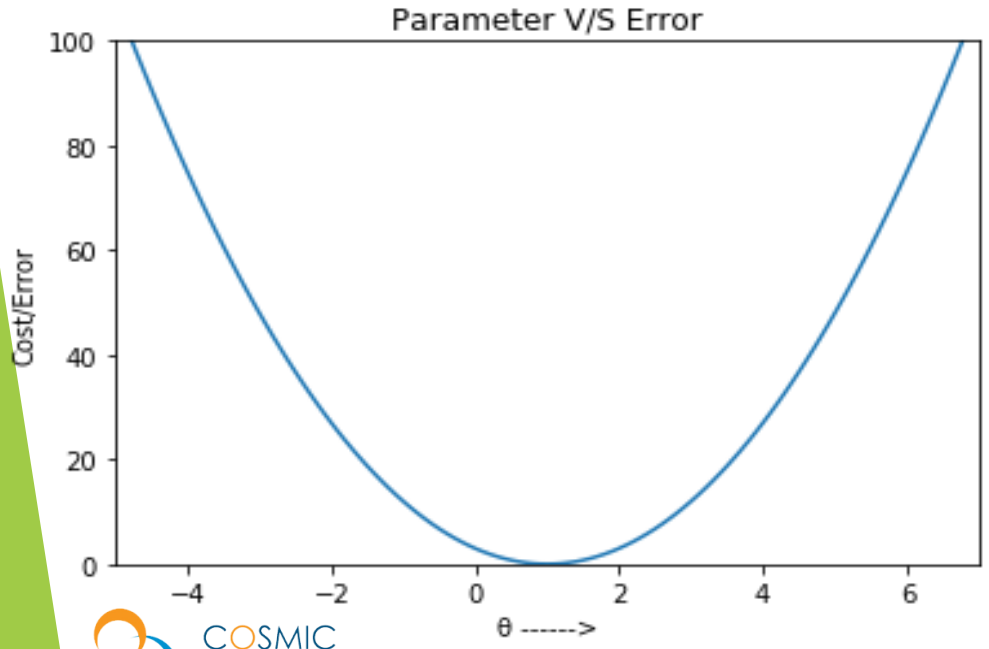
So far,

$h_\Theta(x) = \Theta_0 + \Theta_1 x_1 + \Theta_2 x_2$

Parameters were initialized randomly

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

Compute loss based on all the examples
For that particular weight

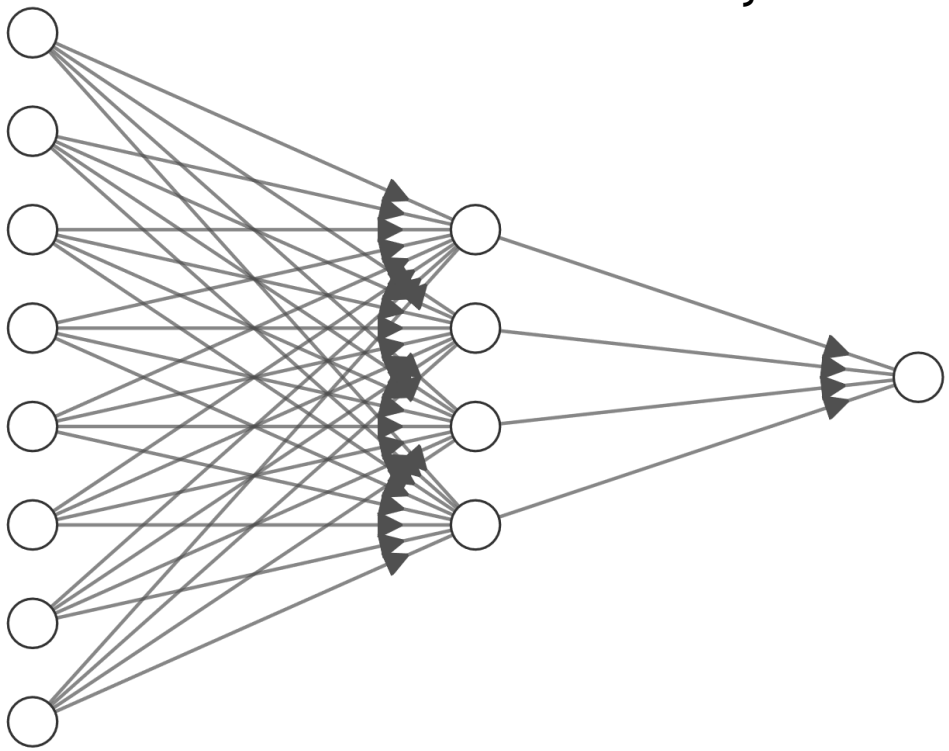


Find Derivative of loss w.r.t parameter
Update the parameters

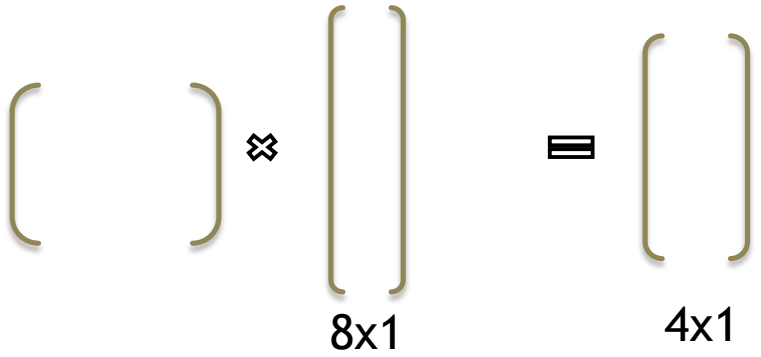
COSMIC SKILLS

Neural Networks

Hidden layers = 1



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

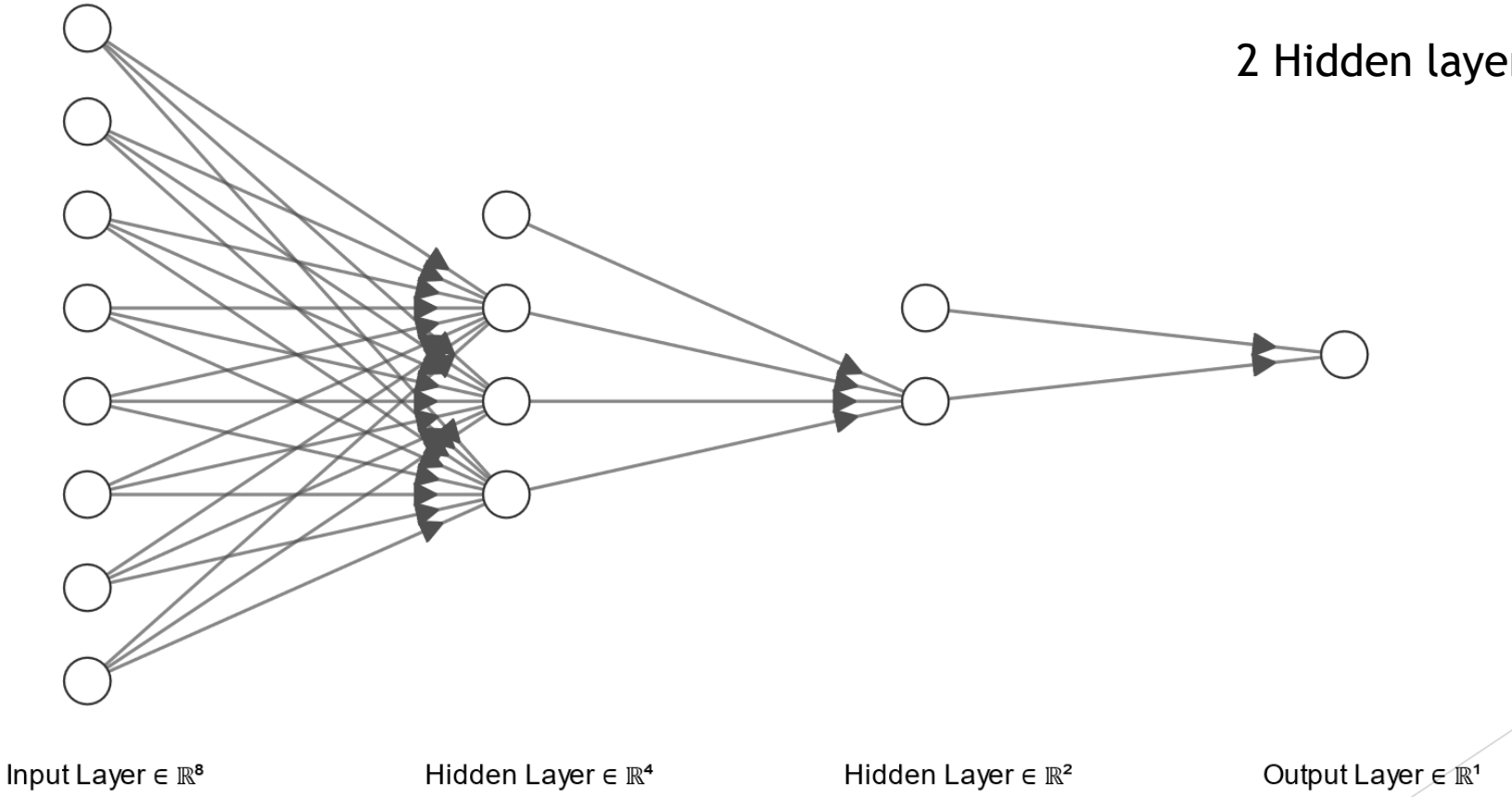


Input Layer $\in \mathbb{R}^8$ Hidden Layer $\in \mathbb{R}^4$ Output Layer $\in \mathbb{R}^1$

COSMIC SKILLS

Fully Connected Network

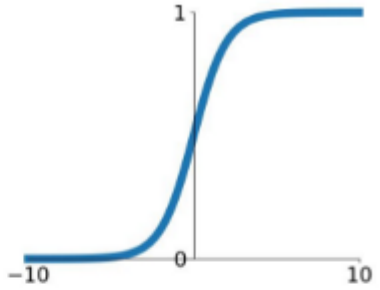
```
model = Sequential()
model.add(Dense(units=4,activation="relu",input_shape=(8,)))
model.add(Dense(units=2,activation="relu"))
model.add(Dense(units=1))
```



Activation Functions

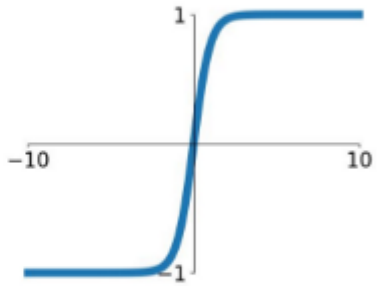
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



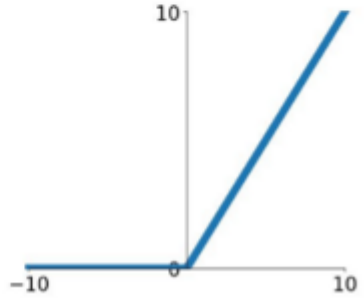
tanh

$$\tanh(x)$$



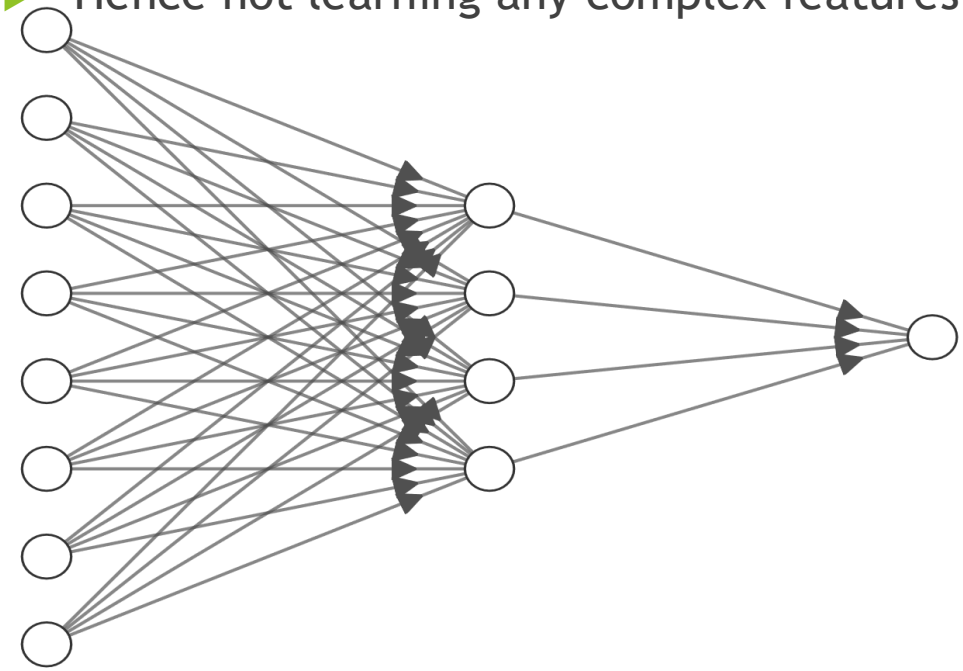
ReLU

$$\max(0, x)$$



Why Non linear activation function ?..

- Equivalent to linear regression (you are just multiplying same thing in different way)
- Hence not learning any complex features

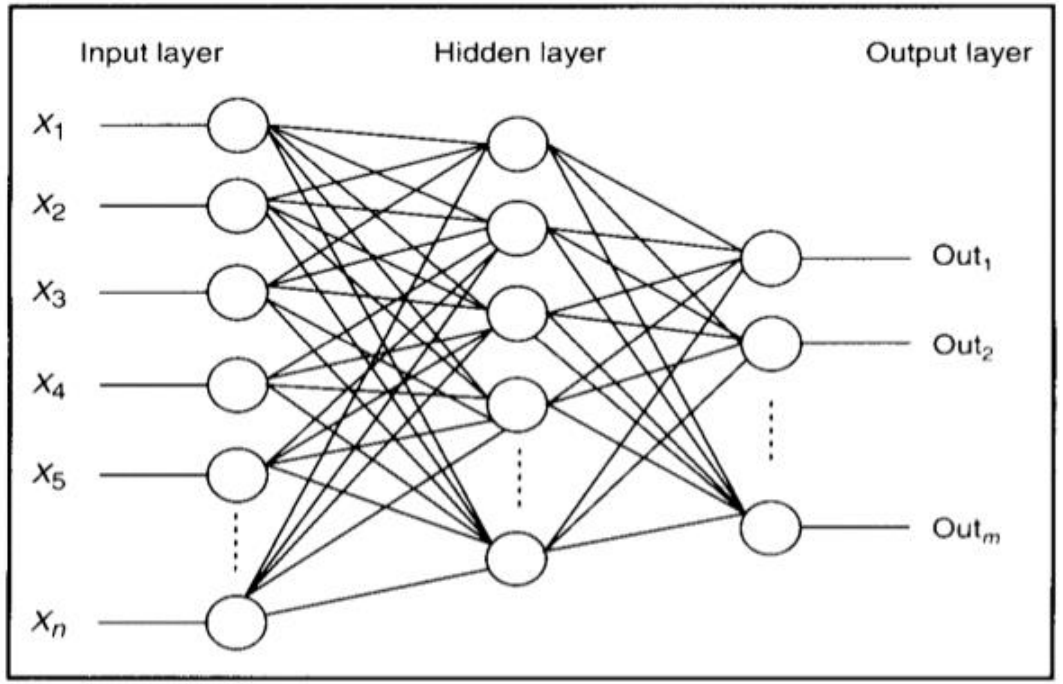


Input Layer $\in \mathbb{R}^8$ Hidden Layer $\in \mathbb{R}^4$ Output Layer $\in \mathbb{R}^1$

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

COSMIC SKILLS

1


Input layer
Hidden layer
Output layer
x_1
x_2
x_3
x_4
x_5
x_n
Out_1
Out_2
Out_m

Why deep learning frameworks??...

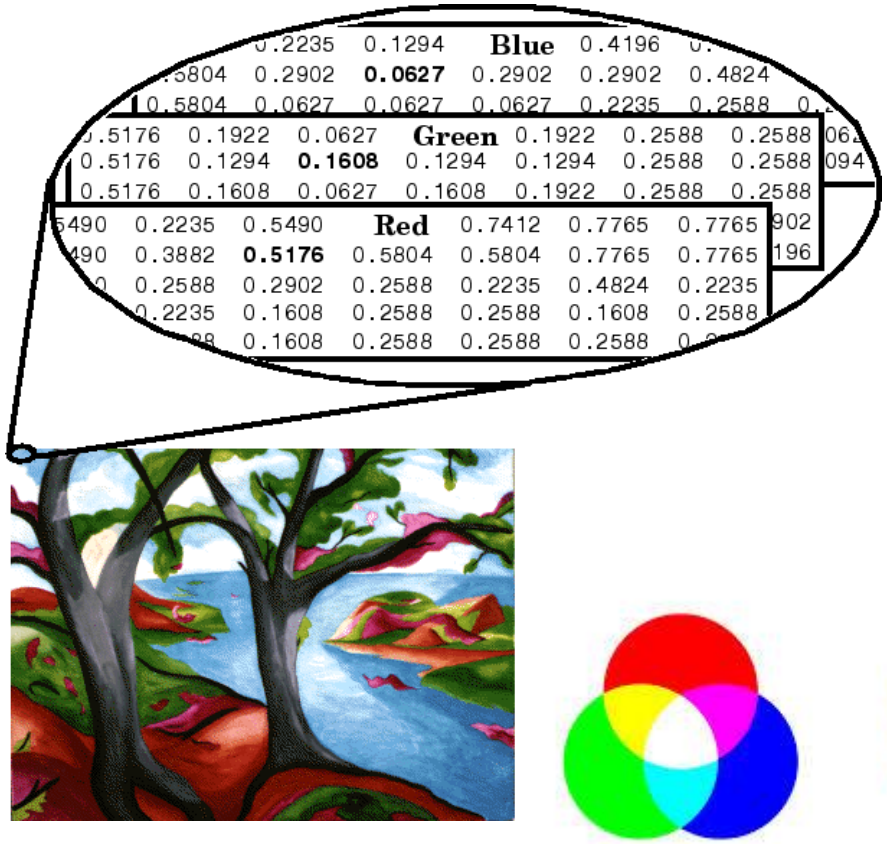
- Easy to code
- Coding reduces hence easy to debug
- Functions are pre-built and are optimized

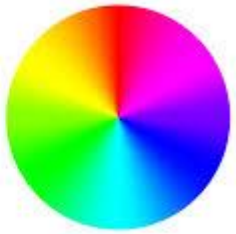
E.g: Simple 3x3 matrix multiplication in pytorch is 50,000 times faster than pure python code

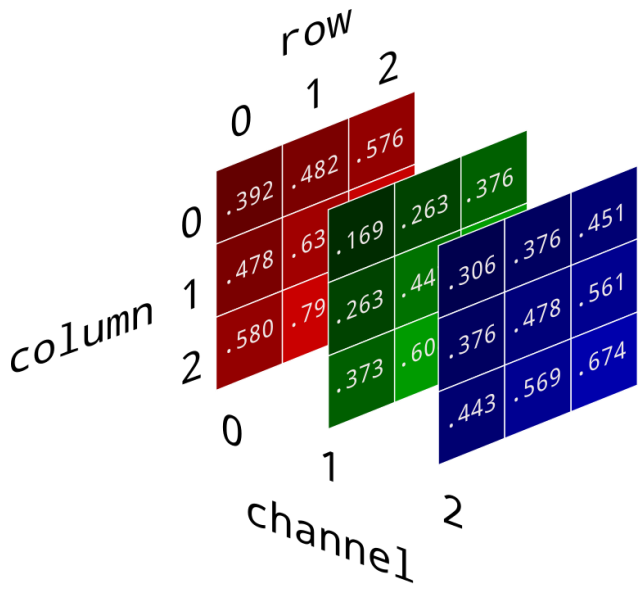
Tensorflow

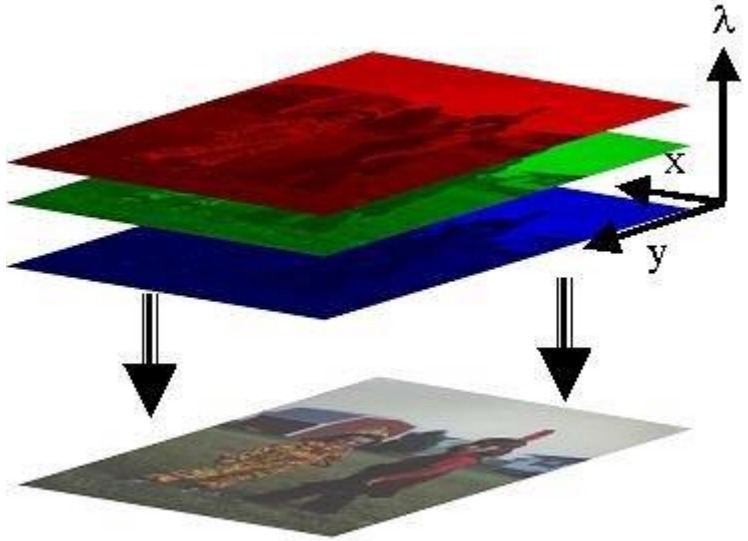
- It is being developed and maintained by Google
- TensorFlow is a free and open-source
- It's **written** in a combination of highly-optimized C++ and CUDA (Nvidia's **language** for programming GPUs)
- A library for defining computation graph and a runtime to execute such graph in different hardware

Representation of Images





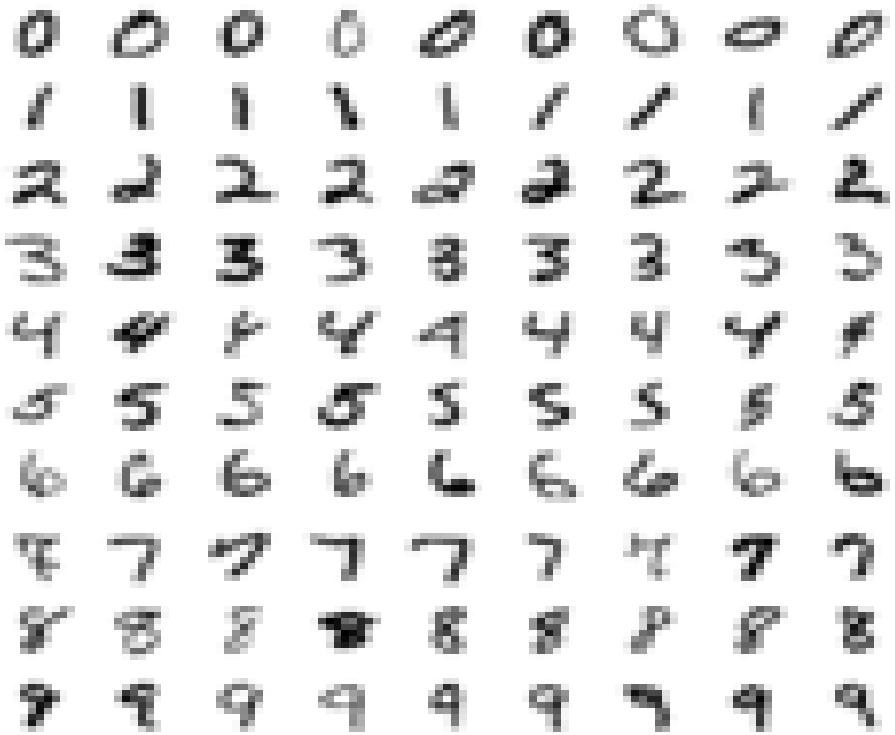


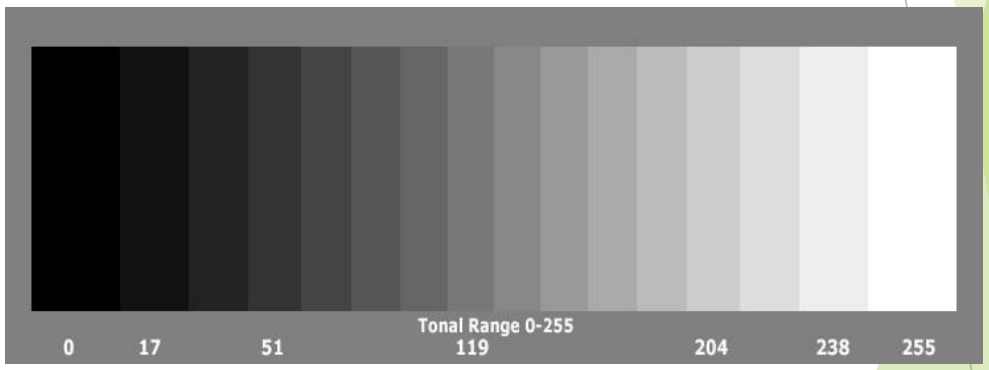


COSMIC SKILLS

Model for digit recognition

- Data set : MNIST





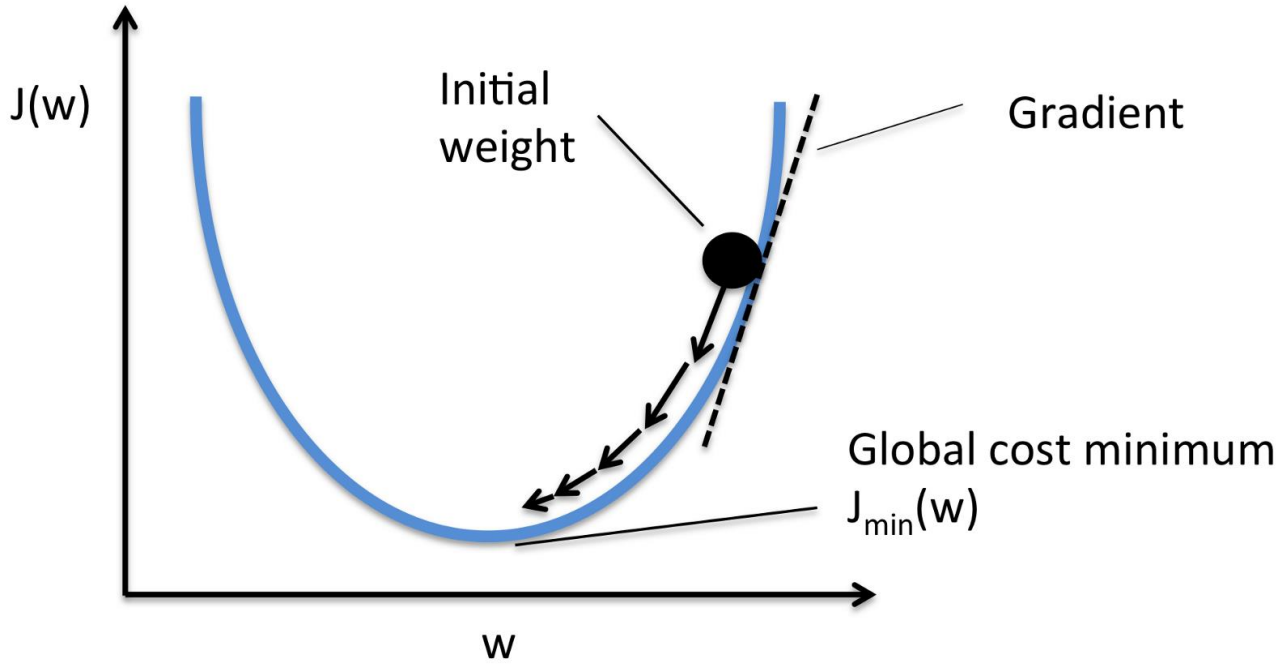
Gradient Descent

Batch Gradient descent

- It considers all the examples in training set to take one step

Mini batch gradient descent

- It considers batch of examples from training set to take each step



$$\text{Repeat until convergence } \{$$

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$$\}$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

COSMIC SKILLS

```
keras.utils.to_categorical(train_y,10)
```

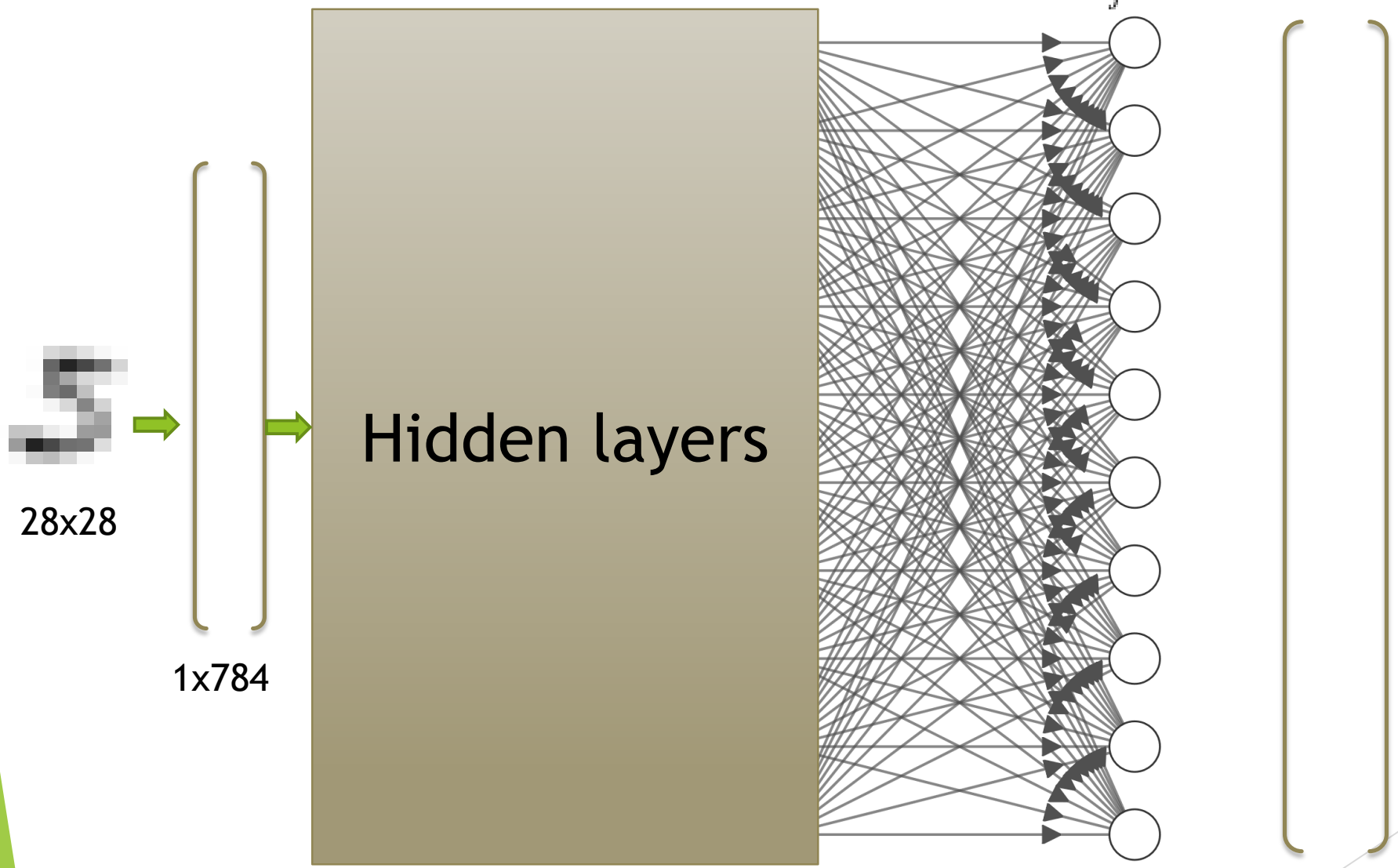
Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

One Hot encoding

Softmax

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$



5. Loss Function

Softmax

$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

$$-\begin{pmatrix}0\\0\\0\\1\\0\end{pmatrix} \times \log\begin{pmatrix}0.23\\0.04\\0.06\\0.6\\0.07\end{pmatrix} = 0.22$$

True Values
(one hot encoded)

Where $f(s)_i$ represents SoftMax function and is given by,

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

Where C represents total number of class and
s_j are the score inferred by the neural network for each class in C.