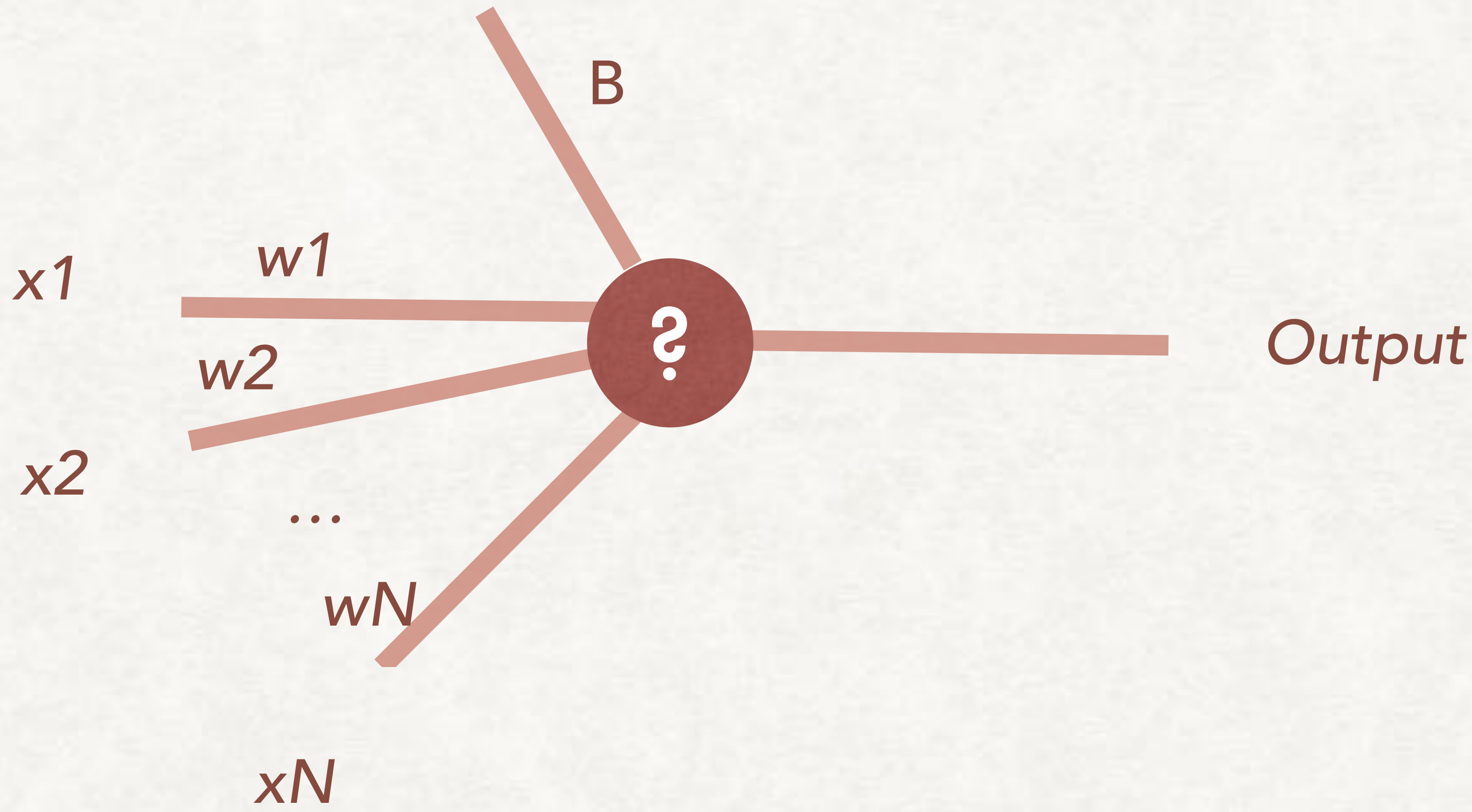
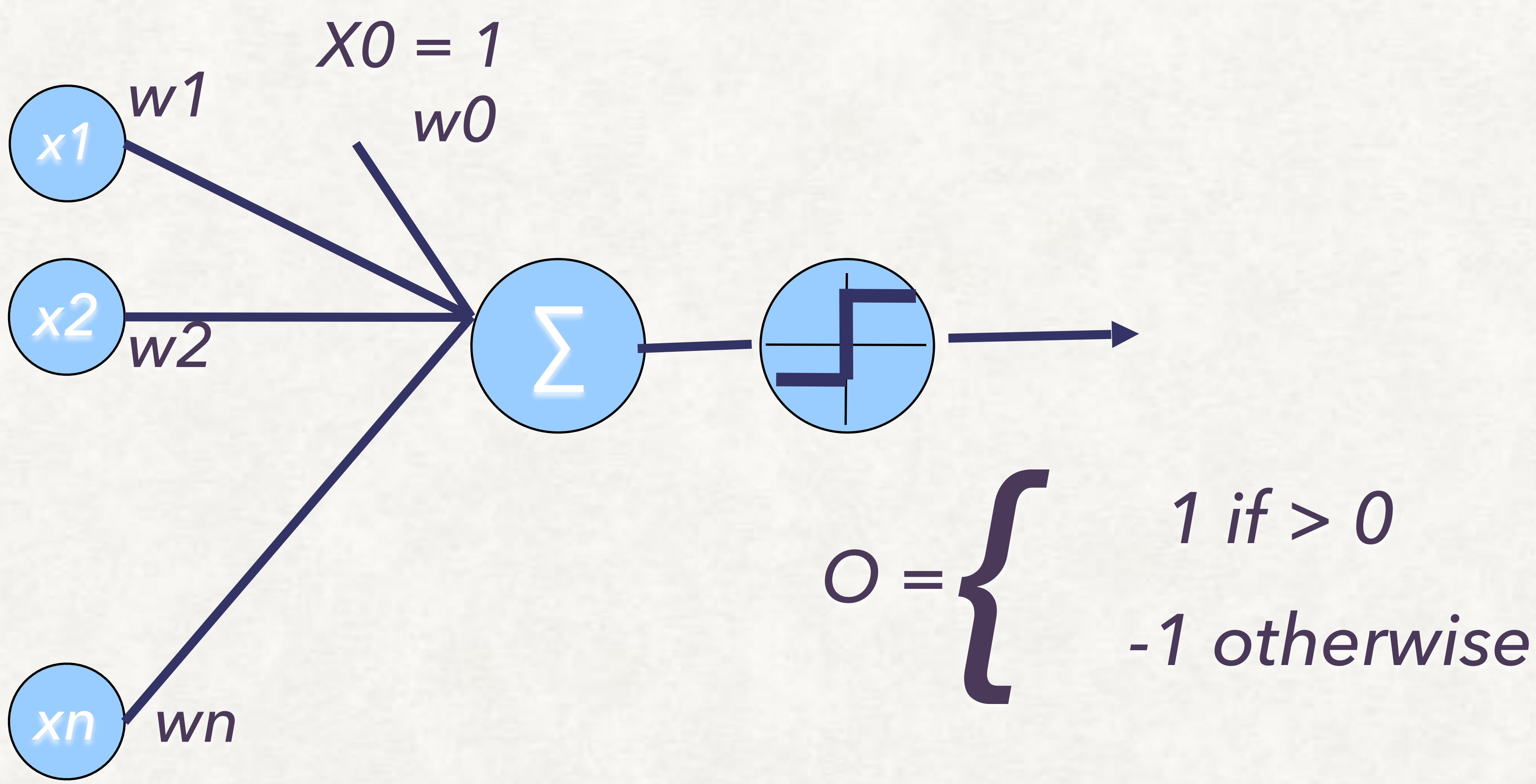
# MORE ON NEURONS

# Activation Functions



# ONE NEURON OR PERCEPTRON



# Sigmoid, Tanh, ReLU Neurons

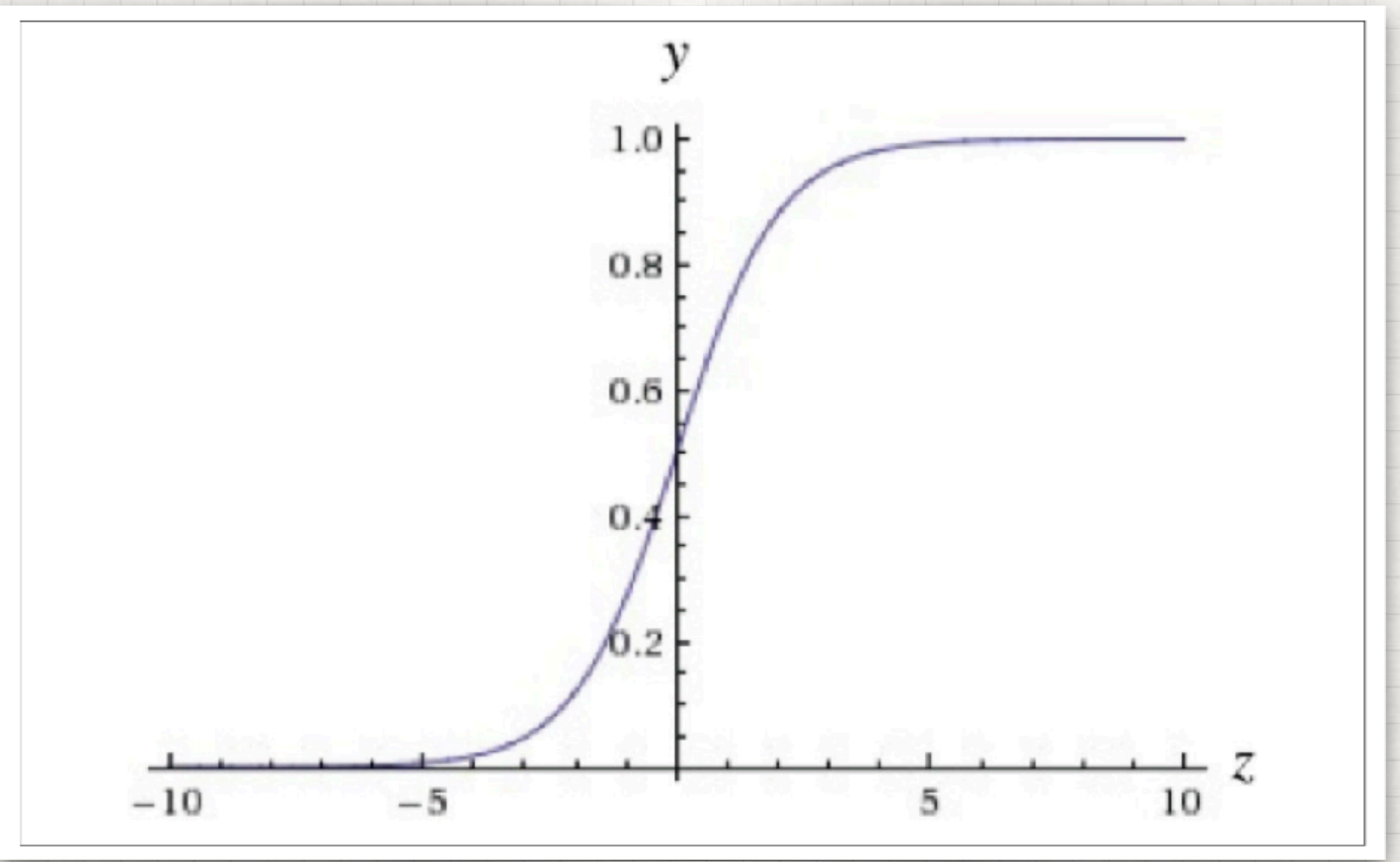
(trying to make things nonlinear)

# SIGMOID

$$f(z) = \frac{1}{1 + e^{-z}}$$

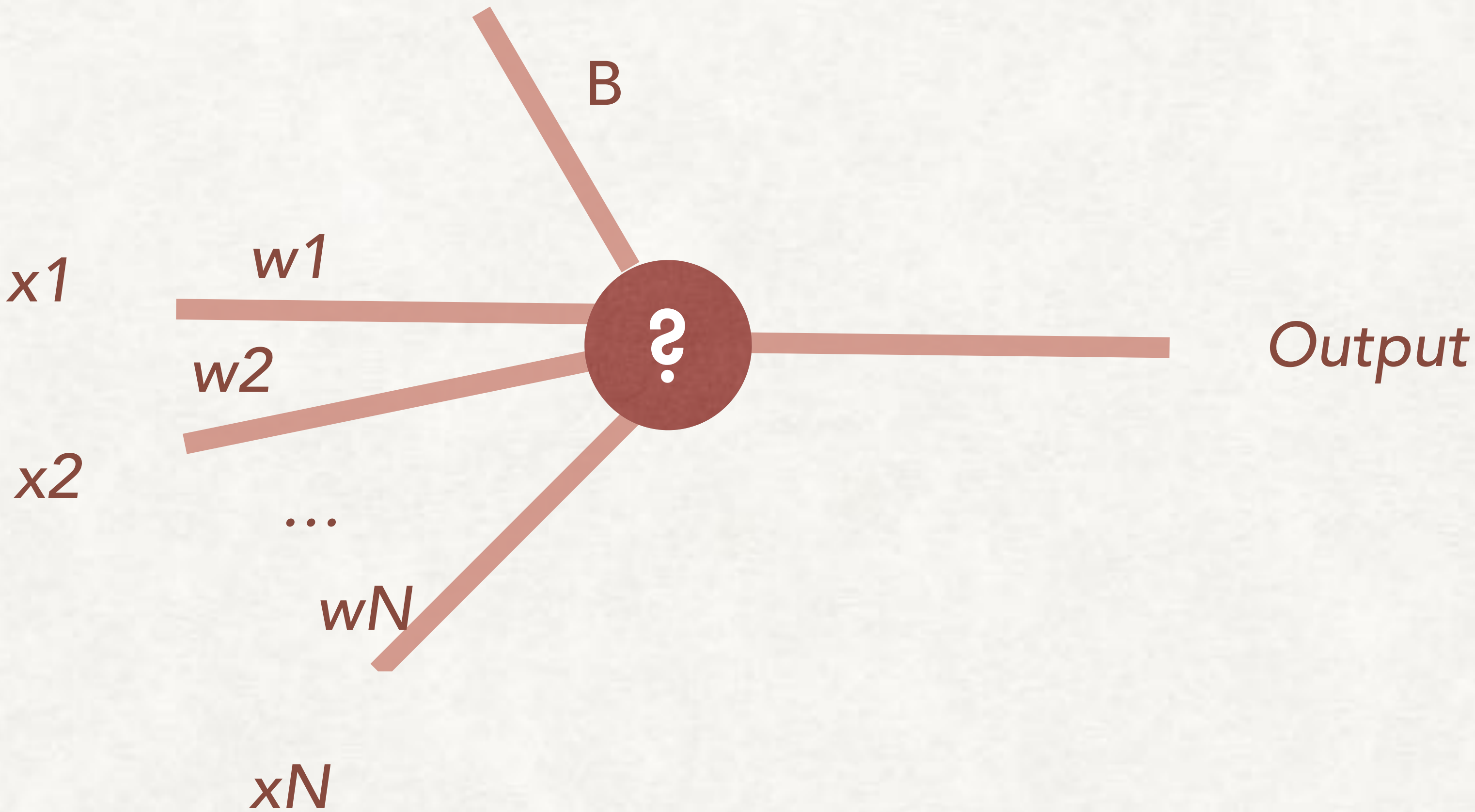
```
>>> f(-10)
4.5397868702434e-05
>>> f(-5)
0.0066928509242848554
>>> f(5)
0.9933071490757153
>>> f(10)
0.9999546021312976
>>>
```

Intuitively, this means that when the logit is very small, the output of a logistic neuron is very close to 0. When the logit is very large, the output of the logistic neuron is close to 1. In-between these two extremes, the neuron assumes an S-shape



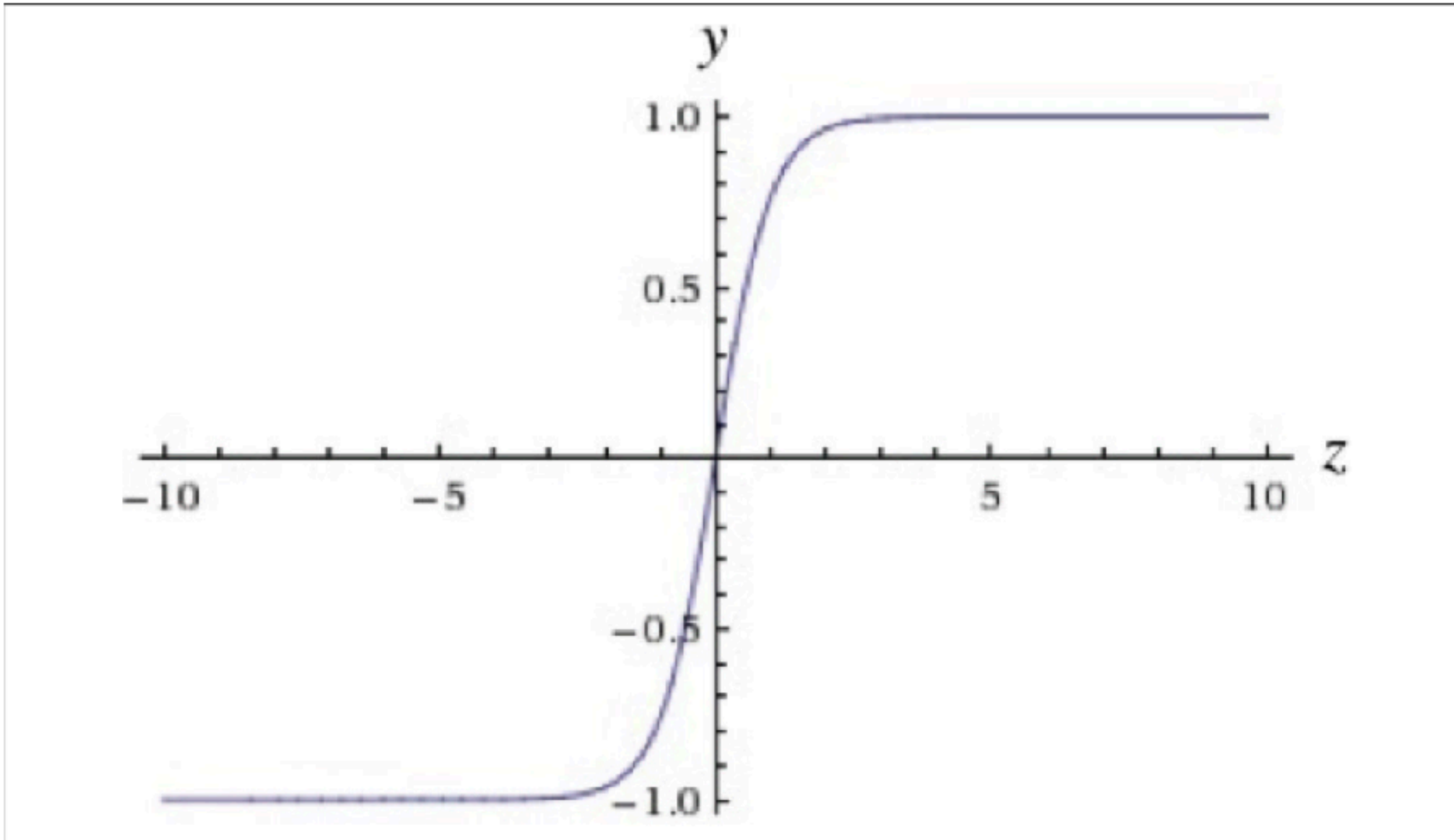
y
1.0
0.8
0.6
0.4
0.2
z
−10
−5
5
10

# Activation Functions



# TANH

SIMILAR BUT INSTEAD OF 0 TO 1 RANGES FROM -1 TO 1


y
1.0
0.5
−0.5
−1.0
z
−10
−5
5
10

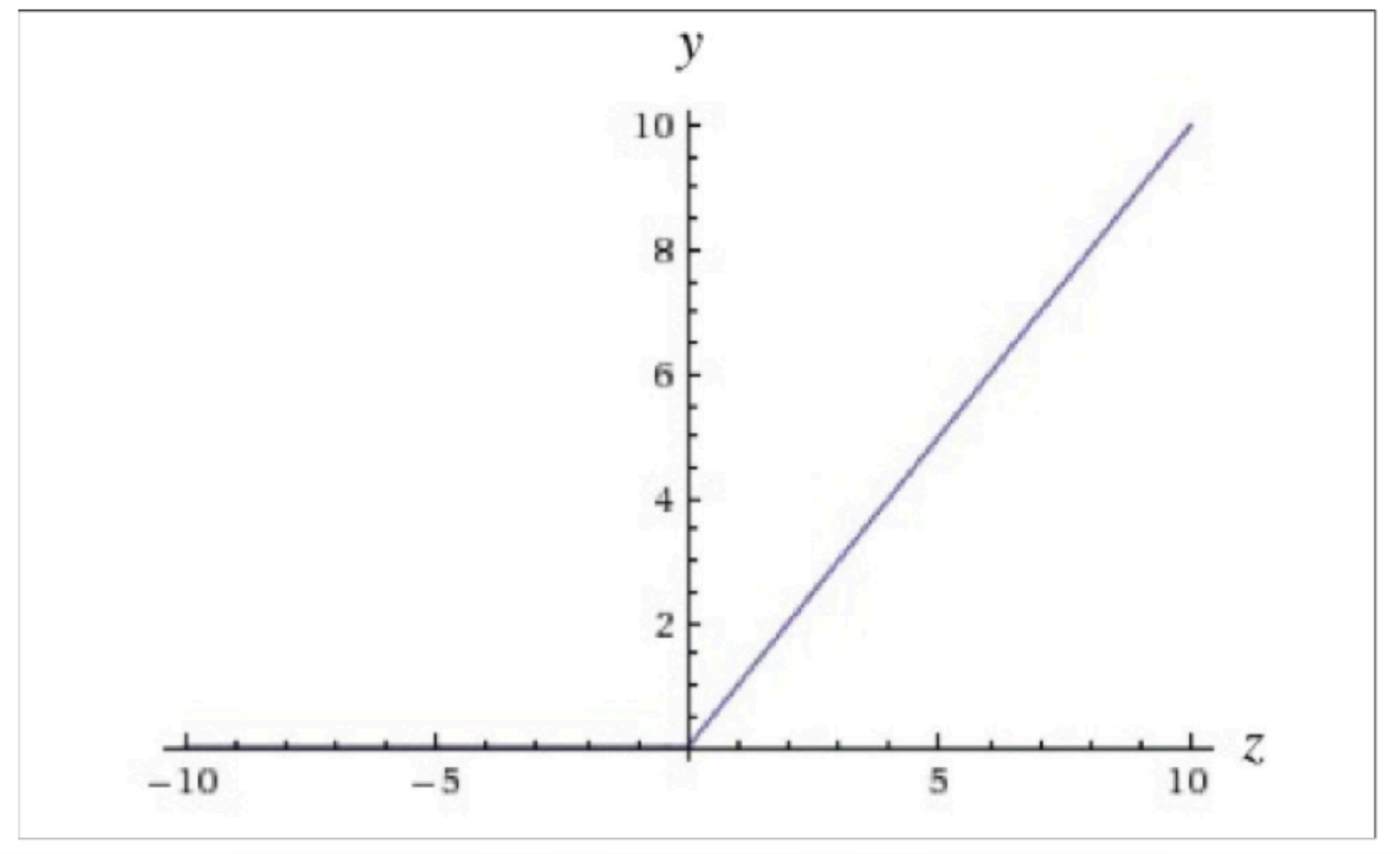
# ReLU

## RESTRICTED LINEAR UNIT NEURON

$$f(z) = \max(0, z)$$

Hockey stick shaped pattern

“The neuron of choice especially for computer vision”



y
10
8
6
4
2
z
−10
−5
5
10

“

All the above work for all neurons regardless of layer
They don’t depend on any other neuron

”

# Softmax Output Layers

$$y_i = \frac{e^{z_i}}{\Sigma_j e^{z_j}}$$

- `Y = tf.nn.softmax(tf.matmul(XX, W) + b)`
- Only output layer
- Depends on all other output neurons
- Probability distribution

$$y_i = \frac{e^{z_i}}{\Sigma_j e^{z_j}}$$

front  8, side 24, back 12, front corner 3

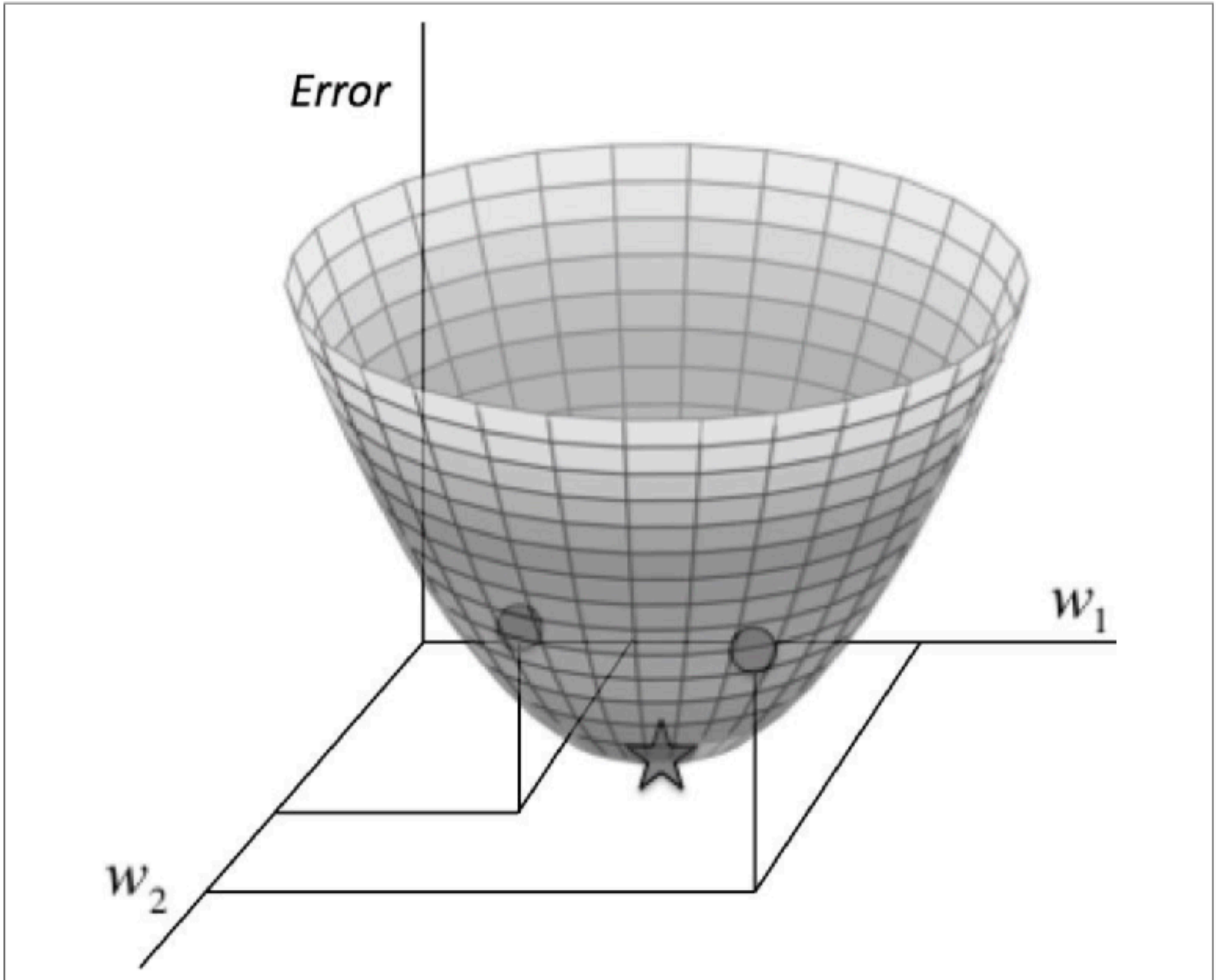
```
>>>
>>> import numpy as np
>>> image = np.array([8,24,12,3])
>>> ytemp = math.e**image
>>> yi = ytemp / np.sum(ytemp)
>>> sum(yi)
0.9999999999999999
>>>
```

```
>>> yi
array([1.12534471e-07, 9.99993743e-01, 6.14417391e-06, 7.58251298e-10])
>>>
```

```
# The model
Y = tf.nn.softmax(tf.matmul(XX, W) + b)
```

```
train_step = tf.train.GradientDescentOptimizer(0.005).minimize(cross_entropy)
```

A foggy mountain


Error
$w_1$
$w_2$

# DELTA RULE

$$\Delta w_k = \sum_i \epsilon x_k^{(i)}(t^{(i)} - y^{(i)})$$

- $t^i$ is the true answer for the $i^{\{th\}}$ training example
- $y^i$ is the prediction our neural network made for the $i^{\{th\}}$ training example
- $x^i$ is the input value
- epsilon is the learning rate

# DELTA RULE

$$\Delta w_k = \sum_i \epsilon x_k^{(i)} (t^{(i)} - y^{(i)})$$

If the neural network correctly predicted this example

- $t^i$ is the true answer for the $i^{\{th\}}$ training example
- $y^i$ is the prediction our neural network made for the $i^{\{th\}}$ training example
- $x^i$ is the input value
- epsilon is the learning rate

Example

# DELTA RULE

Error

$$\Delta w_k = \sum_i \epsilon x_k^{(i)} (t^{(i)} - y^{(i)})$$

If the neural network correctly predicted this example

- $t^i$ is the true answer for the $i^{\{th\}}$ training example
- $y^i$ is the prediction our neural network made for the $i^{\{th\}}$ training example
- $x^i$ is the input value
- epsilon is the learning rate

Demo ray

# TEAM TASK

## EVERYONE IN TEAM MODIFY RAY.PY

- Don't need to print the biases
- Run the training step 1001 times
- Every 10th training step print the accuracy and loss
- Every 100th run the test data and print the epoch accuracy
- Visualize the graph in tensor board
- Be able to explain every line in the program.
- **Congratulations - You've mastered single layer NN!!**