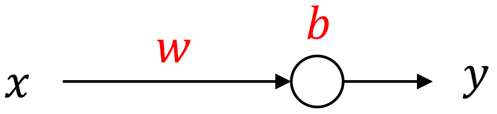
Basics of neural networks

What is a neural network?

An analytical model of output y as a function of input x, containing some fitting parameters

1. Linear Regression Model:

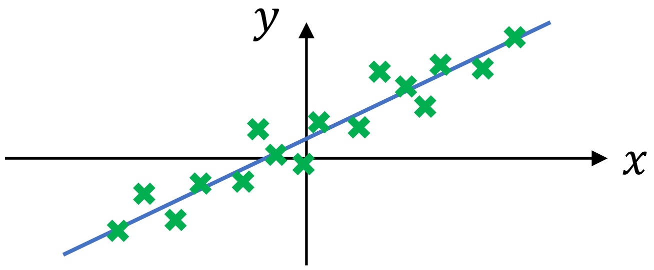
$$\boxed{y = wx + b}$$



Given observations of $\{x_d^i, y_d^i\}_i^n$

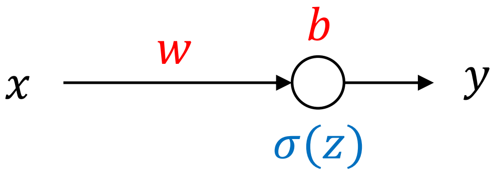
Find the w and b that minimizes

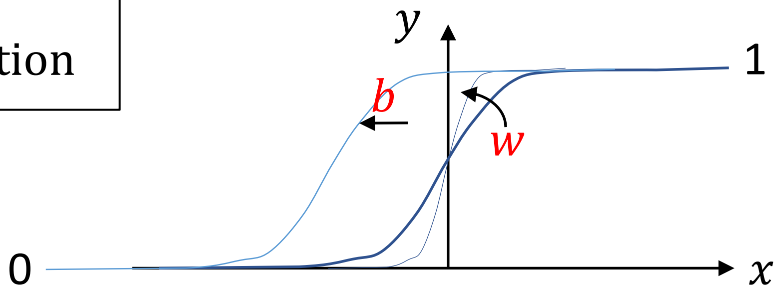
$$J = \sum_{i=1}^{n} \left(y\left(x_d^i\right) - y_d^i\right)^2$$



2. Logistic Regression Model: make output 0 to 1

$$\boxed{\begin{gathered} y = \sigma(wx + b), \\ \text{where } \sigma(z) = \frac{1}{1 + e^{-z}} \text{ is a sigmoid function} \end{gathered}}$$



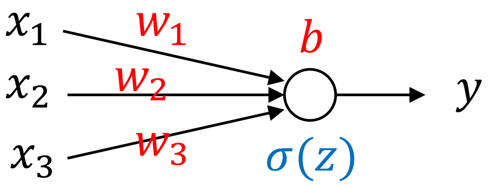


What is a neural network?

An analytical model of output y as a function of input x, containing some fitting parameters

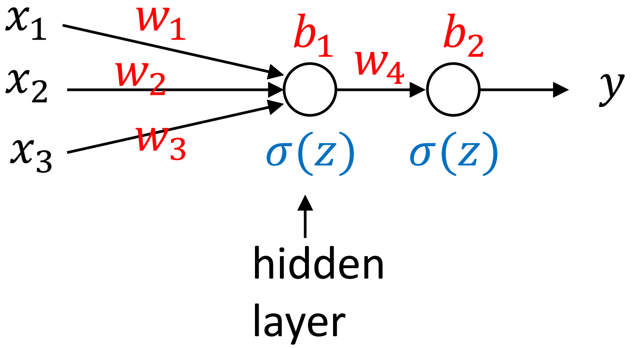
2. Logistic Regression Model: make output -1 to 1

$$y = \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3 + b),$$
$$\text{where } \sigma(z) = \frac{1}{1+e^{-z}} \text{ is a sigmoid function}$$



3. Neural network:

$$y = \sigma(w_4 \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3 + b_1) + b_2),$$
$$\text{where } \sigma(z) \text{ is a nonlinear activation function}$$



Common choices of $\boldsymbol{\sigma(z)}$

$sigmoid(z)$

$\left.\begin{array}{l}\sin(z)\\ \cos(z)\\ \tanh(z)\end{array}\right\}$ Output ranges from -1 to 1

…

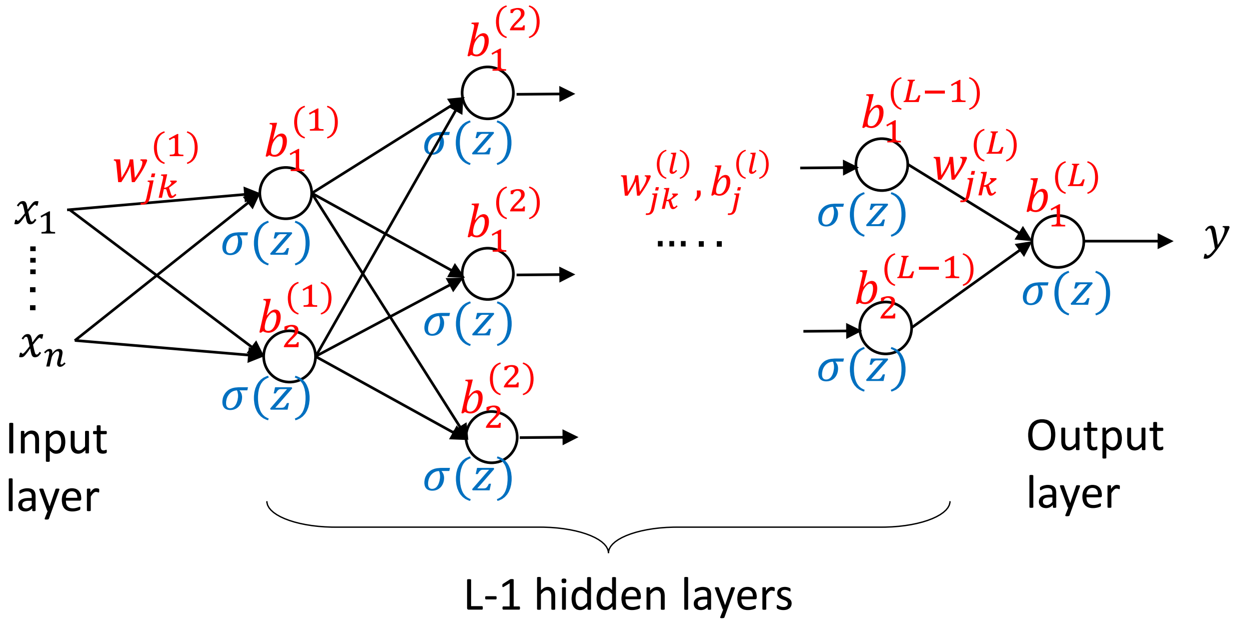
What is a neural network?

An analytical model of output y as a function of input x, containing some fitting parameters

3. Neural network:

More generally…

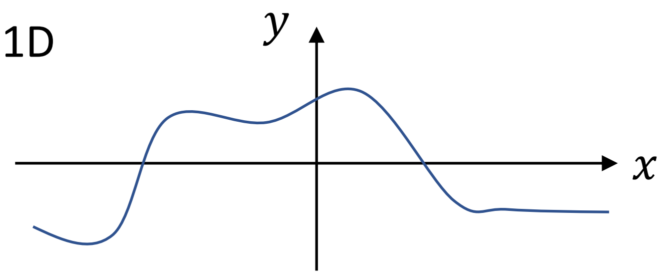
$$\boxed{\begin{gathered} y = fn(w_{jk}^{(l)}, b_j^{(l)}, x_i), \\ \text{where } \sigma(z) \text{ is a nonlinear activation function} \end{gathered}}$$



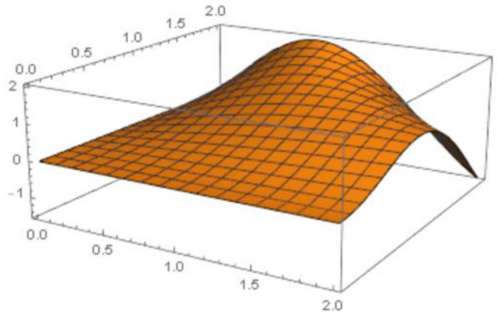
Neural network is a general function approximation

-> Universal function approximation

1D



2D

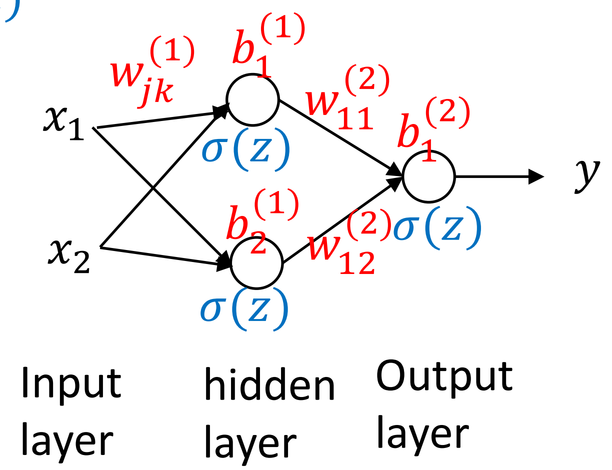


n-Dimension surface….

Why are activation function nonlinear?

Common choices of $\sigma(z)$

$sigmoid(z)$
$\sin(z)$
$\cos(z)$
$\tanh(z)$
…



- If activation function is linear, NN can only represent a linear function

If activation fn is linear, e.g. $\sigma(z) = z$ …

Input x_1, x_2

In hidden layer, j=1,2

$$z_j^{(1)} = \sum_{k=1}^{2} w_{jk}^{(1)} x_k + b_j^{(1)}$$

$$a_j^{(1)} = \sigma\left(z_j^{(1)}\right) = z_j^{(1)}$$

In output layer, j=1

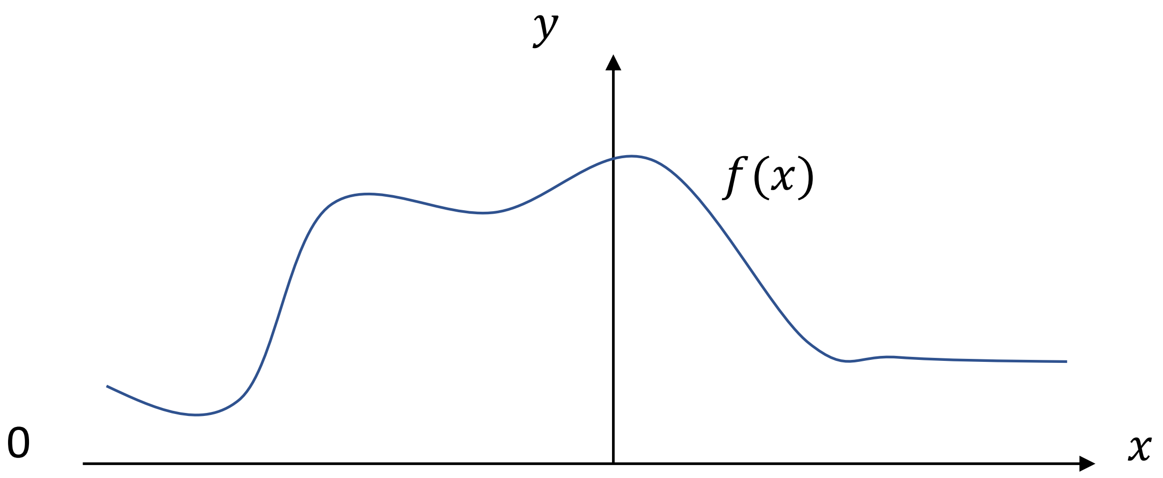
$$z_j^{(2)} = \sum_{k=1}^{2} w_{jk}^{(2)} a_k^{(1)} + b_j^{(2)}$$

$$a_j^{(2)} = \sigma\left(z_j^{(2)}\right) = z_j^{(2)}$$

$$y = a_1^{(2)} = \sum_{k=1}^{2} w_{1k}^{(2)} \left(\sum_{l=1}^{2} w_{kl}^{(1)} x_l + b_k^{(1)} \right) + b_1^{(2)} = Ax_1 + Bx_2 + C$$

NN is a Universal Function Approximator

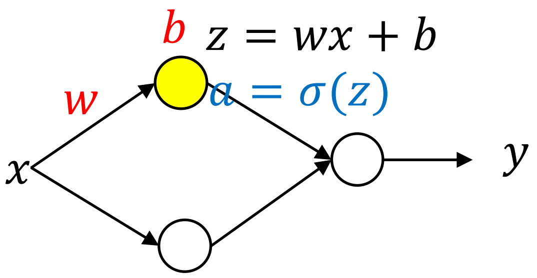
- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)

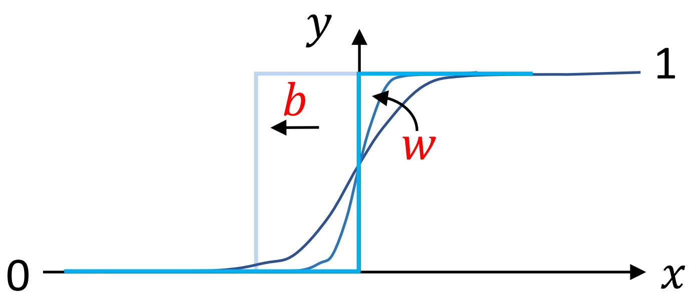


Goal:
Use a NN to approximate $f(x)$

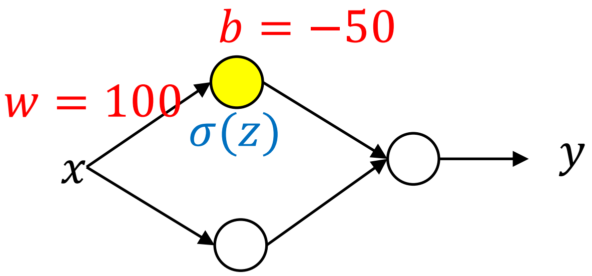
NN is a Universal Function Approximator

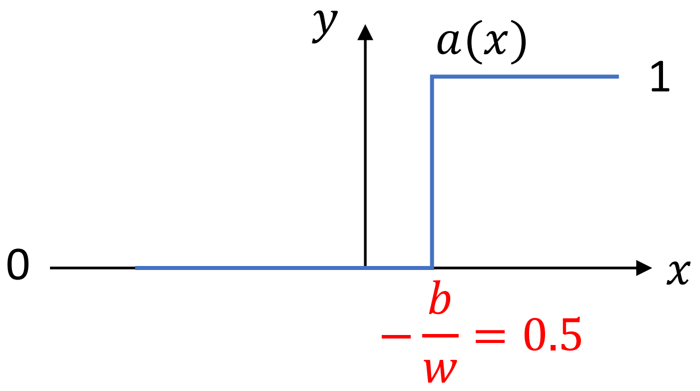
- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)





Large w gives a step

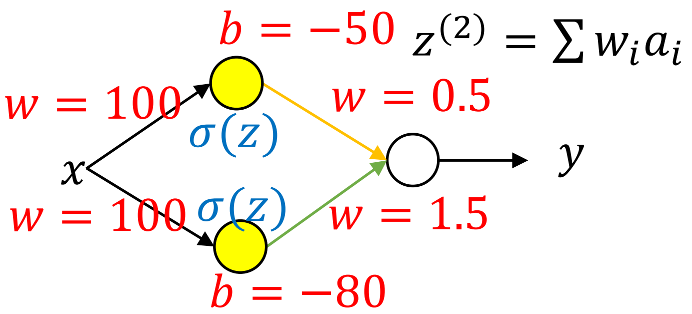


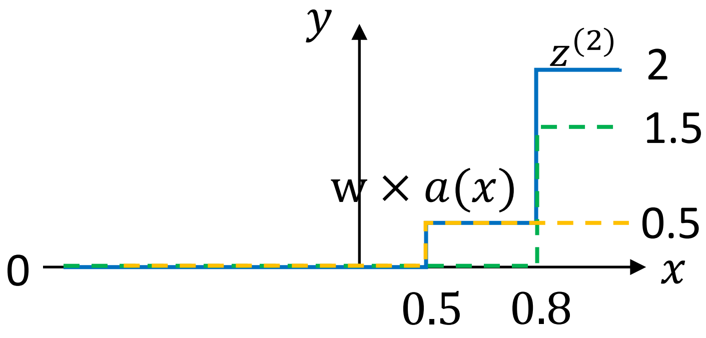


$-\frac{b}{w}$ determines the location of the step

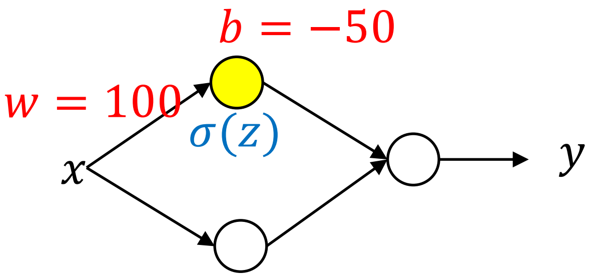
NN is a Universal Function Approximator

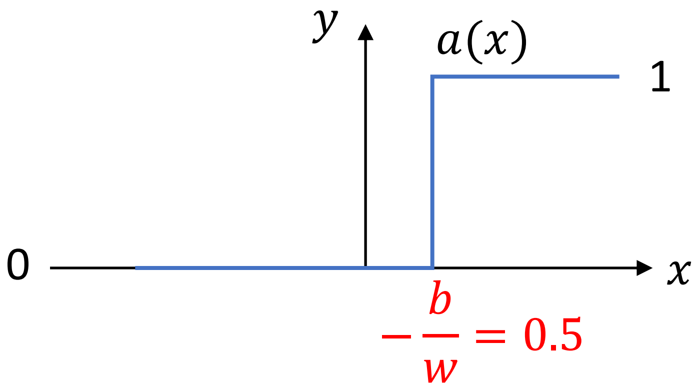
- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)





Superposition of two steps

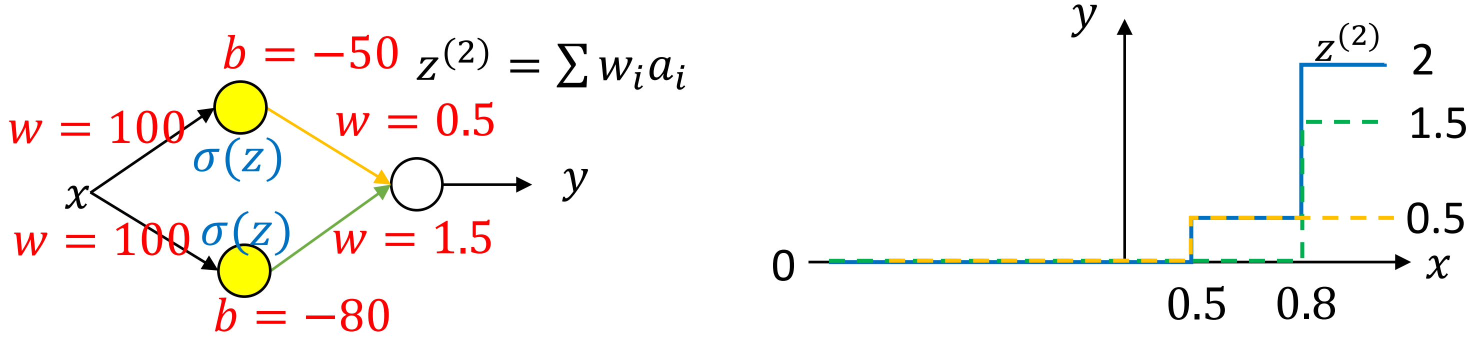




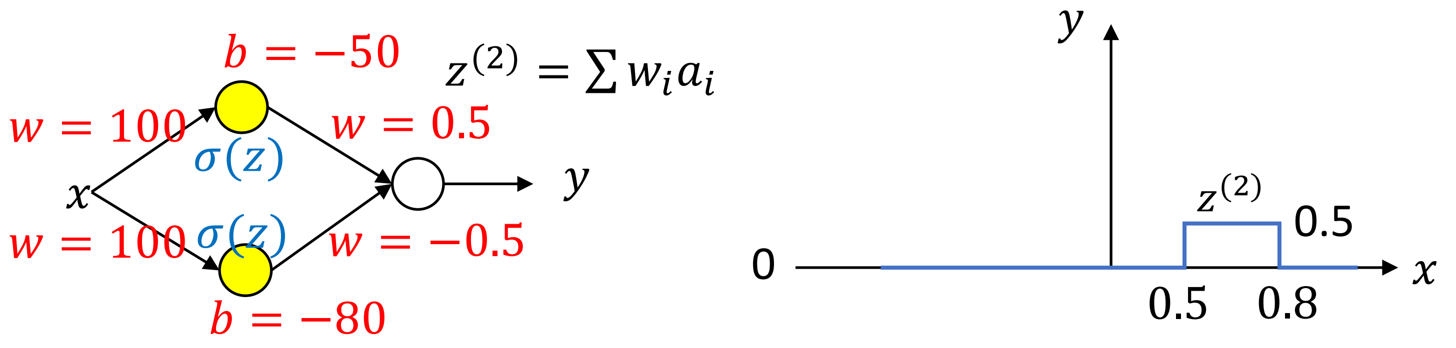
$-\frac{b}{w}$ determines the location of the step

NN is a Universal Function Approximator

- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)



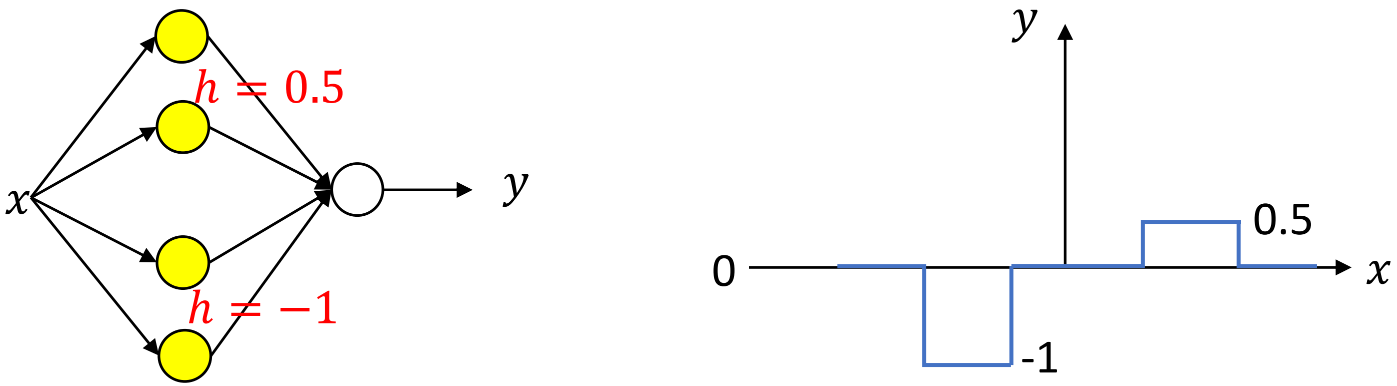
Superposition of two steps

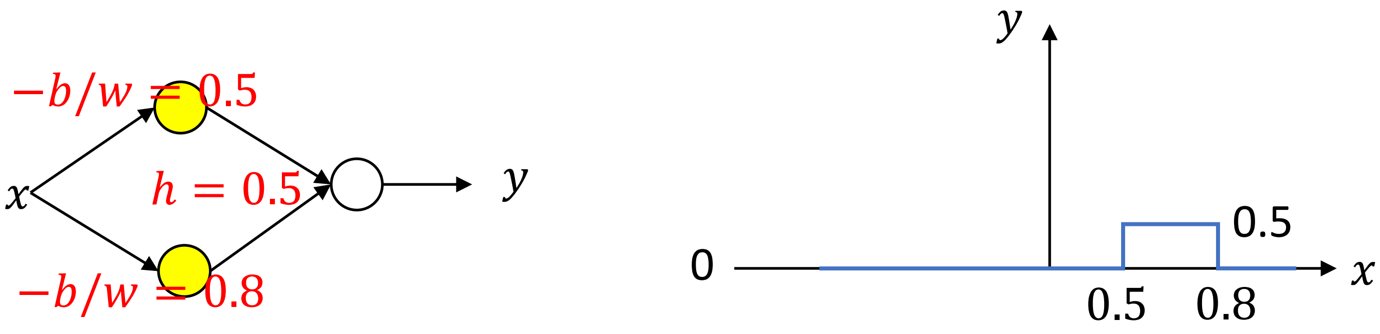


Create a column of height h=0.5

NN is a Universal Function Approximator

- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)

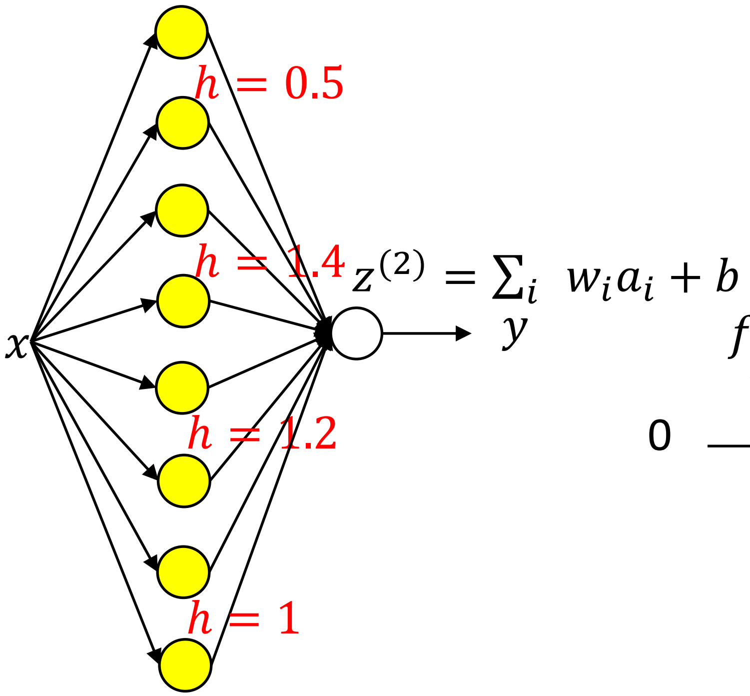


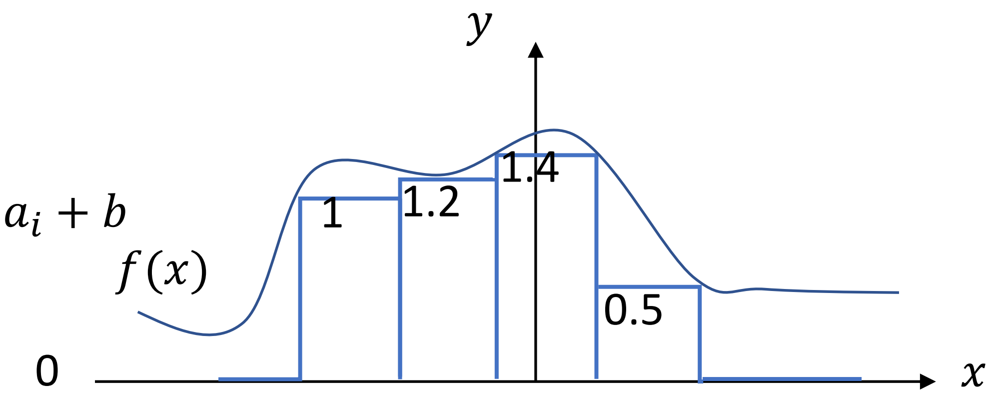


Create a column of height h=0.5

NN is a Universal Function Approximator

- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)

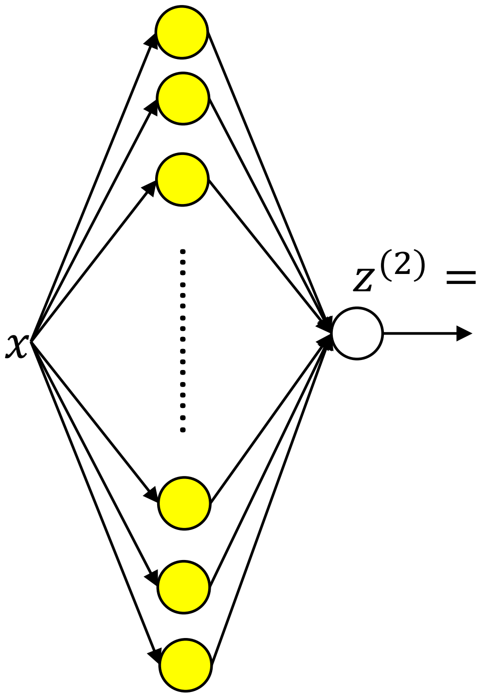




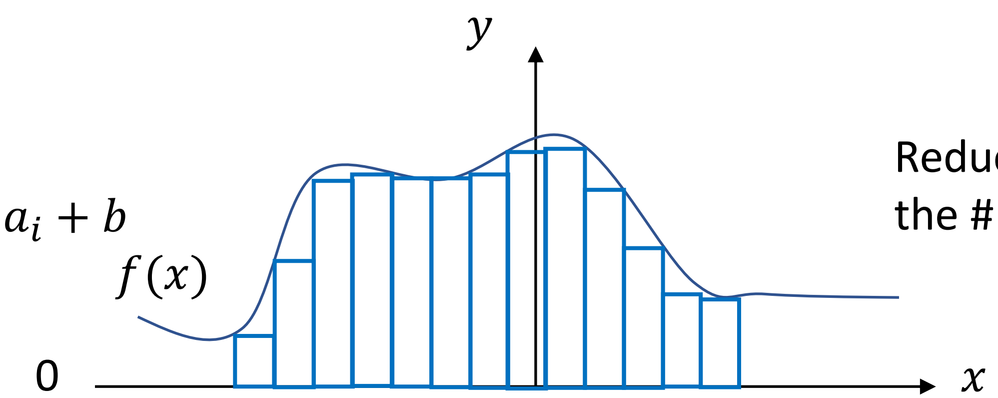
Superposition of columns

NN is a Universal Function Approximator

- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)



$z^{(2)} = \sum_i \; w_i a_i + b$



Reduce column widths + increasing the # of neurons to approximate $f(x)$

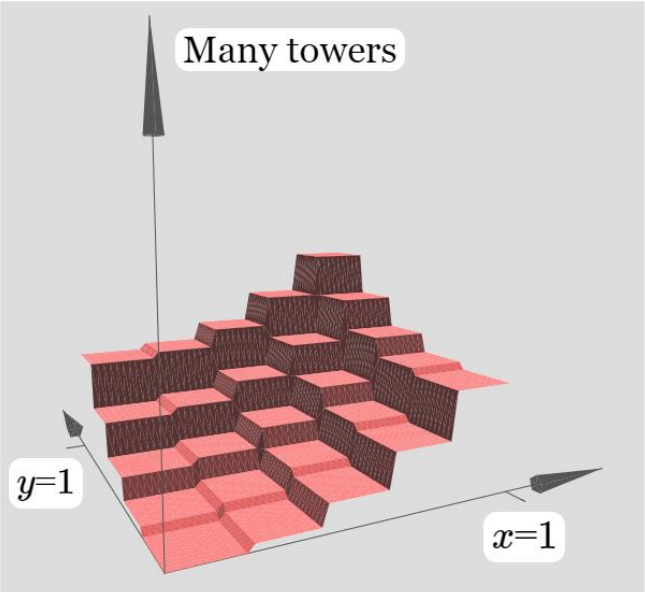
For a given continuous smooth $f(x)$ and an arbitrarily small $\epsilon > 0$
There exist $z^{(2)}$ so that

$$\int |z^{(2)}(x) - f(x)| \, dx < \epsilon$$

NN is a Universal Function Approximator

- Approximate a 2D surface

What should be the input/output units of NN?



http://neuralnetworksanddeeplearning.com/chap4.html

NN is a Universal Function Approximator

In summary

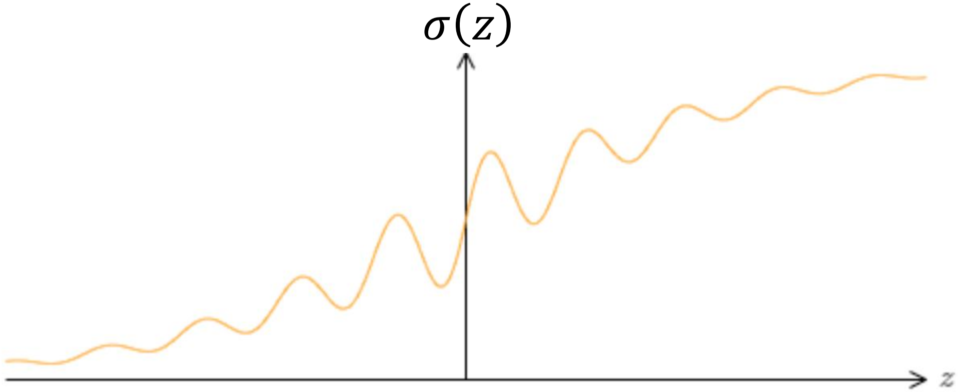
- Dimension of the approximated surface is determined by …

  Number of input units

- Step size is determined by …

  Number of hidden units

- Could the following activation function instead of a sigmoid function approximate a step?

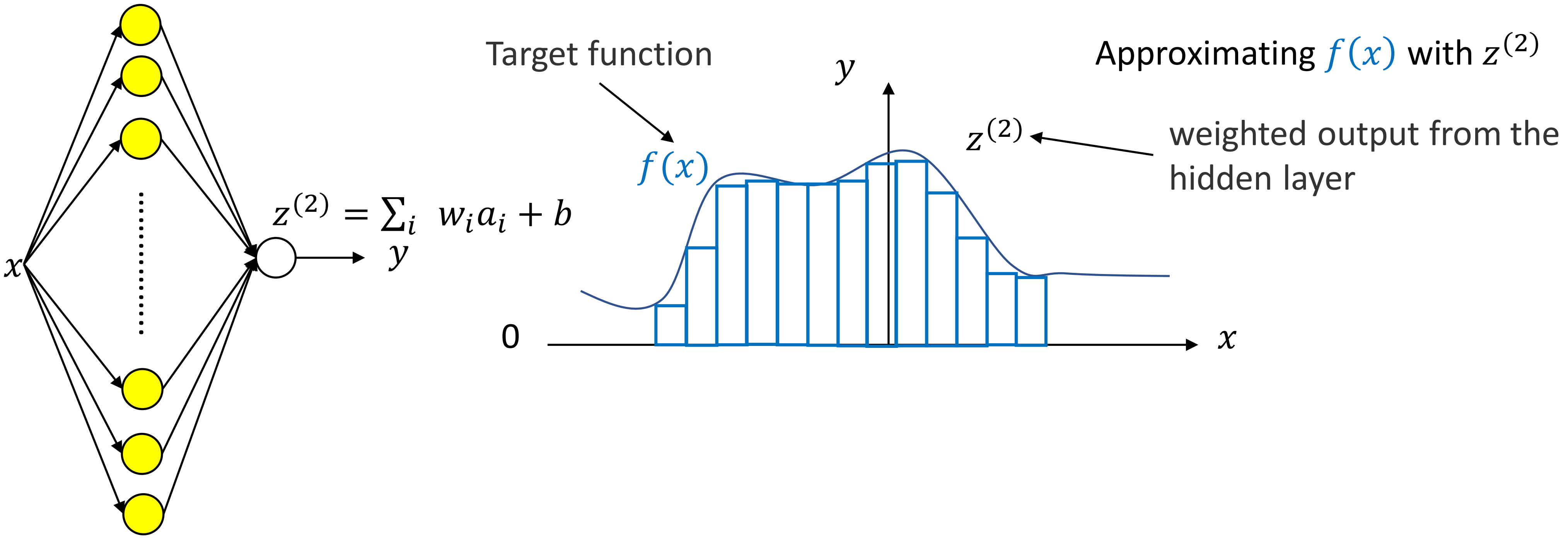


$\sigma(z) \to s_1, \; z \to \infty$
$\sigma(z) \to s_2, \; z \to -\infty$
$s_1 \neq s_2$

- Could activation function $\sigma(z) = z$ instead of a sigmoid function approximate a step?

NN is a Universal Function Approximator

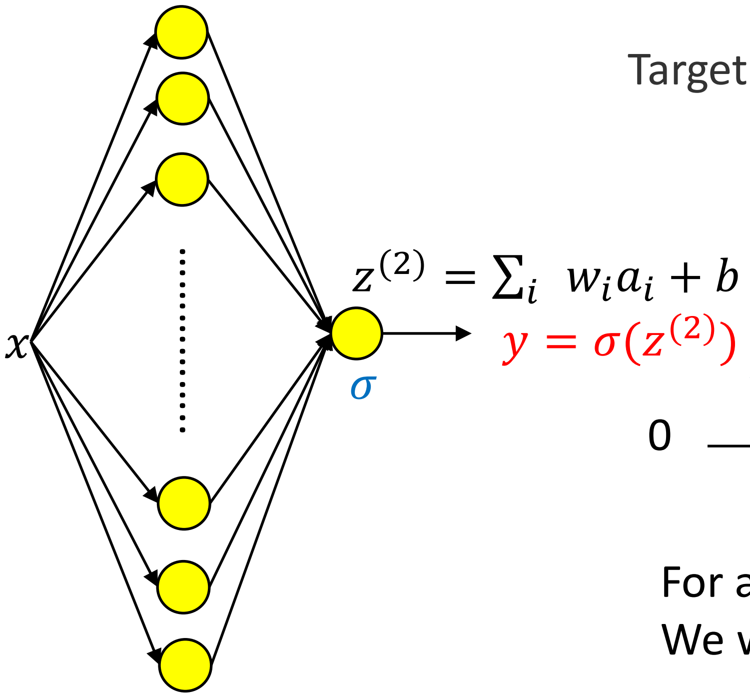
- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)

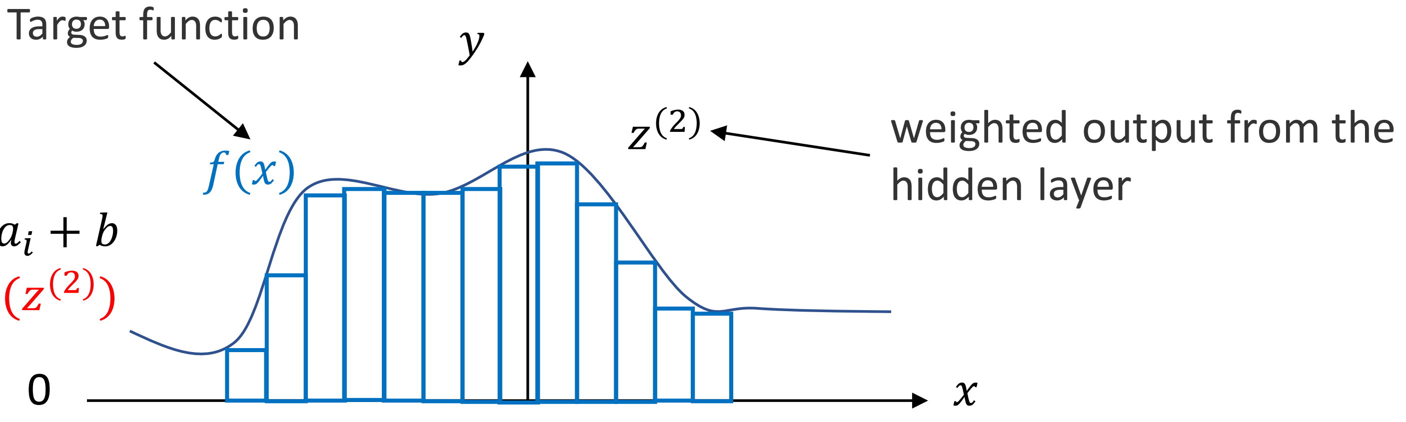


NN is a Universal Function Approximator

- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)

Instead of finding $z^{(2)} \approx f(x)$,
we want $y = \sigma(z^{(2)}) \approx f(x)$



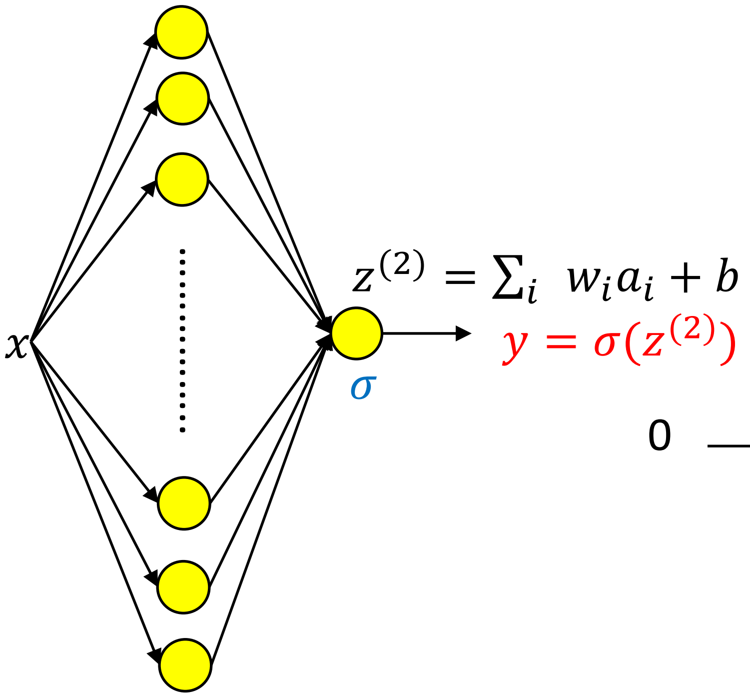


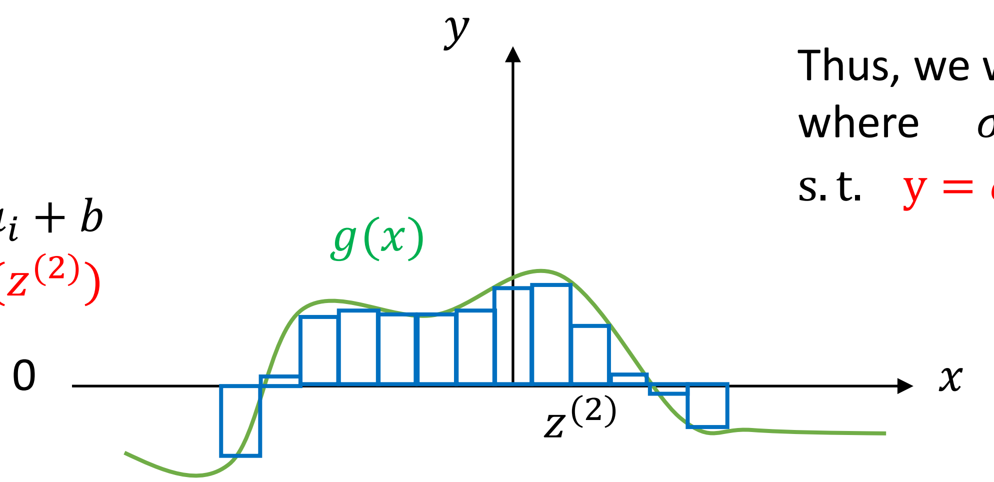
For a given continuous smooth $f(x)$ and an arbitrarily small $\epsilon > 0$
We want to find NN prediction $y(x)$ so that

$$\int |y(x) - f(x)|\, dx < \epsilon$$

NN is a Universal Function Approximator

- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)





Instead of finding $z^{(2)} \approx f(x)$,
we want $y = \sigma\left(z^{(2)}\right) \approx f(x)$

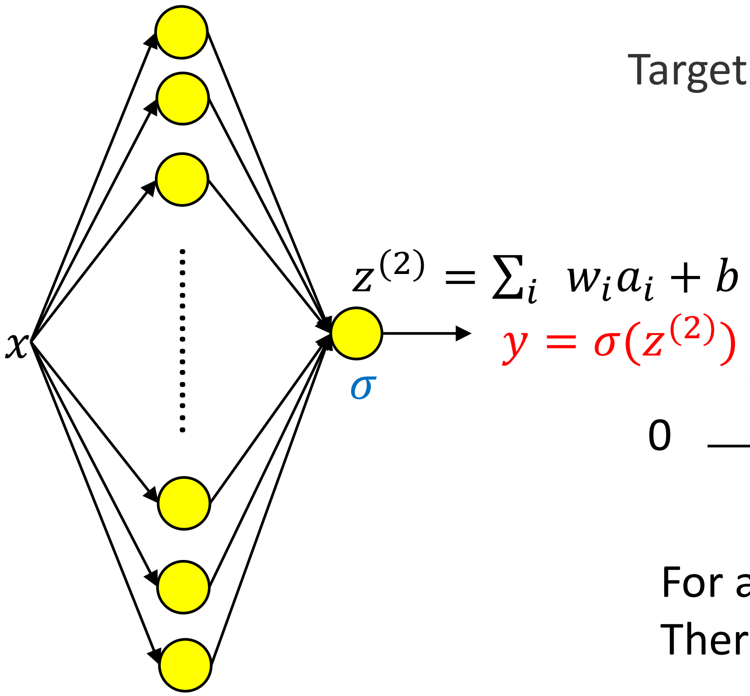
Thus, we want to find $z^{(2)} \approx g(x)$
where $\sigma(g(x)) = f(x)$
s.t. $y = \sigma\left(z^{(2)}\right) \approx \sigma(g(x)) = f(x)$

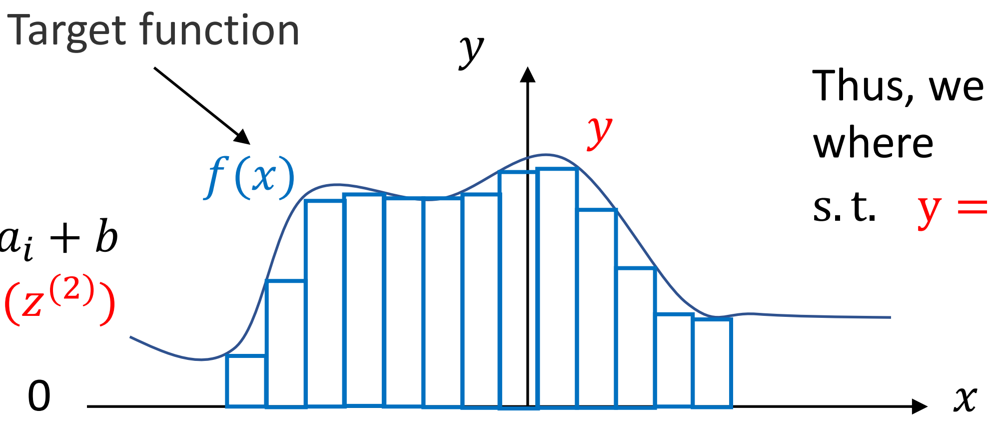
NN is a Universal Function Approximator

- NN can approximate continuous and smooth functions
  A visual proof (for sigmoid activation)

Instead of finding $z^{(2)} \approx f(x)$,
we want $y = \sigma\left(z^{(2)}\right) \approx f(x)$

Thus, we want to find $z^{(2)} \approx g(x)$
where $\sigma(g(x)) = f(x)$
s.t. $y = \sigma\left(z^{(2)}\right) \approx \sigma(g(x)) = f(x)$





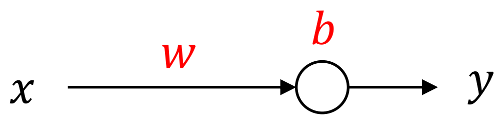
For a given continuous smooth $f(x)$ and an arbitrarily small $\epsilon > 0$
There exist NN prediction $y(x)$ so that

$$\int |y(x) - f(x)|\, dx < \epsilon$$

Now we know it is possible to tune weights and biases in a NN to approximate functions. We still need an automated method to find the correct weights and biases.

How to find w and b?

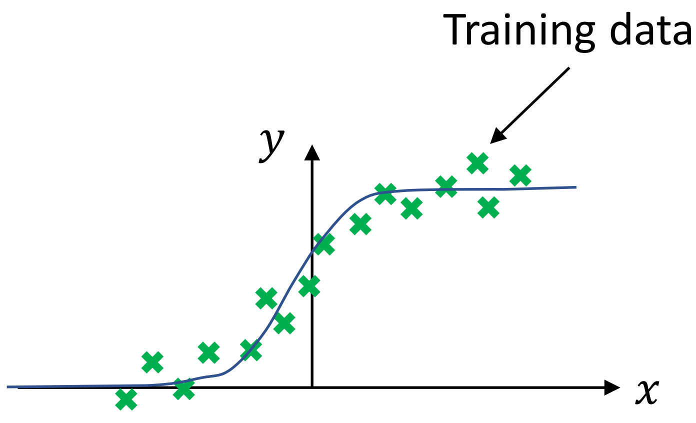
- E.g., Model $y = \sigma(wx + b)$



Given observations of $\{x_d^i, y_d^i\}_i^m$

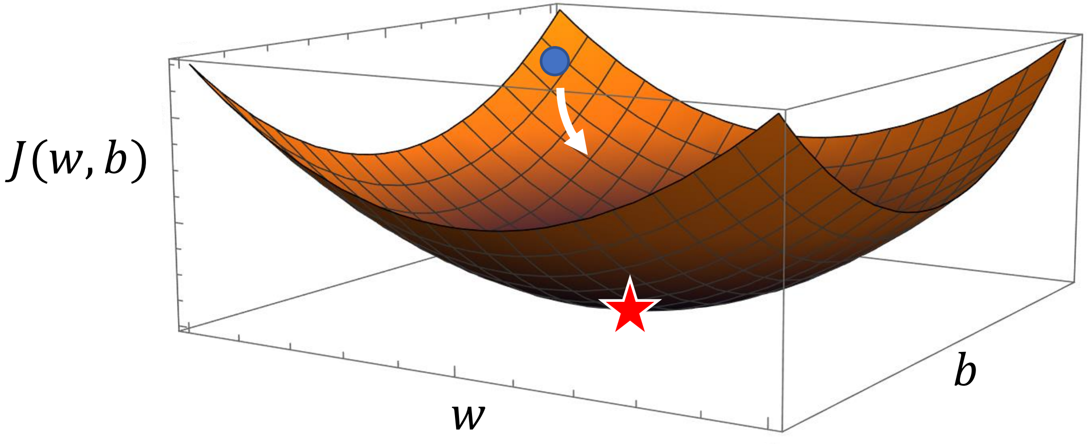
Our goal is to find the model parameters w, b that minimizes the cost function

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2$$



How to find w and b? Gradient descent

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \left(y\left(x_d^i\right) - y_d^i \right)^2$

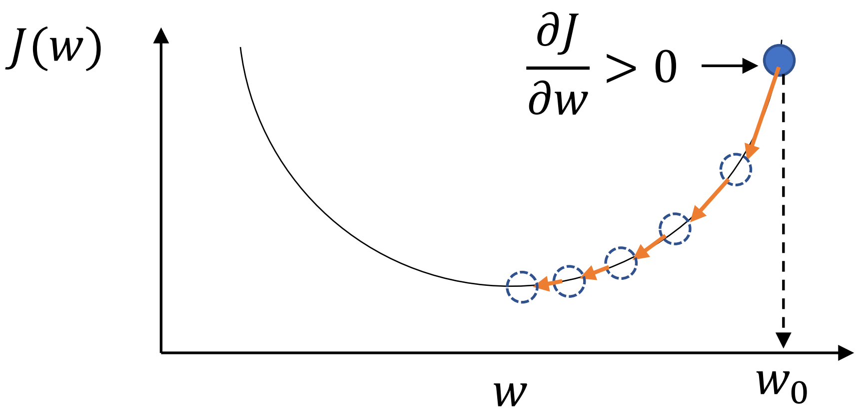


Find the w, b that minimizes $J(w, b)$

i.e. $\frac{\partial J}{\partial w} = \frac{\partial J}{\partial b} = 0$

How to find w and b? Gradient descent

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w, b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2$



I. calculate the slope $\frac{\partial J}{\partial w}$ corresponds to an initial w

II. Adjust w according to the local slope

$w_{new} = w_{old} - \alpha\,\frac{\partial J}{\partial w}$, $\alpha > 0$ is the learning rate

III. Iterate until $\frac{\partial J}{\partial w} = 0$

https://developers.google.com/machine-learning/crash-course/fitter/graph

Summary

- Universal function approximator
- Gradient descent: a method to find weights and biases that minimize J