Different approaches of differentiation

$$l_1 = x$$
$$l_{n+1} = 4l_n(1 - l_n)$$

$$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$$

Questions

Using python to write a function using for-loop and the idea of automatic differentiation that allow to compute the derivative of f(x) at any x

Try both ***forward*** and ***backward*** mode

```
def fdot(x):
    f = x
    df = 1
    for i = 1 to 3:
        f = # to do#
        df = # to do #
    return df
```

Different approaches of differentiation

$$l_1 = x$$
$$l_{n+1} = 4l_n(1 - l_n)$$

$$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$$

Forward mode $\quad \dot{l_i} = \dfrac{dl_i}{dx}$

$$\begin{aligned}
l_1 &= x & \dot{l_1} &= 1 \\
l_2 &= 4l_1(1 - l_1) & \dot{l_2} &= 4\dot{l_1} - 8l_1\dot{l_1} \\
l_3 &= 4l_2(1 - l_2) & \dot{l_3} &= 4\dot{l_2} - 8l_2\dot{l_2} \\
l_4 &= 4l_3(1 - l_3) & \dot{l_4} &= 4\dot{l_3} - 8l_3\dot{l_3} \\
&\vdots & &\vdots \\
y = l_n &= 4l_{n-1}(1 - l_{n-1}) & \dot{l_n} &= 4\dot{l_{n-1}} - 8l_{n-1}\dot{l_{n-1}}
\end{aligned}$$

```
def dydx(x,n):
    f = x
    df = 1
    for i = 1 to n:
        f = 4*f*(1-f)
        df = 4*df - 8*f*df
    return f, df
```

Different approaches of differentiation

$$l_1 = x$$
$$l_{n+1} = 4l_n(1 - l_n)$$

$$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$$

Backward mode $\quad \bar{l_i} = \dfrac{dy}{dl_i}$

$$l_1 = x \qquad \bar{l_1} = \frac{dl_2}{dl_1}\bar{l_2} = 4 - 8l_1$$
$$l_2 = 4l_1(1 - l_1) \qquad \bar{l_2} = \frac{dl_3}{dl_2}\bar{l_3} = 4 - 8l_2$$
$$l_3 = 4l_2(1 - l_2) \qquad \bar{l_3} = \frac{dl_4}{dl_3}\bar{l_4} = 4 - 8l_3$$
$$l_4 = 4l_3(1 - l_3)$$
$$\vdots \qquad \vdots$$
$$\bar{l_{n-1}} = \frac{dl_n}{dl_{n-1}}\bar{l_n} = 4 - 8l_{n-1}$$
$$y = l_n = 4l_{n-1}(1 - l_{n-1}) \qquad \bar{l_n} = 1$$

```
def fdot(x,n):
    f[0] = x
    for i = 1 to n-1:
        f [i]= 4*f*(1-f)

    for i = 0 to n-1:
        df [i]= 4-8f[n-i]
    return f[n], df[n]
```

Differential equation

Neural network → Physics-informed neural network

Back propagation

?

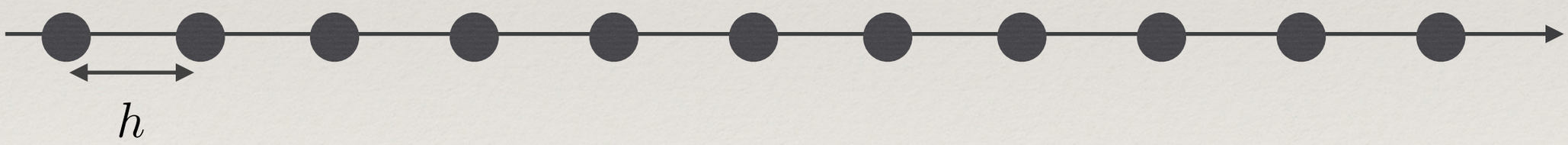
Automatic differentiation

Differential equation

Difficulty $\boxed{\dfrac{du}{dx}} = u$ Boundary condition $u(0) = 1$

Differential equation ⟶ Algebraic equation

Numerical method: **Finite difference**



$$\frac{du}{dx} = \frac{u(x_n) - u(x_{n-1})}{h}$$

$$\frac{du}{dx} = u \quad \Longrightarrow \quad \frac{u(x_n) - u(x_{n-1})}{h} = u(x_{n-1})$$

$$x_{n+1} = x_n + h$$

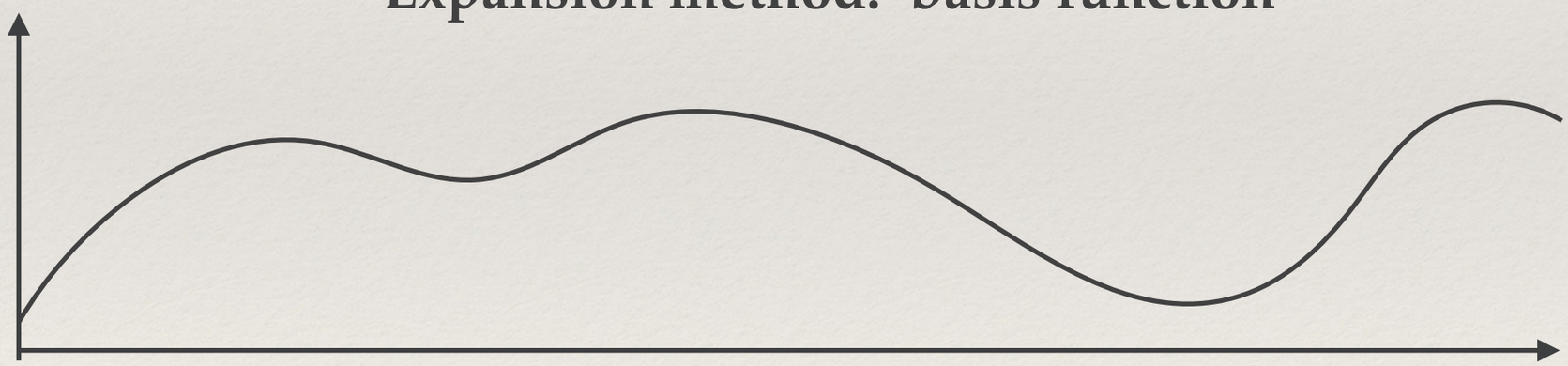
$$u(x_n) = (1+h)u(x_{n-1}) = (1+h)^n u(x_0)$$

Differential equation

Difficulty $\boxed{\frac{du}{dx}} = u$ Boundary condition $u(0) = 1$

Differential equation ⟶ Algebraic equation

Expansion method: basis function



$u(x, a_n) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... = \sum_{n=0}^{N} a_n x^n$ Power series expansion

$u(x, a_n, b_n) = \sum_{n=0}^{N} a_i \cos(nx) + b_i \cos(nx)$ Fourier series expansion

1-layer network

Known derivative

$$\frac{du}{dx} = u \qquad \text{Boundary condition} \qquad u(0) = 1 \quad \longrightarrow \quad u = e^x$$

Power series expansion $$u(x, a_n) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... = \sum_{i=0}^{\infty} a_n x^n$$

$$\frac{du}{dx}(x, a_n) = a_1 + 2a_2 x + 3a_3 x^2 + ... = \sum_{n=1}^{\infty} n a_n x^{n-1}$$

$$\sum_{n=1}^{\infty} n a_n x^{n-1} = \sum_{n=0}^{\infty} a_n x^n \qquad \text{should hold for every real value of x}$$

$$\sum_{n=0}^{\infty} (n+1) a_{n+1} x^n - \sum_{n=0}^{\infty} a_n x^n = 0$$

Coefficient equation

$$x^0 : \quad a_1 - a_0 = 0$$

$$x^1 : \quad 2a_2 - a_1 = 0$$

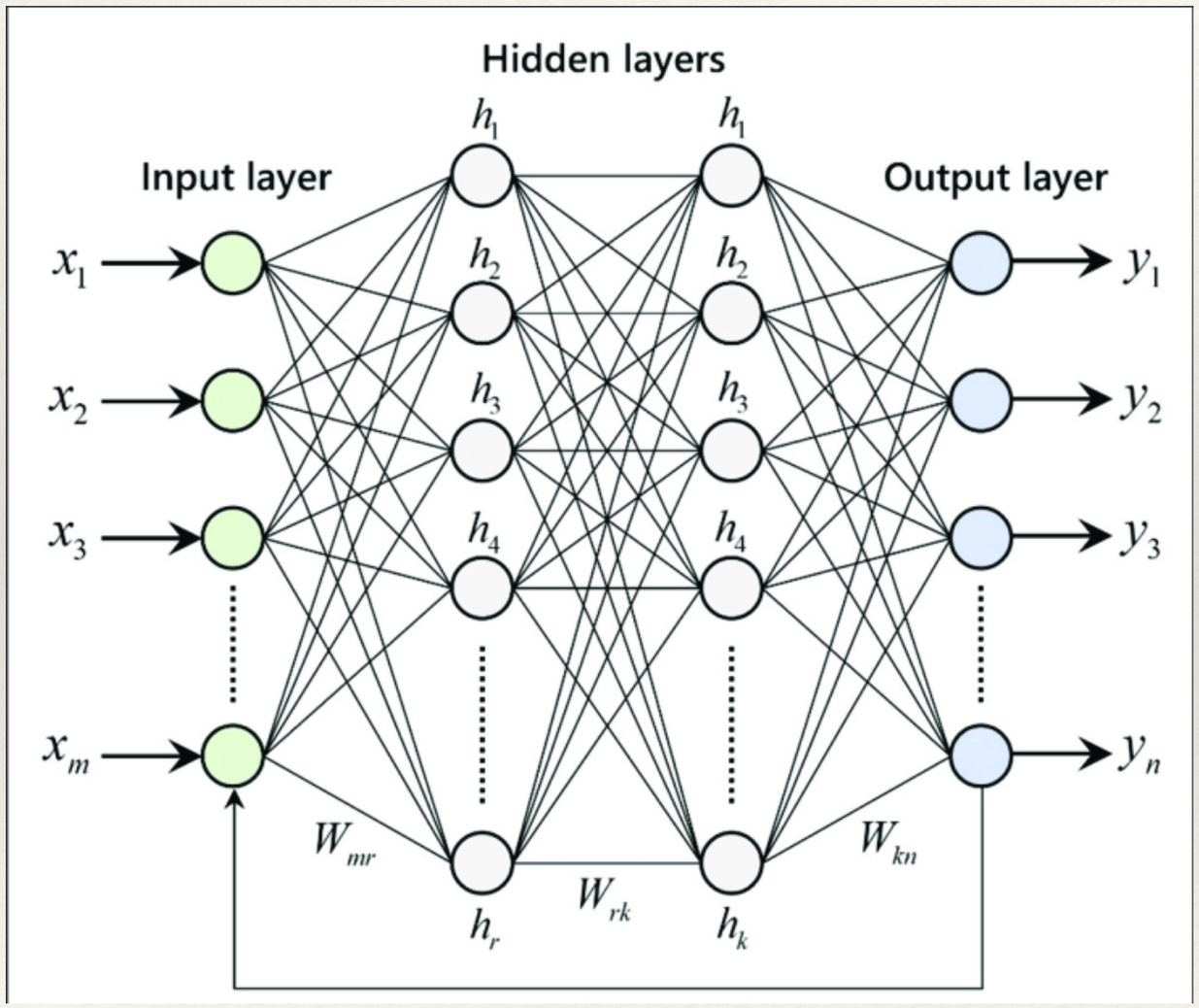
$$x^n : \quad n a_n - a_{n-1} = 0 \quad \Longrightarrow \quad a_n = \frac{a_{n-1}}{n} = \frac{a_0}{n!} = \frac{1}{n!}$$

Given the boundary condition:

$$u(0) = a_0 = 1$$

Final solution $$u(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... = e^x$$

Weight initialization



$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

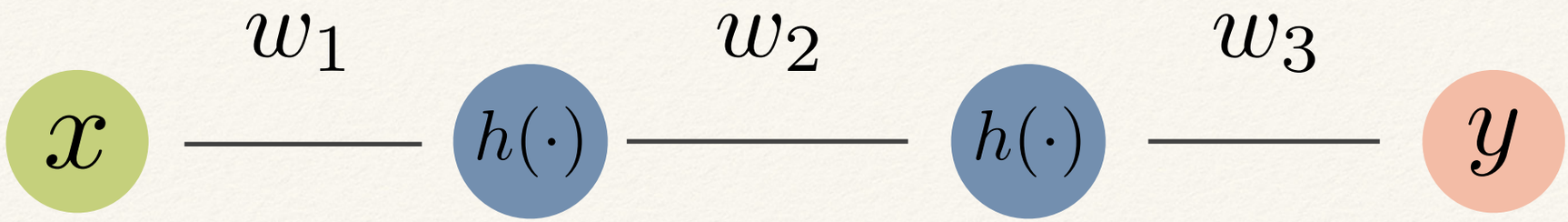
How to initialize the weight ?

$w_{jk}^{(n)} = 0$ all be **zero** ?

$w_{jk}^{(n)} = c$ all be the **same** ?

What are their magnitude ?

$w_{jk}^{(n)} \sim O(0.01)$ $w_{jk}^{(n)} \sim O(1000)$



$$a_1 = x \qquad a_2 = h(w_1 \cdot a_1) \qquad a_3 = h(w_2 \cdot a_2) \qquad y = w_3 a_3$$

Entire neural network

$$y = w_3 \cdot h[w_2 \cdot h(w_1 \cdot x)]$$

Loss function: mean squared error

$$J(y(x, w)) = (y(x, w) - y_0)^2$$

$$\frac{dJ}{dw_2} = 2(y - y_0)\frac{dy}{dw_2} = 2(y - y_0)w_3\frac{da_3}{dw_2} = 2(y - y_0) \cdot \boxed{w_3} h'(w_2 a_2) \cdot a_2$$

$$\frac{dJ}{dw_1} = 2(y - y_0)\frac{dy}{dw_1} = 2(y - y_0)w_3\frac{da_3}{dw_1} = 2(y - y_0) \cdot \boxed{w_3} h'(w_2 a_2) \cdot \boxed{w_2}\frac{da_2}{dw_1}$$

$w_{jk}^{(n)} = 0$ initialized to be all **zero**

$$\frac{dJ}{dw_2} = 0 \qquad \frac{dJ}{dw_1} = 0$$

Layers = [1, 10, 10, 1] Iteration = 4000

Weights at 1 —> 2

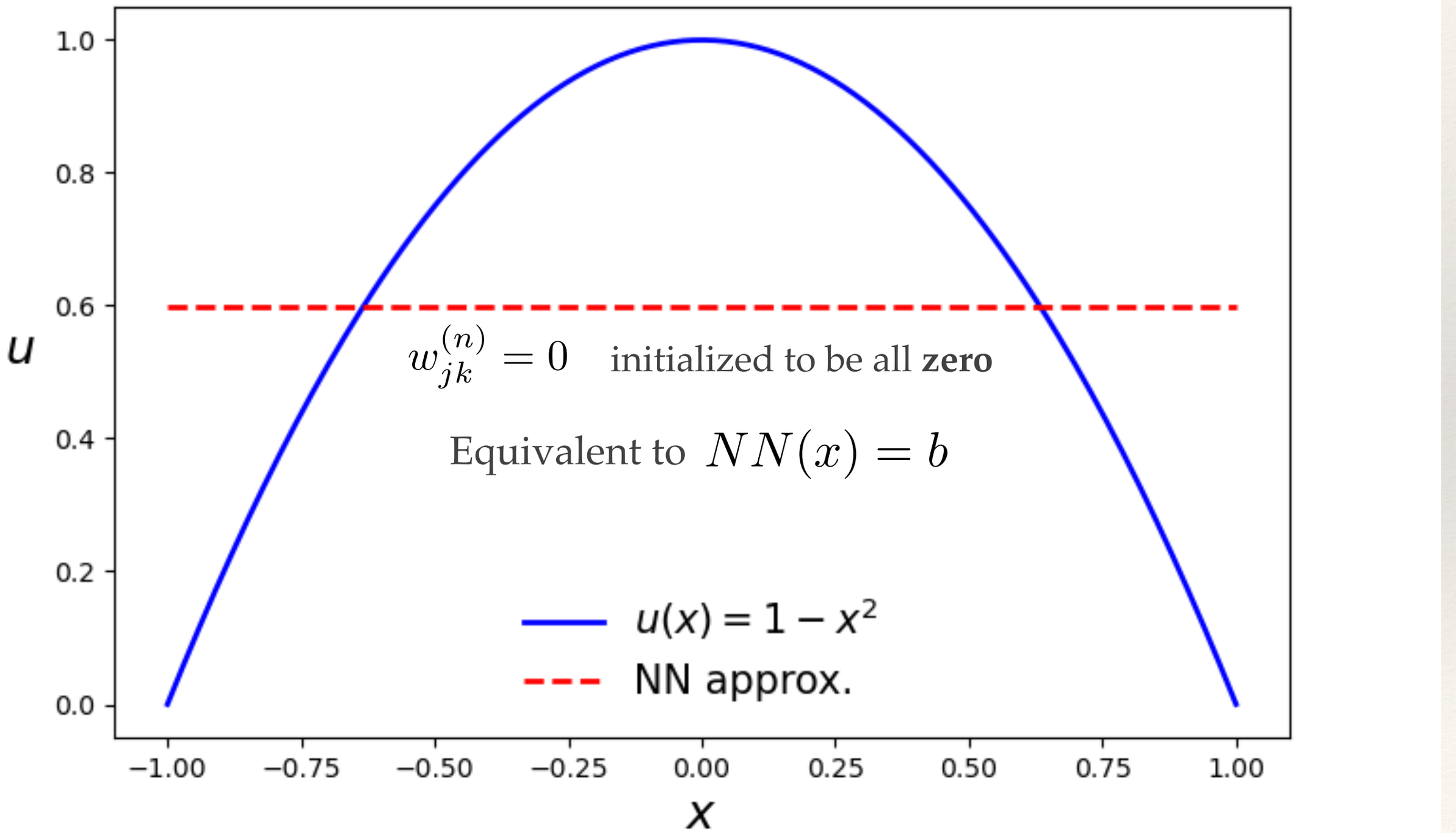
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

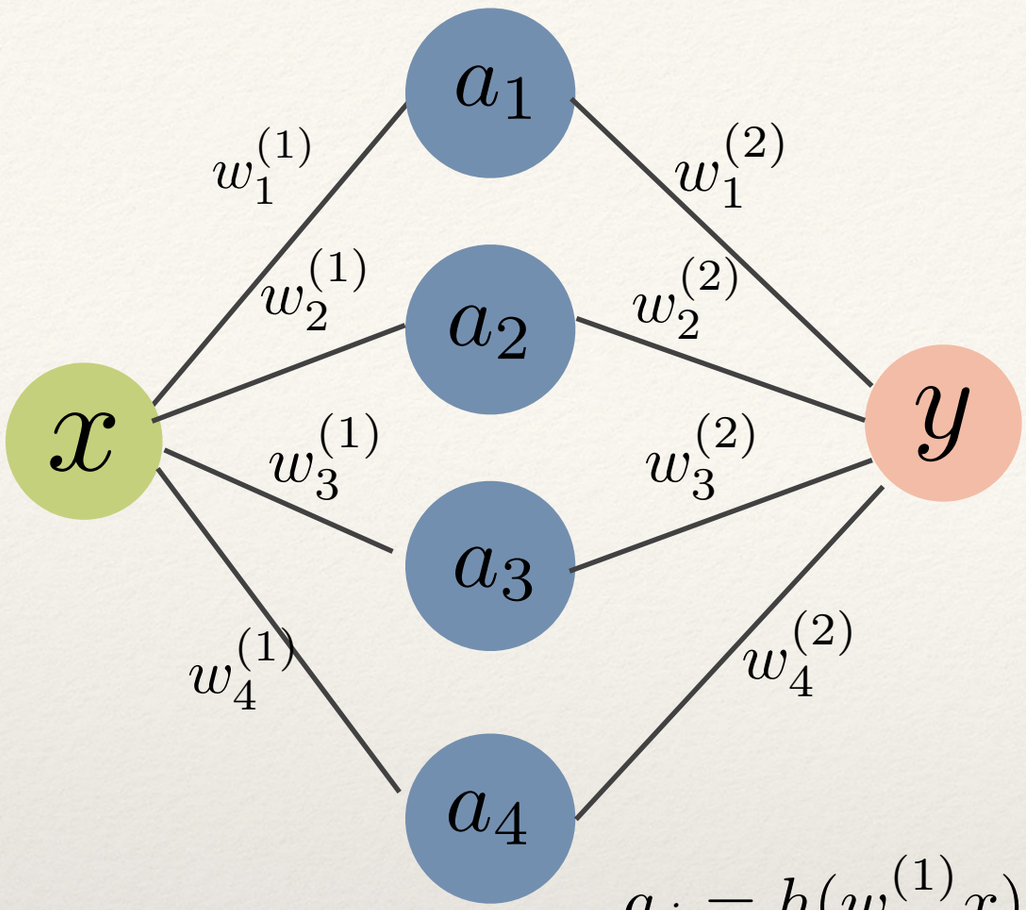
Weights at 3 —> 4

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

Weights at 2 —> 3

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```





$$a_j = h(w_j^{(1)} x)$$

$$w_{jk}^{(n)} = c \qquad \text{all be the } \textbf{same} \text{ ?}$$

Loss function: mean squared error

$$J(y(x,w)) = (y(x,w) - y_0)^2$$

$$\frac{dJ}{dw_j^{(2)}} = 2(y - y_0)\frac{dy}{dw_j^{(2)}} = 2(y - y_0)a_j \qquad \text{also be the } \textbf{same} \text{ for all j}$$

$$\frac{dJ}{dw_j^{(1)}} = 2(y - y_0)\frac{dy}{dw_j^{(1)}} = 2(y - y_0)w_j^{(2)}\frac{da_j}{dw_j^{(1)}} \qquad \text{also be the } \textbf{same} \text{ for all j}$$

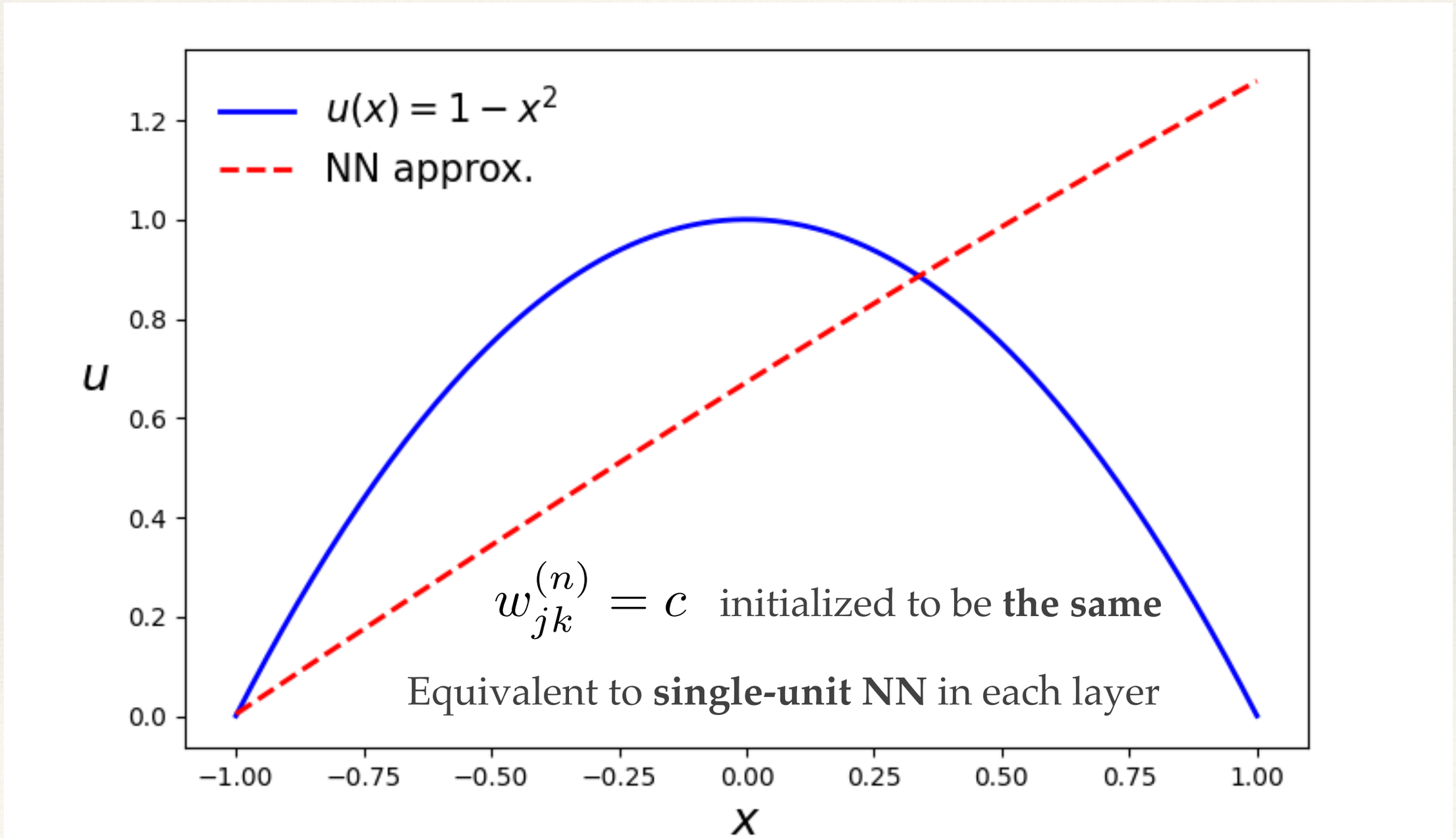
Layers = [1, 10, 1] Iteration = 4000

Weights at 1 —> 2

```
array([[0.2296614, 0.2296614, 0.2296614, 0.2296614, 0.2296614, 0.2296614,
        0.2296614, 0.2296614, 0.2296614, 0.2296614]], dtype=float32)
```

Weights at 2 —> 3

```
array([[0.3037359, 0.3037359, 0.3037359, 0.3037359, 0.3037359, 0.3037359,
        0.3037359, 0.3037359, 0.3037359, 0.3037359]], dtype=float32)
```

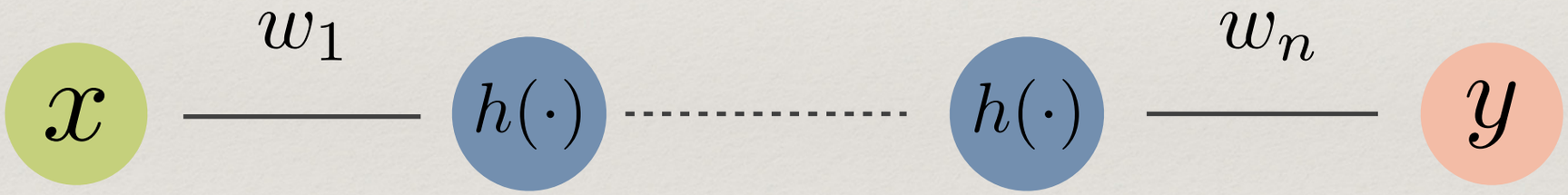


Weight initialization $w_{jk}^{(n)}$ should be **different** to break the symmetry

$w_{jk}^{(n)}$ follow Gaussian distribution $N(\mu, \sigma^2)$

What are their magnitude ?

$w_{jk}^{(n)} \sim O(0.01)$ $w_{jk}^{(n)} \sim O(1000)$



$h(x) = x$

Gradient **vanishing**

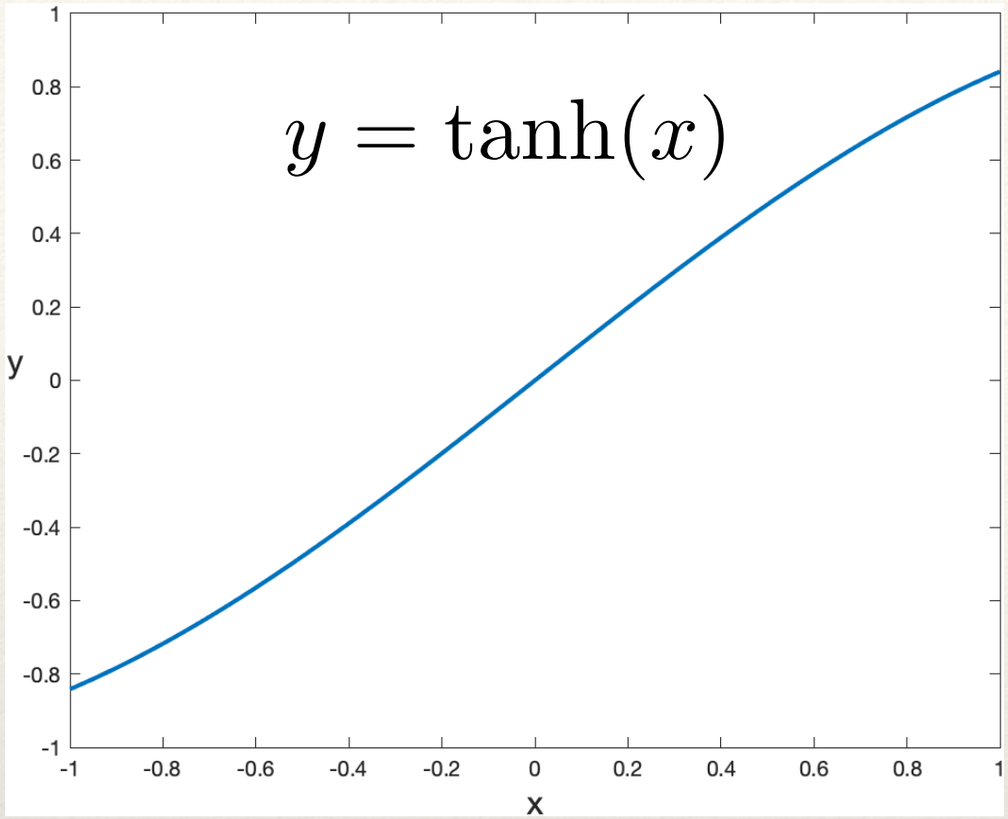
$$\frac{dJ}{dw_1} = 2(y - y_0)\frac{dy}{dw_1} = 2(y - y_0) \cdot (w_n \cdot w_{n-1} .... \cdot w_1)x$$

$= 2(y - y_0)2^n x \qquad w_j = 2$

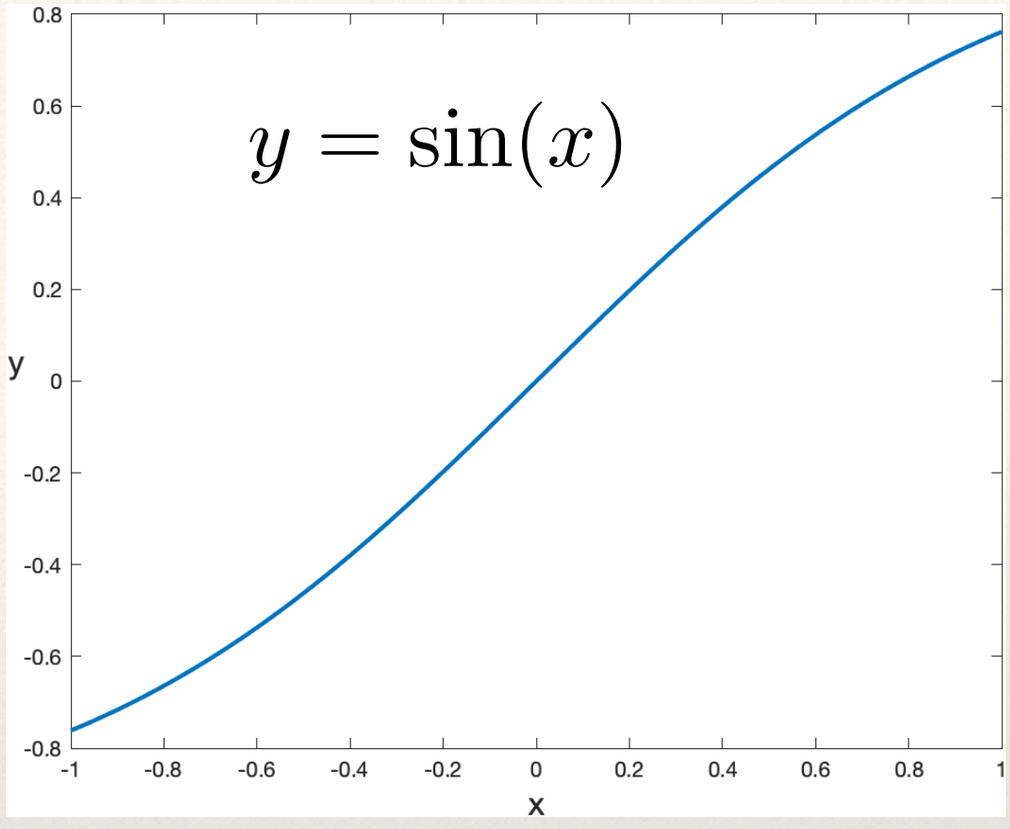
$= 2(y - y_0)\frac{1}{2^n}x \qquad w_j = 0.5$

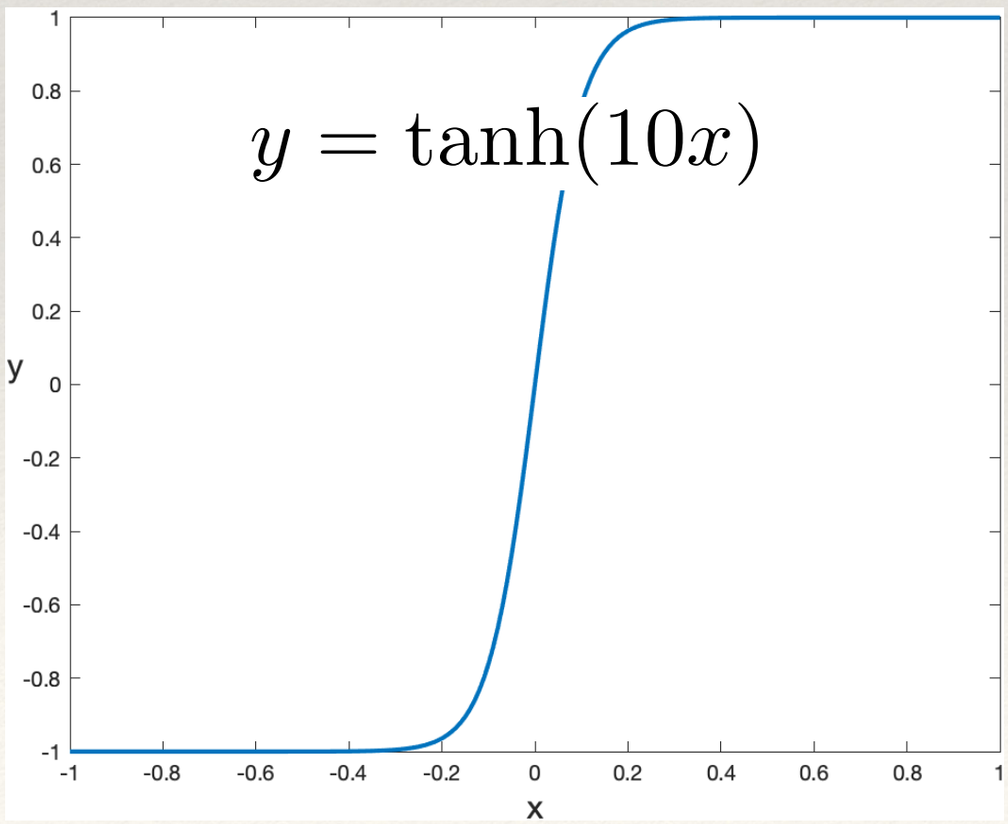
Gradient **exploding**

Activation function

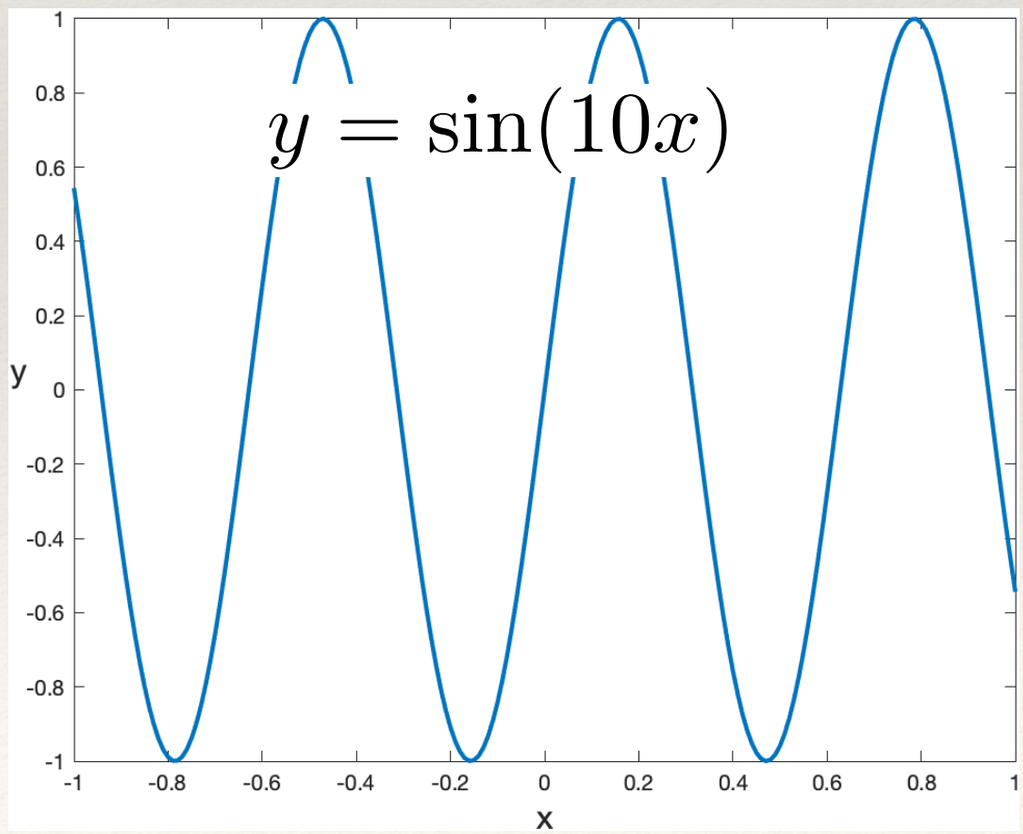


$w \sim O(1)$



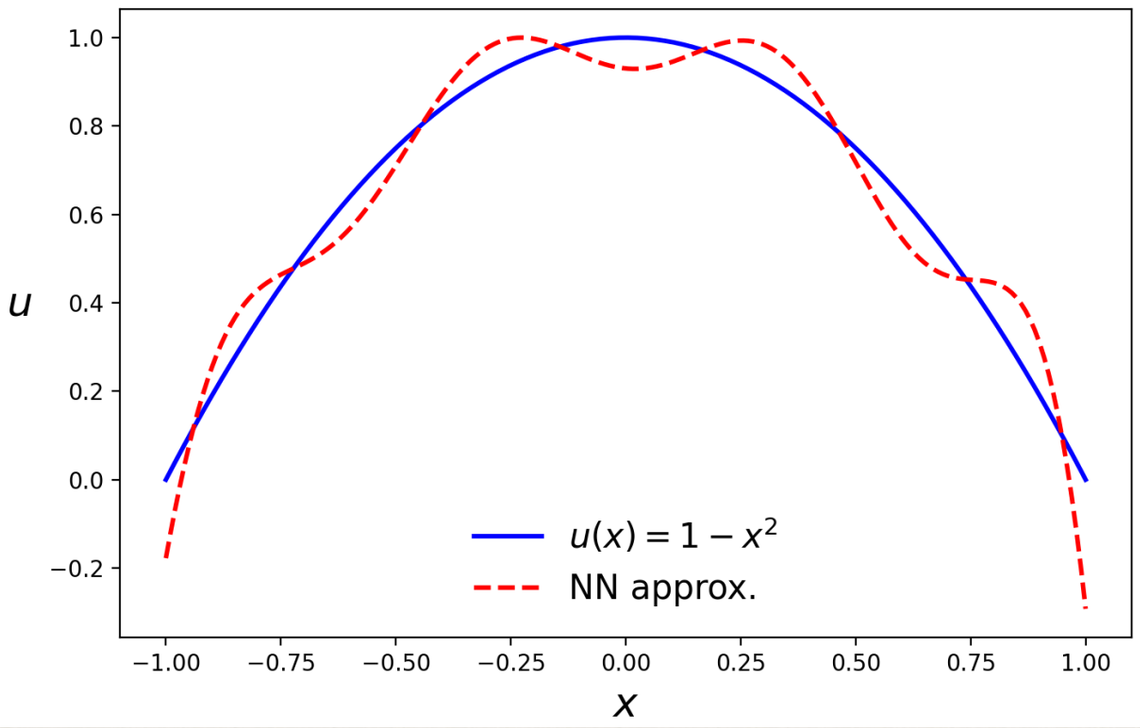


$w \sim O(10)$

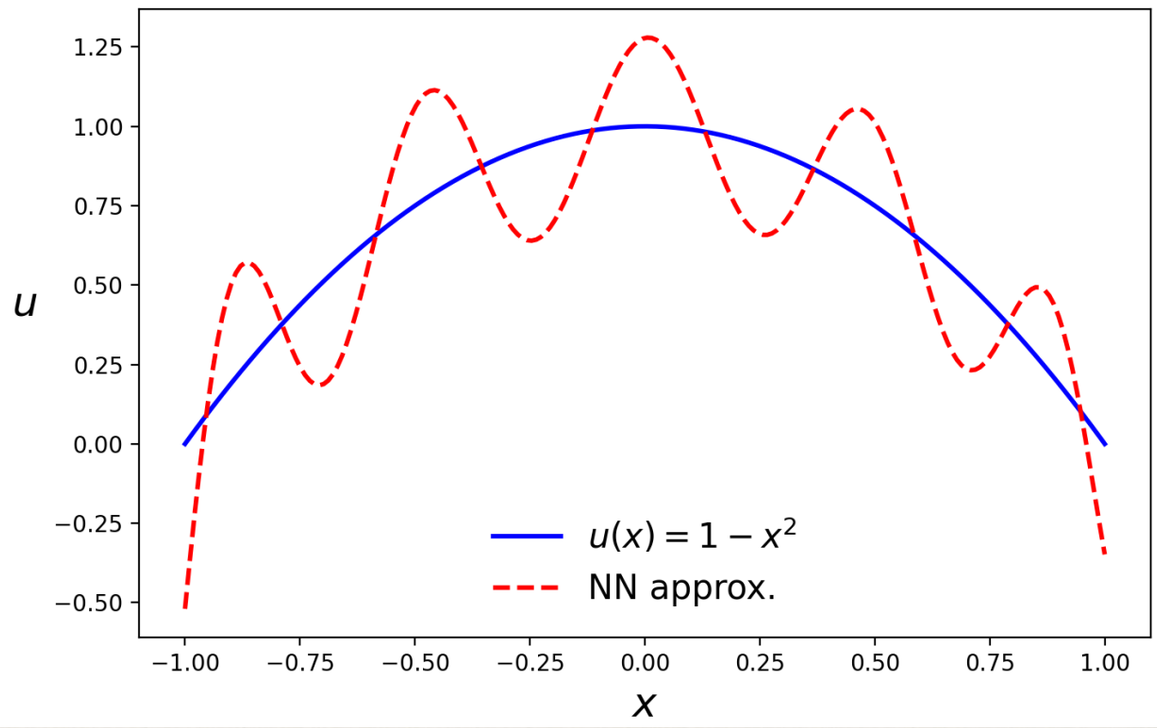


Layers = [1, 10, 1] Iteration = 4000

$w \sim O(10)$

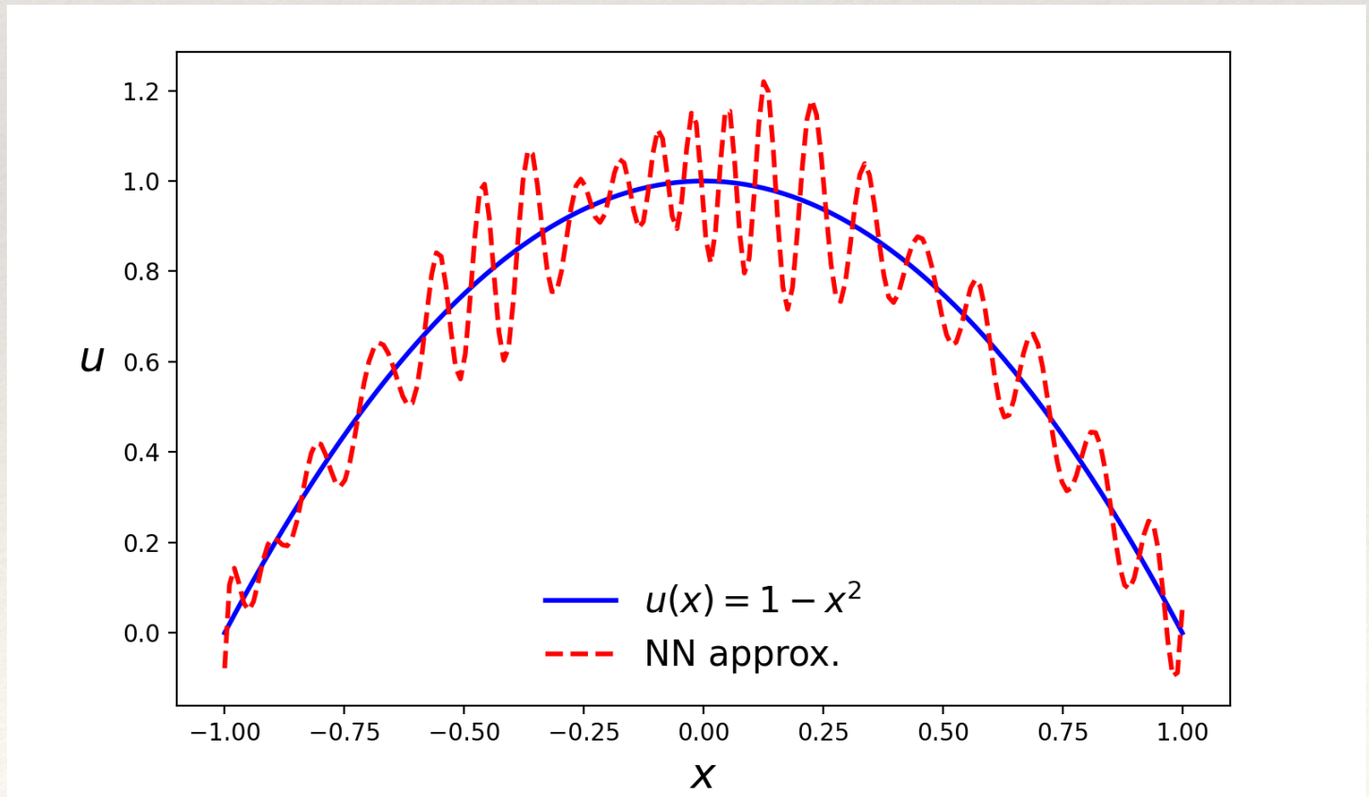


$w \sim O(20)$



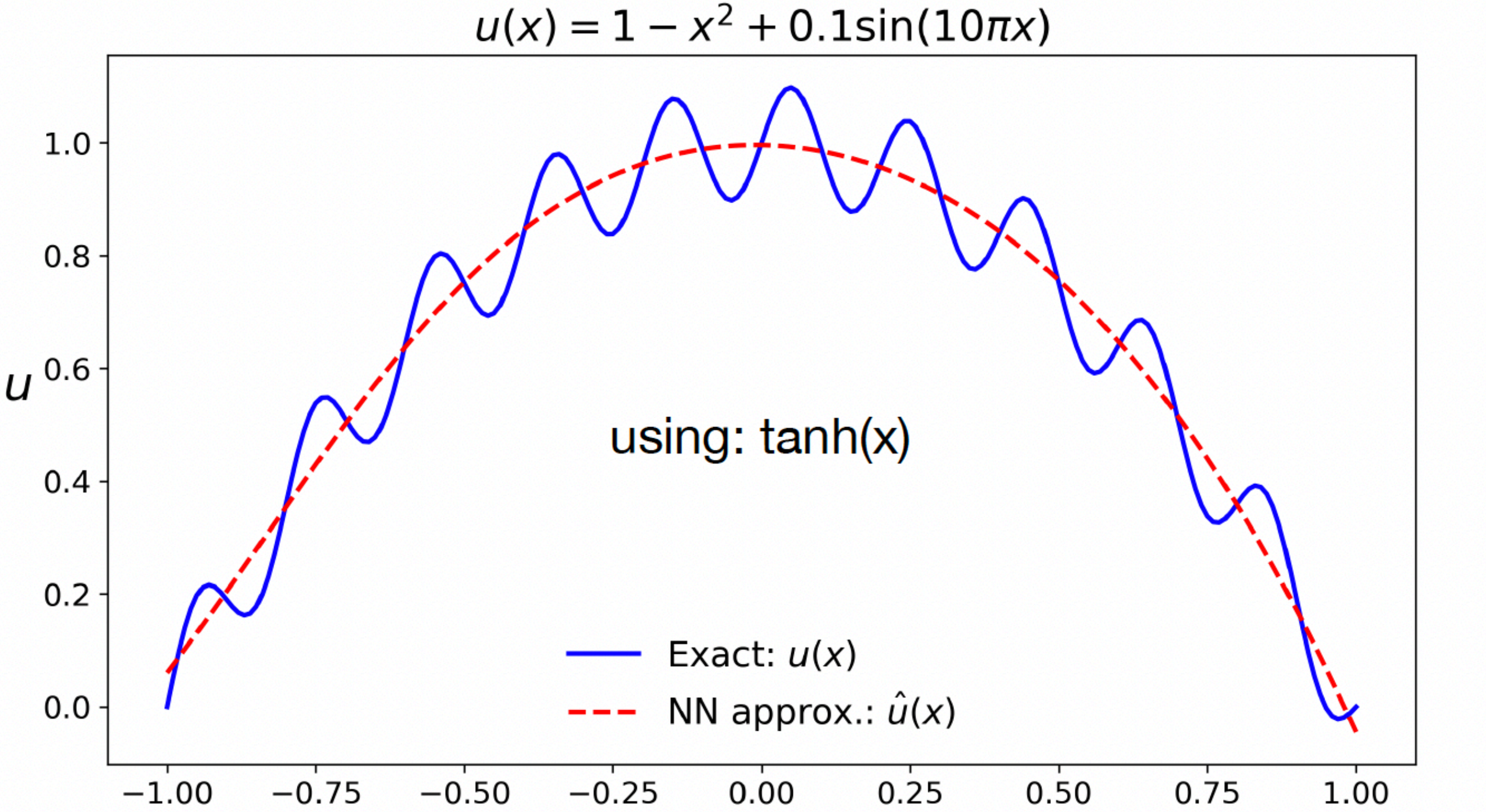
Layers = [1, 10, 10, 1]

$w \sim O(10)$



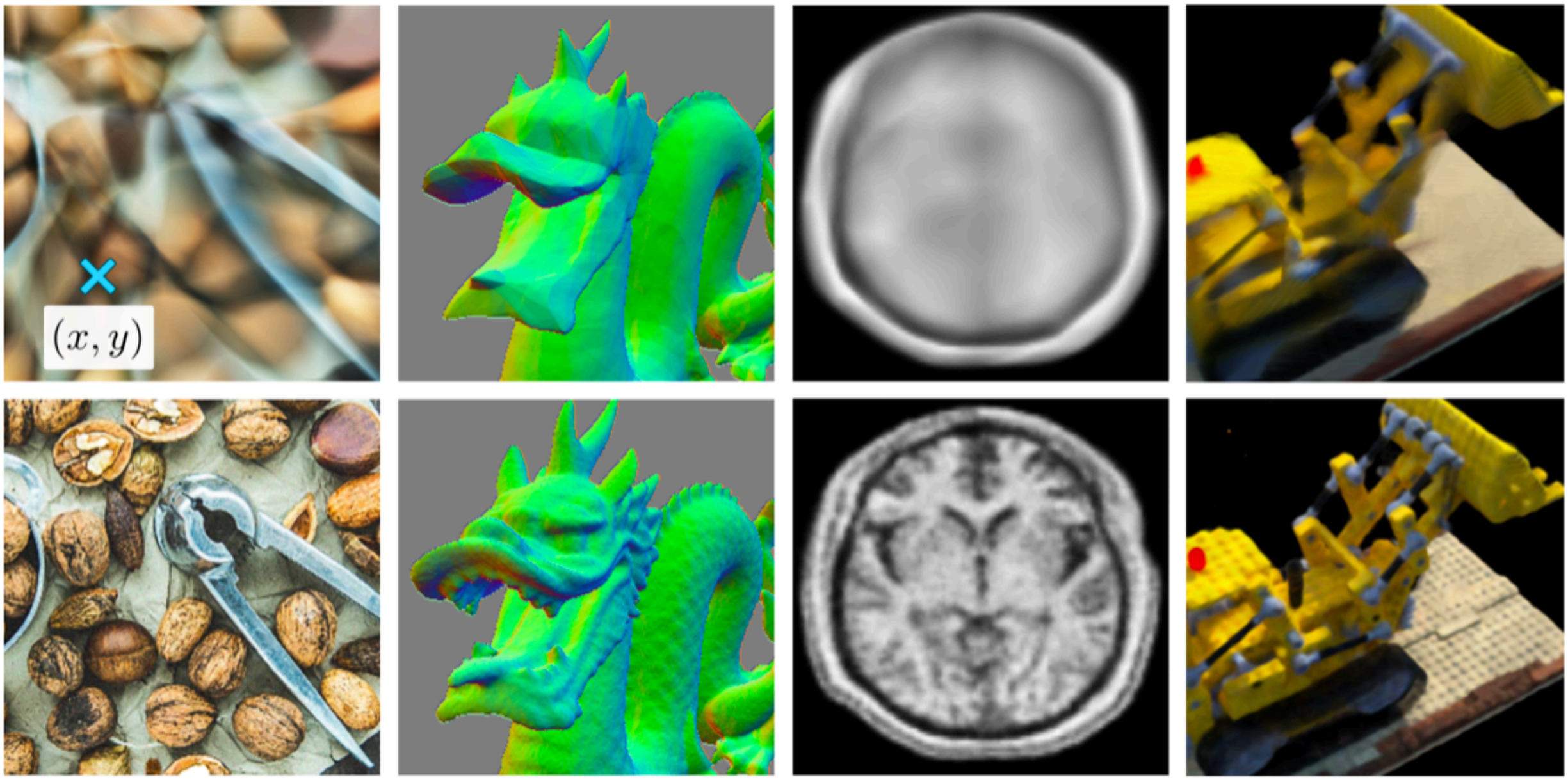
Layers = [1, 10, 1] Iteration = 4000

$$w \sim O(1)$$



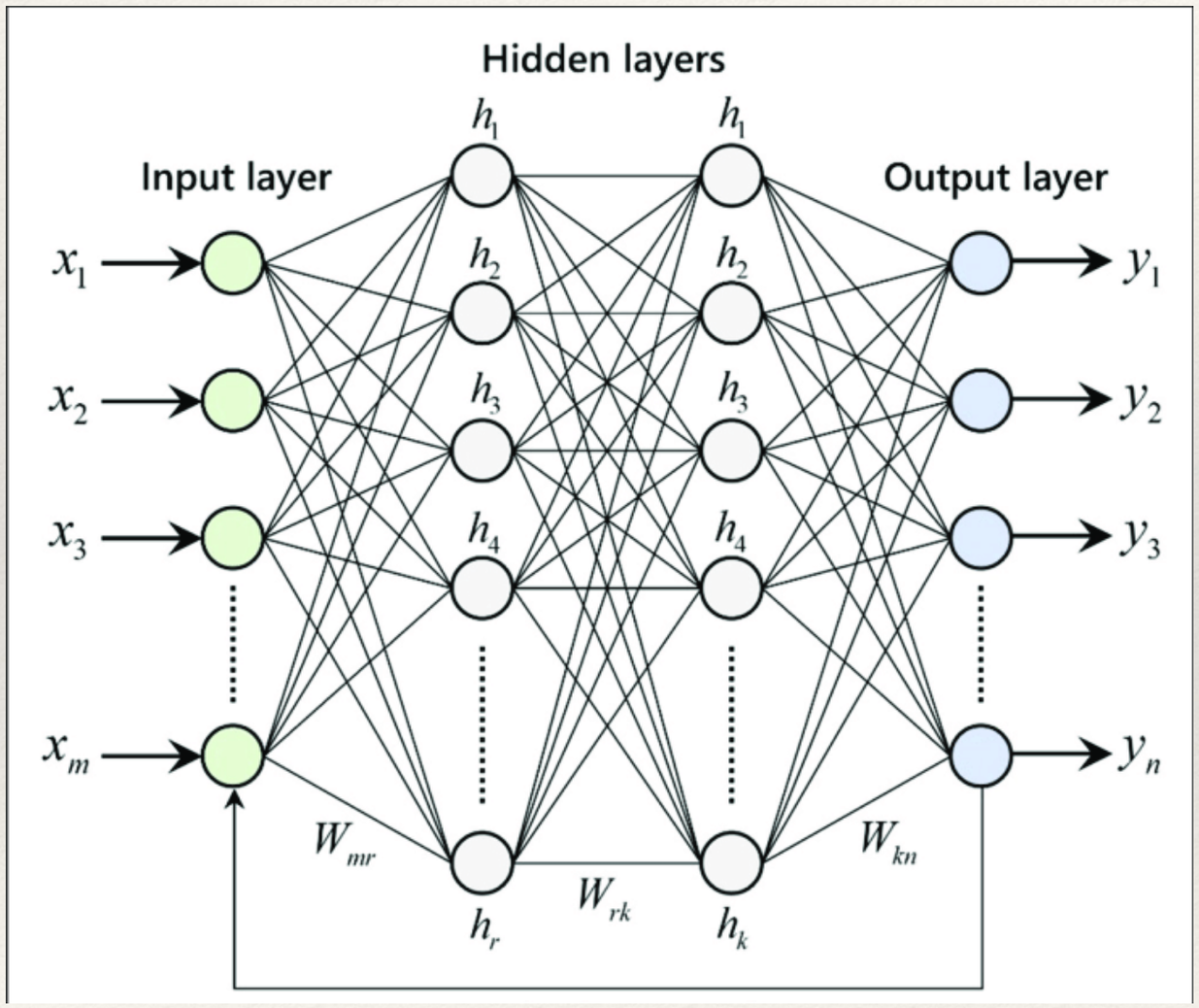
https://bmild.github.io/fourfeat/index.html

$$w \sim O(1)$$



$$w \geq O(1)$$

Weight initialization



$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

$w_{jk}^{(l)}$ should be **different**

Avoid gradient vanishing and exploding

$w_{jk}^{(l)}$ follow Gaussian distribution $N(\mu, Var)$

mean $\mu = 0$ Variance $Var^{(l)} = \frac{1}{n_l}$

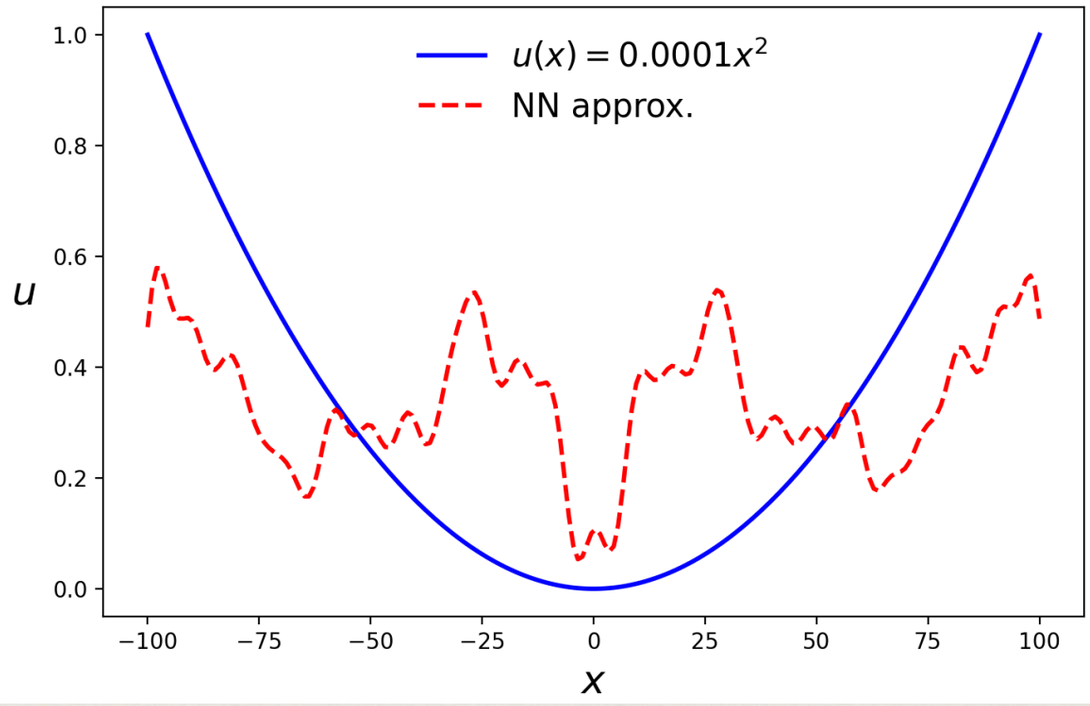
Probability theory: $x_j^{(n)} = \sum_{i=1} w_{ji}^{(n-1)} x_i$ $Var(x_j^{(l)}) = n_{l-1}\, Var(w_{ji}^{(l-1)}) \cdot Var(x_i)$

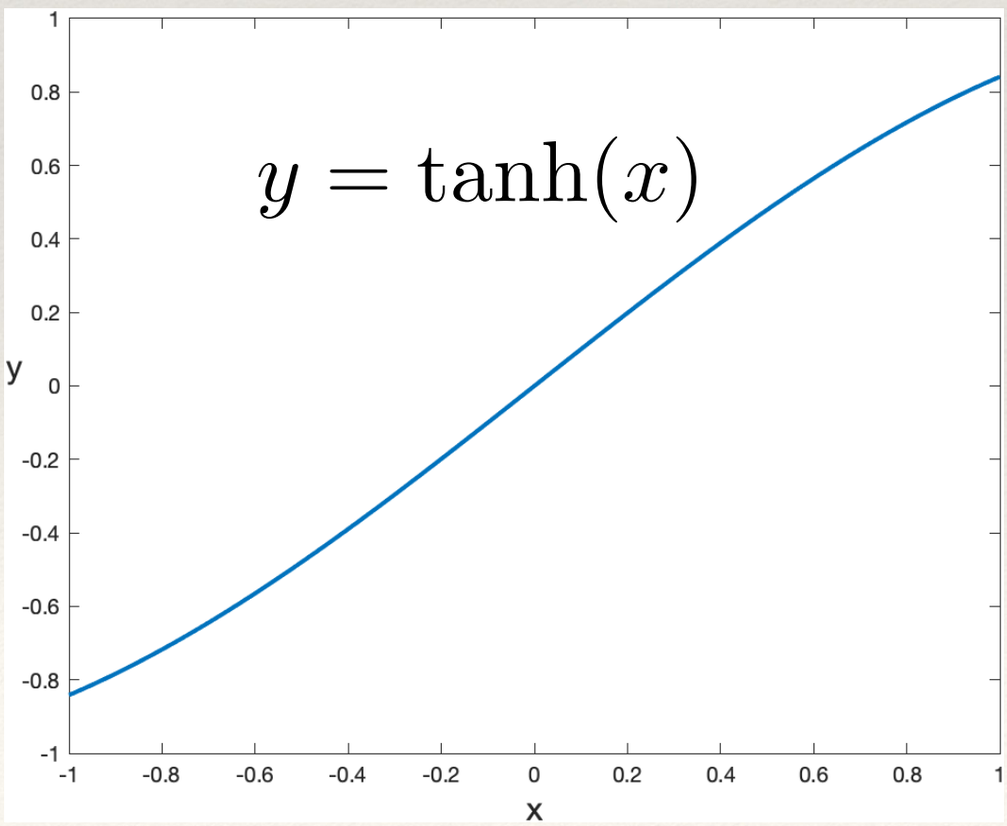
$w_{jk}^{(l)}$ follow Gaussian distribution $N(\mu = 0, Var = \frac{1}{n_l})$

Limitation ?

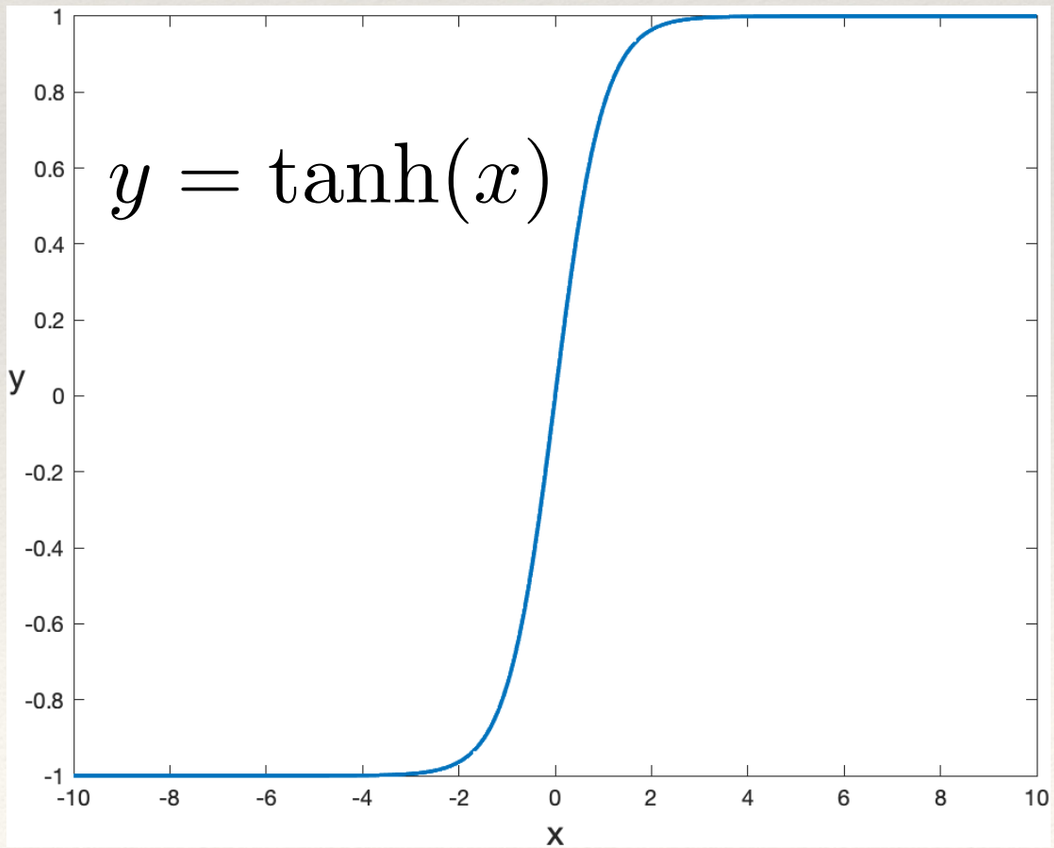
Input: $x \in [-100, 100]$

Output: $u = 0.0001x^2 \in [-1, 1]$





$w \sim O(1)$



$w_{jk}^{(l)}$ follow Gaussian distribution $N(\mu = 0, Var = \frac{1}{n_l})$

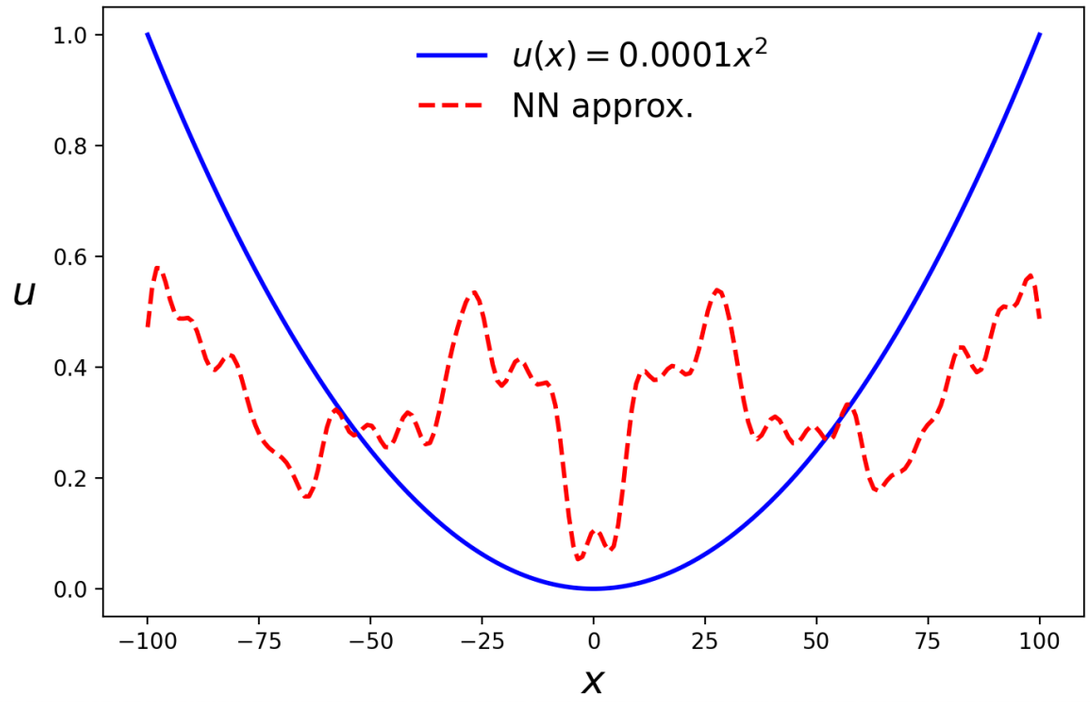
Limitation ?

Input: $x \in [-100, 100]$

Output: $u = 0.0001x^2 \in [-1, 1]$

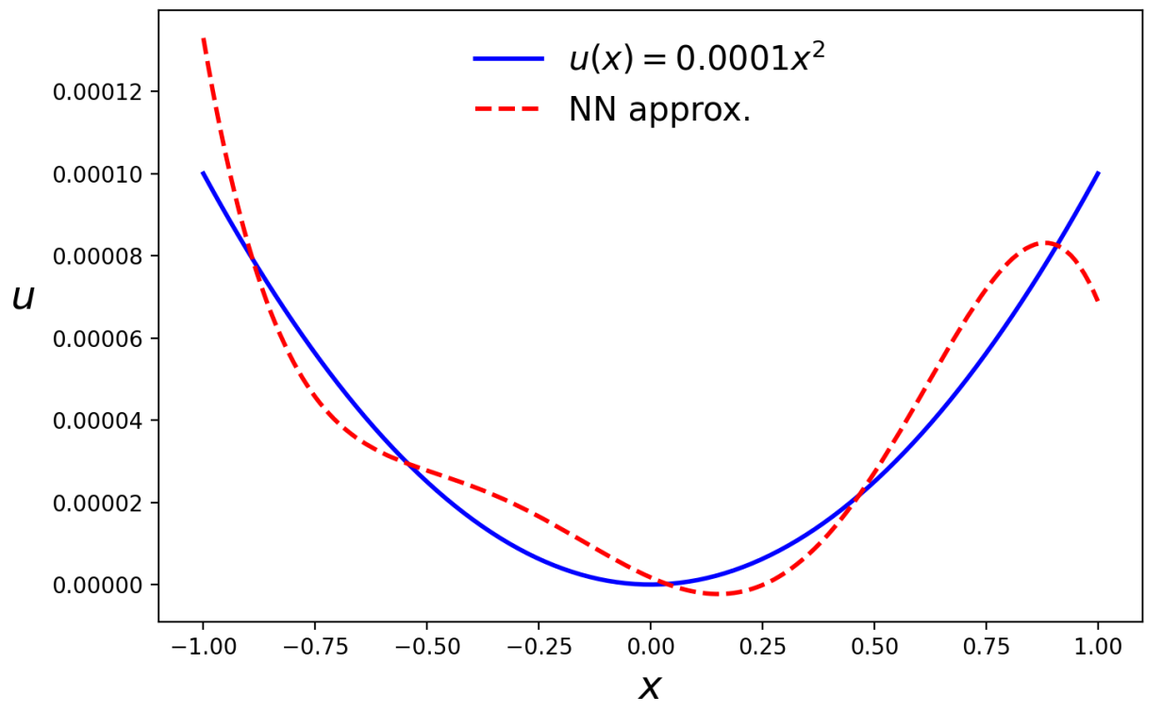
NN is very **sensitive** to scale

Normalization



Input: $x \in [-1, 1]$

Output: $u = 0.0001x^2 \sim O(10^{-4})$

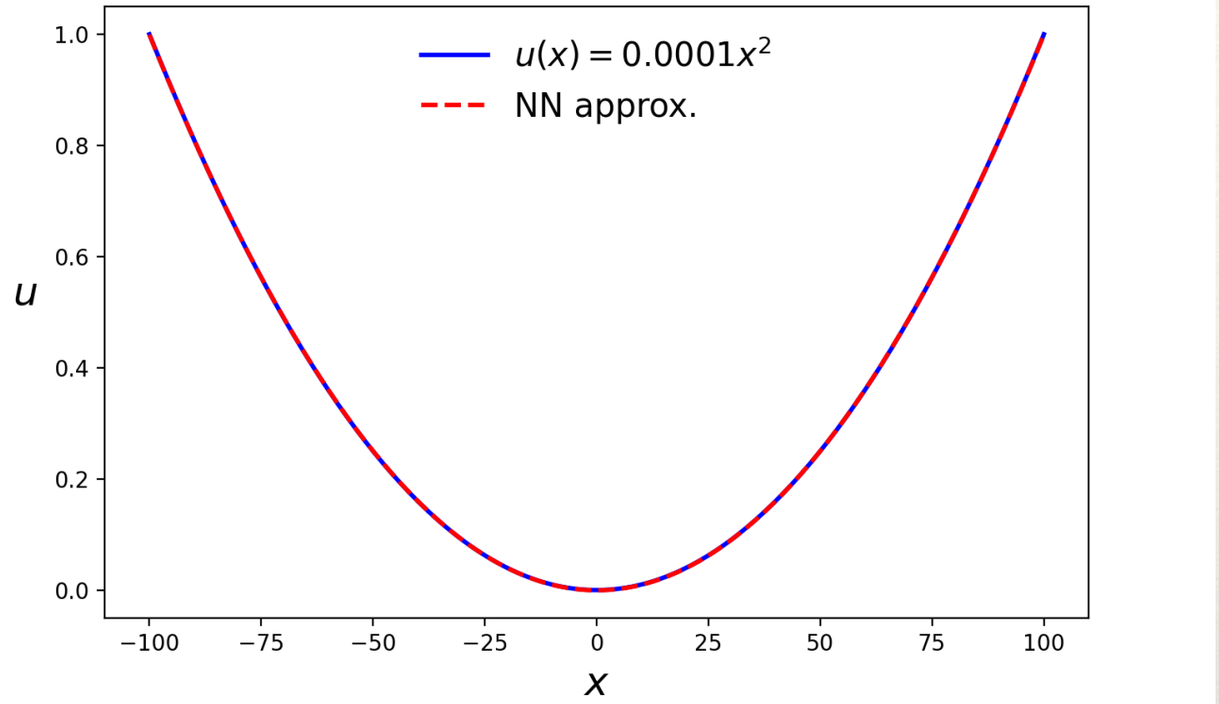


$w_{jk}^{(l)}$ follow Gaussian distribution $N(\mu = 0, Var = \frac{1}{n_l})$

Input: $x \in [-100, 100]$

Output: $u = 0.0001x^2 \in [-1, 1]$

$x' = x/100$	Training x' and u

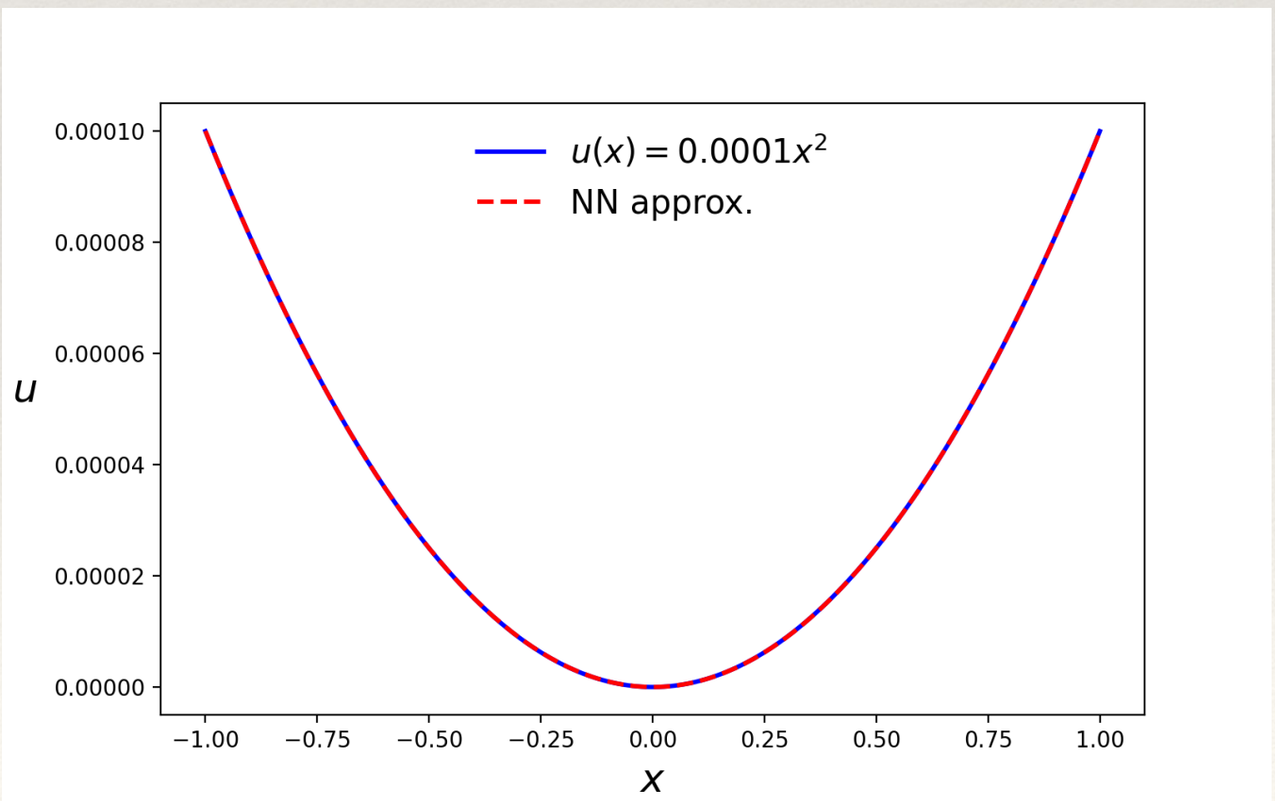


Normalization

Input: $x \in [-1, 1]$

Output: $u = 0.0001x^2 \sim O(10^{-4})$

$u' = 10000u$	Training x and u'

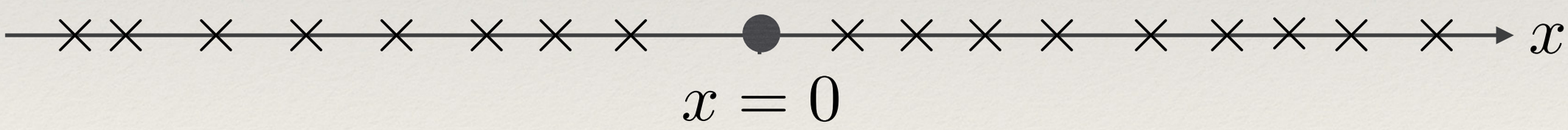


Weight of equation loss

$$loss_d = \frac{1}{N_d}\sum_{i=1}^{N_d}[\hat{u}(x_i) - u_i]^2 \qquad loss_e = \frac{1}{N_c}\sum_{j=1}^{N_c} f^2(x_j, u(x_j))$$

Loss function: $loss = (1-\lambda)loss_d + \lambda loss_e$

$$\frac{du}{dx} = x \qquad \text{Boundary condition} \quad u(0) = 0 \implies u = 0.5x^2$$



● Data point (evaluate boundary conditions)

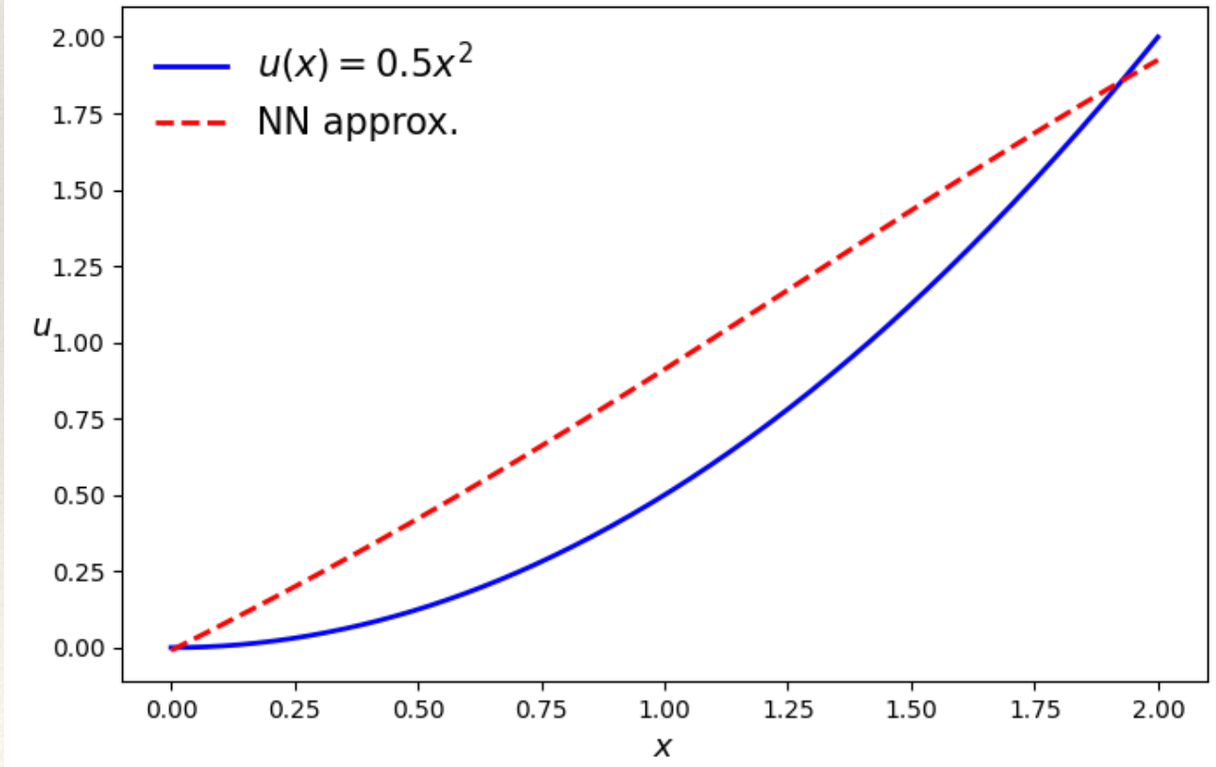
× Collocation points (evaluate equation balance)

$$\frac{du}{dx} = x \qquad \text{Boundary condition} \qquad u(0) = 0 \implies u = 0.5x^2$$

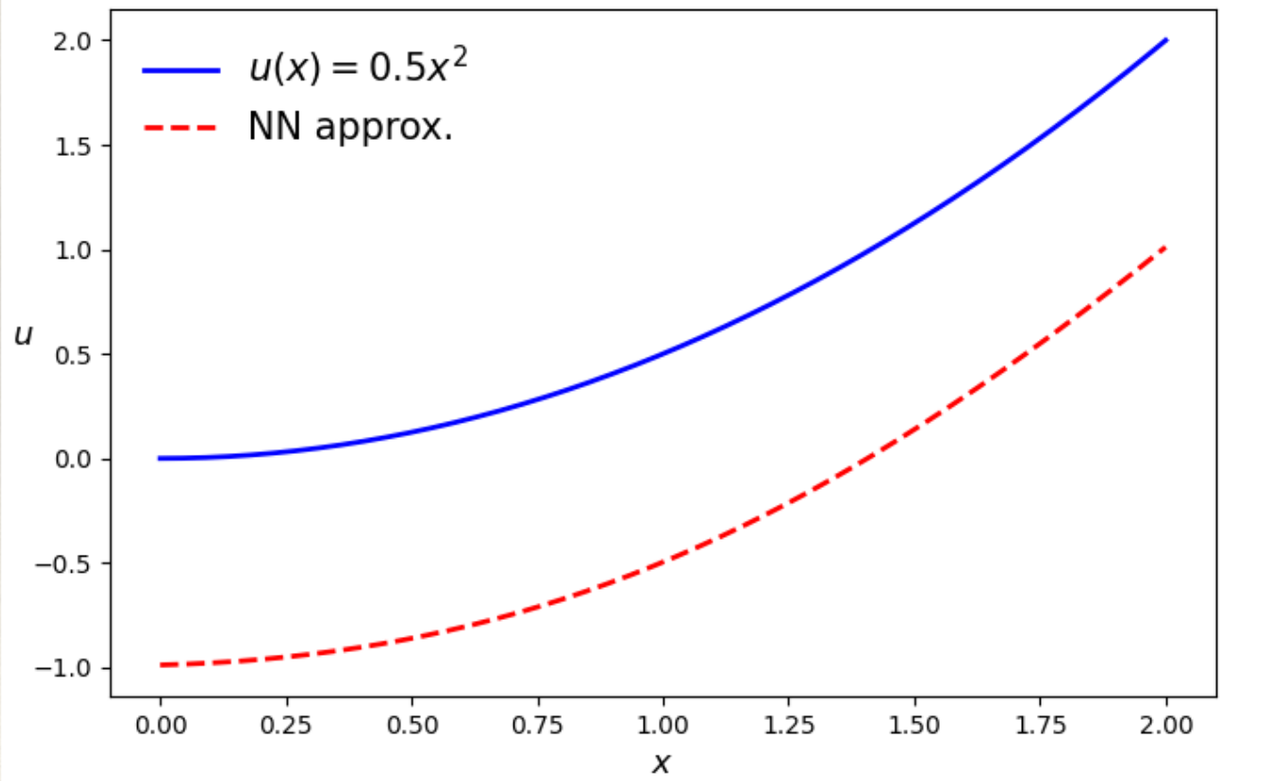
$$loss_d = [u(0) - u_0]^2 \qquad loss_e = \sum_j^N f^2(x_j, u(x_j))$$

Loss function: $loss = (1 - \lambda) loss_d + \lambda loss_e$

$\lambda = 0.000001$



$\lambda = 0.999999$



Layers: [1, 10, 10, 1]

Iteration: 4000

Loss function: $loss = (1-\lambda)loss_d + \lambda loss_e$

Rule of thumb: λ is selected to make $(1-\lambda)loss_d \approx \lambda loss_e$

$$\Longrightarrow \quad \frac{\lambda}{1-\lambda} \approx \frac{loss_d}{loss_e}$$

$$\frac{du}{dx} = u \qquad \text{Boundary condition} \quad u(0) = 1 \qquad \longrightarrow \qquad u = e^x$$

Ground truth: $u(x)$ NN prediction: $\hat{u}(x)$

$$loss_d = [\hat{u}(0) - u(0)]^2 \qquad loss_e = \frac{1}{N_c}\sum_{j=1}^{N_c}\left(\frac{d\hat{u}(x_j)}{dx} - \hat{u}(x_j)\right)^2$$

We introduce $\hat{u}(x) = u(x) + \delta u$

$$loss_d = [\hat{u}(0) - u(0)]^2 = [\delta u(0)]^2 \sim O(\delta^2)$$

$$loss_e = \frac{1}{N_c}\sum_{j=1}^{N_c}\left(\frac{d}{dx}[u(x_j) - \delta u(x_j)] - [u(x_j) - \delta u(x_j)]\right)^2 = \frac{1}{N_c}\sum_{j=1}^{N_c}\left(\frac{d}{dx}\delta u(x_j) - \delta u(x_j)\right)^2 \sim o(\delta^2)$$

At which circumstance $$||loss_d|| \neq ||loss_e||$$

$$100\frac{du}{dx} = 100u \qquad \text{Boundary condition} \quad u(0) = 1 \quad \longrightarrow \quad u = e^x$$

$$loss_e = \frac{1}{N_c}\sum_{j=1}^{N_c} 100\left(\frac{d}{dx}\delta u(x_j) - \delta u(x_j)\right)^2 \sim o(100\delta^2)$$

1D Euler equation $$\rho(uu_x) = -p_x$$ Key point: **Normalization of data**

$$\rho = 10^3 \ \mathrm{kg/m^3}$$

Rule of thumb: normalize/non-dimensionalize the equation by the magnitude of the **largest** term to make all terms in the normalized equation less than 1

$$U = u/U_0 \qquad X = x/L \qquad P = p/P_0$$

$$\frac{\rho U_0^2}{L}UU_X = -\frac{P_0}{L}P_X \qquad f = UU_X + \frac{P_0}{\rho U_0^2}P_X = 0$$

At which circumstance $$||loss_d|| \neq ||loss_e||$$

2D Euler equation

$$\rho(uu_x + vu_y) = -p_x$$

$$\rho(vu_x + vv_y) = -p_y$$

$$X = x/L \qquad U = u/U_0 \qquad P = p/P_0$$

$$Y = y/L \qquad V = v/V_0$$

Assuming $U_0 \geq V_0$

$$\frac{\rho U_0^2}{L}(UU_X + \frac{V_0}{U_0}VU_Y) = -\frac{P_0}{L}P_X \qquad f_1 = (UU_X + \frac{V_0}{U_0}VU_Y) + \frac{P_0}{\rho U_0^2}P_X$$

$$\frac{\rho V_0 U_0}{L}(UV_X + \frac{V_0}{U_0}VV_Y) = -\frac{P_0}{L}P_Y \qquad f_2 = (UV_X + \frac{V_0}{U_0}VV_Y) + \frac{P_0}{\rho U_0 V_0}P_Y$$