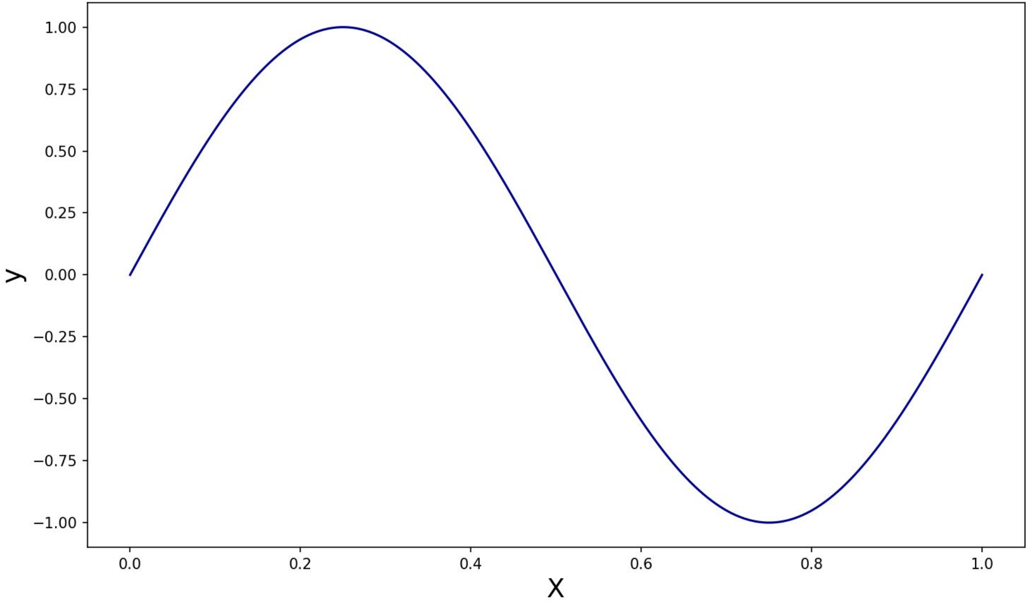
# Neural Networks

# Example:

Let's say that we are trying to fit data that comes from this data generation process:

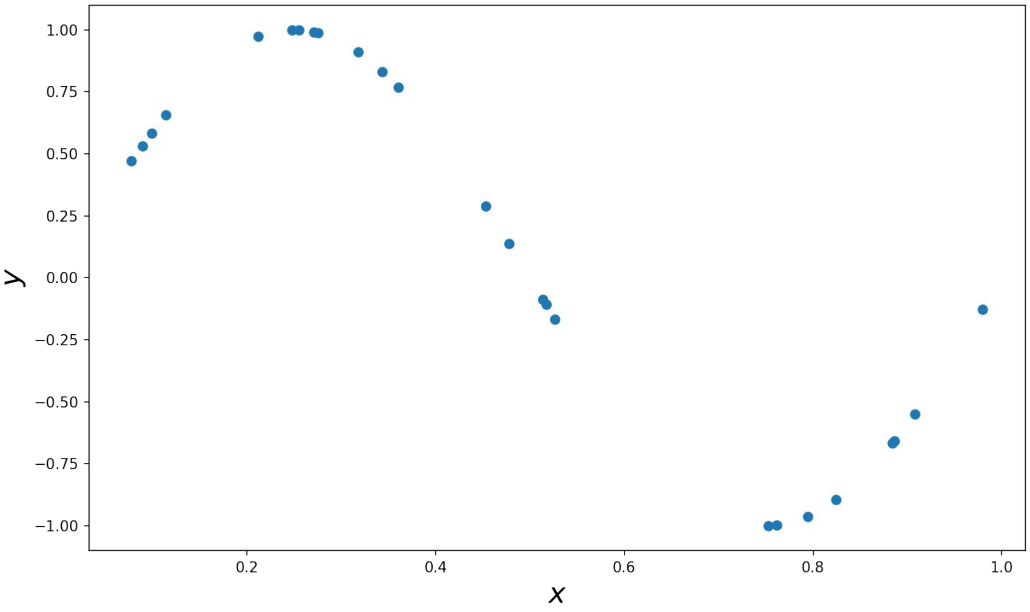
X: Uniform distribution on the interval [0,1]

Y = sin(2$\pi$X) (no noise)



# Example:

We have a training dataset with 25 points:



# Recall: Polynomial Regression

One option we can try is polynomial regression, say a degree 2 polynomial:

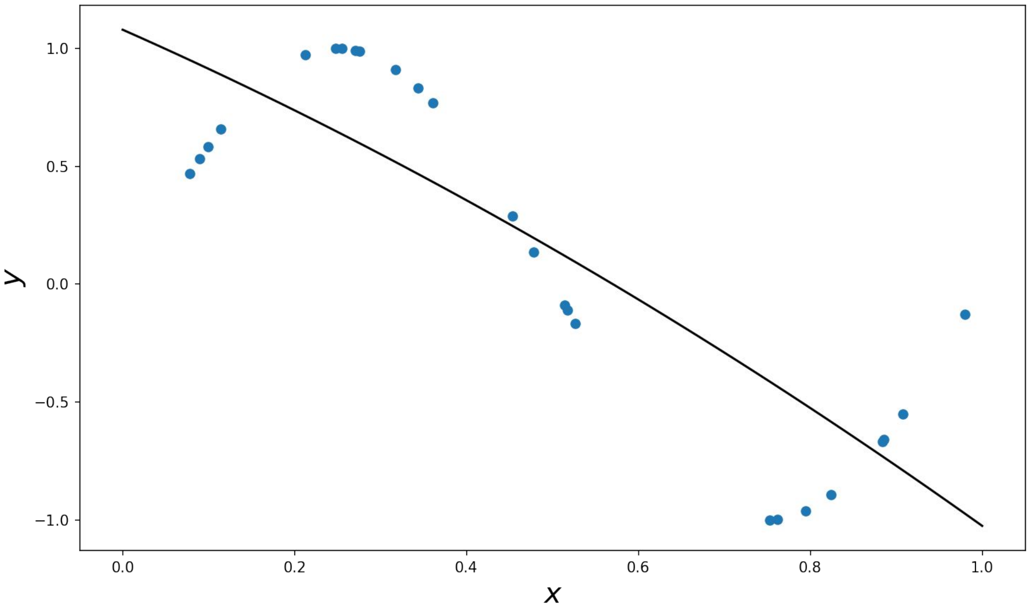
$$\hat{f}(x) = a + bx + cx^2$$

# Example:

We have a training dataset with 25 points:

When we fit this data, we get the equation

$$\hat{f}(x) = 1.08 - 1.61x - 0.49x^2$$



What if instead of using $x$ and $x^2$ we allow a bit more flexibility?

$$\hat{f}(x) = a + bx + cx^2$$

What if instead of using $x$ and $x^2$ we allow a bit more flexibility?

$$\hat{f}(x) = a + bx + cx^2$$

$$\Downarrow$$

$$\hat{f}(x) = a + b \cdot g_1(x) + c \cdot g_2(x)$$

What if instead of using $x$ and $x^2$ we allow a bit more flexibility?

$$\hat{f}(x) = a + bx + cx^2$$

$$\hat{f}(x) = a + b \cdot g_1(x) + c \cdot g_2(x)$$

Where $g_i(x) = \tanh(\beta_0^{(i)} + \beta_1^{(i)} \cdot x)$ for constants $\beta_j^{(i)}$

# Example:

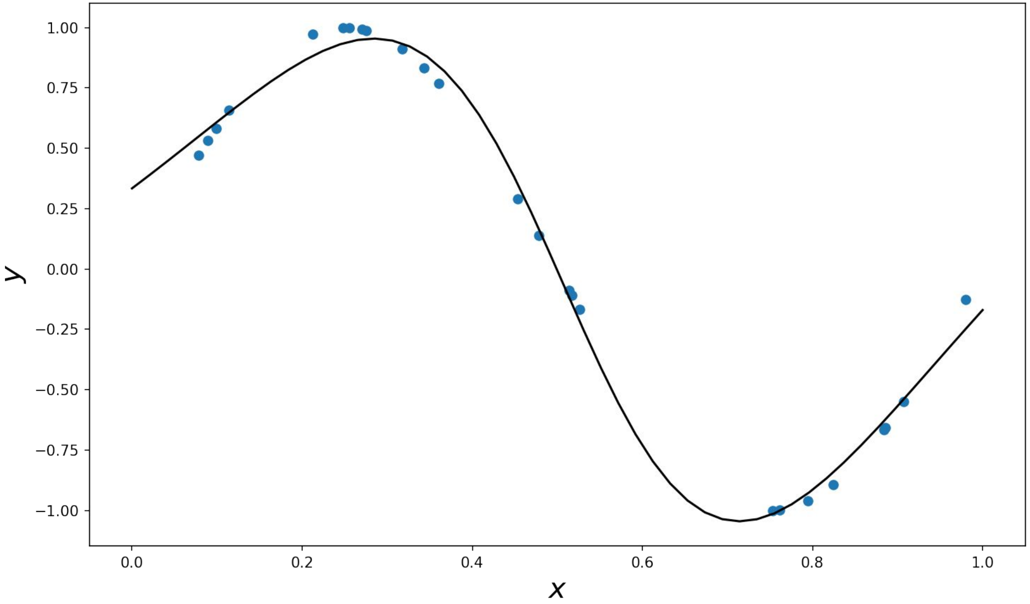
We have a training dataset with 25 points:

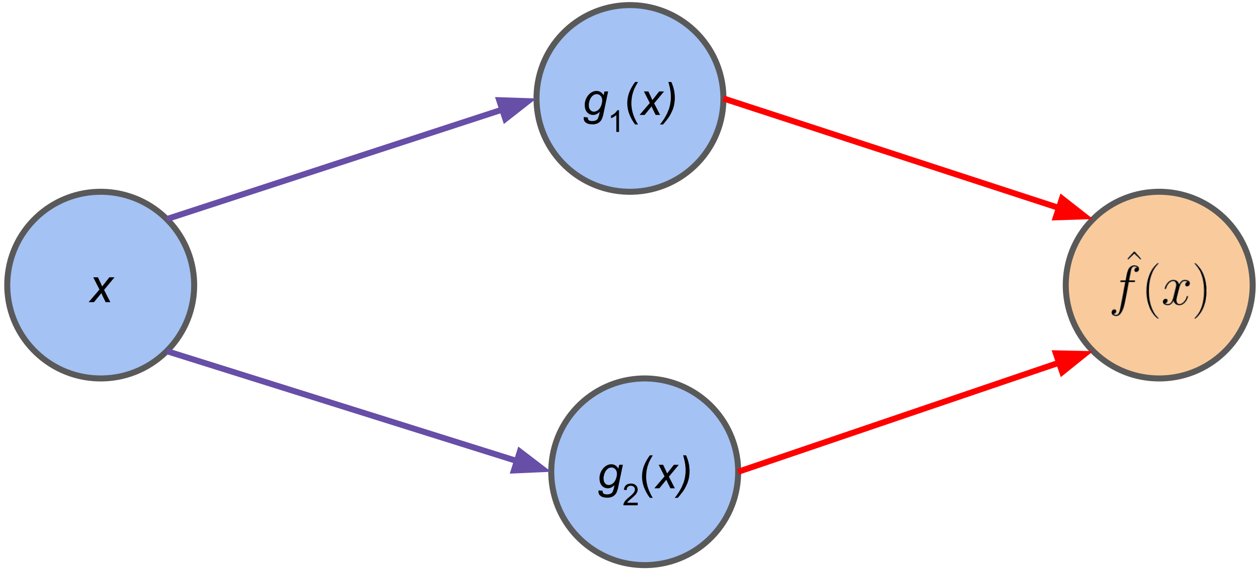
When we fit this data, we get the equation

$$\hat{f}(x) = 0.38 - 3.16 \cdot g_1(x) + 4.22 \cdot g_2(x)$$

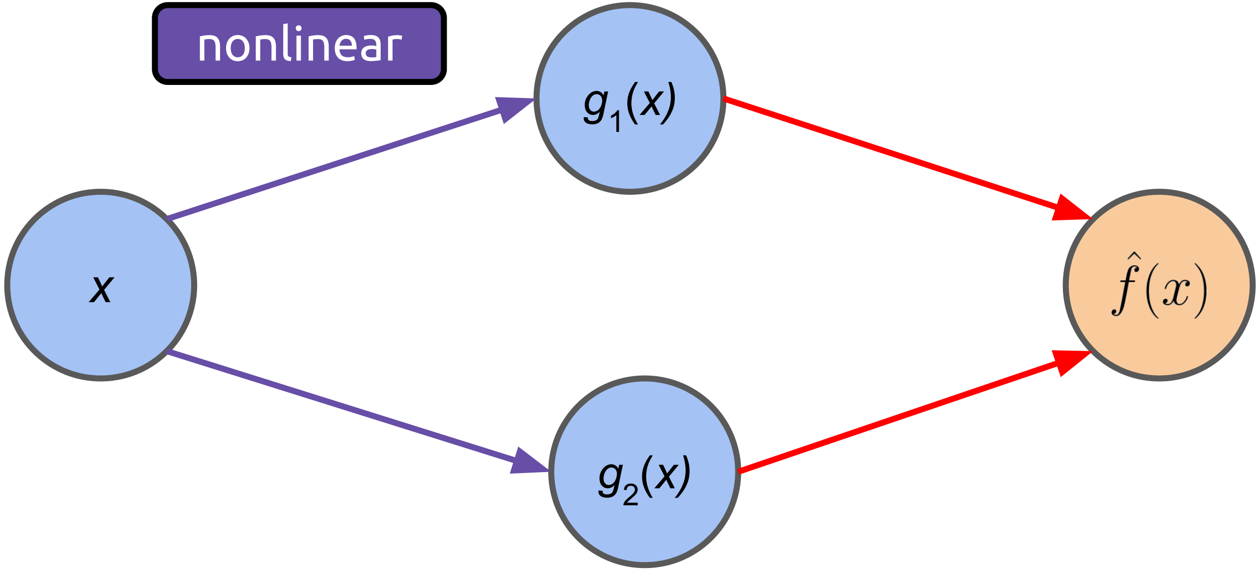
$$g_1(x) = \tanh(-2.43 + 4.74x)$$

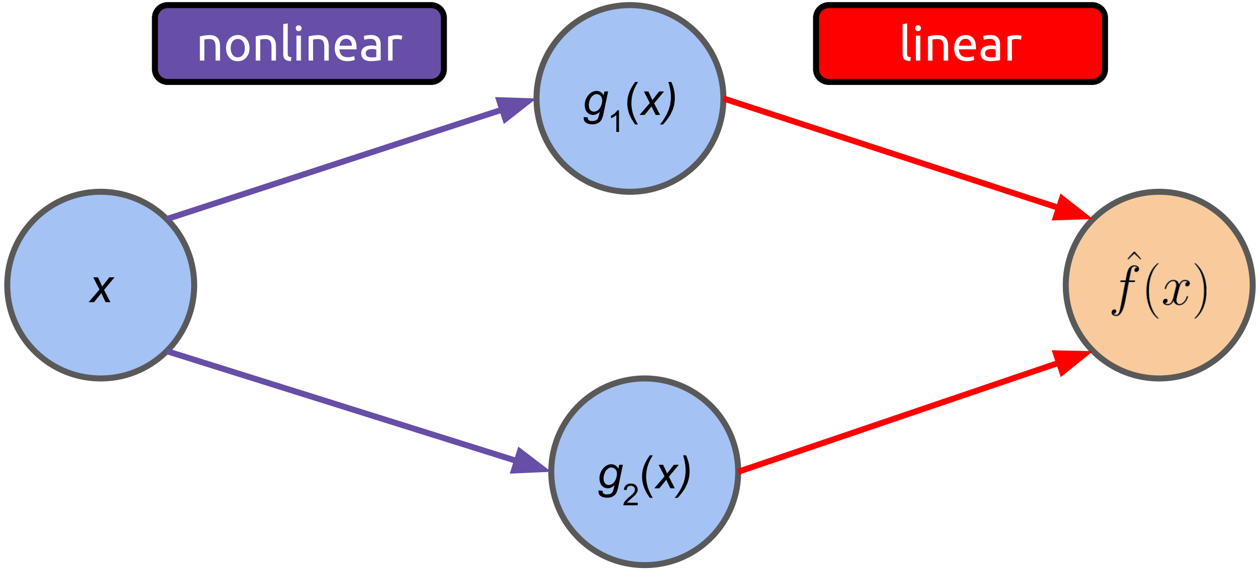
$$g_2(x) = \tanh(-0.97 + 1.66x)$$

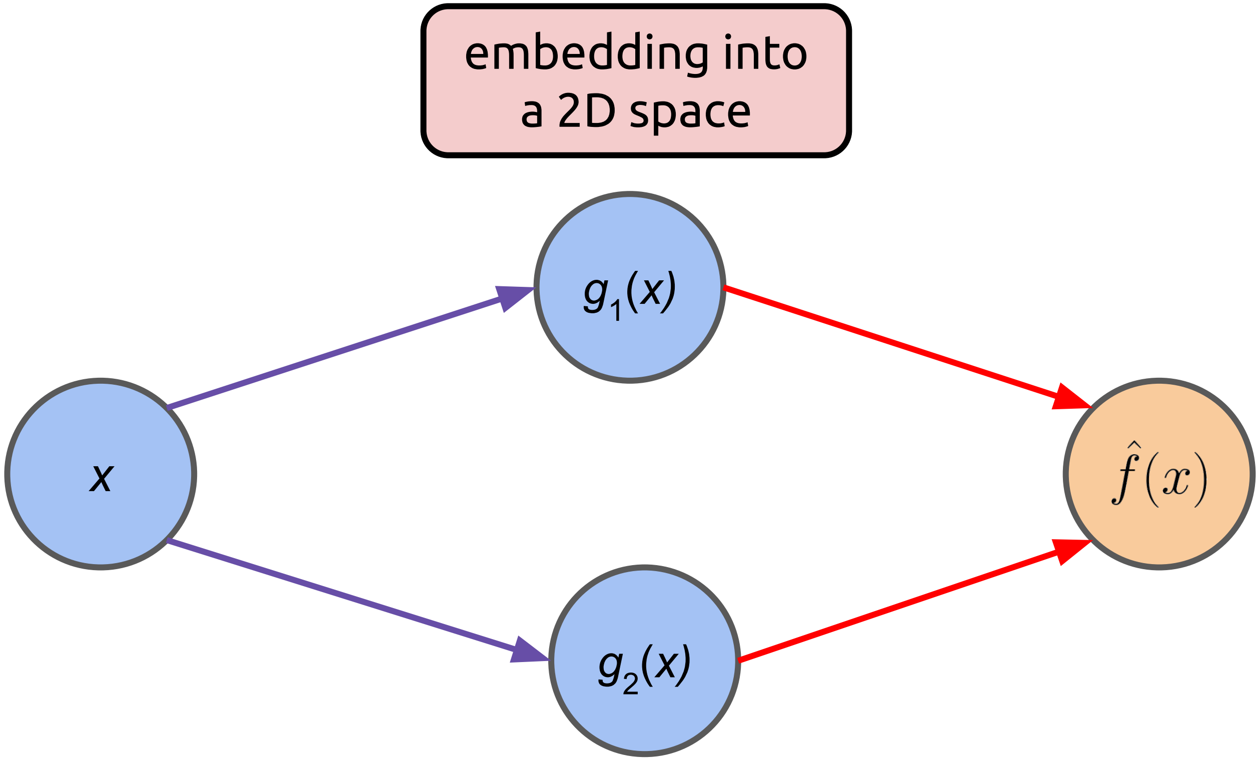


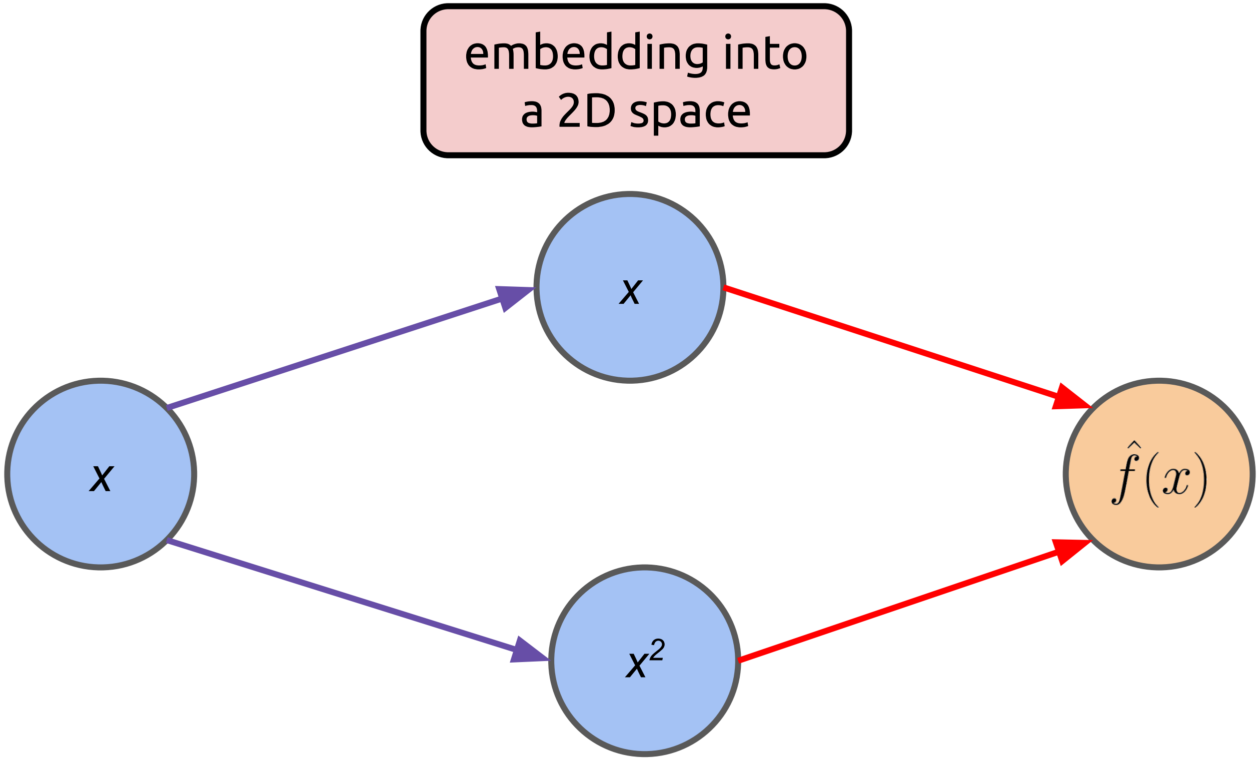


$x$
$g_1(x)$
$g_2(x)$
$\hat{f}(x)$


nonlinear
$x$
$g_1(x)$
$g_2(x)$
$\hat{f}(x)$

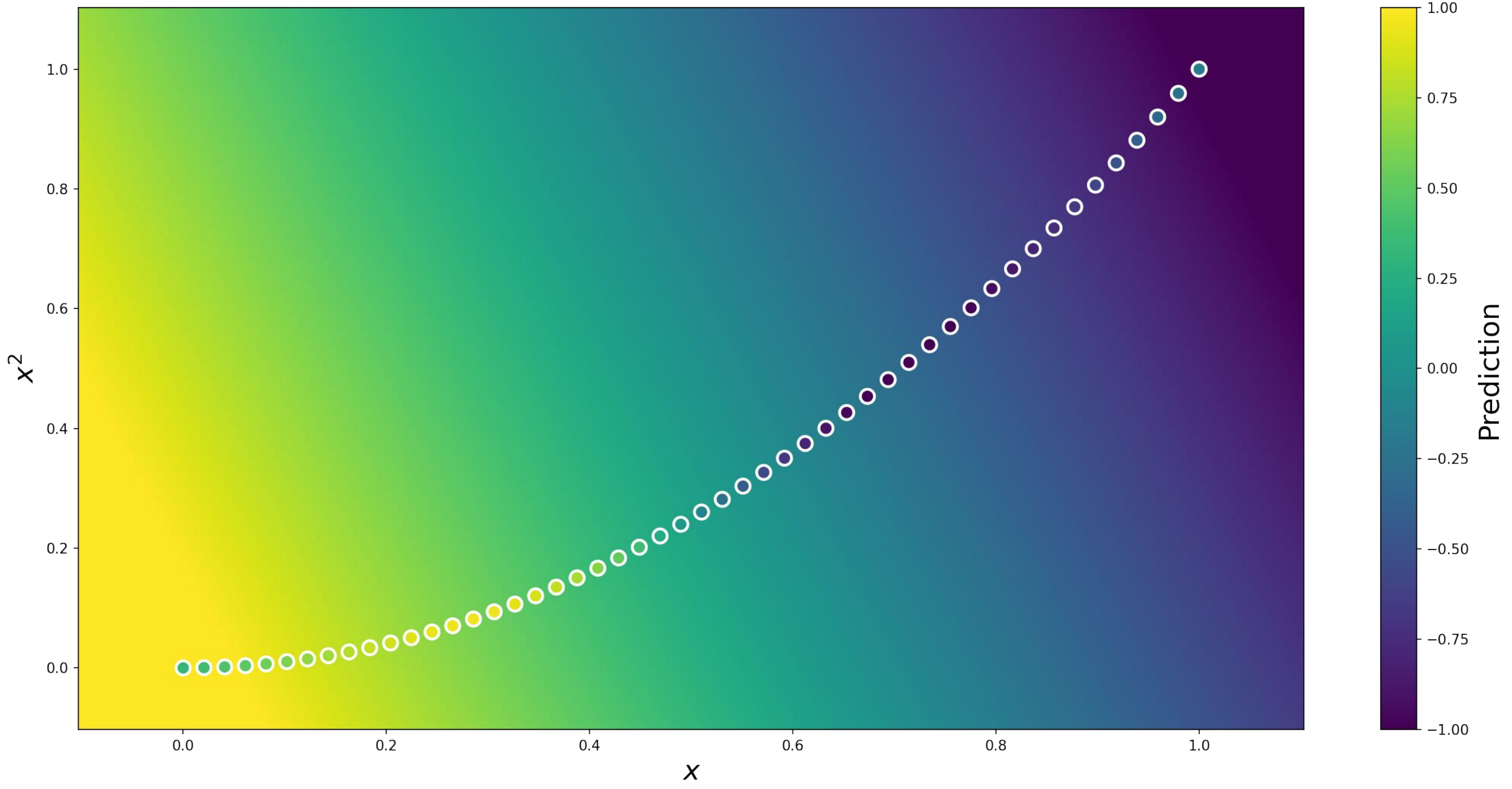

nonlinear
linear
$x$
$g_1(x)$
$g_2(x)$
$\hat{f}(x)$


embedding into
a 2D space
$x$
$g_1(x)$
$g_2(x)$
$\hat{f}(x)$

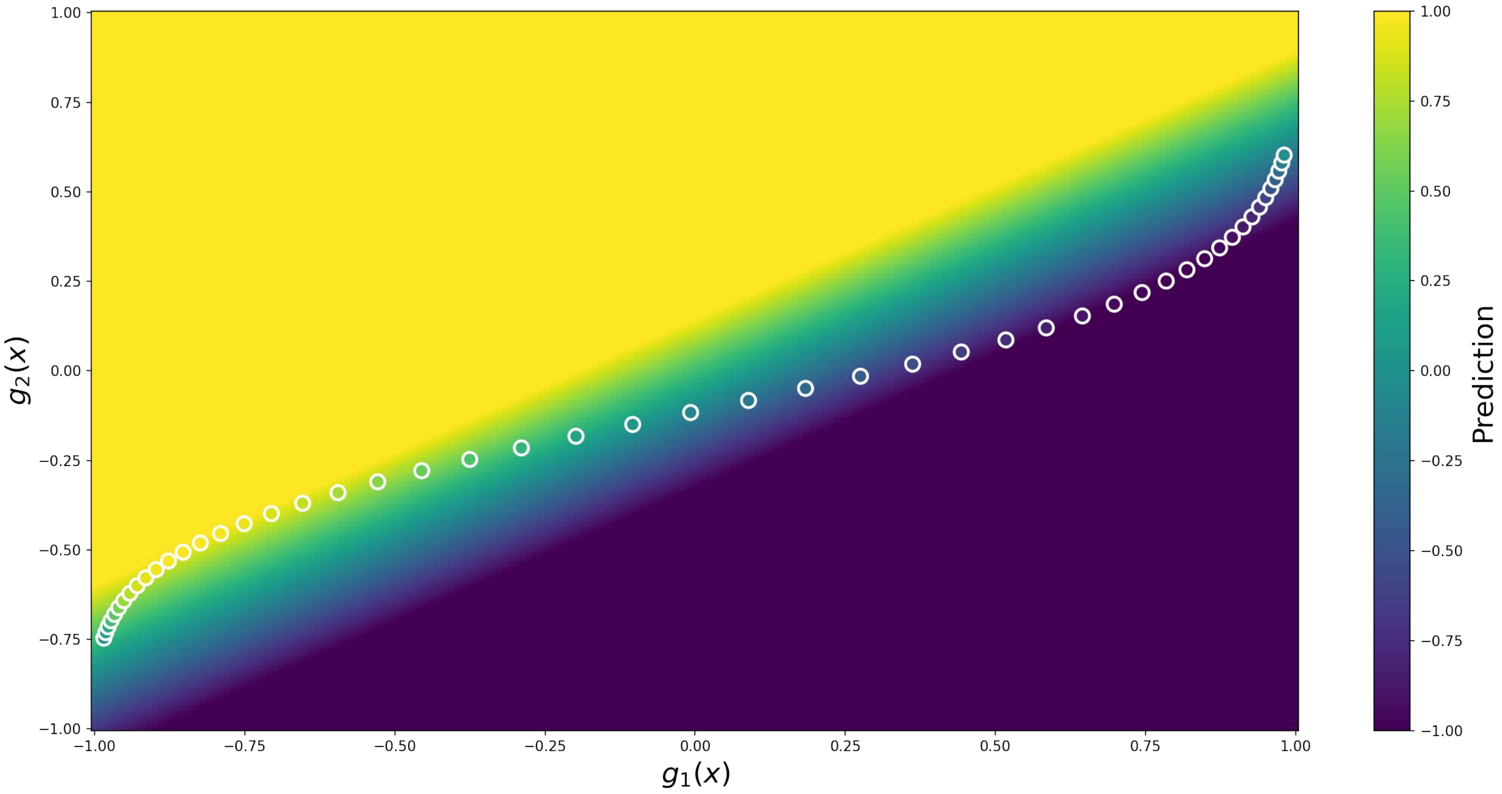



Polynomial Regression
Version

# Polynomial Regression

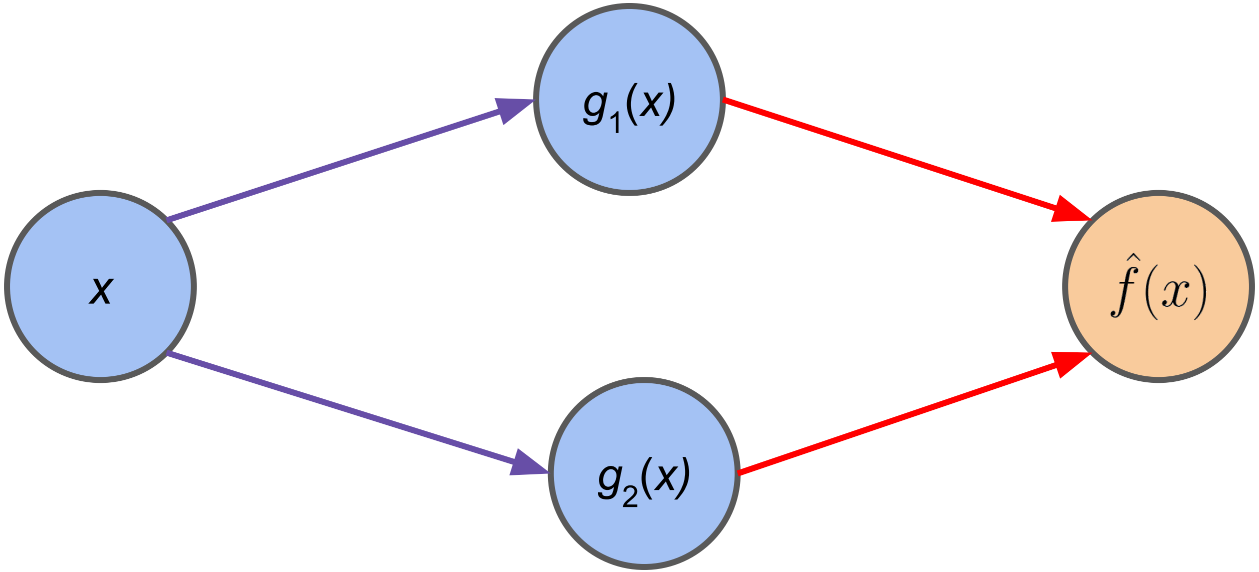


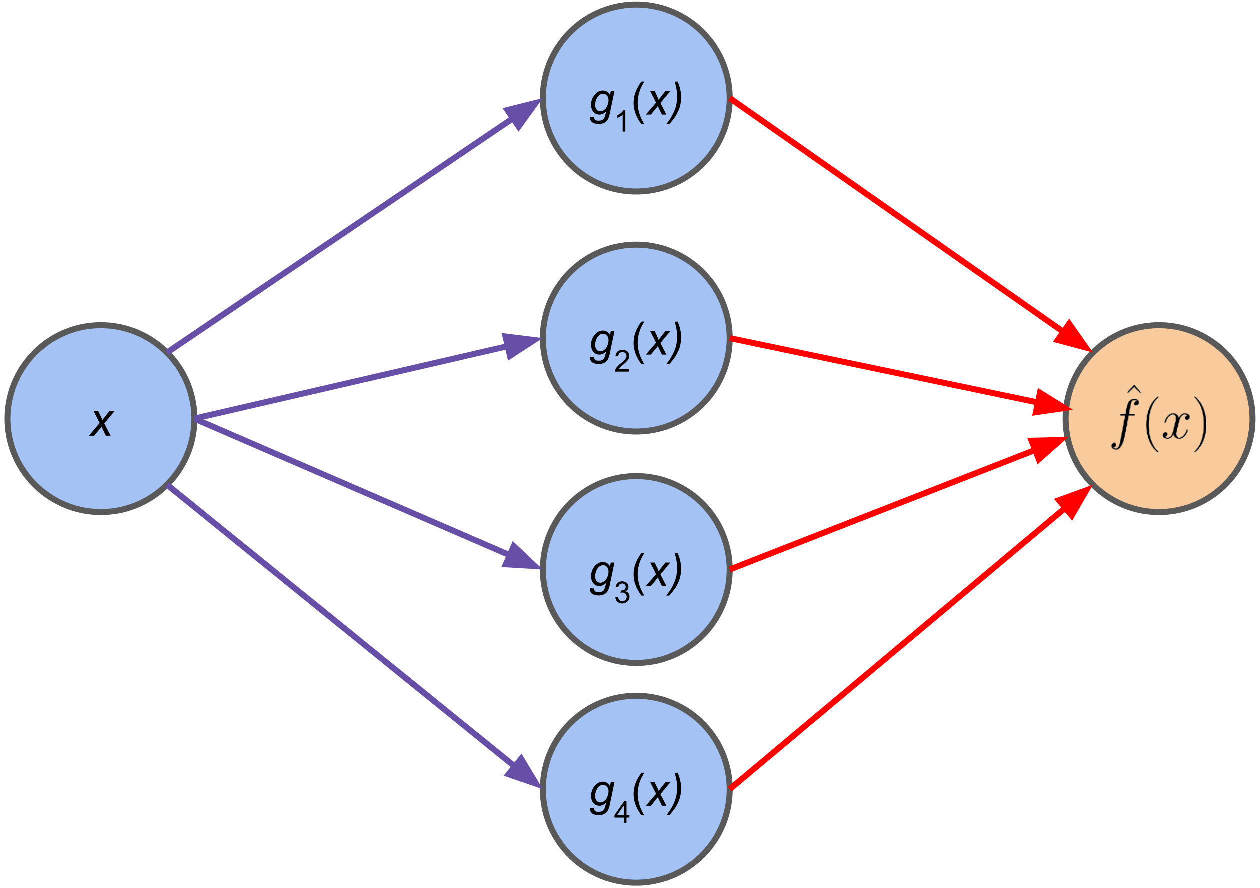
# Alternative Embedding

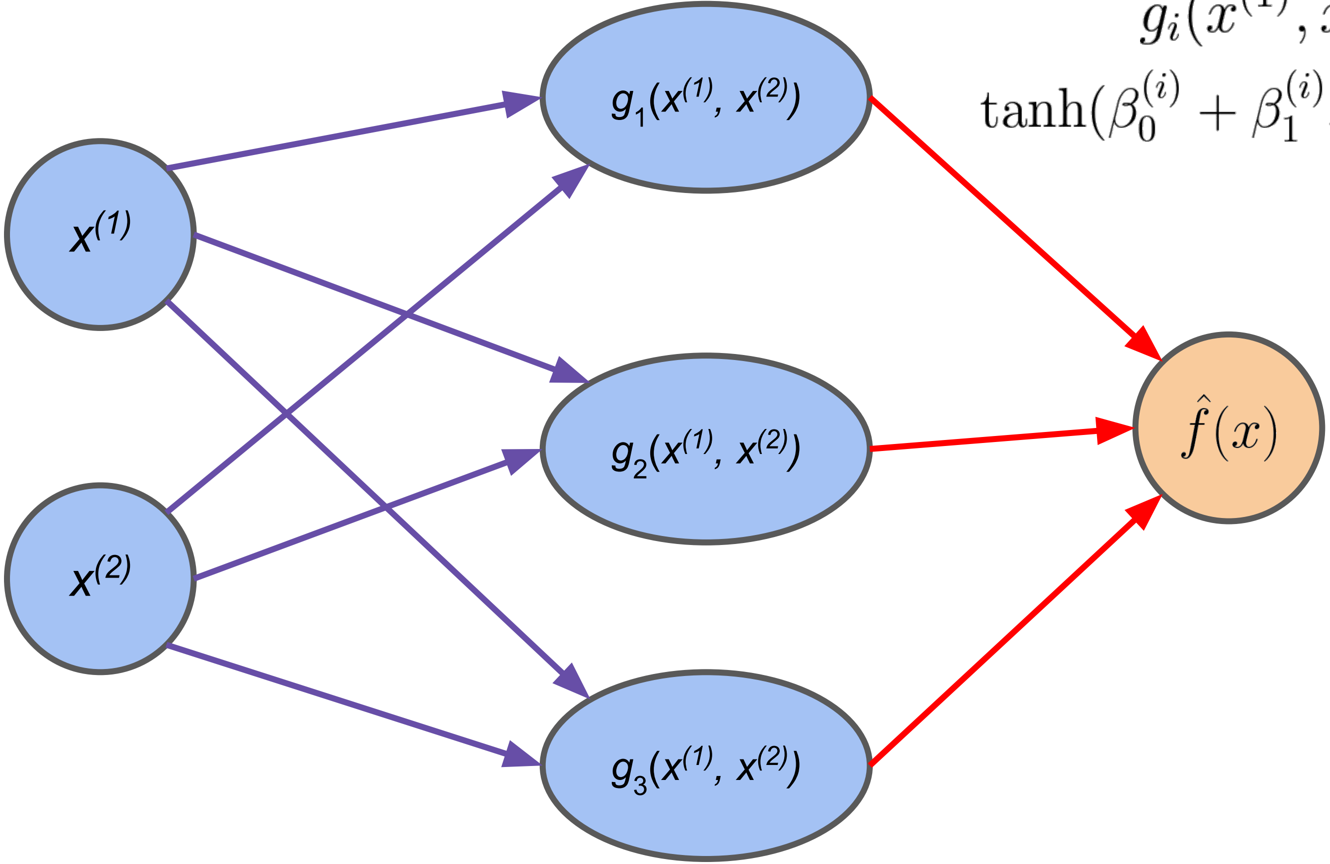


**Terminology:**

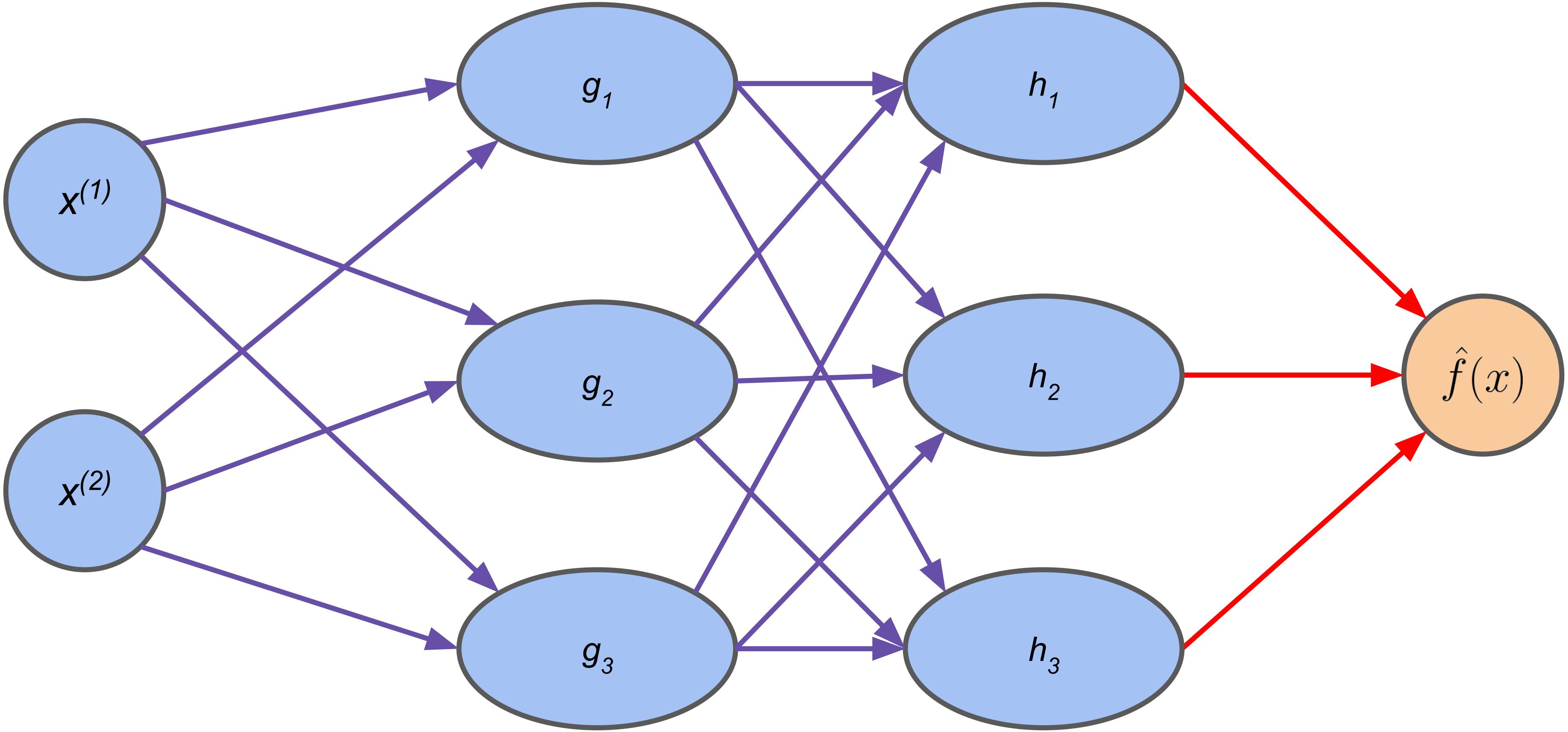
- The model we used is called a **neural network**.
- Each node is a **neuron**.
- The nonlinear function we used in each neuron (*tanh*) is called an **activation function.**




$x$
$g_1(x)$
$g_2(x)$
$g_3(x)$
$g_4(x)$
$\hat{f}(x)$




$$g_i(x^{(1)}, x^{(2)}) = \tanh(\beta_0^{(i)} + \beta_1^{(i)} x^{(1)} + \beta_2^{(i)} x^{(2)})$$

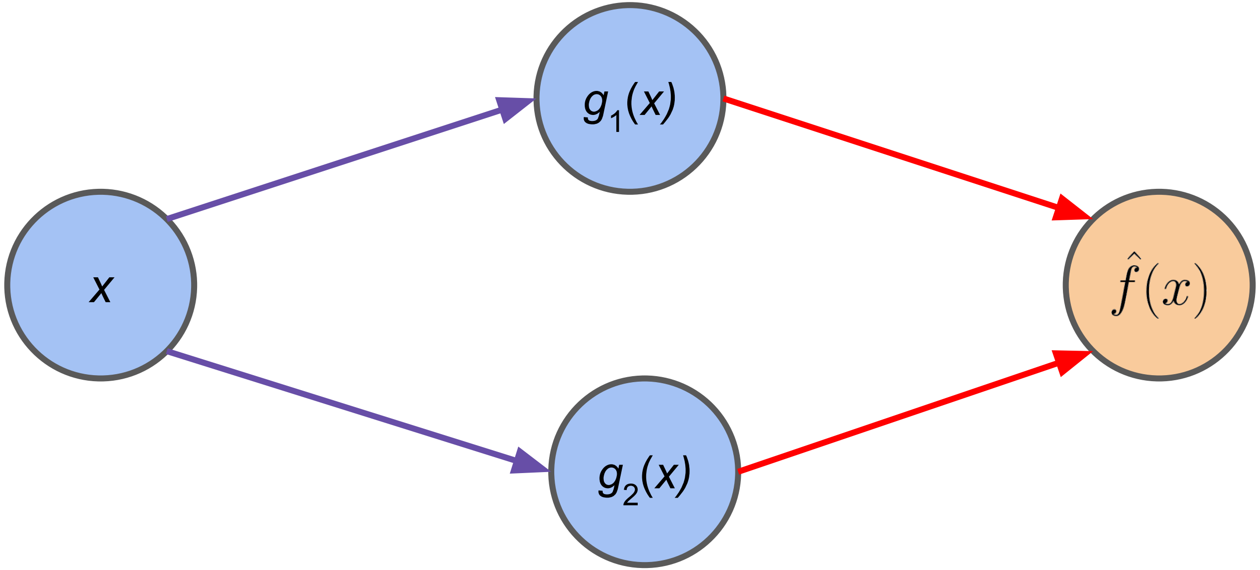

$x^{(1)}$
$x^{(2)}$
$g_1$
$g_2$
$g_3$
$h_1$
$h_2$
$h_3$
$\hat{f}(x)$

# Neural Networks

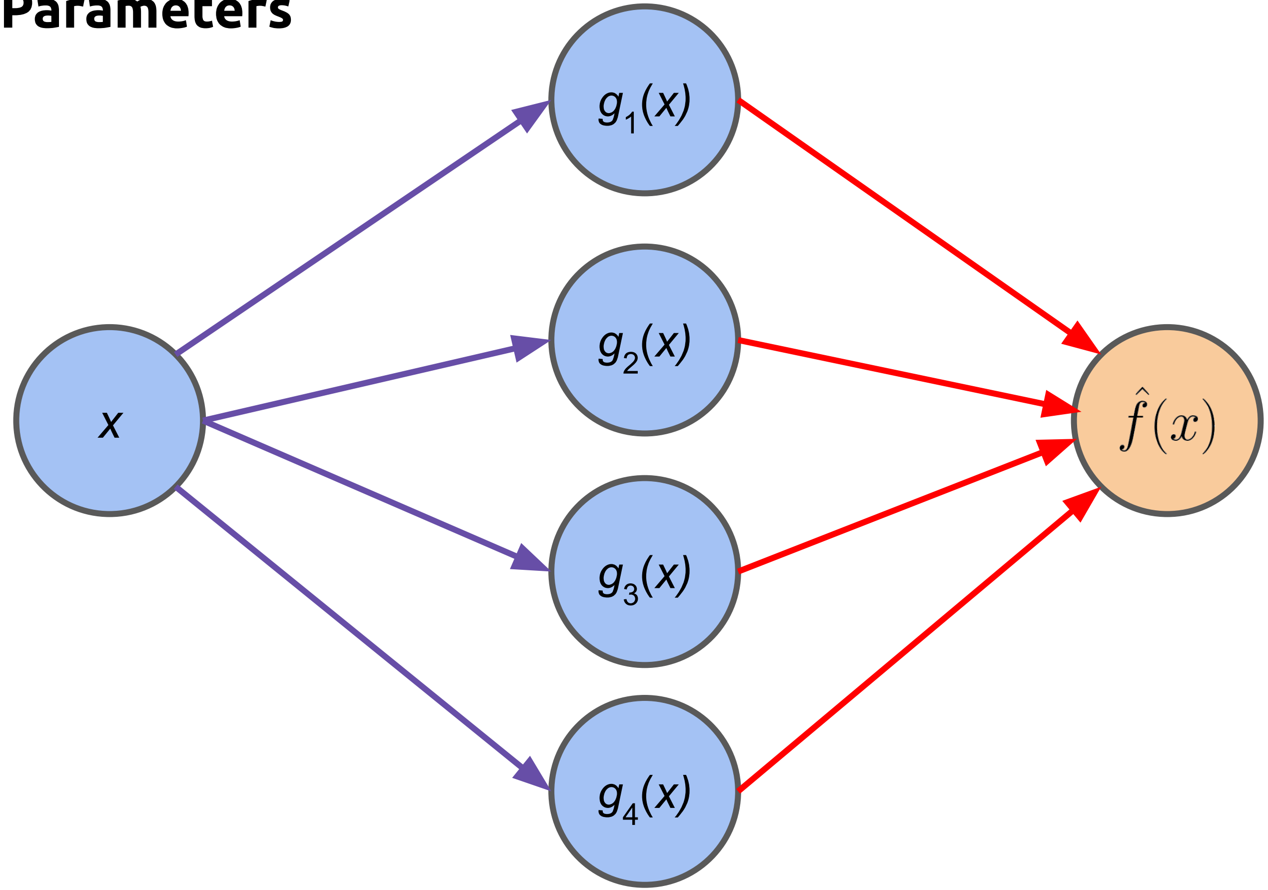
These models are very flexible, so can fit very complicated functions.

However, this comes at the risk of overfitting!

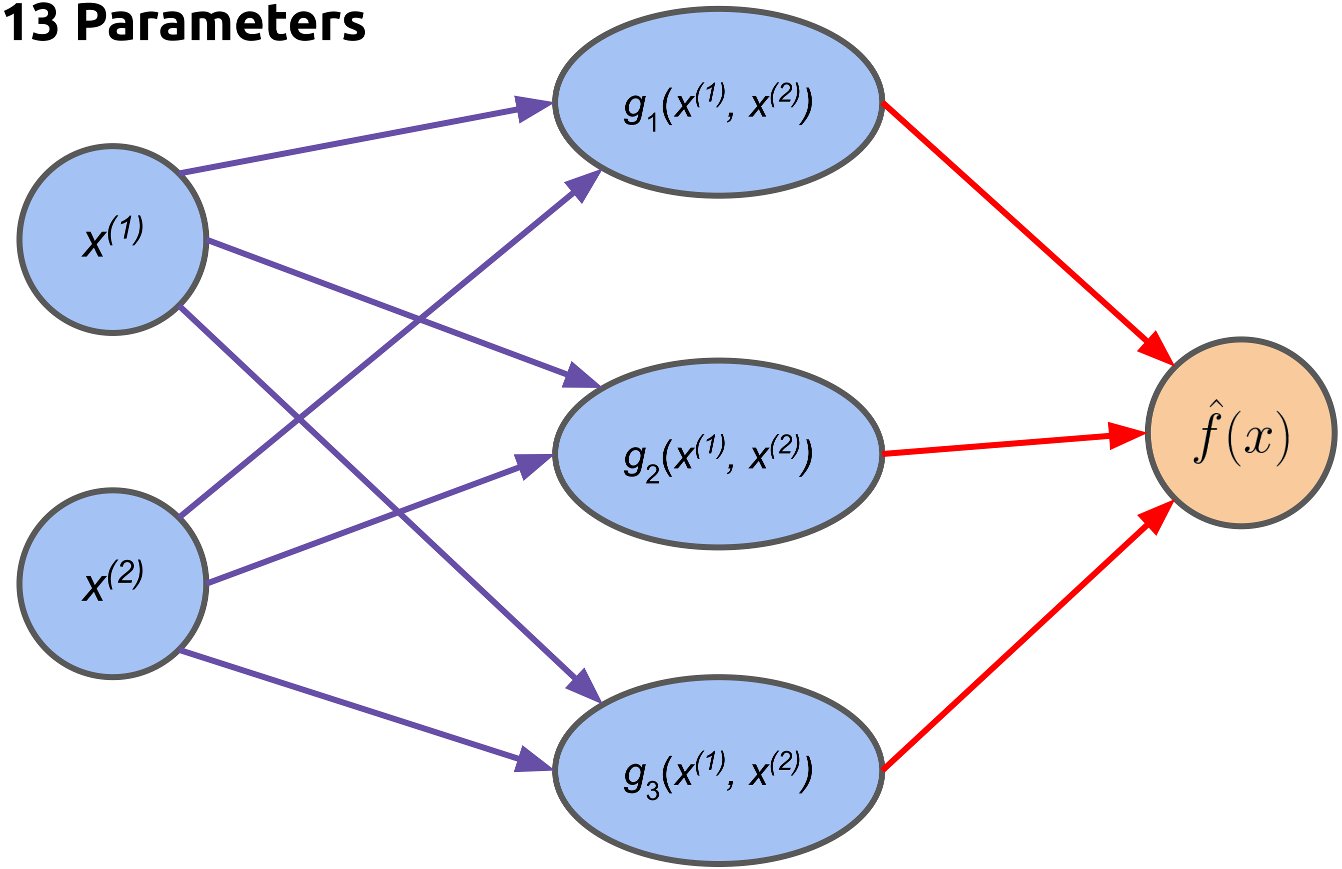
# 7 Parameters



**13 Parameters**



## 13 Parameters



**25 Parameters**