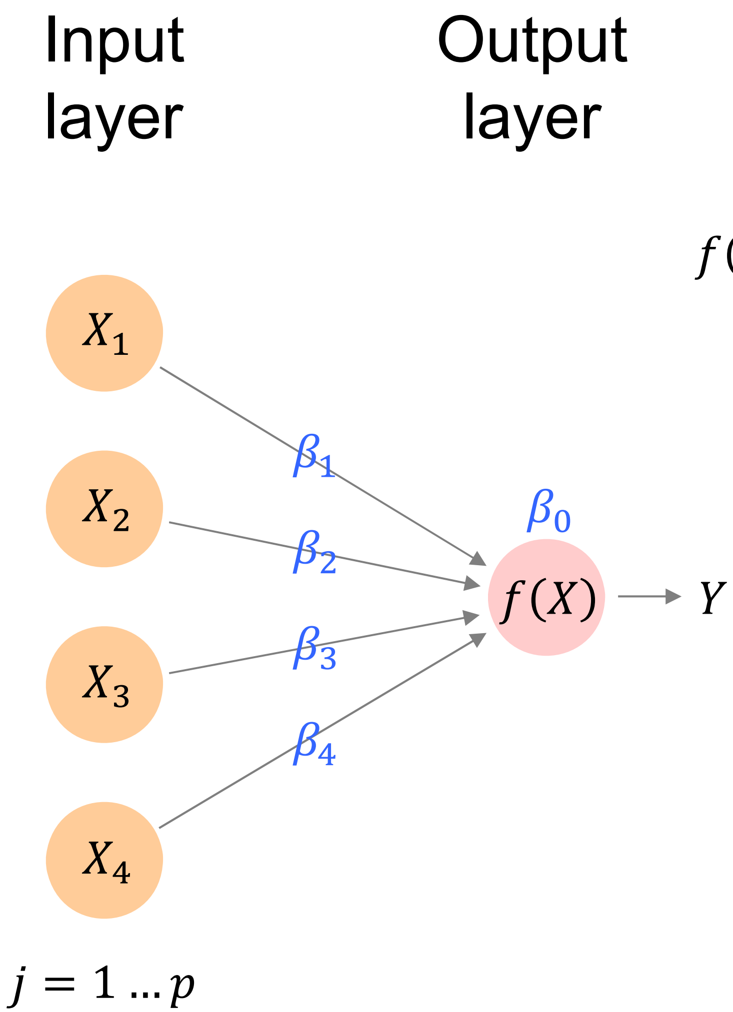
Today

- Neural networks and deep learning
  - Single layer neural networks
  - Multi-layer neural networks
  - Convolutional neural networks
  - Other: transformers, U-net, RNN, LSTM

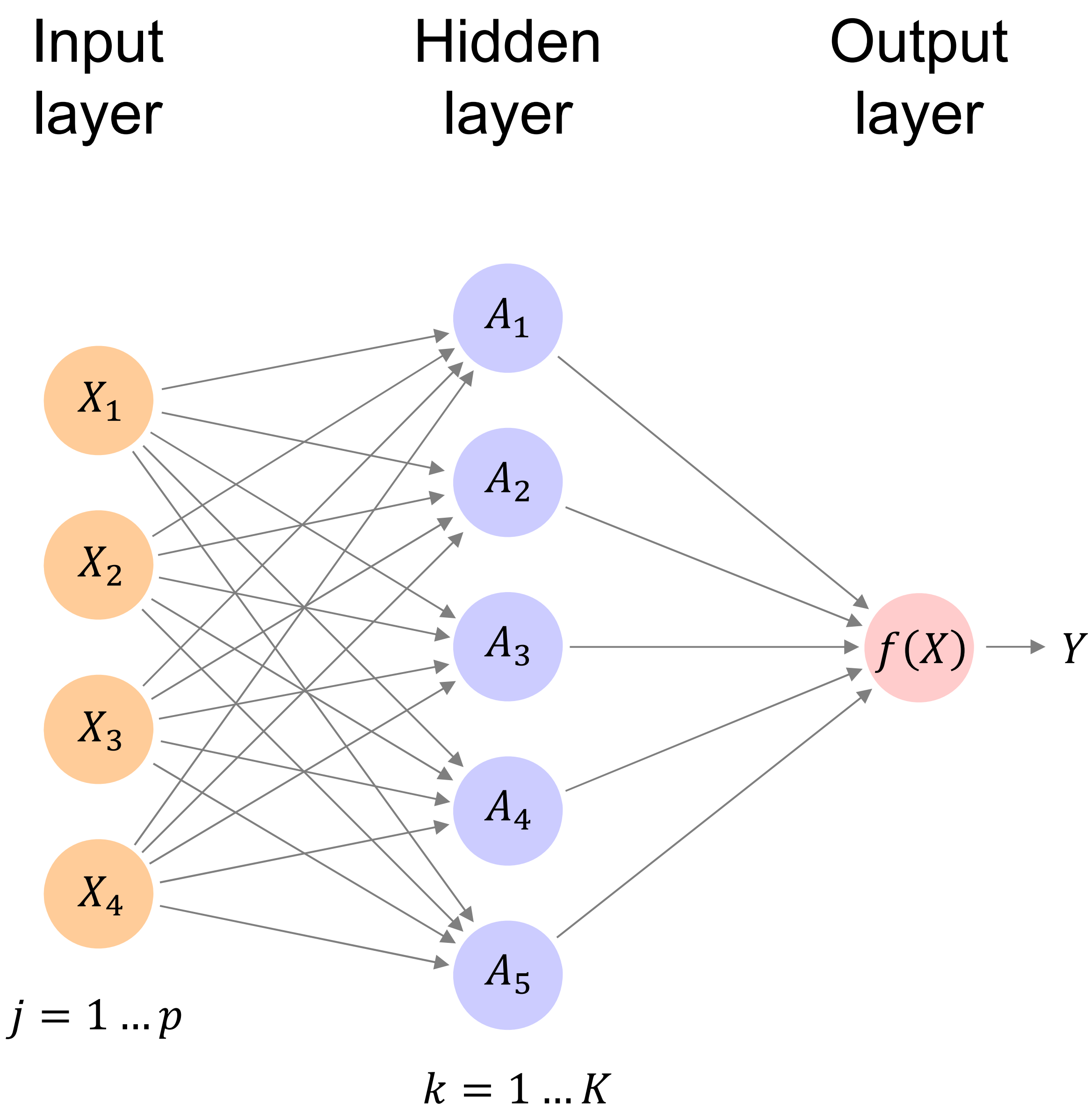
Multiple linear regression



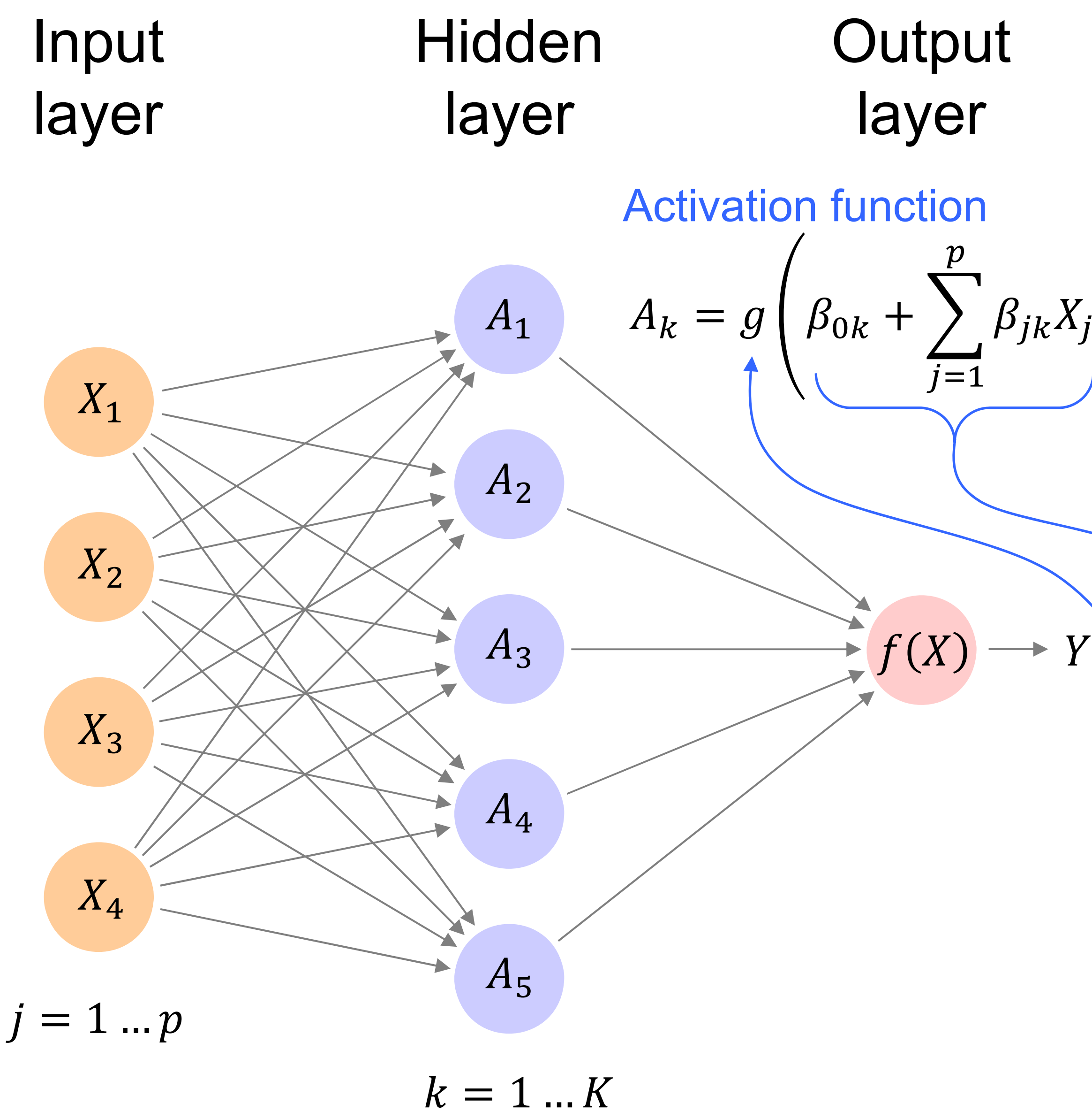
$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4$$

$$= \beta_0 + \sum_{j=1}^{p} \beta_j X_j$$

Single layer NN



Single layer NN



Activation function

$$A_k = g\left(\underbrace{\beta_{0k} + \sum_{j=1}^{p} \beta_{jk} X_j}\right)$$

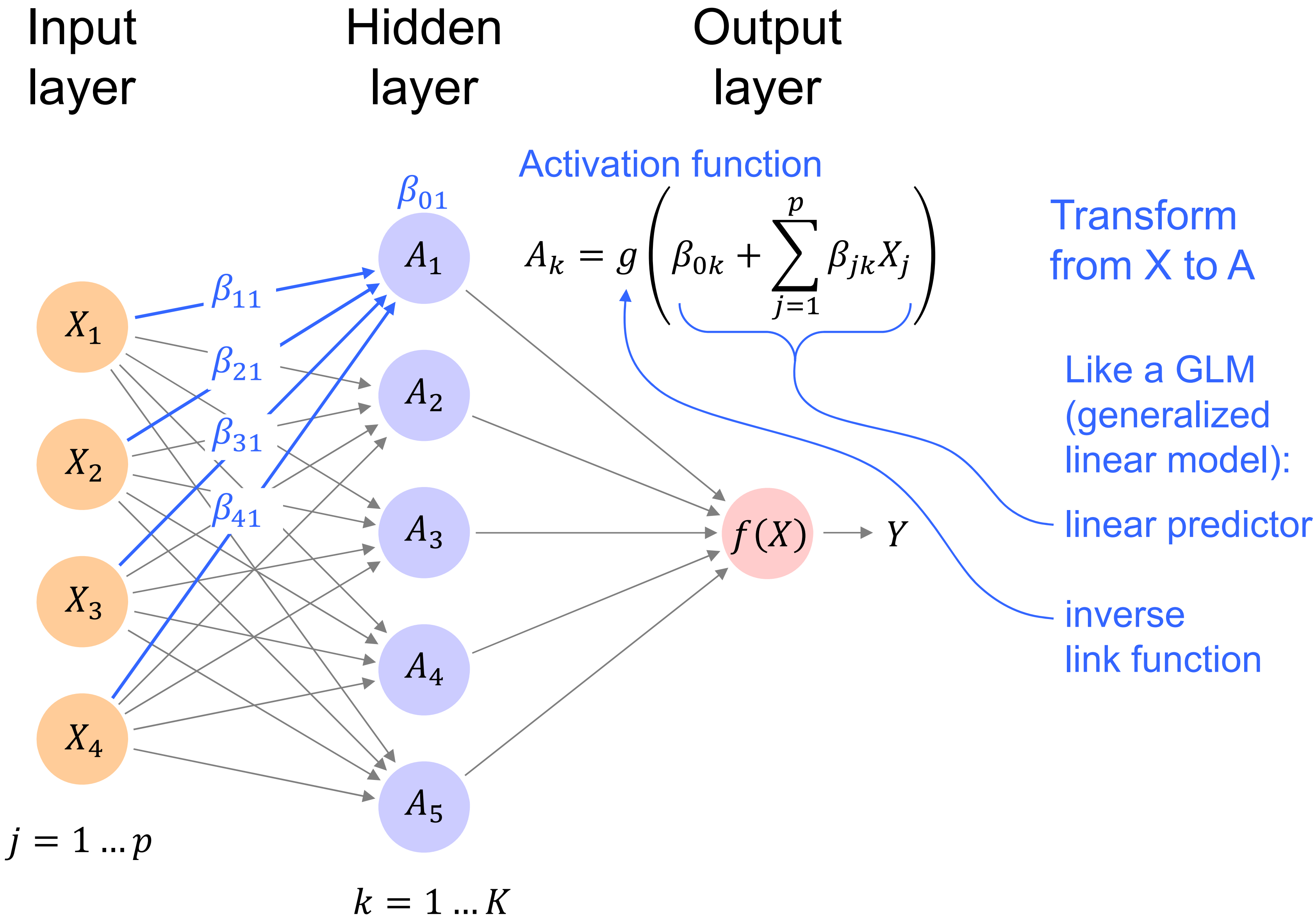
Transform from X to A

Like a GLM (generalized linear model):

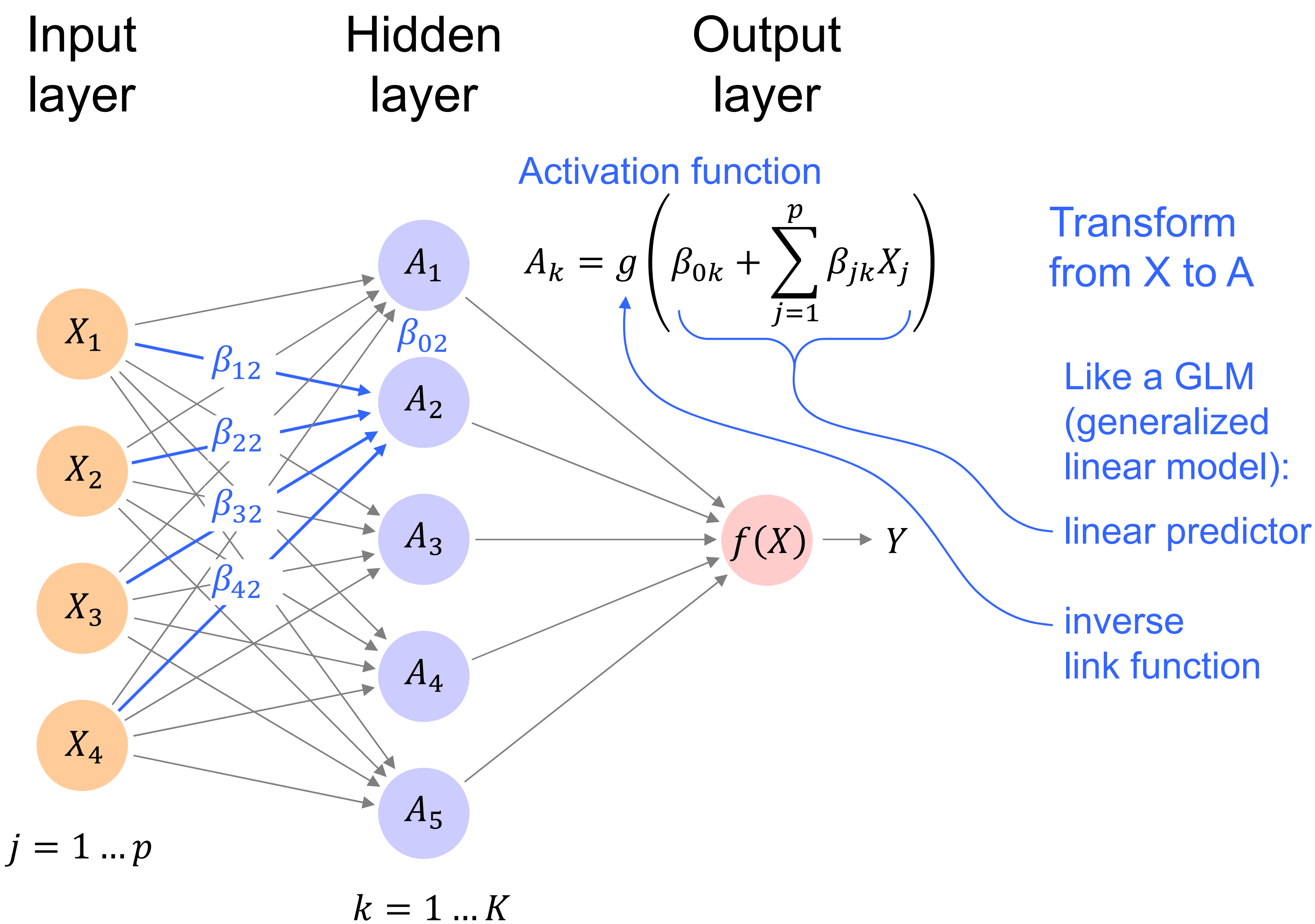
linear predictor

inverse link function

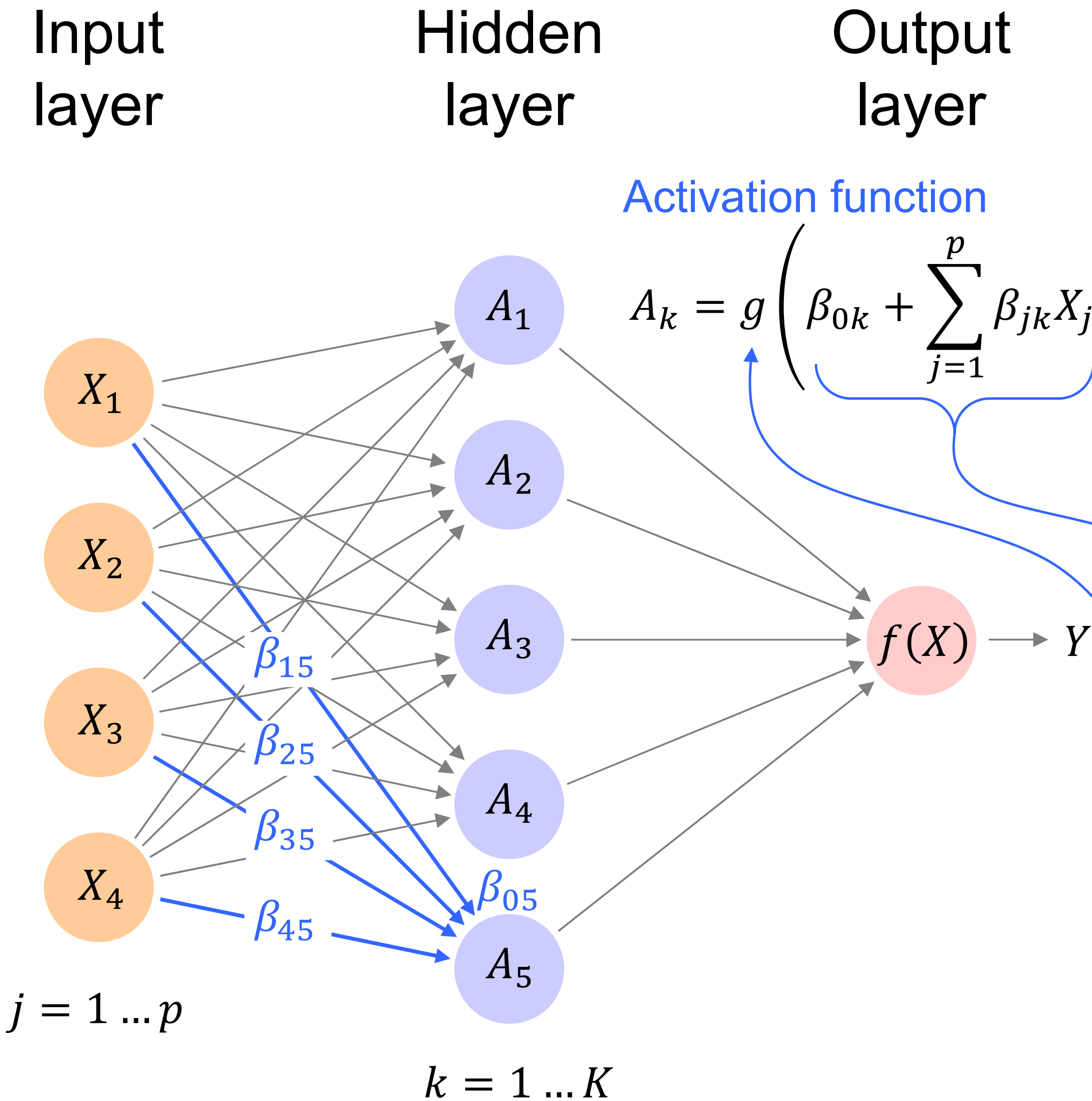
Single layer NN



Single layer NN



Single layer NN



Activation function

$$A_k = g\left(\beta_{0k} + \sum_{j=1}^{p} \beta_{jk} X_j\right)$$

Transform from X to A

Like a GLM (generalized linear model):

linear predictor

inverse link function

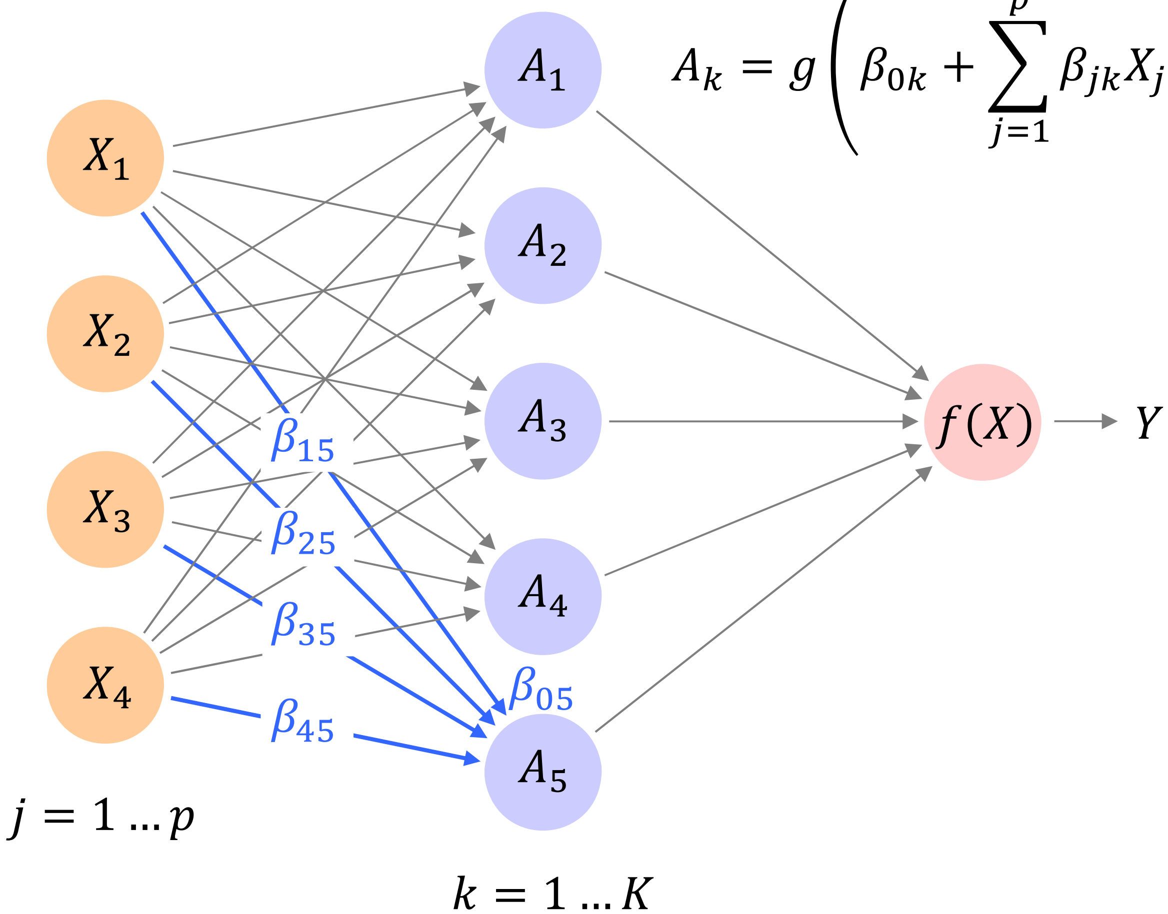
Single layer NN

aka
(logit link)$^{-1}$

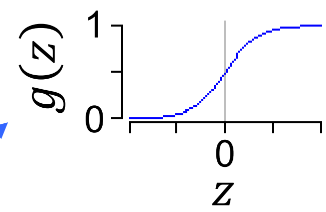
Input layer | Hidden layer | Output layer

Activation function

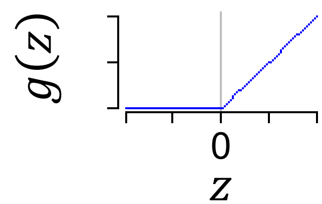
$$A_k = g\left(\beta_{0k} + \sum_{j=1}^{p} \beta_{jk} X_j\right)$$



Sigmoid

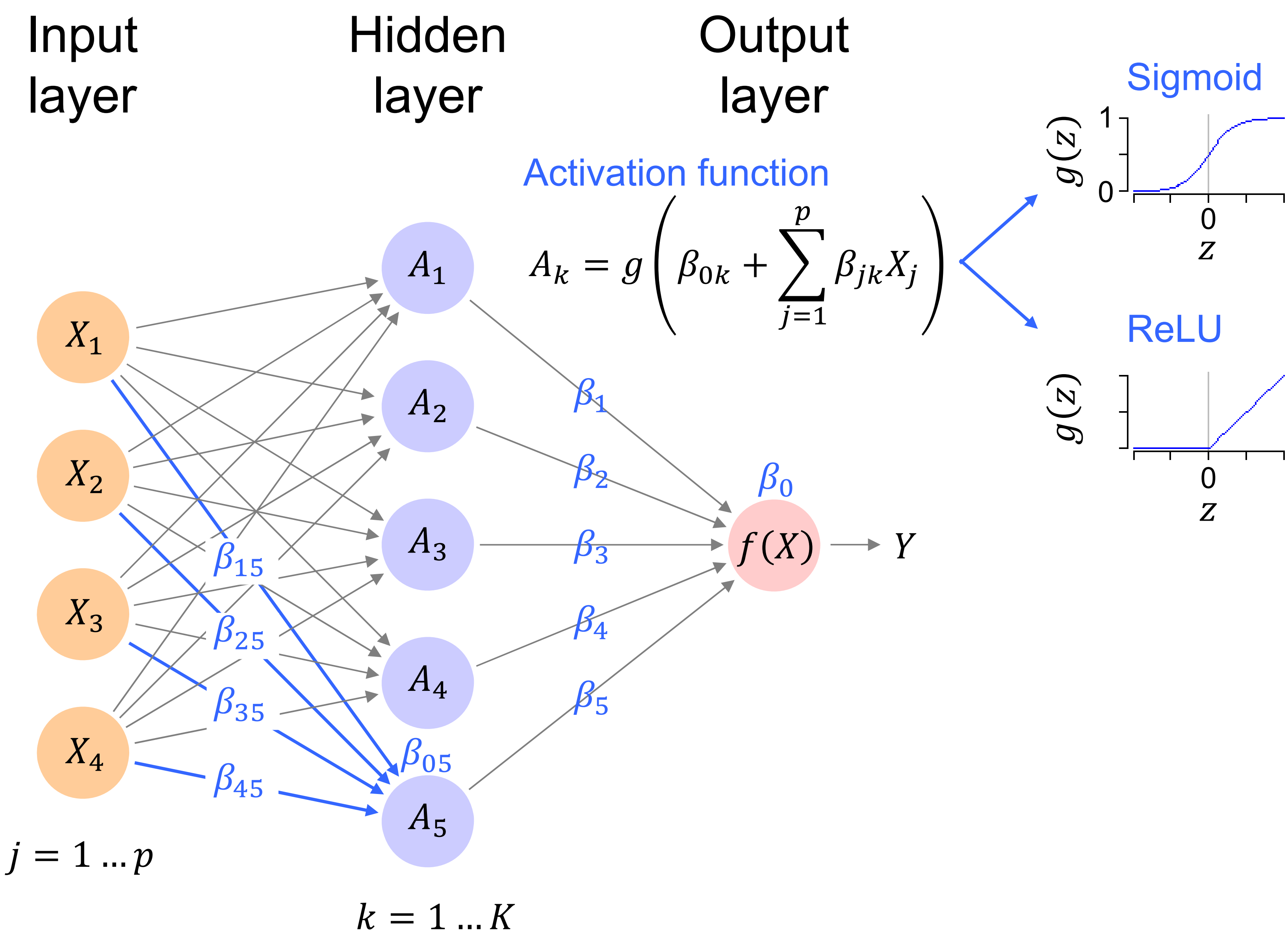


ReLU

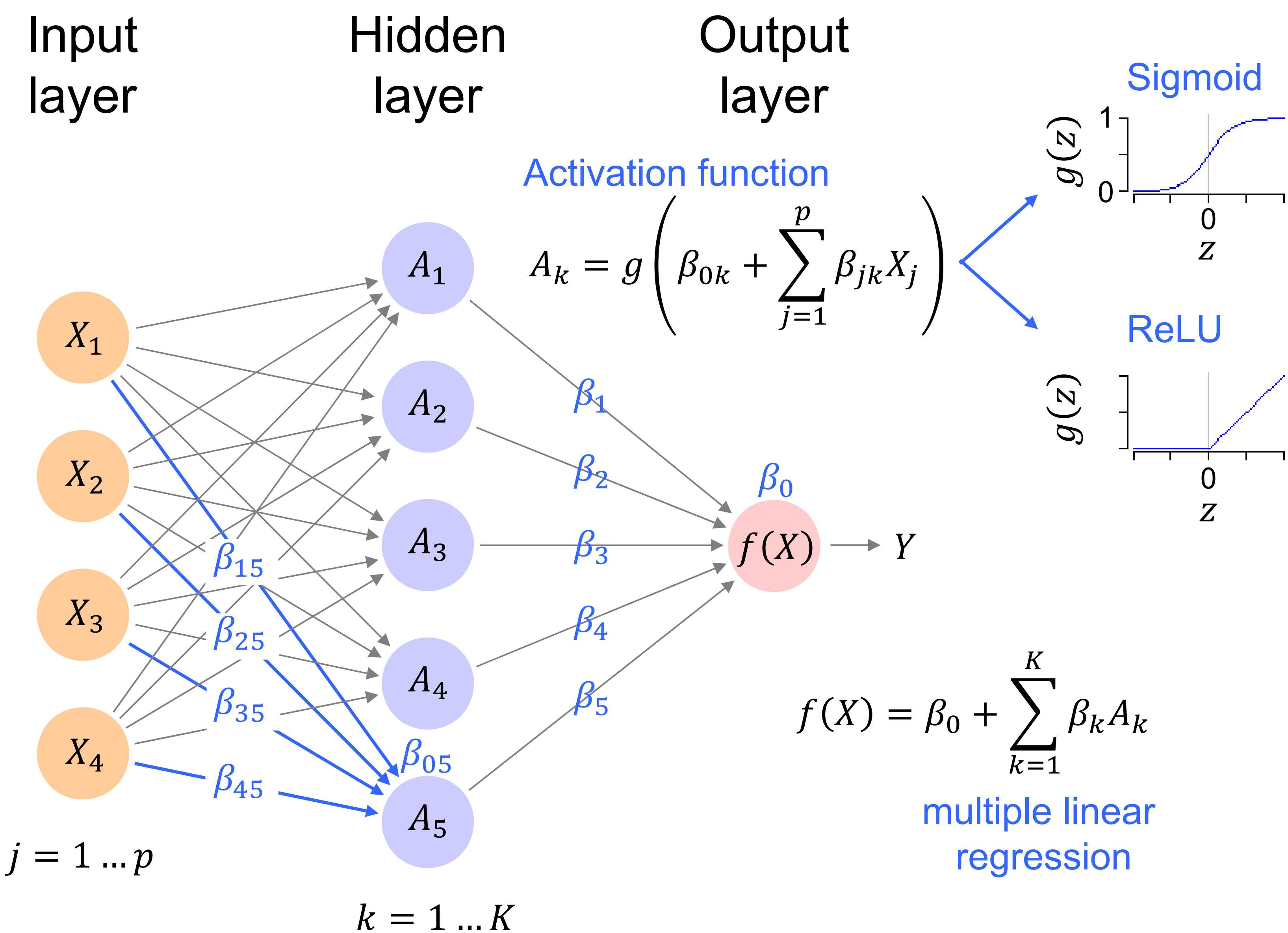


Rectified
linear unit

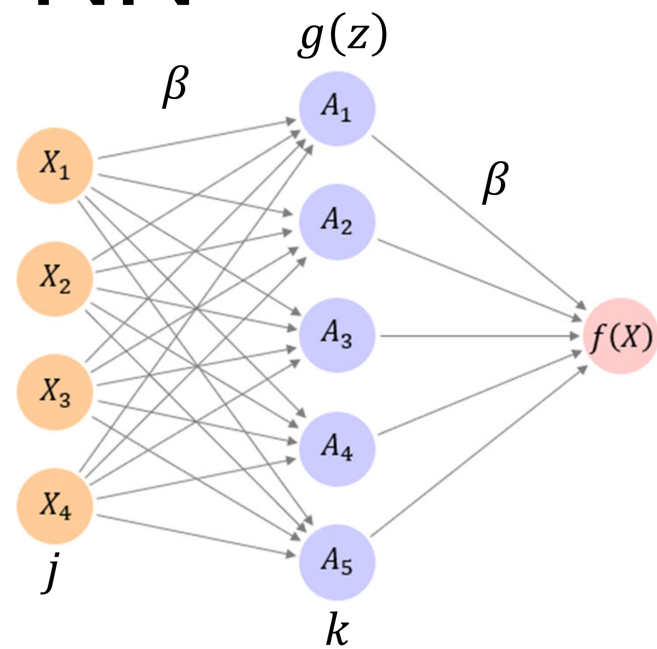
Single layer NN



Single layer NN



Single layer NN



Model algorithm

define $g(z)$

load x_j

set K

set parameters: $\beta_{..}$

for each activation unit k in 1:K

    calculate linear predictor: $z_k = \beta_{0k} + \sum_j \beta_{jk} x_j$

    calculate nonlinear activation: $A_k = g(z_k)$

calculate linear model: $f(x) = \beta_0 + \sum_k \beta_k A_k$

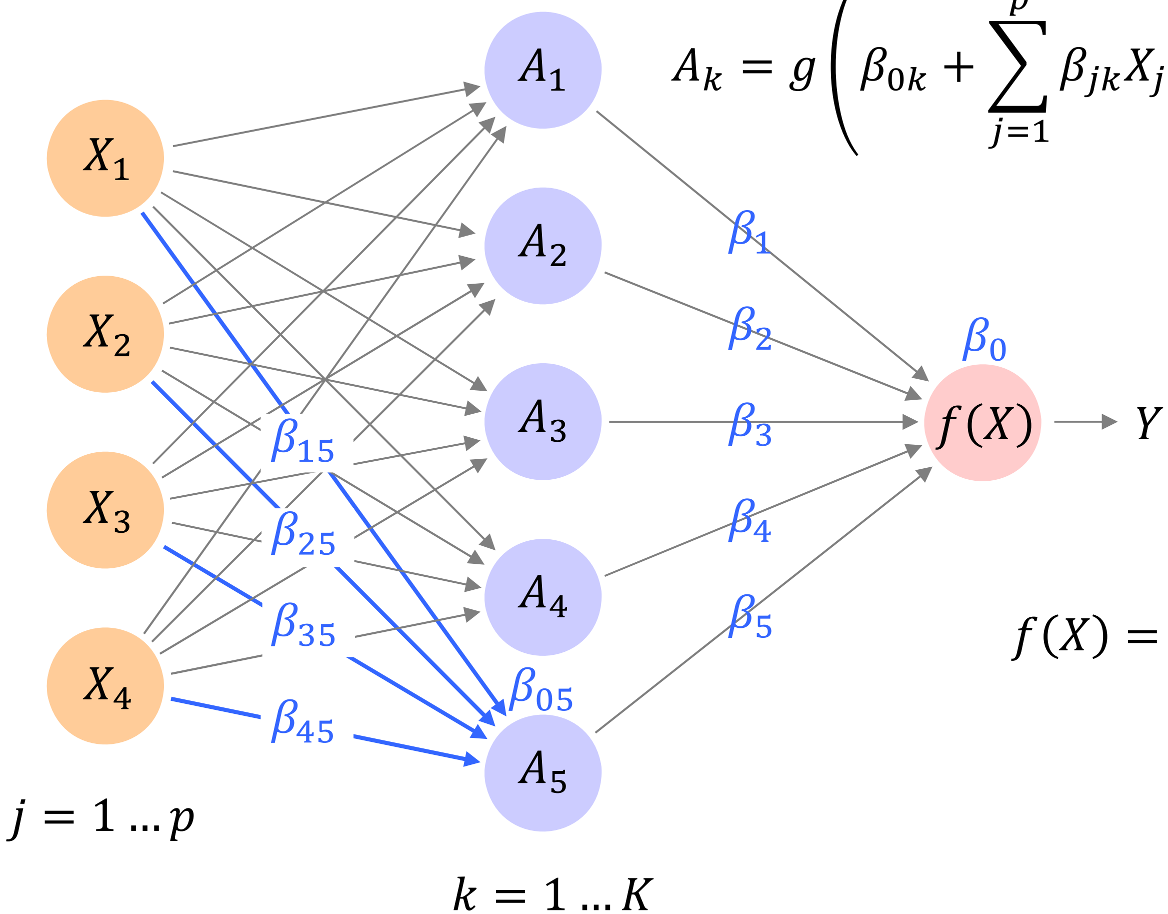
return $f(x)$

Single layer NN (generalizing)

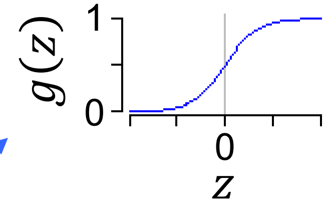
Input layer | Hidden layer | Output layer

Activation function

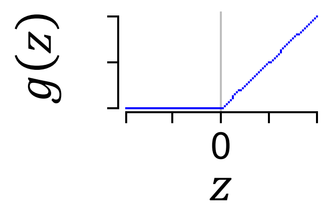
$$A_k = g\left(\beta_{0k} + \sum_{j=1}^{p} \beta_{jk} X_j\right)$$



Sigmoid

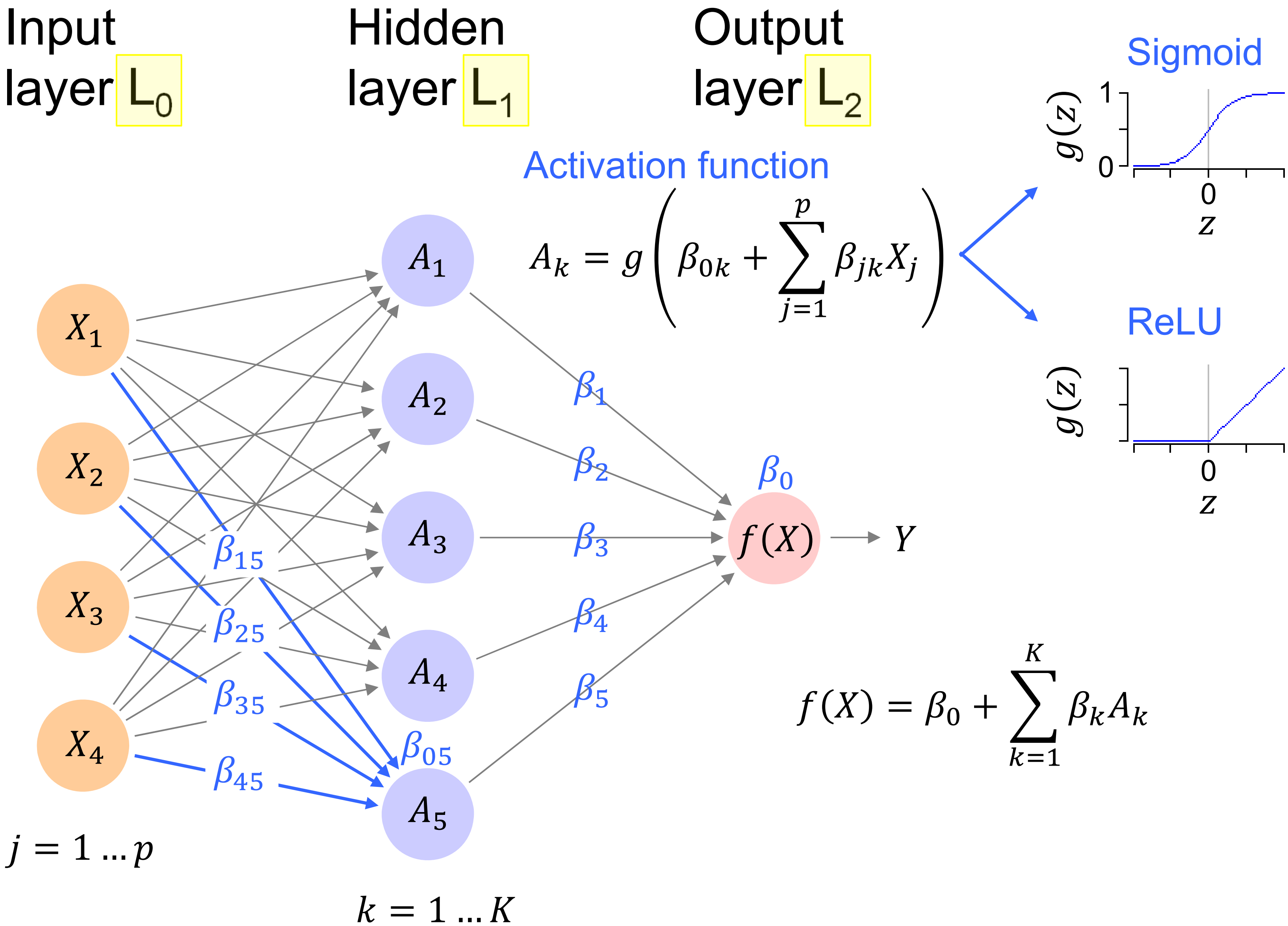


ReLU



$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k$$

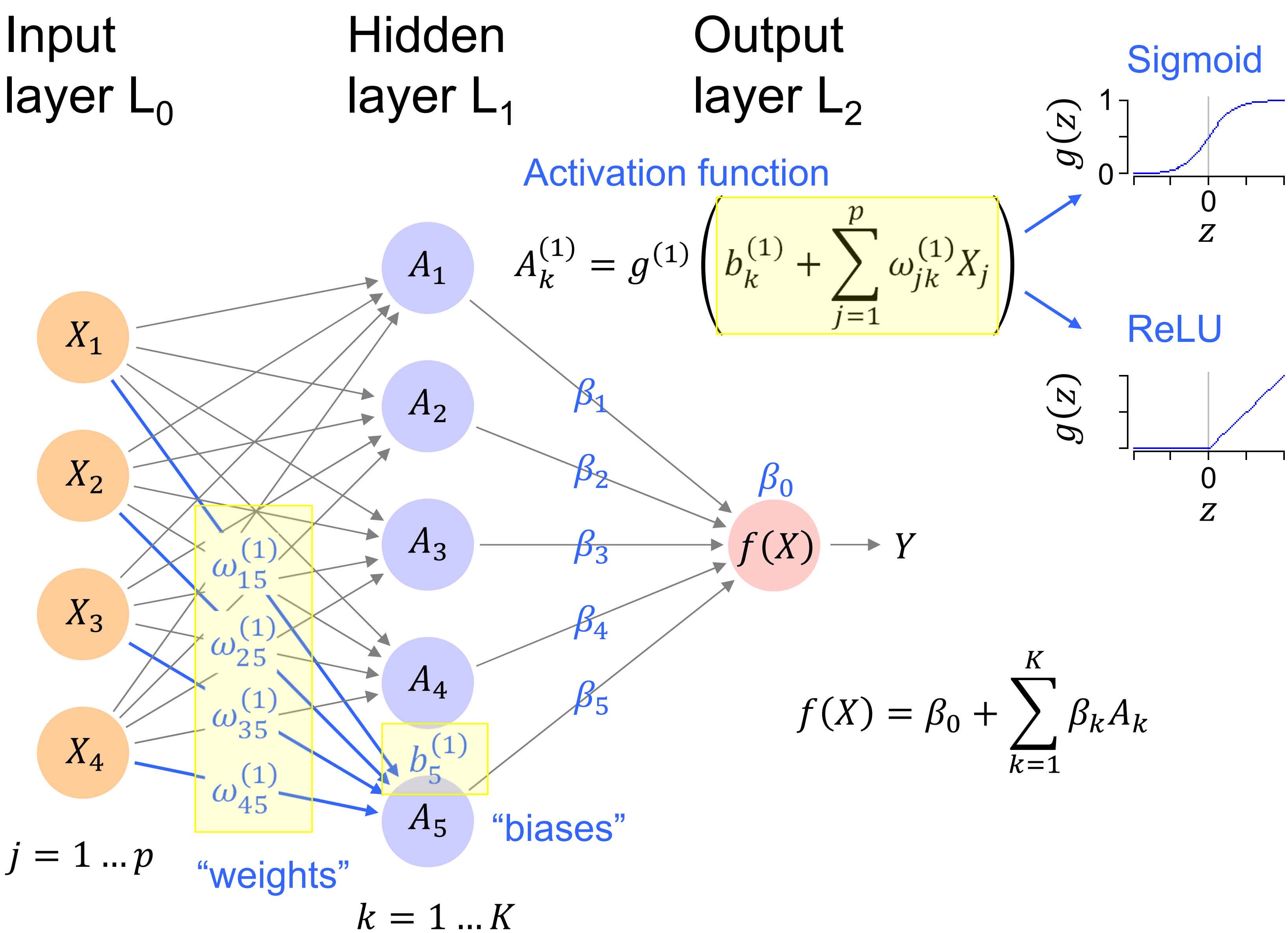
Single layer NN (generalizing)



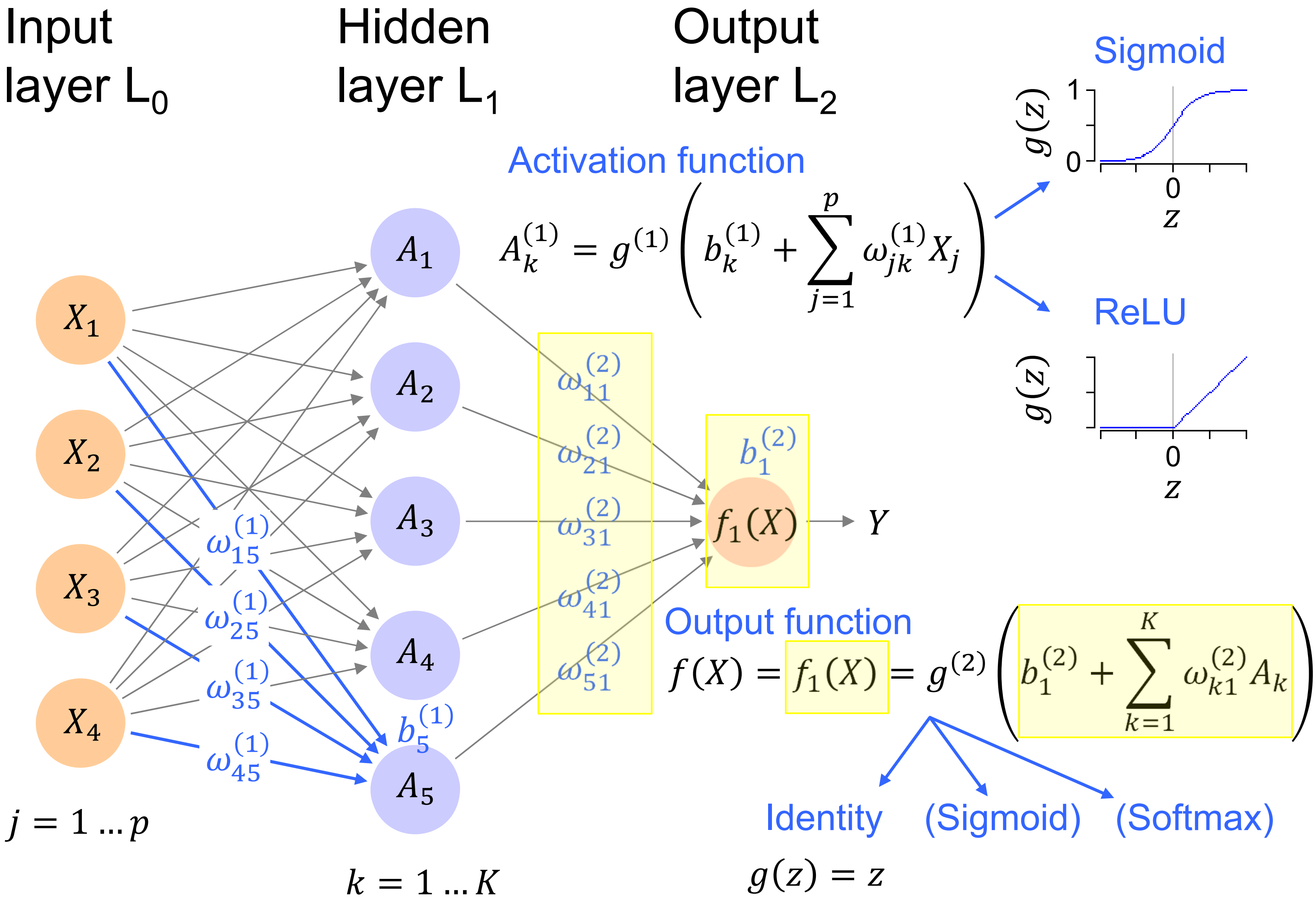
$$A_k = g\left(\beta_{0k} + \sum_{j=1}^{p} \beta_{jk} X_j\right)$$

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k$$

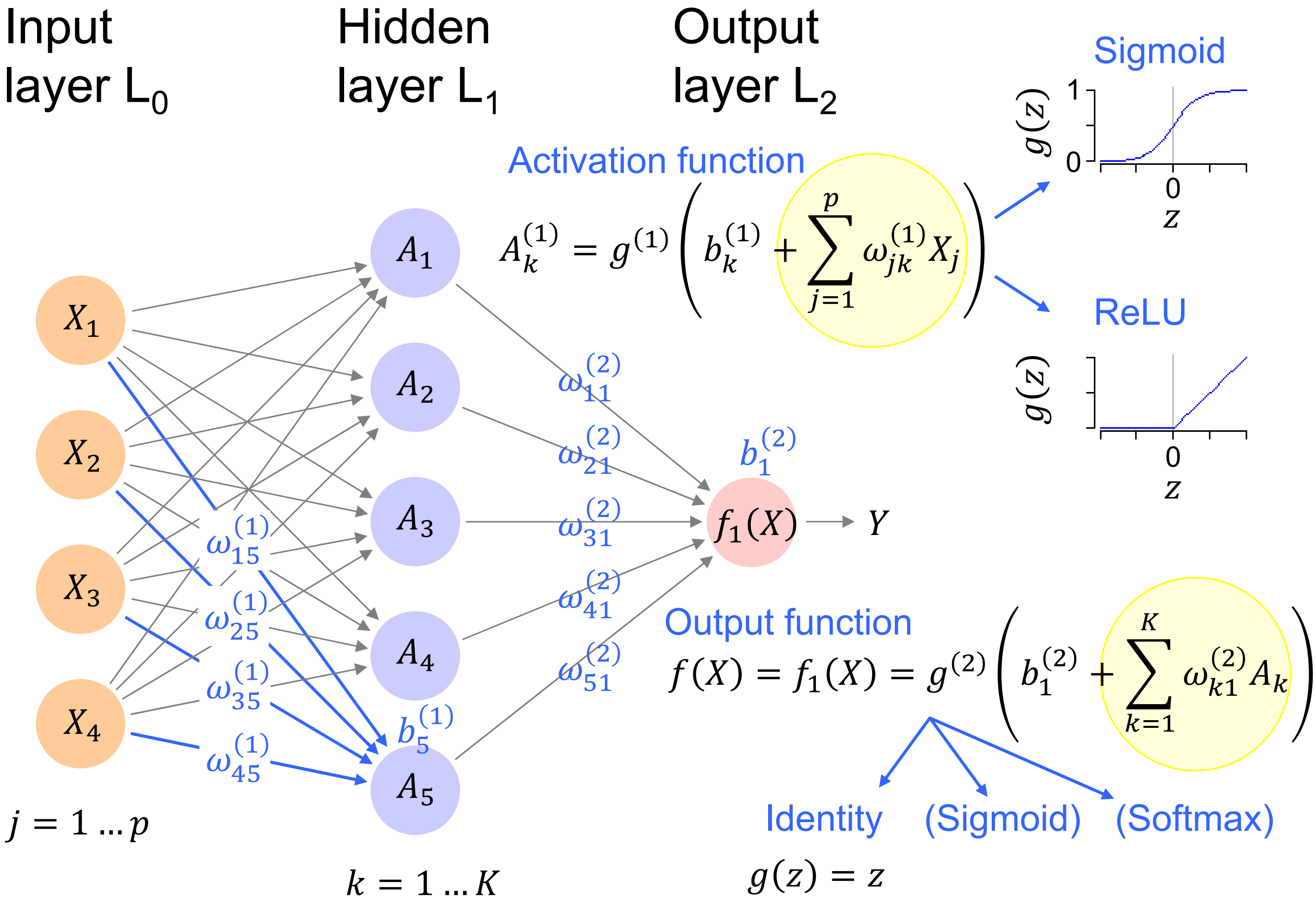
Single layer NN (generalizing)



Single layer NN (generalizing)



Single layer NN (generalizing)



Single layer NN (generalizing)

$$\sum_{j=1}^{p} \omega_{jk}^{(1)} X_j$$

multiply down columns
then add across rows

$$\begin{array}{ccccccc}
\omega_{1k} & & \omega_{2k} & & \omega_{3k} & & \omega_{4k} \\
* & + & * & + & * & + & * \\
X_1 & & X_2 & & X_3 & & X_4
\end{array}$$

data rows
i = 1...n

$$\begin{array}{cccc}
x_{11} & x_{12} & x_{13} & x_{14} \\
x_{21} & x_{22} & x_{23} & x_{24} \\
\vdots & \vdots & \vdots & \vdots \\
x_{n1} & x_{n2} & x_{n3} & x_{n4}
\end{array}$$

data columns
j = 1...p

Matrix multiplication

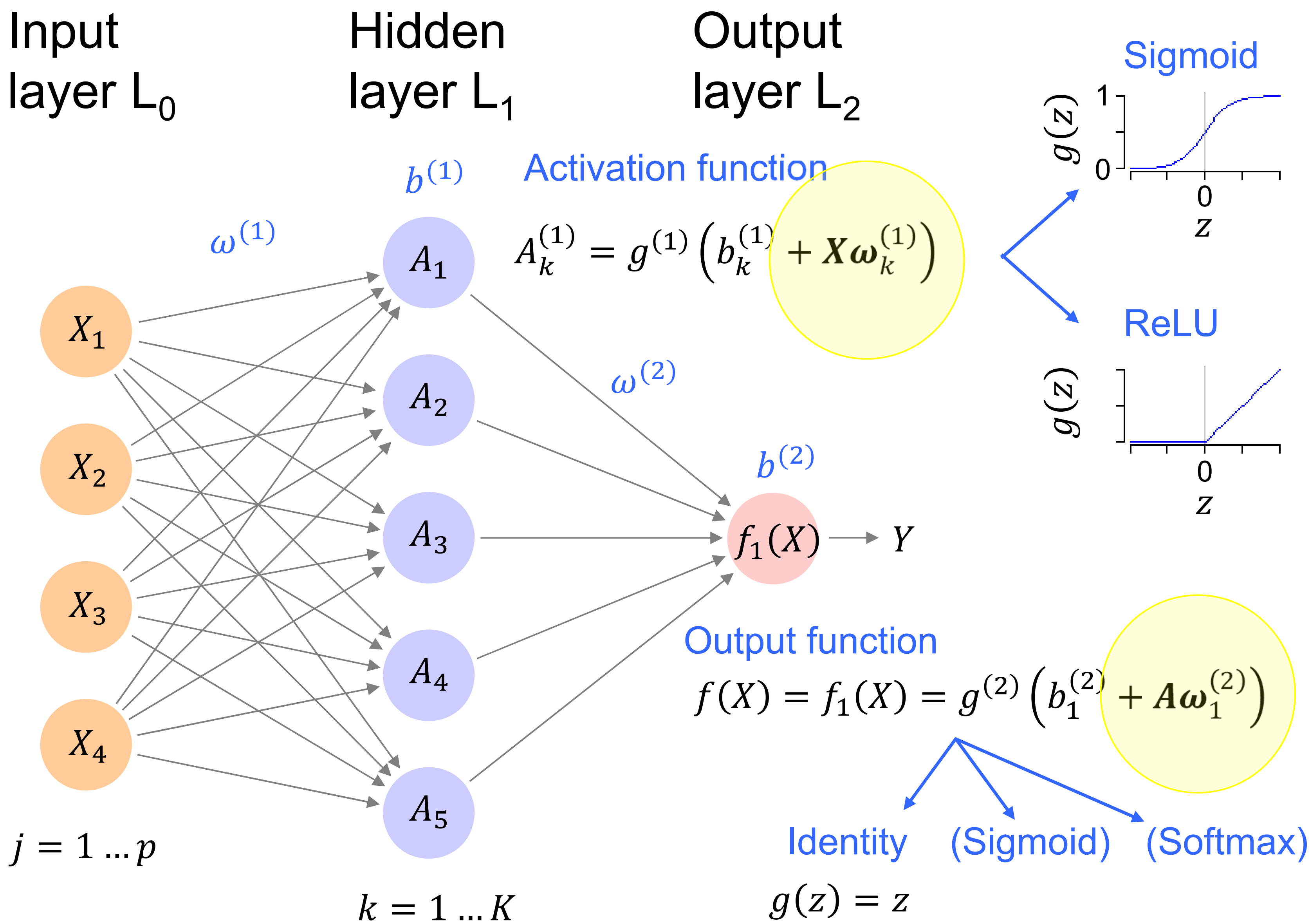
$$\boldsymbol{X\omega}_k$$

n x p   p x 1

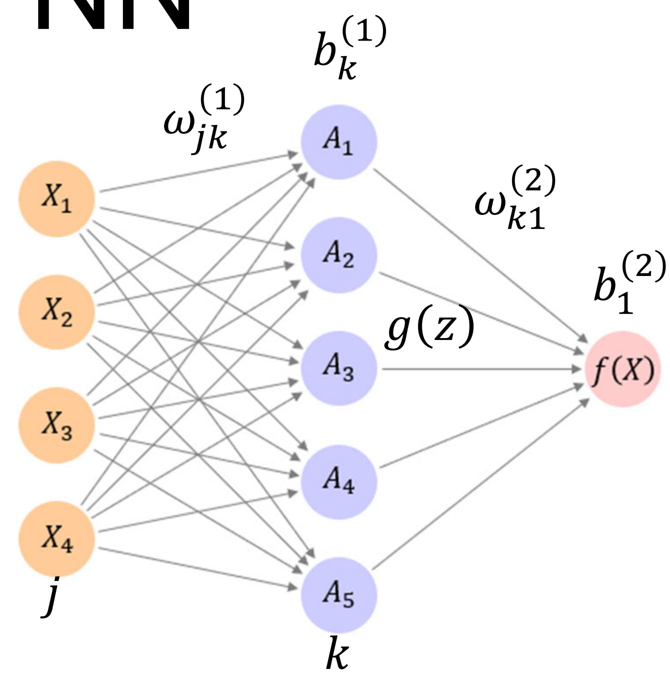
→ n x 1

R: `x %*% w`

Single layer NN (generalizing)



Single layer NN



Model algorithm

define $g(z)$

load and prepare x_j

set K

set $\omega_{jk}^{(1)}$, $b_k^{(1)}$, $\omega_{k1}^{(2)}$, $b_1^{(2)}$

for each activation unit k in 1:K

  calculate linear predictor: $z_k = b_k^{(1)} + \boldsymbol{X\omega}_k^{(1)}$

  calculate nonlinear activation: $A_k = g(z_k)$

calculate linear model: $f(x) = b_1^{(2)} + \boldsymbol{A\omega}_1^{(2)}$

return $f(x)$

Code: ants_neural_net.R

Training algorithm

Loss function (e.g. MSE)

$$\mathrm{MSE}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f_\theta(x_i)\right)^2$$

$$\theta = \begin{matrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{10} \\ \beta_{11} \\ \vdots \end{matrix}$$

Training algorithm

Stochastic gradient descent

guess θ (typically random)

set λ (learning rate)

for iterations (e.g. until $\mathrm{MSE}(\theta)$ stops decreasing)

  randomly sample the data

  calculate gradient of $\mathrm{MSE}(\theta)$: $\frac{\delta \mathrm{MSE}(\theta)}{\delta\theta}$

  Method: back propagation

  $\theta \leftarrow \theta - \lambda \frac{\delta \mathrm{MSE}(\theta)}{\delta\theta}$

Training algorithm

Stochastic gradient descent (mini batch)

guess θ (typically random)

set λ (learning rate)

for many epochs

    randomly partition data into batches

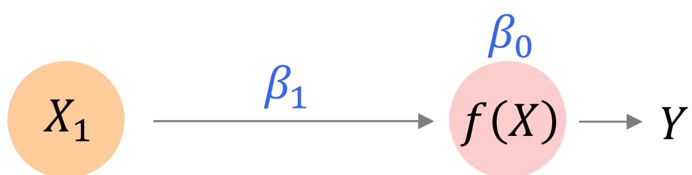
    for each batch

        calculate gradient of $\mathrm{MSE}(\theta)$: $\frac{\delta \mathrm{MSE}(\theta)}{\delta \theta}$

        $\theta \leftarrow \theta - \lambda \frac{\delta \mathrm{MSE}(\theta)}{\delta \theta}$

Training algorithm

e.g. simplest neural net!



$$f(X) = \beta_0 + \beta_1 X_1$$

Code: mini_batch_stoch_gradient_descent.R