STATISTICAL AND MACHINE LEARNING FOR BIG GEOSPATIAL DATA: Part III

Abhi Datta
Johns Hopkins University
Department of Biostatistics

Overview of Part III

Introduction to feed-forward neural networks
- Terminology and network architecture
- Computational techniques

Neural networks for geospatial analysis
- Residual kriging
- Added spatial features
- Issues: Not modeling spatial correlation

NN-GLS: Combining neural networks and Gaussian processes for spatial data
- Representation as graph-neural network
- Estimation and prediction

Non-linear regression

$Y_i = m(X_i) + \epsilon_i$

Many choices for modeling m

- Basis functions
- GAM
- Regression trees and random forests

Non-linear regression

$Y_i = m(X_i) + \epsilon_i$

Many choices for modeling m

- Basis functions
  - Curse of dimensionality with increase in covariate dimension
- GAM
  - Cannot model interactions
- Regression trees and random forests
  - Estimates are discontinuous
  - Slow for larger datasets due to requiring brute force grid search for tree partitioning

Non-linear regression

$$Y_i = m(X_i) + \epsilon_i$$

Many choices for modeling m

- Basis functions
- GAM
- Regression trees and random forests
- Neural networks

Feed-forward Neural networks

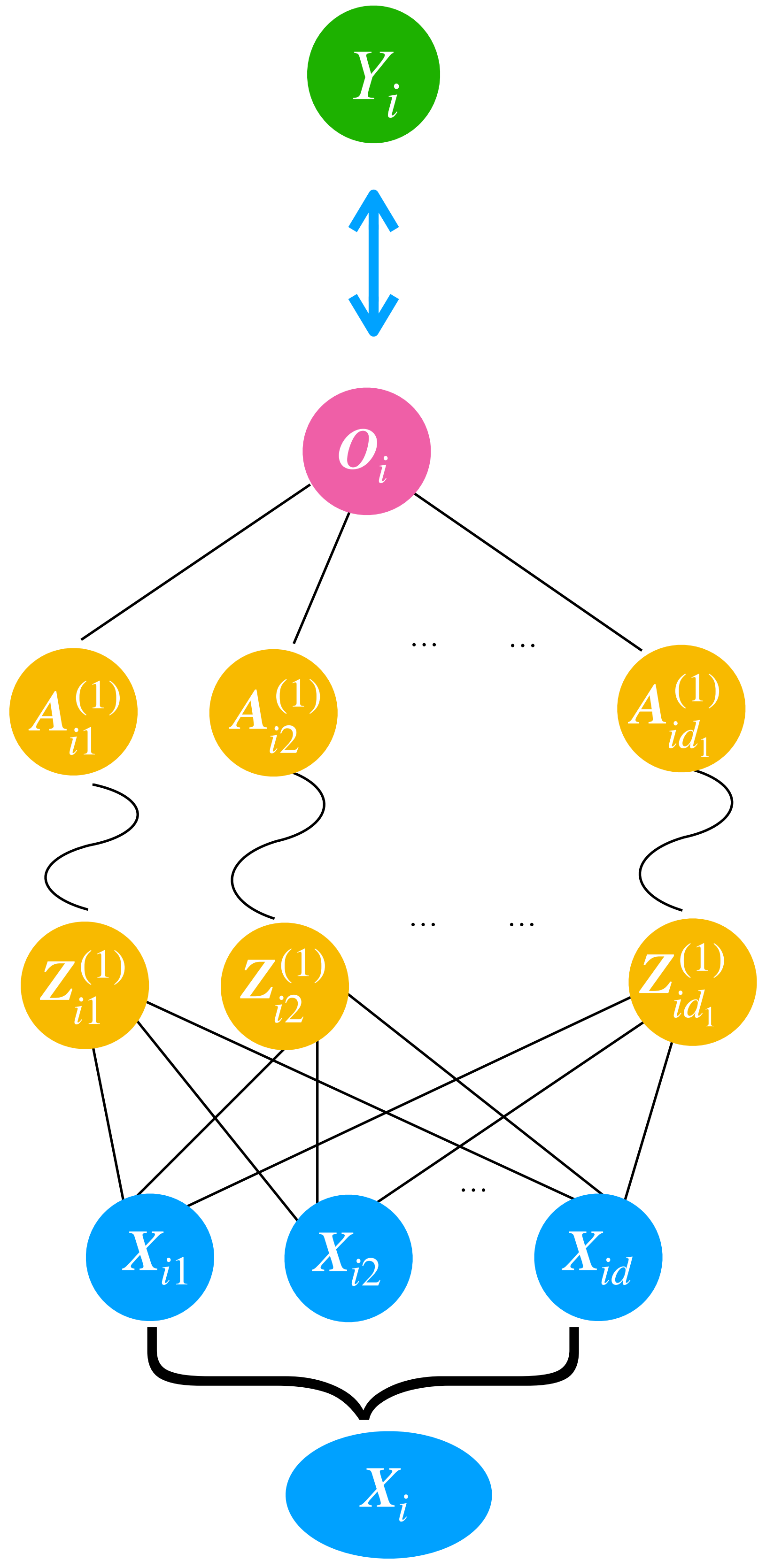
Single-layer perceptron

$$Y_i = m(X_i) + \epsilon_i$$

Single layer perceptron model for m:

$$m(X_i) = \beta' g_1(W_1 * X_i)$$

Response



Covariates X_i

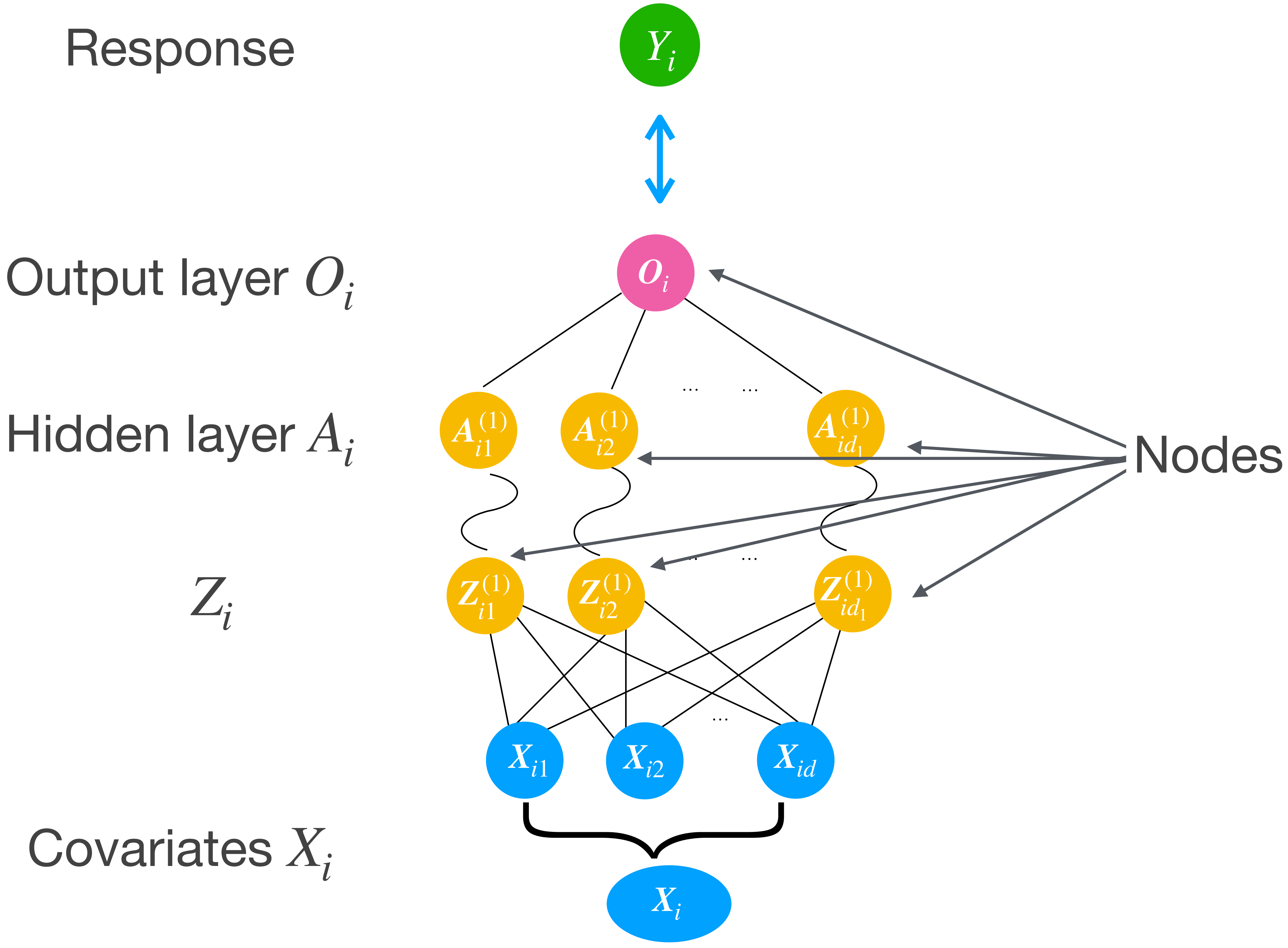
Feed-forward Neural networks

Nodes, layers, width

$$Y_i = m(X_i) + \epsilon_i$$

Single layer perceptron model for m:

$$m(X_i) = \beta' g_1(W_1 * X_i)$$



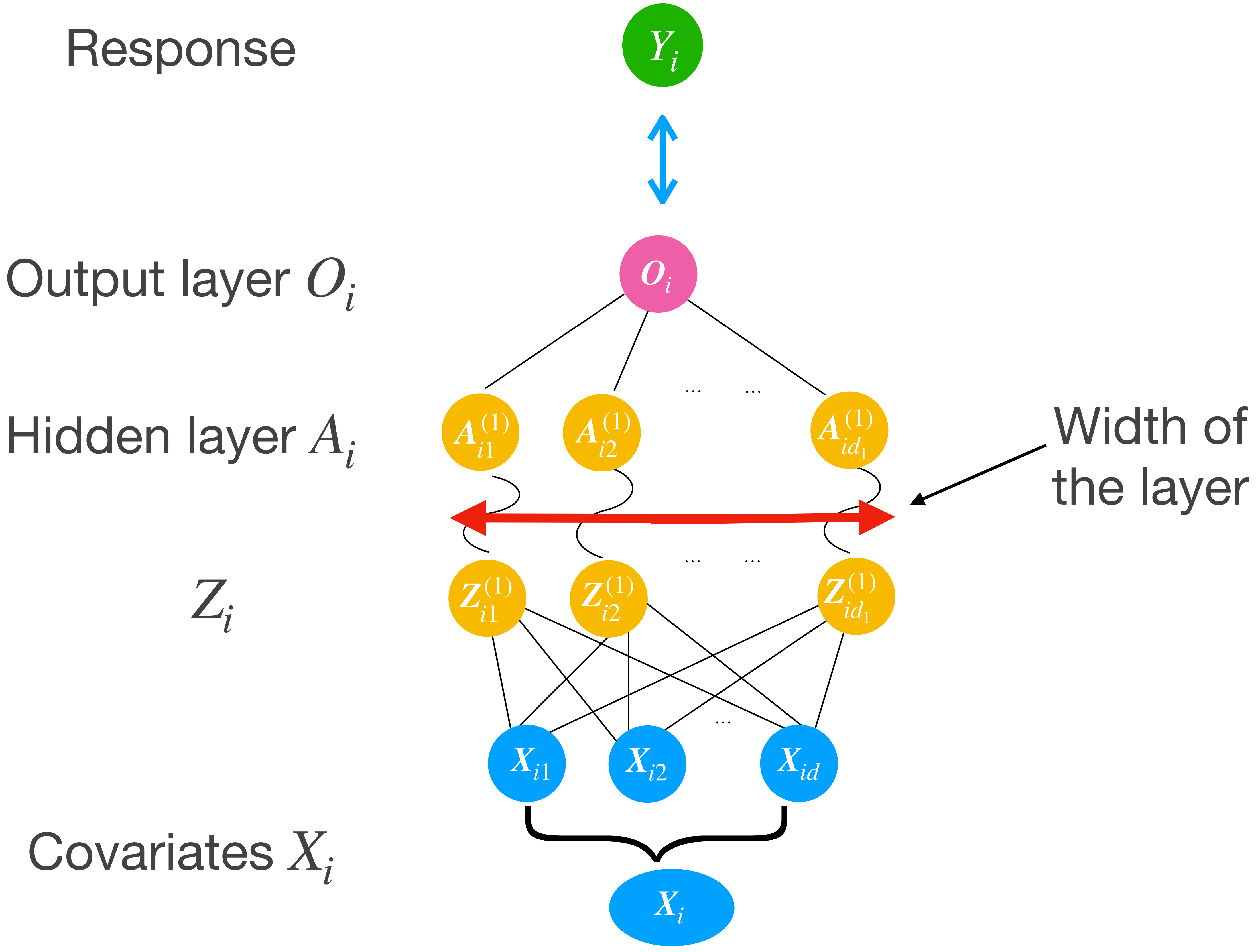
Feed-forward Neural networks

Nodes, layers, width

$$Y_i = m(X_i) + \epsilon_i$$

Single layer perceptron model for m:

$$m(X_i) = \beta' g_1(W_1 * X_i)$$



Feed-forward Neural networks

Weights and biases

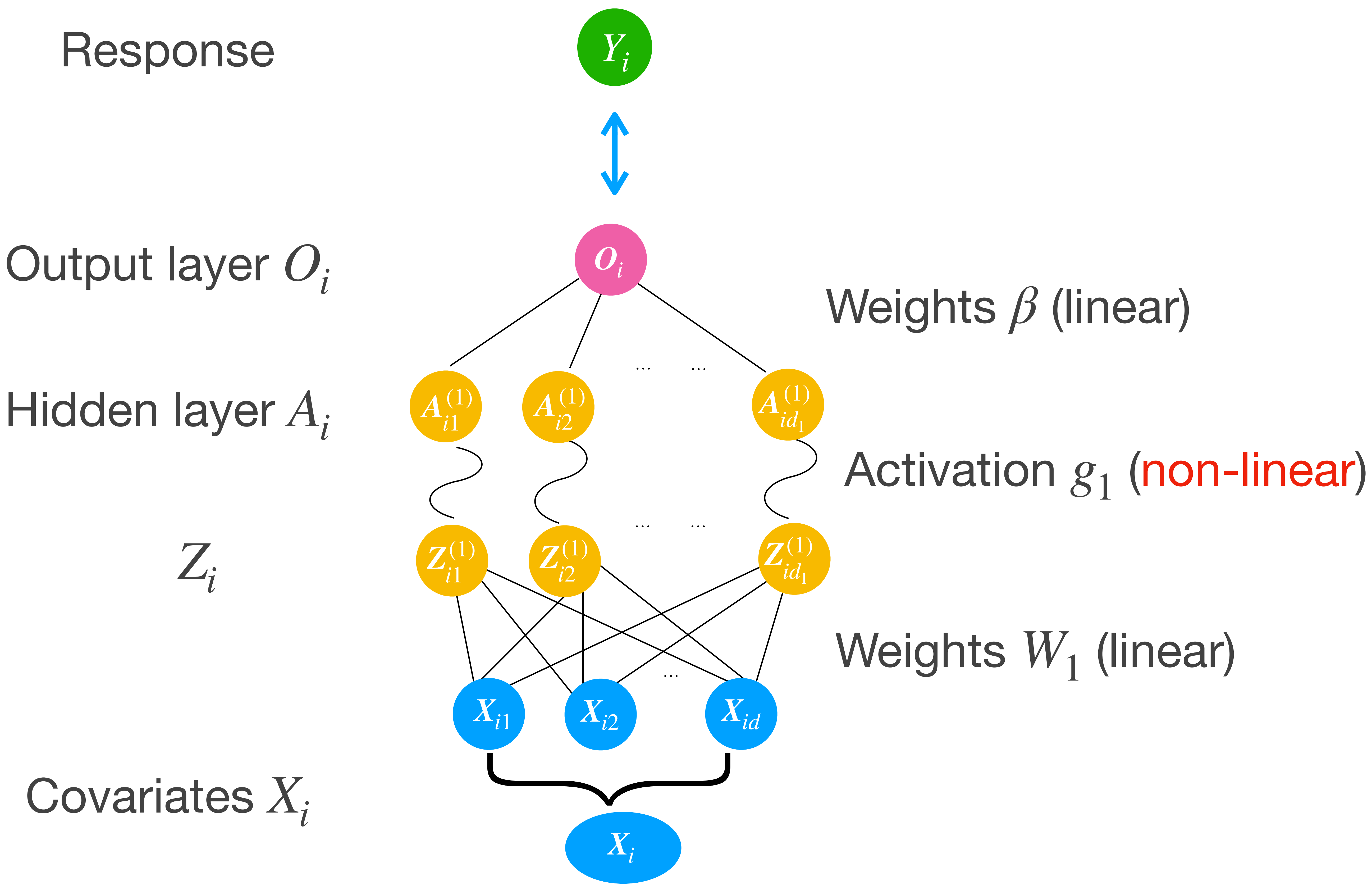
$$Y_i = m(X_i) + \epsilon_i$$

Single layer perceptron model for m:

$$m(X_i) = \beta' g_1(W_1 * X_i)$$

W_1 and β are the weights (coefficients)

Weights are unknown and are estimated



Feed-forward Neural networks

Weights and biases

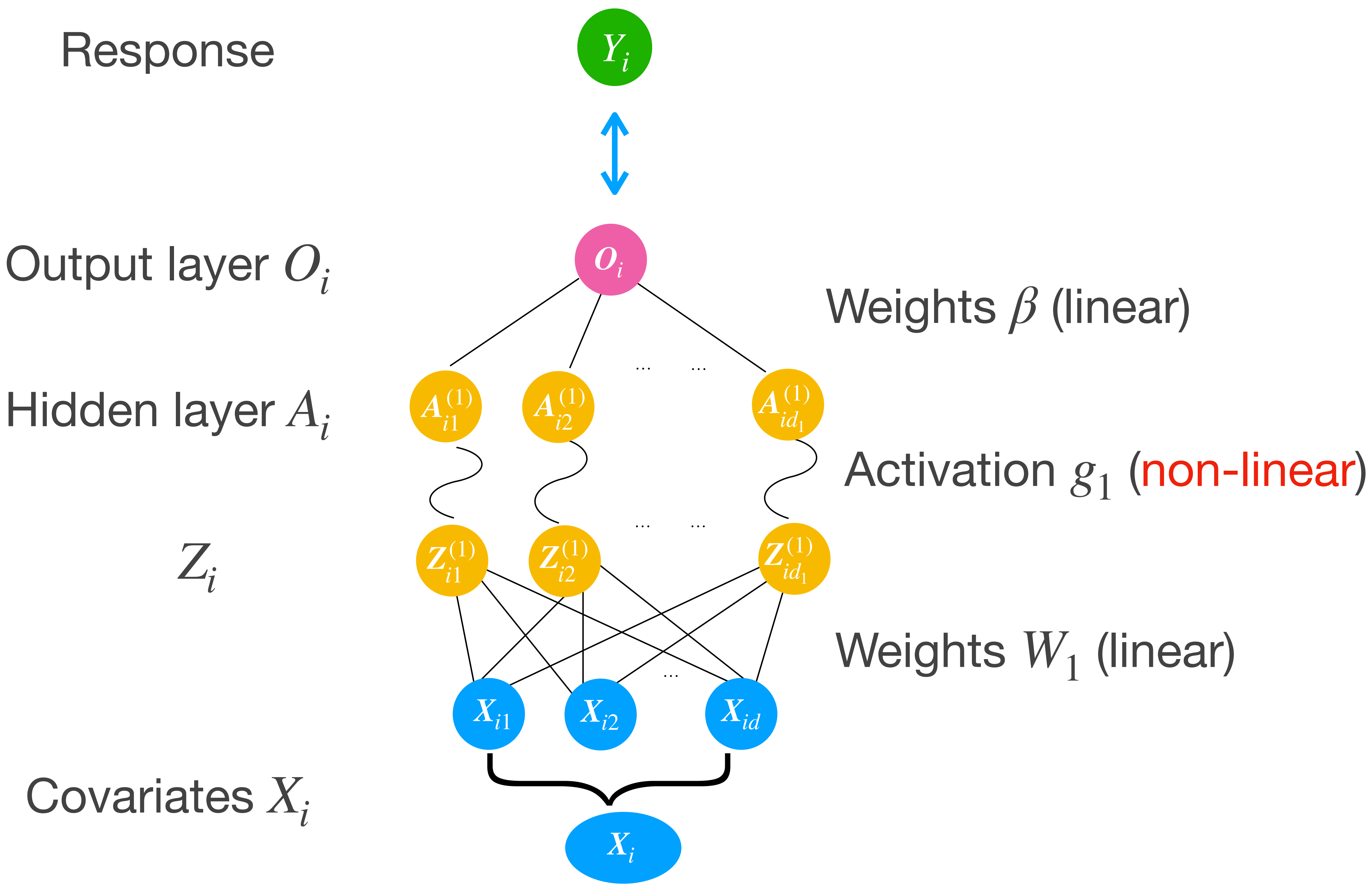
$$Y_i = m(X_i) + \epsilon_i$$

Single layer perceptron model for m:

$$m(X_i) = \beta' g_1(W_1 * X_i)$$

W_1 and β are the weights (coefficients)

Weights are unknown and are estimated



Often, an intercept is included in X_i and each hidden layer.

The coefficients corresponding to the intercepts is often called biases

Feed-forward Neural networks

Activation function

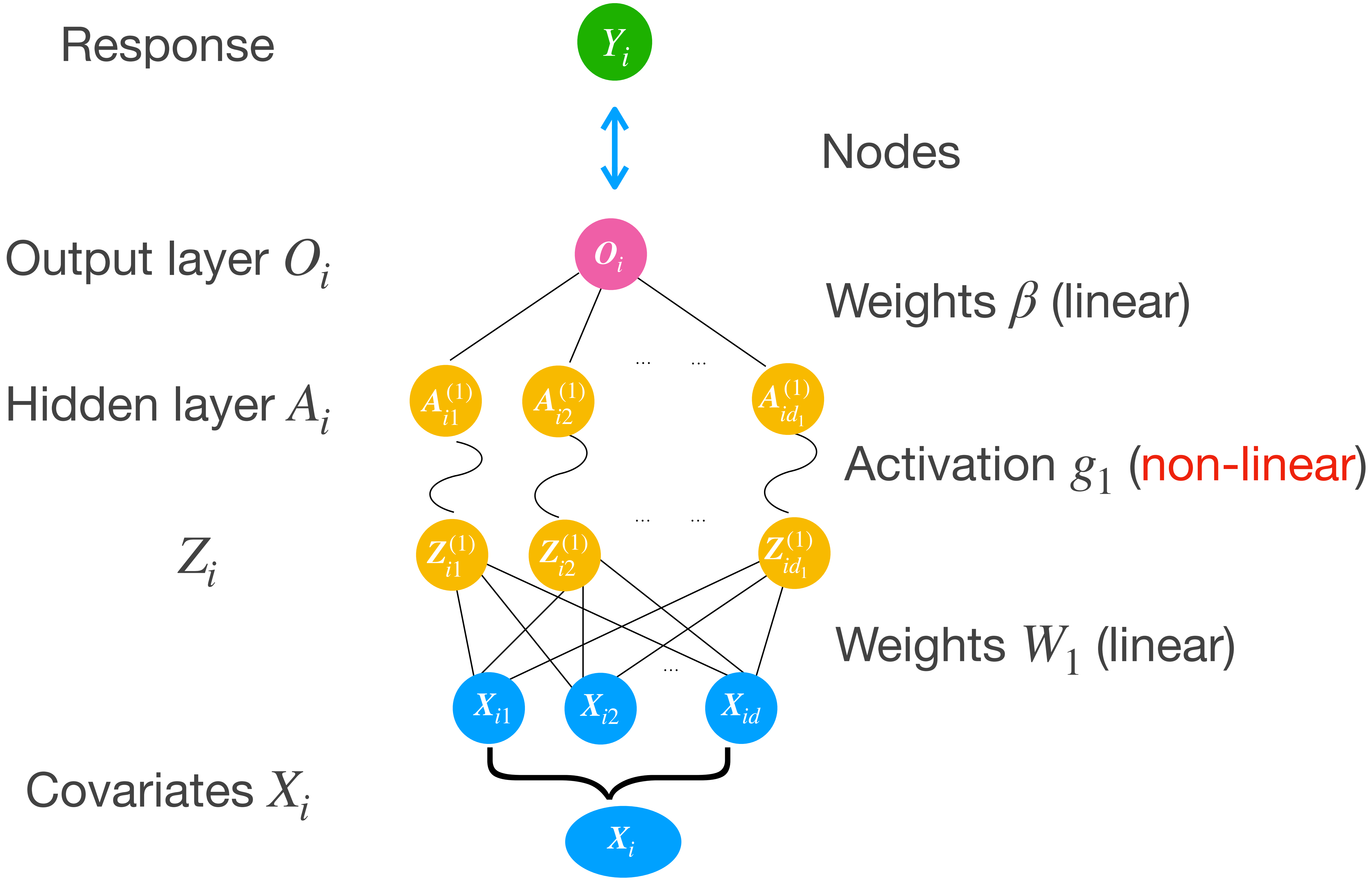
$$Y_i = m(X_i) + \epsilon_i$$

Single layer perceptron model for m:

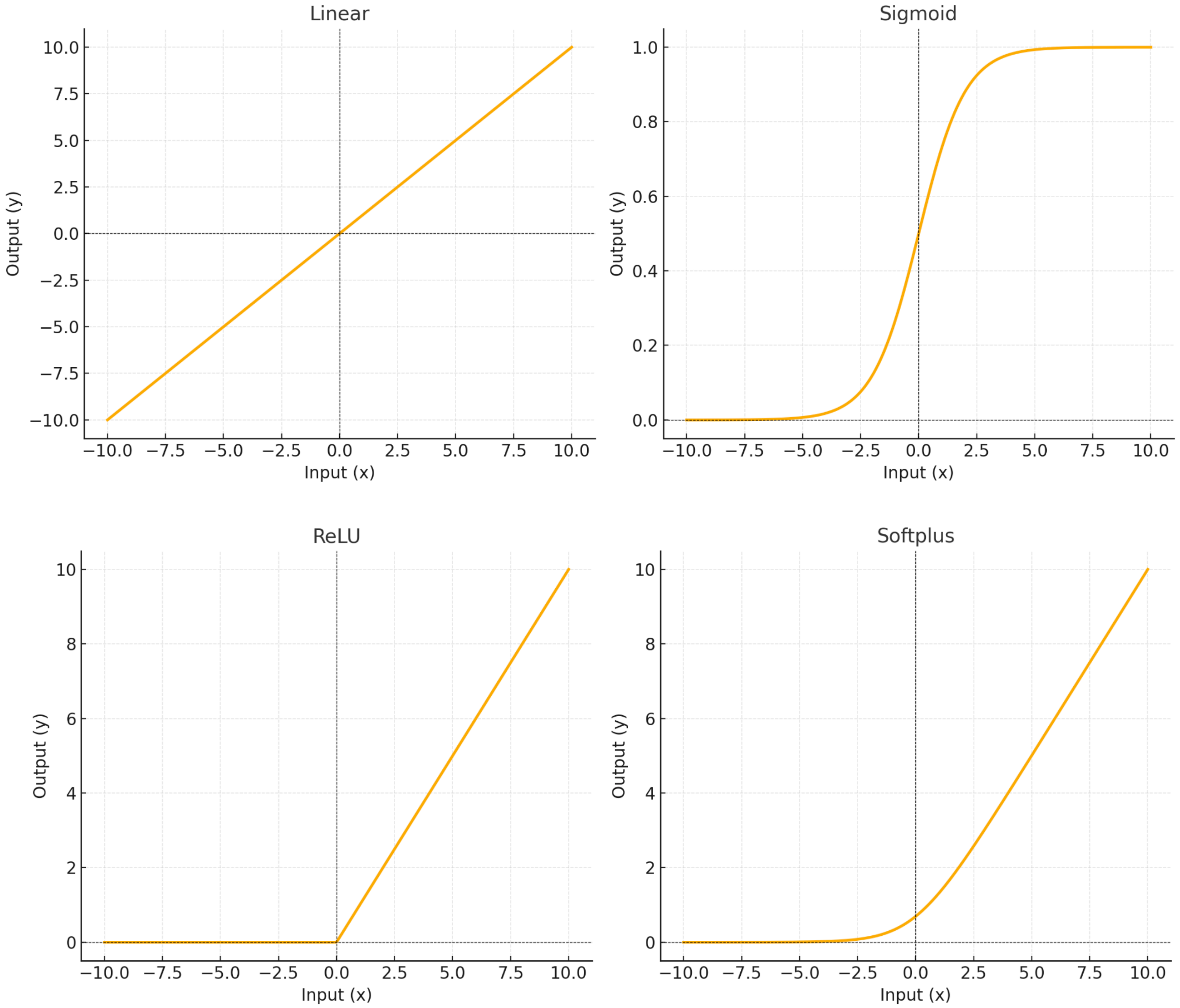
$$m(X_i) = \beta' g_1(W_1 * X_i)$$

W_1 and β are the weights (unknown)

g_1 is a known non-linear function called the link or activation function



Activation functions



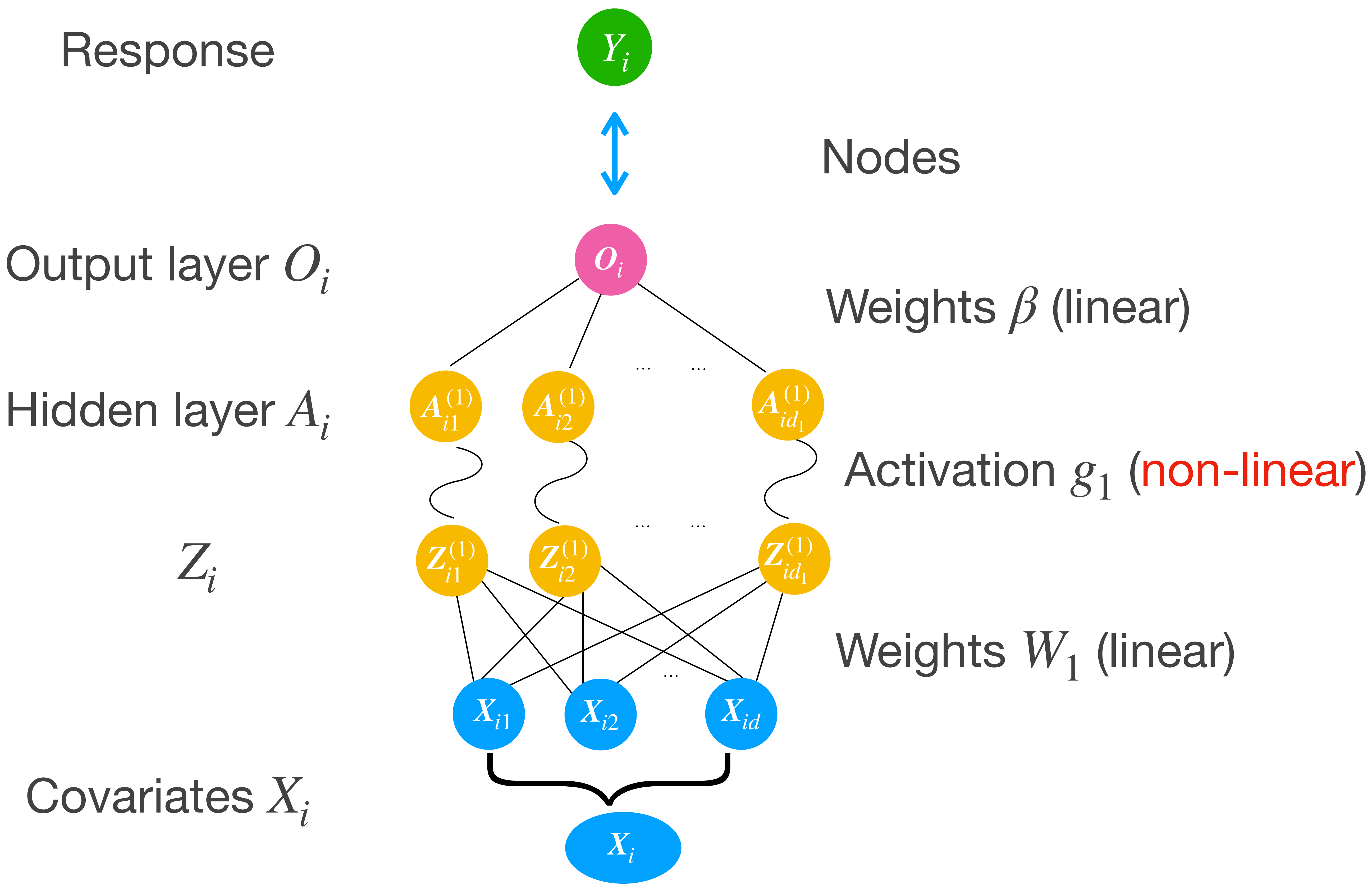
Feed-forward Neural networks

$Y_i = m(X_i) + \epsilon_i$

Single layer perceptron model for m:

$$m(X_i) = \beta' g_1(W_1 * X_i)$$

The output layer $O_i = m(X_i)$ is fitted to the response Y_i to estimate the weights W_1 and β



Feed-forward Neural networks

Single layer perceptron function:

$$m(X) = \beta' g_1(W_1 * X)$$

Universal approximation theorem: Any continuous function can be approximated to any degree of accuracy using a single layer perceptron with any non-polynomial activation function (Stinchcombe et al, 1989 and others)

Feed-forward Neural networks

Single layer perceptron function:

$$m(X) = \beta' g_1(W_1 * X)$$

Universal approximation theorem: Any continuous function can be approximated to any degree of accuracy using a single layer perceptron with any non-polynomial activation function (Stinchcombe et al, 1989 and others)

May need a very wide hidden layer with many nodes for good approximation

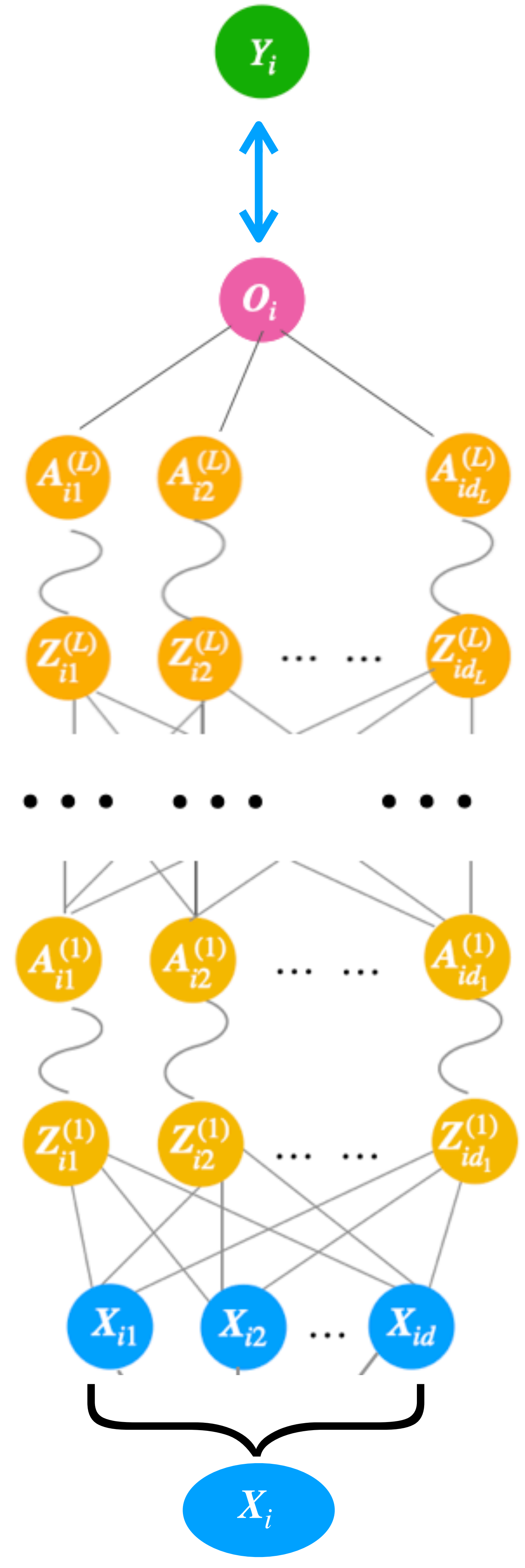
Feed-forward Neural networks

Multi-layer perceptron

$$Y_i = m(X_i) + \epsilon_i$$

Multi-layer perceptron (MLP):

$$m(X_i) = \beta^\top g_L(W_L * g_{L-1}(W_{L-1} * \ldots g_1(W_1 * X_i)\ldots)$$



Feed-forward Neural networks

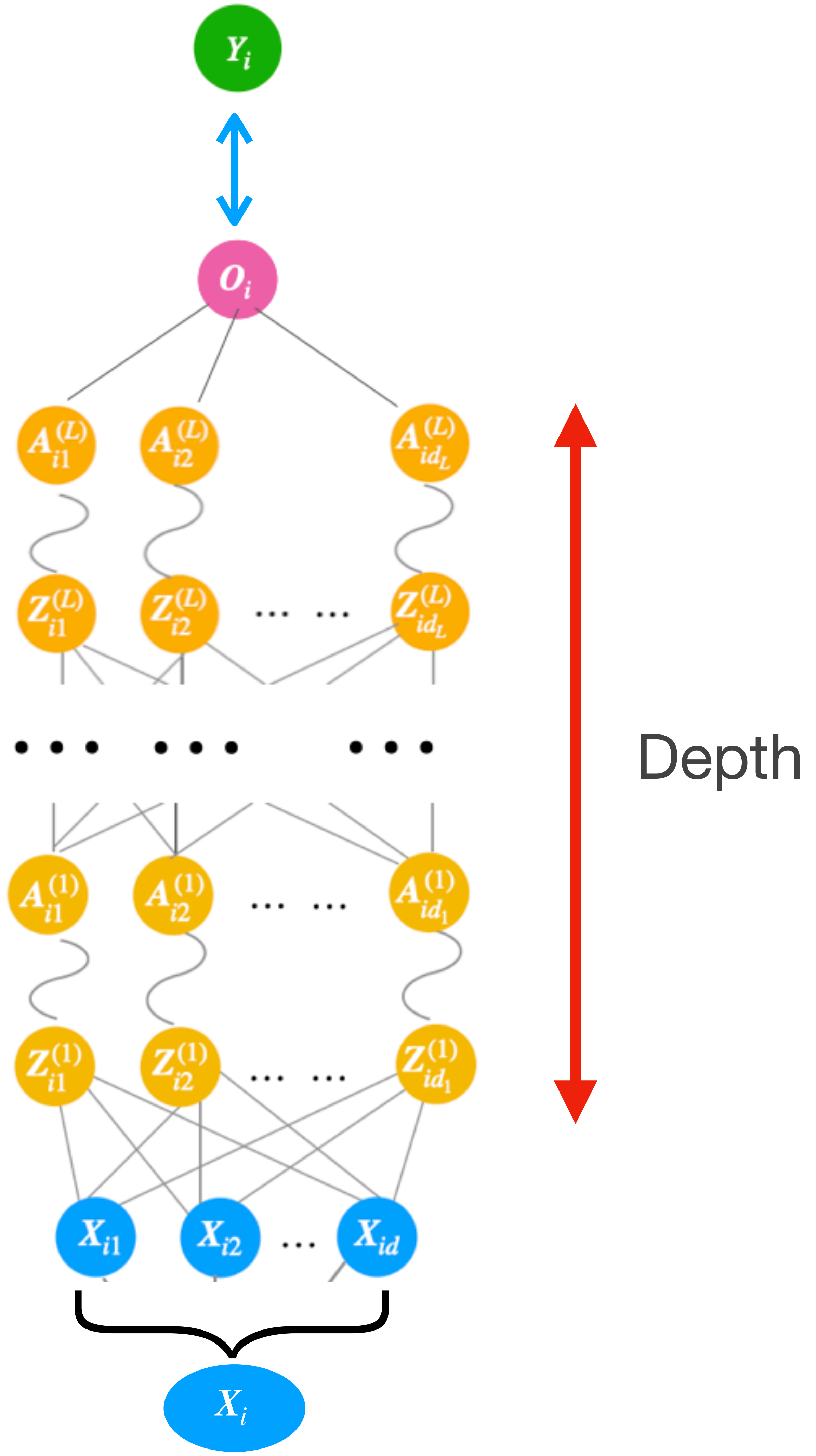
Depth

$$Y_i = m(X_i) + \epsilon_i$$

Multi-layer perceptron (MLP):

$$m(X_i) = \beta^\top g_L(W_L * g_{L-1}(W_{L-1} * \ldots g_1(W_1 * X_i)\ldots)$$

L hidden layers (network depth)



Feed-forward Neural networks

$Y_i = m(X_i) + \epsilon_i$

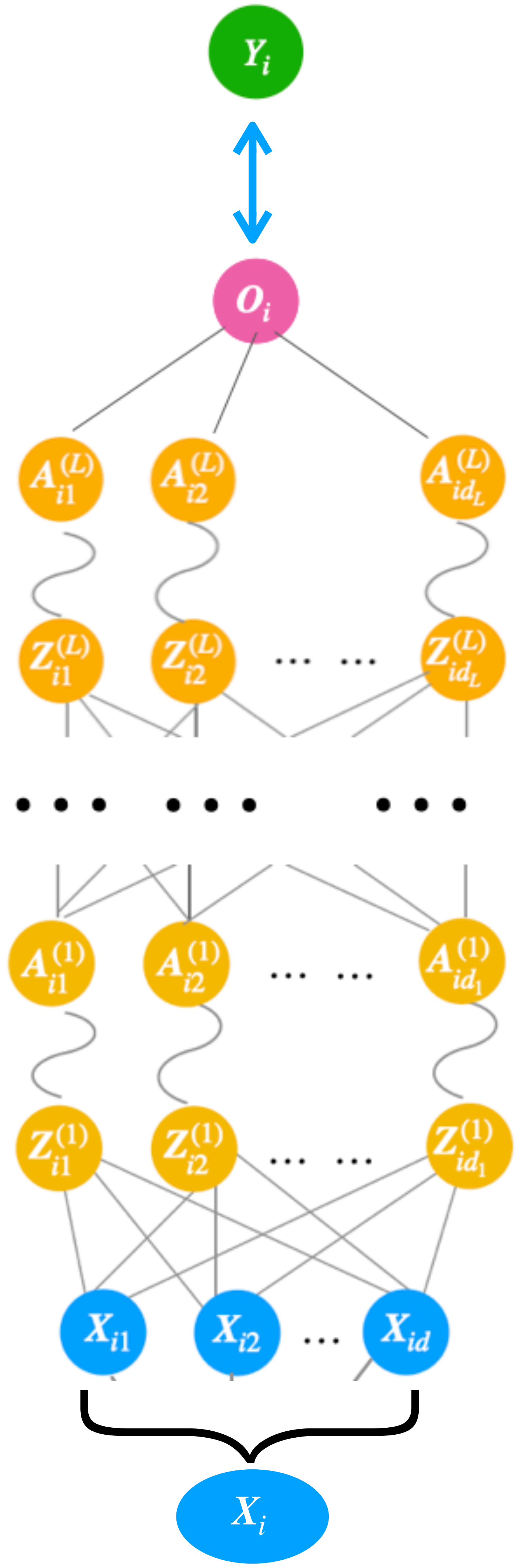
Multi-layer perceptron (MLP):

$m(X_i) = \beta^\top g_L(W_L * g_{L-1}(W_{L-1} * \ldots g_1(W_1 * X_i)\ldots)$

L hidden layers (network depth)

Weights W_l's and β are unknown

Activations g_l's are known



Feed-forward Neural networks

$Y_i = m(X_i) + \epsilon_i$

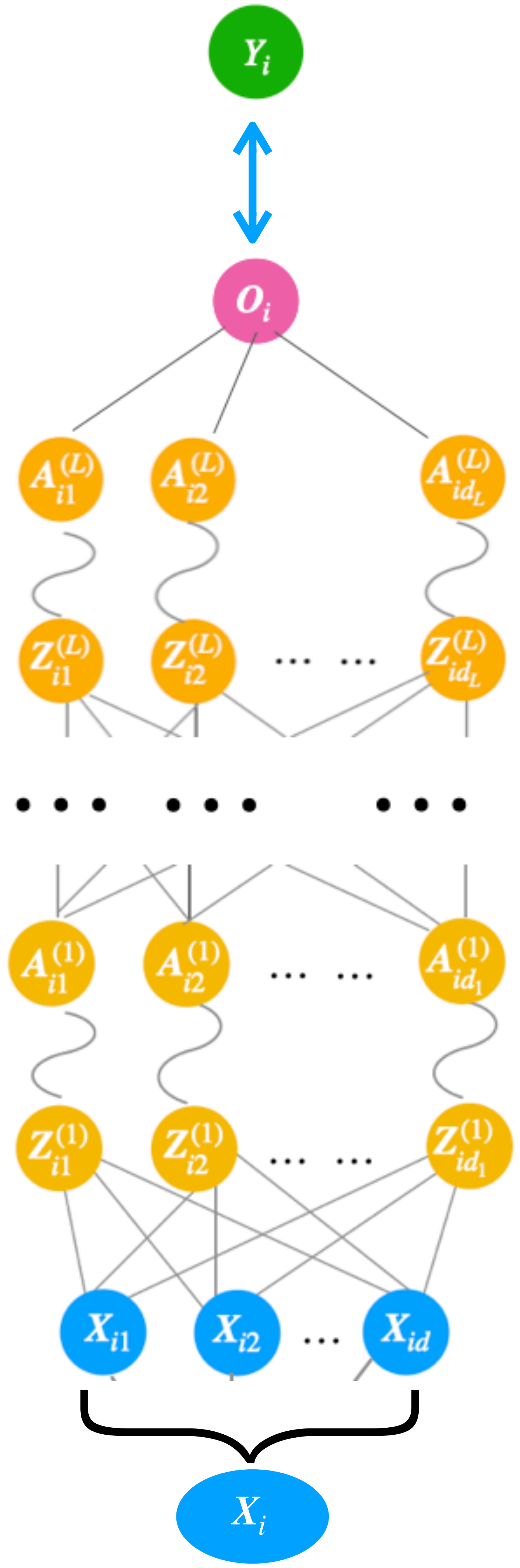
Multi-layer perceptron (MLP):

$m(X_i) = \beta^\top g_L(W_L * g_{L-1}(W_{L-1} * \ldots g_1(W_1 * X_i)\ldots)$

L hidden layers (network depth)

Weights are unknown, activations are known

The output layer $O_i = m(X_i)$ is fitted to the response to estimate the weights $W_1, W_2, \ldots, W_L,$ and β



Estimation in Neural networks

Gradient descent

$\Psi = (W_1, \ldots, W_L, \beta)$ is the collection of all the weight parameters

The output layer $m(X_i) = O_i = O(X_i, \Psi)$

Loss function used is $\ell(\Psi) = \sum_{i=1}^{n} (Y_i - m(X_i))^2 = \sum_{i=1}^{n} (Y_i - O_i)^2$

Parameters updated using gradient descent, e.g., $(\beta^{t+1} = \beta^t - \gamma \frac{\partial \mathscr{L}}{\partial \beta})$

γ is the learning rate, controls how quickly the model learns

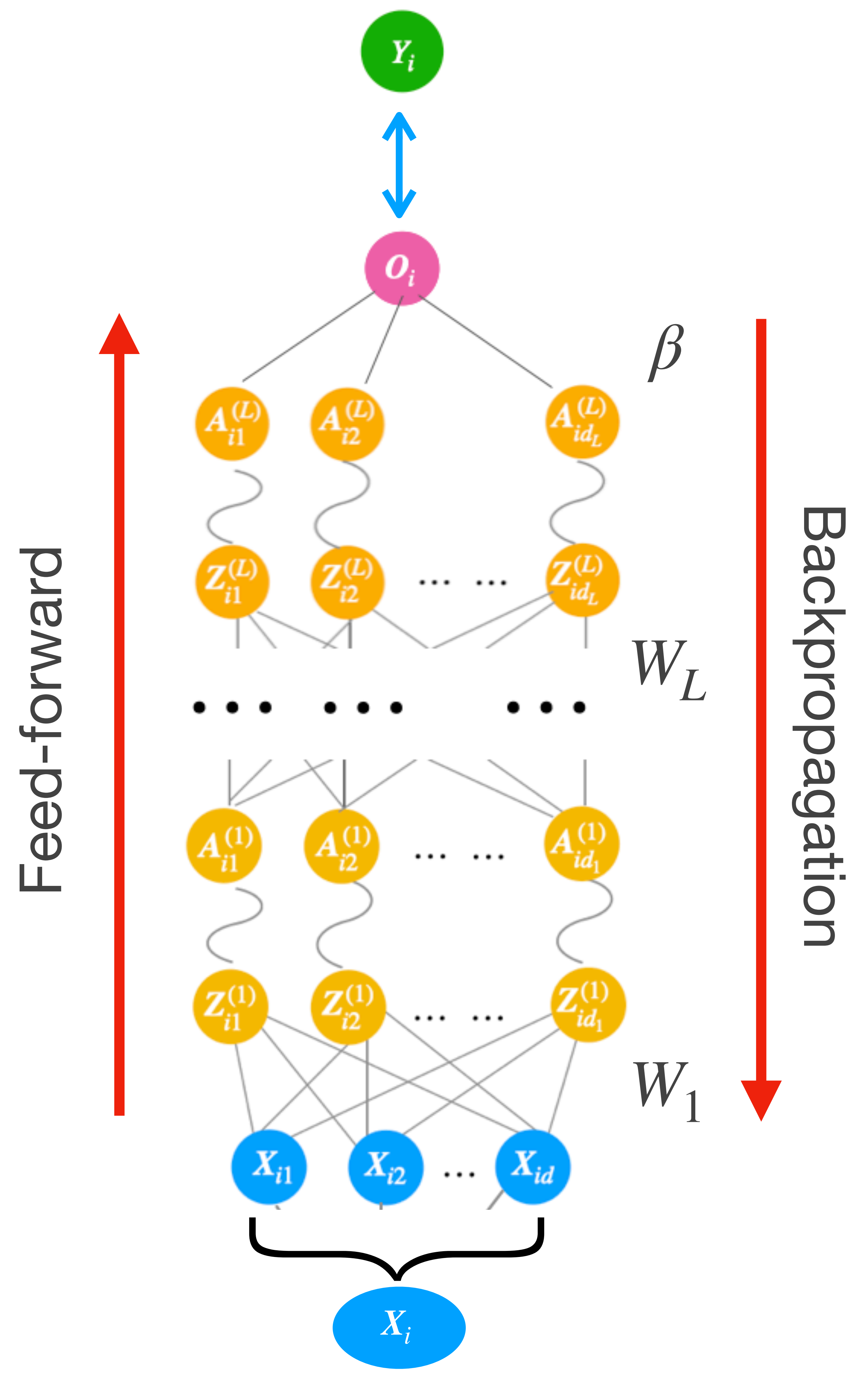
Estimation in Neural networks

Backpropagation and feed-forward

Parameters of last layers updated first which are then used to update parameters of previous layers

Updated parameter values Ψ then fed-forward into the network to get update O_i and evaluate the loss function $\ell(\Psi)$

Process is repeated iteratively until stopping criterion is reached (loss flattens out)



Estimation in Neural networks

Minibatching and stochastic gradient descent

Loss function $\ell(\Psi) = \sum_{i=1}^{n} (Y_i - m(X_i))^2 = \sum_{i=1}^{n} (Y_i - O_i)^2$

Mini-batching loss:

$$\ell_B(\Psi) = \sum_{i \in B} (Y_i - O_i)^2$$

B is a mini-batch (subsample), cycle over all such disjoint mini-batches

Stochastic gradient descent (SGD) = mini-batch size of 1

Minibatching or SGD leads to considerable speedup in estimation

Success of Neural networks

Theory:

Consistency of 1-layer neural networks for non-linear regression (Shen et al. 2023)

Deep neural networks (many layers) with ReLU activation outperforms basis functions and wavelets (Schmidt-Hieber, 2020)

Highly active area of research: Farell et al. 2021, Fan et al. 2023 and others

Most work considers regression for data with iid errors and neural network architectures that do not make adjustments for dependence

What is the impact of ignoring data correlation on performance of neural nets ?

Challenges of neural networks for dependent data

Non-linear regression for dependent data:

$Y_i = m(X_i) + \epsilon_i$, ϵ_i are dependent are errors

Loss function $\ell(\Psi) = \sum_{i=1}^{n} (Y_i - m(X_i))^2 = \sum_{i=1}^{n} (Y_i - O_i)^2 = (Y - O)'(Y - O)$

Loss function is essentially the OLS loss

Does not account for dependence in the Y_i's

Neural networks for geospatial analysis

Common strategies:

1. Residual kriging: Estimates a non-linear regression function $E(Y) = m(X)$ using Neural networks.

Kriging on the residuals $Y_i - \widehat{m}(X_i)$ for spatially-informed predictions.

Demyanov et al. 1998, Seo et al. 2015, Tarasov et al. 2018 and others

Spatial dependence is completely ignored during estimation

Neural networks for geospatial analysis

Common strategies:

2. Added spatial features:

Creates a set $B(s)$ of spatial features / covariates (spatial co-ordinates, pairwise distances, basis functions, etc.).

Estimates a non-linear regression function $E(Y) = g(X, B(s))$ using neural network.

Gray et al., 2022; Chen et al., 2024; Wang et al., 2019

Prediction only! Cannot estimate the spatial effect $m(X)$

Does not directly model spatial correlation. Curse of dimensionality from many added features.

Neural networks for geospatial analysis

3. Model based approach: $Y_i = m(X_i) + w_i + \epsilon_i^*, w \sim GP(0,C), \epsilon_i^* \sim_{iid} N(0,\tau^2)$

Model the non-linear m using a multi-layer perceptron: $m(X_i) = O_i = O(\Psi, X_i)$

Retains all advantages of the traditional spatial mixed models

- Interpretability and parsimony of GP
- Estimation of mean and spatial prediction (kriging)

Neural networks with GLS loss

3. Model based approach: $Y_i = m(X_i) + \epsilon_i, \epsilon \sim N(0,\Sigma), \Sigma = C(\theta) + \tau^2 I\,.$

Marginal model: $Y \sim N(m(X), \Sigma) = N(O(\Psi), \Sigma)$

For a given Σ, MLE of Ψ can be obtained by minimizing GLS loss:

$$\widehat{\Psi} = \arg\min_{\Psi} \ell_G(\Psi) \text{ where } \ell_G(\Psi) = (Y - O(\Psi))'\,\Sigma^{-1}(Y - O(\Psi))$$

In practice, Ψ can be estimated using gradient descent based on $\ell_G(\Psi)$

NN-GLS: Neural network parameter estimation using GLS loss

Neural networks with GLS loss

Challenges with neural network with the GLS loss

$$\ell_G(\Psi) = (Y - O(\Psi))' \, \Sigma^{-1}(Y - O(\Psi))$$

Unlike the OLS loss $\sum_i (Y_i - O_i)^2$, the GLS loss is not additive over datapoints

and not amenable to minibatching

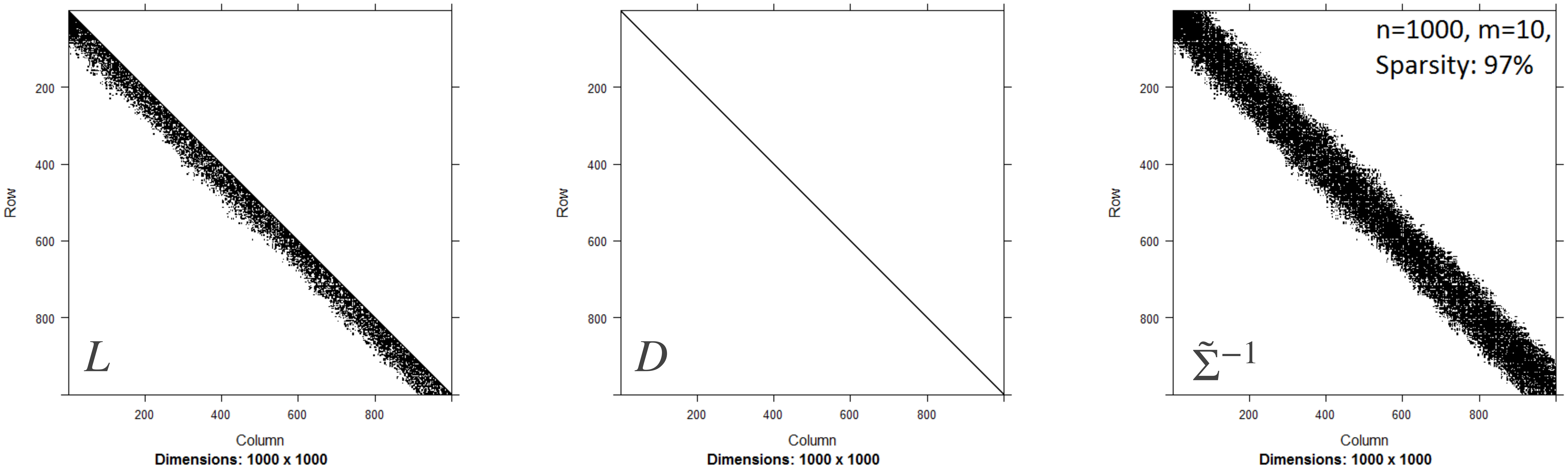
Evaluating Σ^{-1} is expensive ($O(n^3)$)

Σ contains unknown spatial parameters θ

Nearest Neighbor Gaussian Processes

The NNGP precision matrix admits the factorization $\tilde{\Sigma}^{-1} = L'DL$

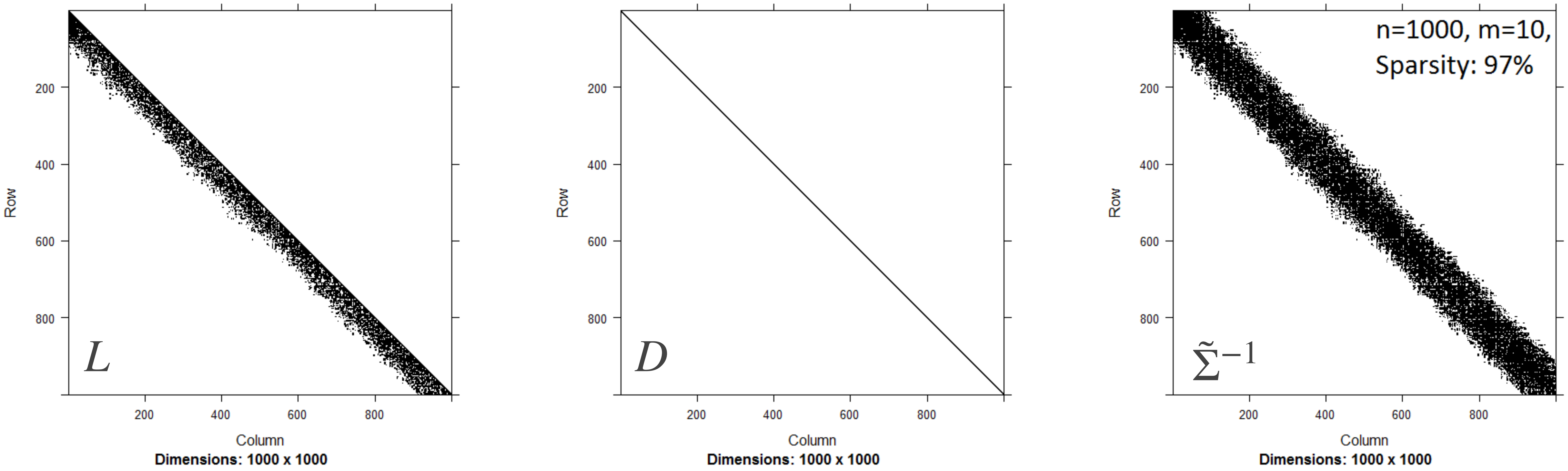
- D is diagonal with entries d_i
- L is lower triangular and row sparse
  - Sparsity determined by the nearest-neighbor DAG



Nearest Neighbor Gaussian Processes

Use GLS loss with covariance $\tilde{\Sigma}$ from Nearest Neighbor Gaussian Process (NNGP)

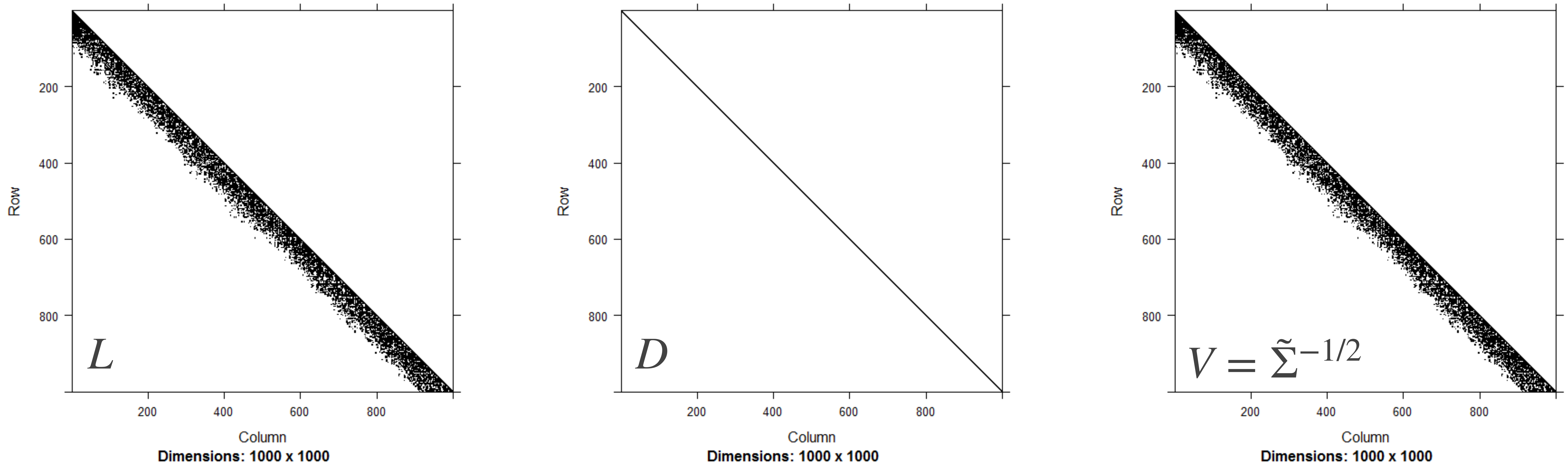
$\tilde{\Sigma}^{-1} = L'DL$, D is diagonal with entries d_i, L is lower triangular and row sparse



Nearest Neighbor Gaussian Processes

The Cholesky factor $V = \tilde{\Sigma}^{-1/2} = D^{1/2}L$ can be computed in $O(n)$ time

V has the same sparsity as L



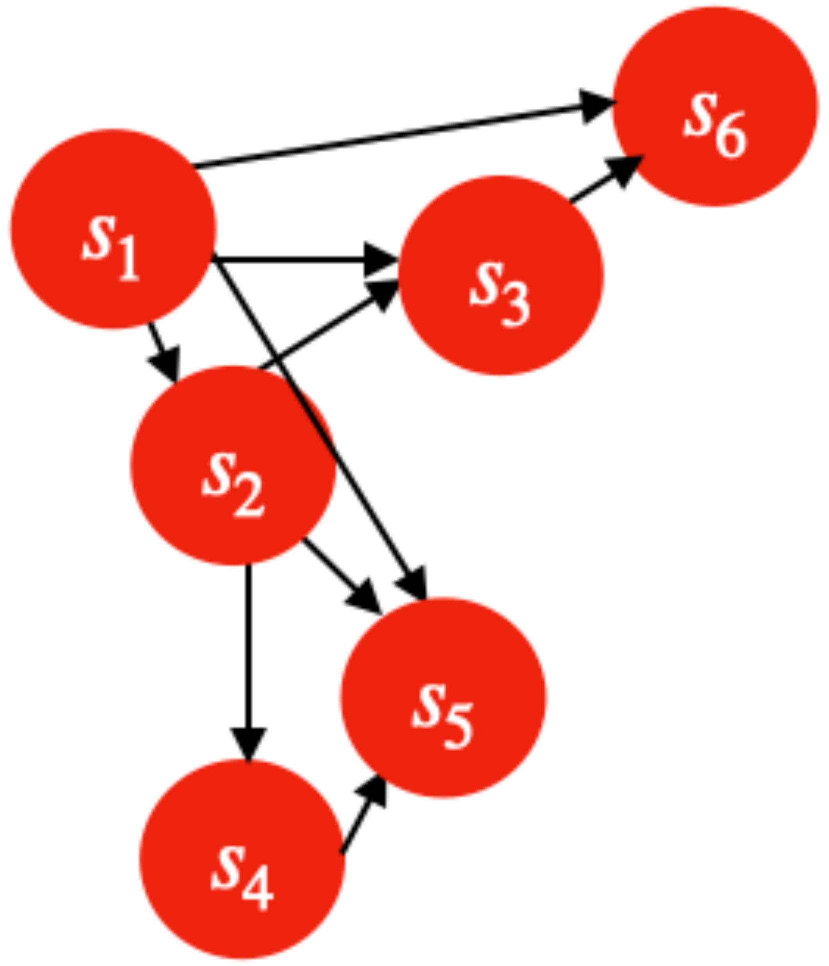
Nearest Neighbor Gaussian Processes

The Cholesky factor V has same sparsity as L

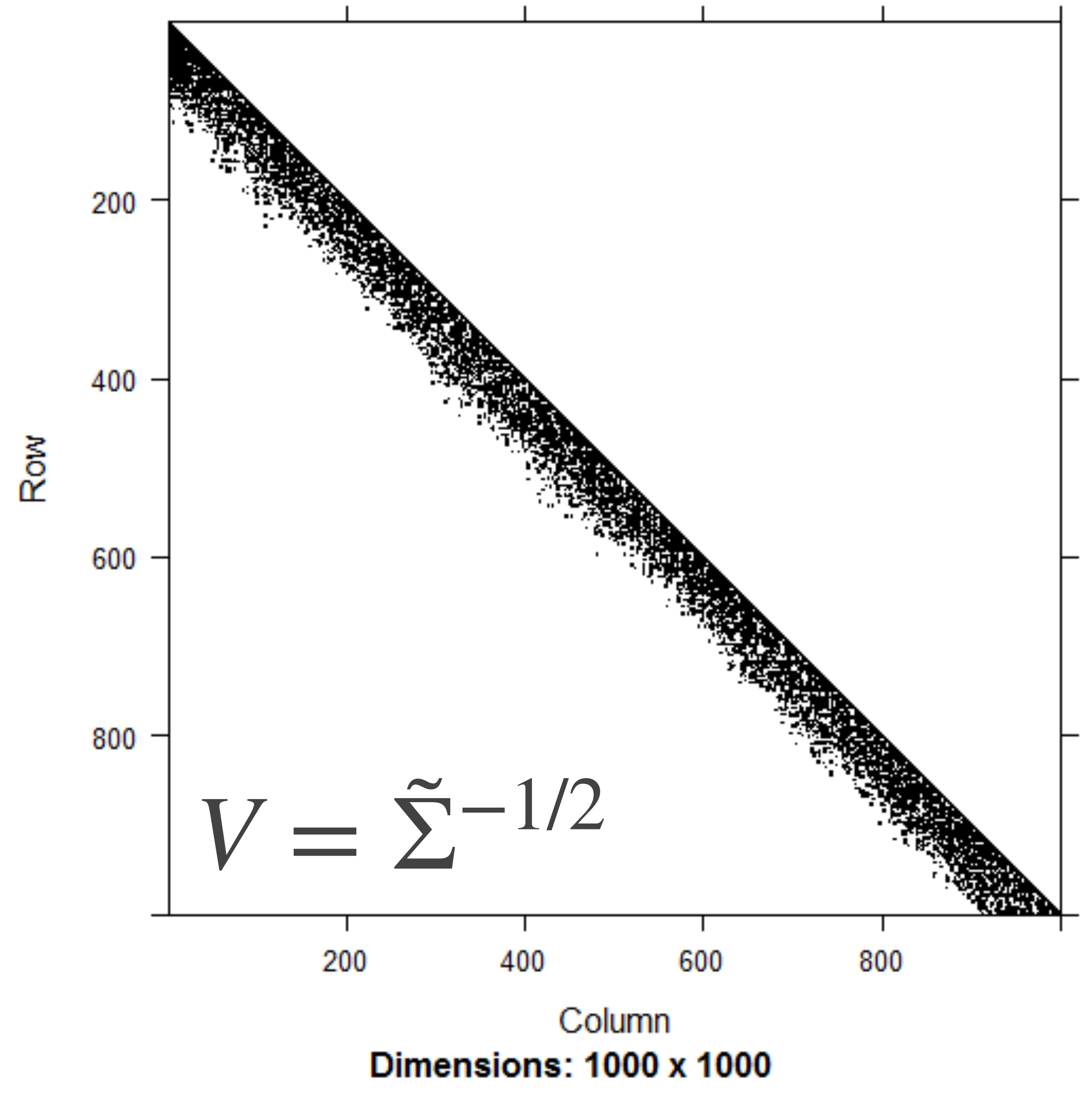
Sparsity determined by the m-nearest neighbor directed acyclic graph (DAG)

$V_{ij} = 0$ unless $i \to j$ or $i = j$

Non-zero V_{ij}'s are nearest-neighbor kriging weights and depend on θ



2-NN DAG



GLS loss using NNGP covariance

NN-GLS loss with NNGP covariance matrix: $(Y - O)' \, \tilde{\Sigma}^{-1}(Y - O)$

GLS loss between Y and O =

OLS loss between decorrelated response $Y* = VY$ and $O* = VO$ with $V = \tilde{\Sigma}^{-1/2}$

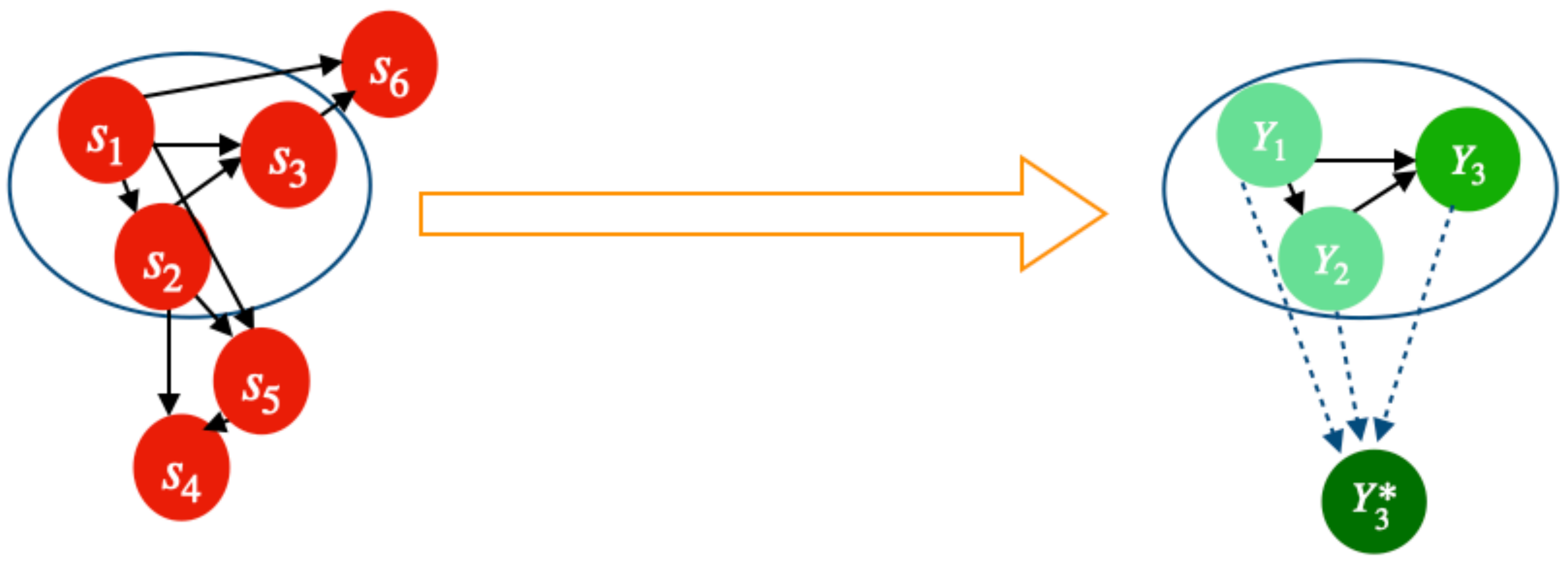
NN-GLS loss: $\sum_i (Y_i^* - O_i^*)^2$ where $Y_i^* = v_i(\theta)^T Y_{N*(i)}$

Non-zero V_{ij}'s

Y_i and its neighbors $Y_{N(i)}$

GLS loss using NNGP covariance

NN-GLS loss: $\sum_i (Y_i^* - O_i^*)^2$, $Y_i^* = v_i(\theta)^T Y_{N*(i)}$ is the decorrelated response



2-NN DAG

Decorrelated response

Decorrelation in NNGP = Multiplication by the sparse Cholesky factor V

= Graph convolution on the nearest neighbor DAG with convolution weights $v_i(\theta)$

Graph neural network

Graph neural networks (GNN) are used when variables have a graphical relationship

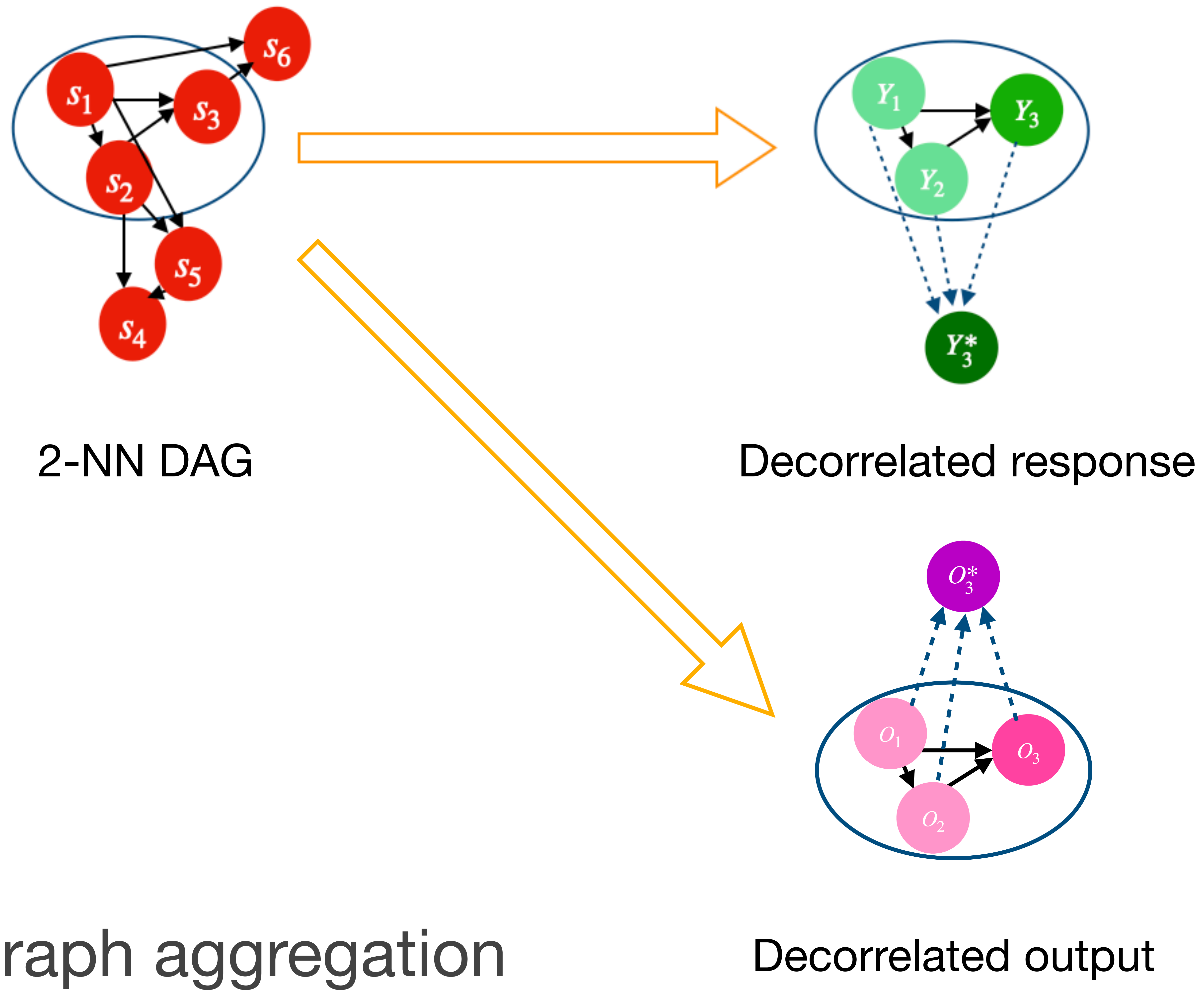
Graph convolution: New nodes are created by aggregating variables over their graph neighborhoods

NN-GLS as a graph neural network (GNN)

NN-GLS loss: $\sum_i (Y_i^* - O_i^*)^2$

$$Y_i^* = v_i(\theta)^T Y_{N^*(i)}$$

$$O_i^* = v_i(\theta)^T O_{N^*(i)}$$



Both Y_i^* and O_i^* are created by graph aggregation

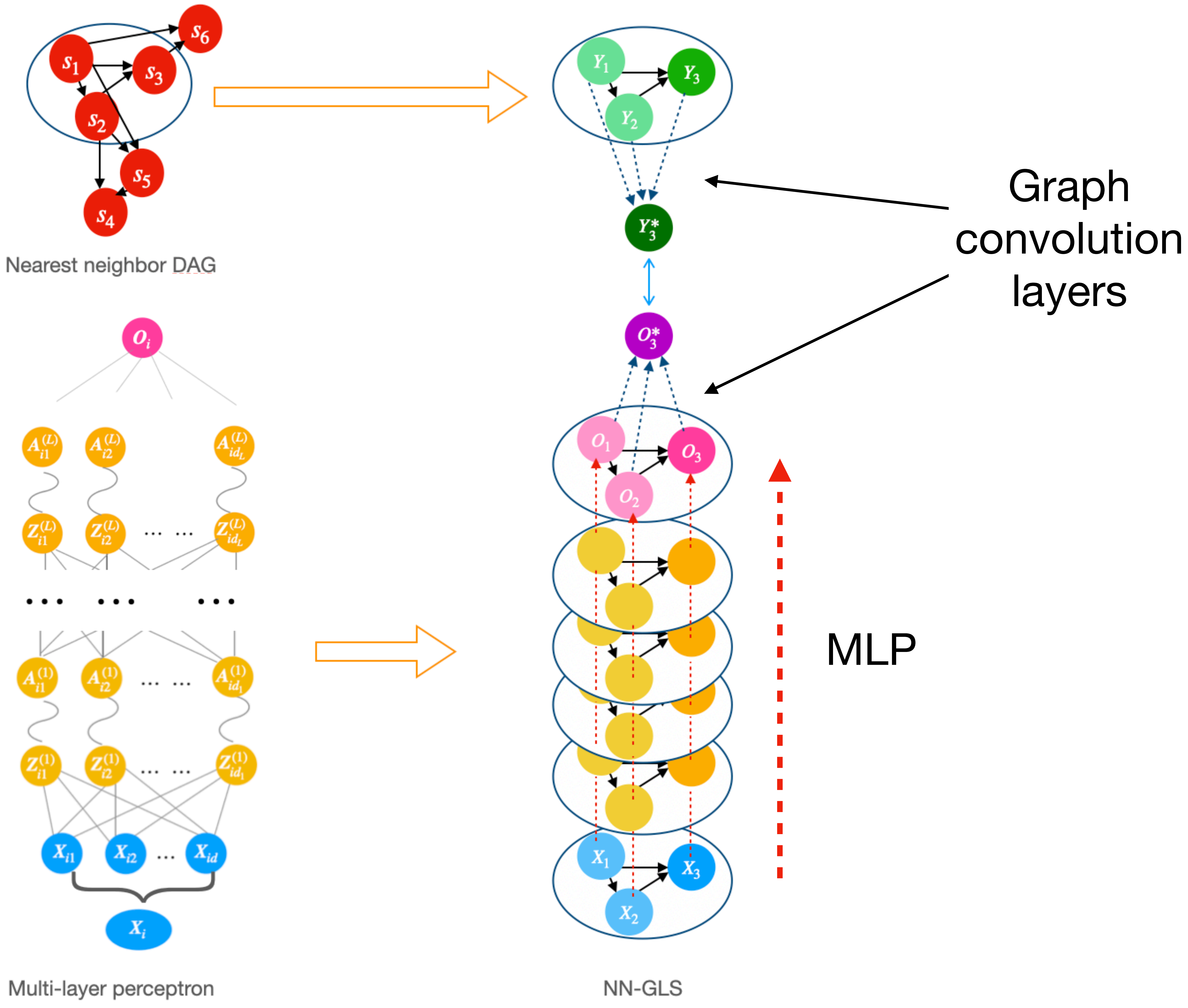
NN-GLS as a graph neural network (GNN)

NN-GLS model with NNGP covariance: $Y \sim N(m(X), \tilde{\Sigma})$

Can be represented as a special type of GNN

Multi-layer perceptron for modeling the mean m

Modeling covariance $\tilde{\Sigma}$ is equivalent to adding two graph aggregation layers based on NN-DAG and kriging weights



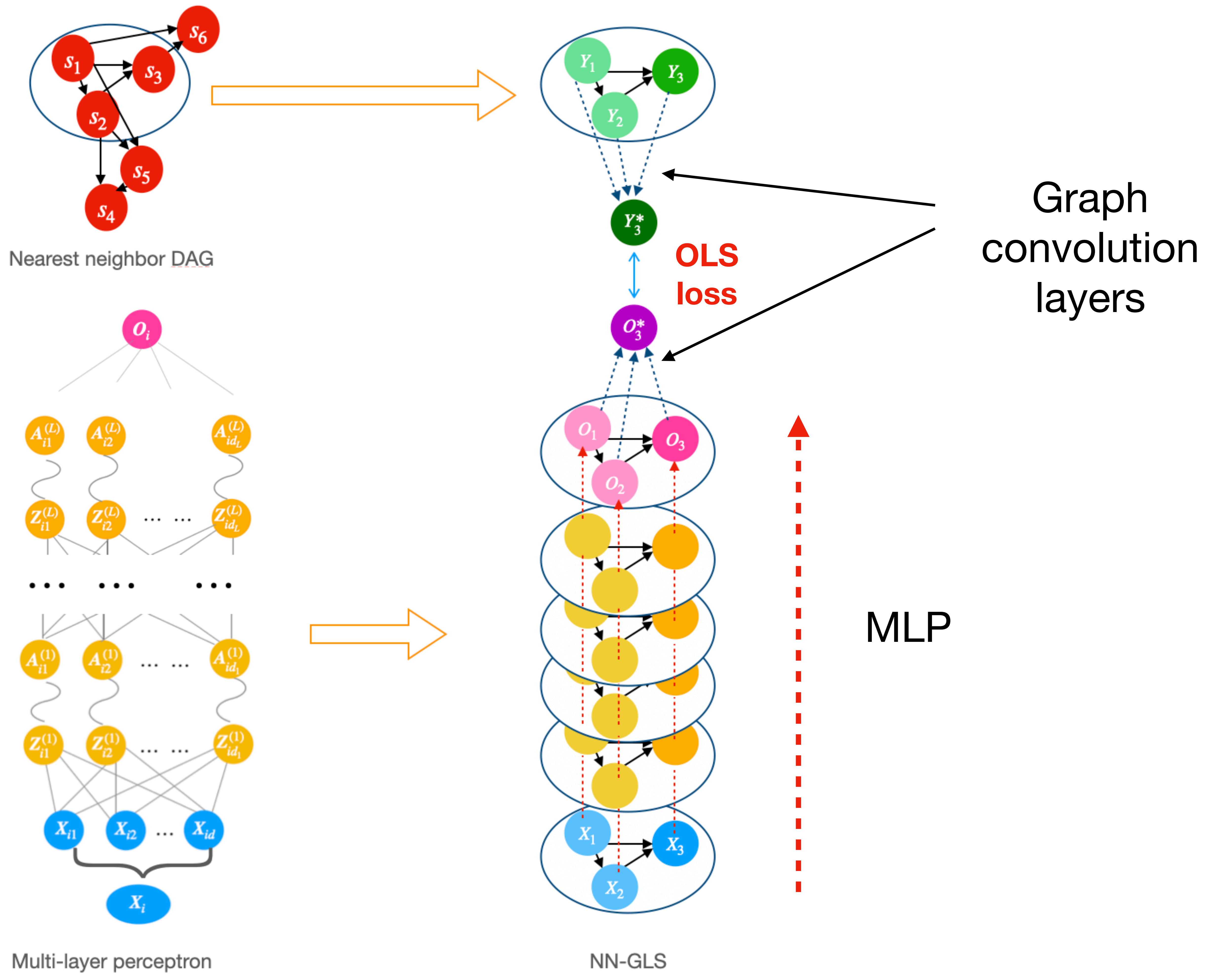
NN-GLS as a graph neural network (GNN)

Mini-batching:

The OLS loss $\sum_{i=1}^{n} (Y_i^* - O_i^*)^2$ can be split into minibatches

MLP parameters (weights) updated using minibatch GLS loss:

$$\sum_{i \in B} (Y_i^* - O_i^*)^2$$

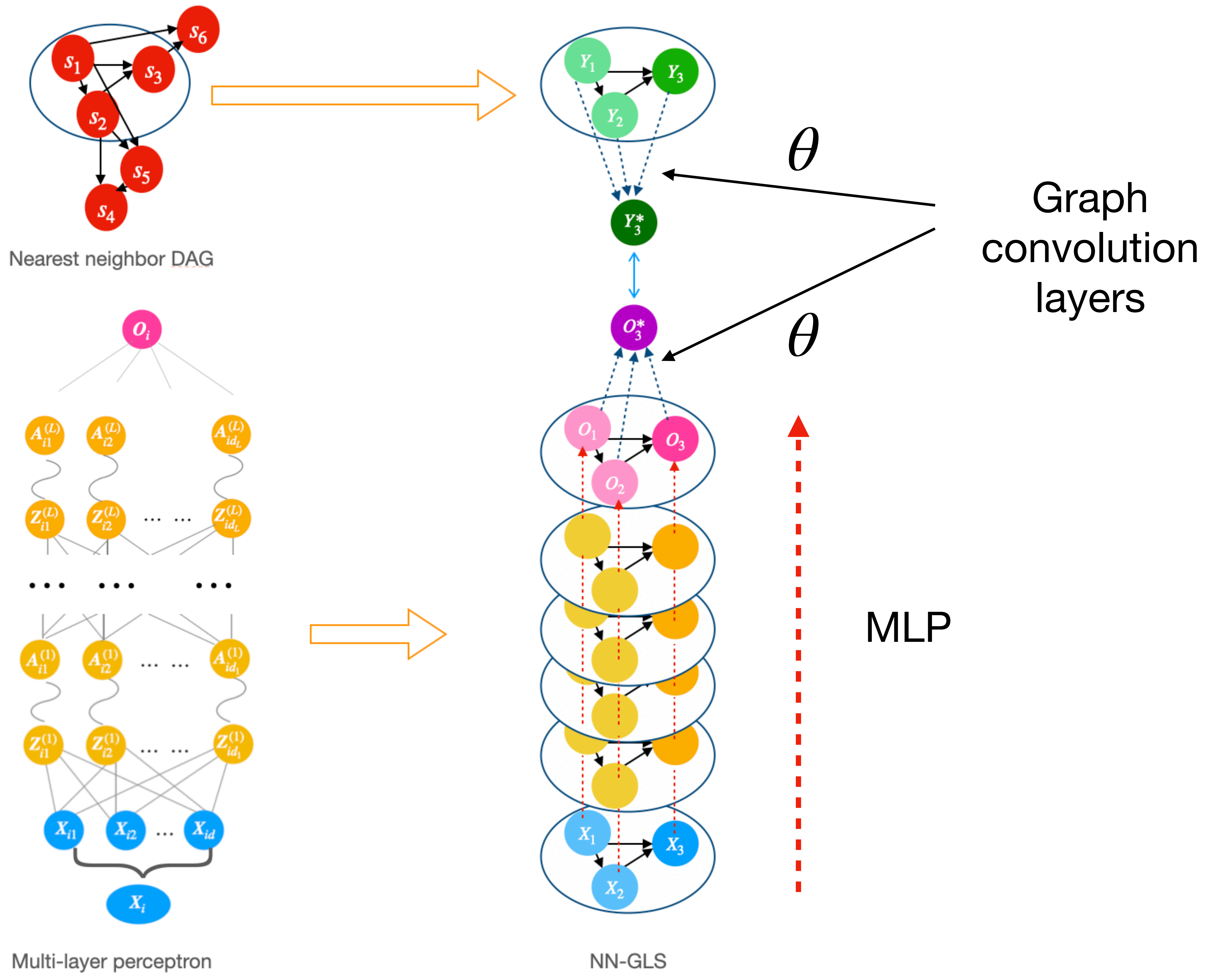


NN-GLS as a graph neural network (GNN)

Spatial parameter estimation:

Spatial covariance parameters θ only appear in the two graph convolution layers as kriging-based graph convolution weights

Negative log-likelihood from the model $Y \sim N(m(X), \tilde{\Sigma})$ for updating θ is GLS loss + $\log(\det(\tilde{\Sigma}))$



NN-GLS as a graph neural network (GNN)

Prediction (kriging):

For NN-GLS using NNGP, predictive distribution at a new location s_0 is given by

$$Y(s_0) \mid Y, \theta, \beta = N\left(\mu(s_0), \sigma^2(s_0)\right)$$

$N_0 = m$ nearest neighbors of s_0 among $s_1, \ldots, s_n$

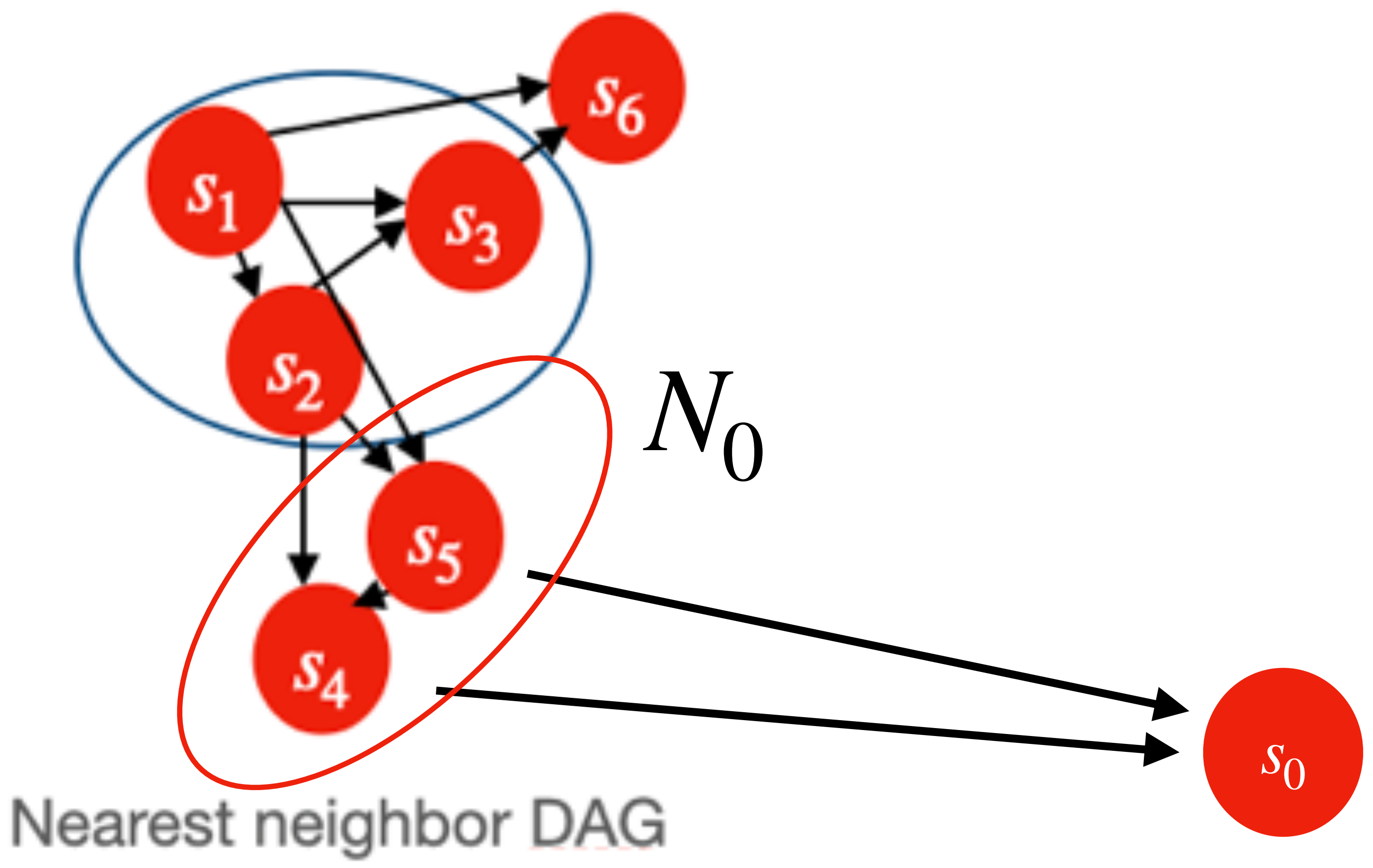
Kriging mean: $\mu(s_0) = \widehat{m}(X(s_0)) + C(s_0, N_0)\Sigma^{-1}_{N_0,N_0}(Y_{N_0} - \widehat{m}(X_{N_0}))$

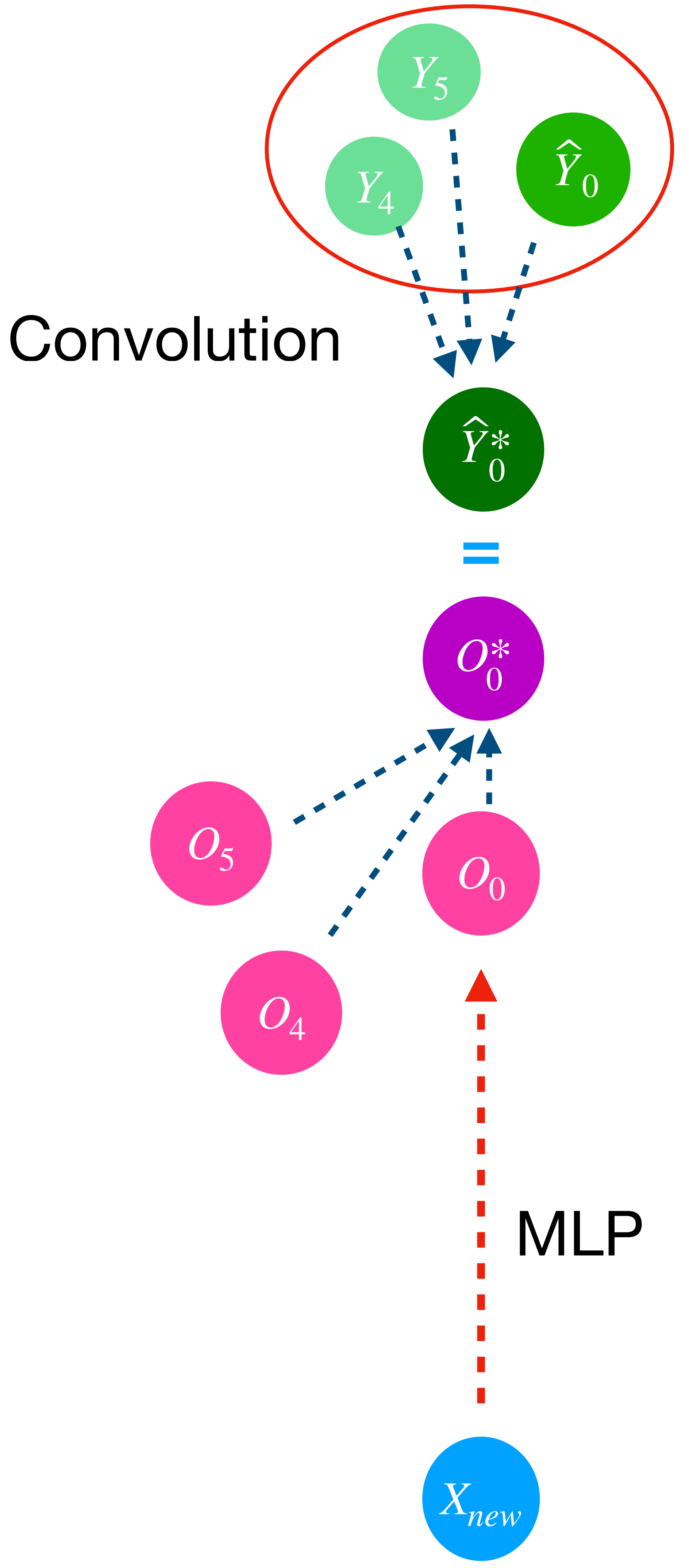
Kriging variance: $\sigma^2(s_0) = C(s_0, s_0) + \tau^2 - C(s_0, N_0)\Sigma^{-1}_{N_0,N_0}C(N_0, s_0)$

$\widehat{m}$ is the MLP estimate of m

NN-GLS as a graph neural network (GNN)

Prediction (kriging):



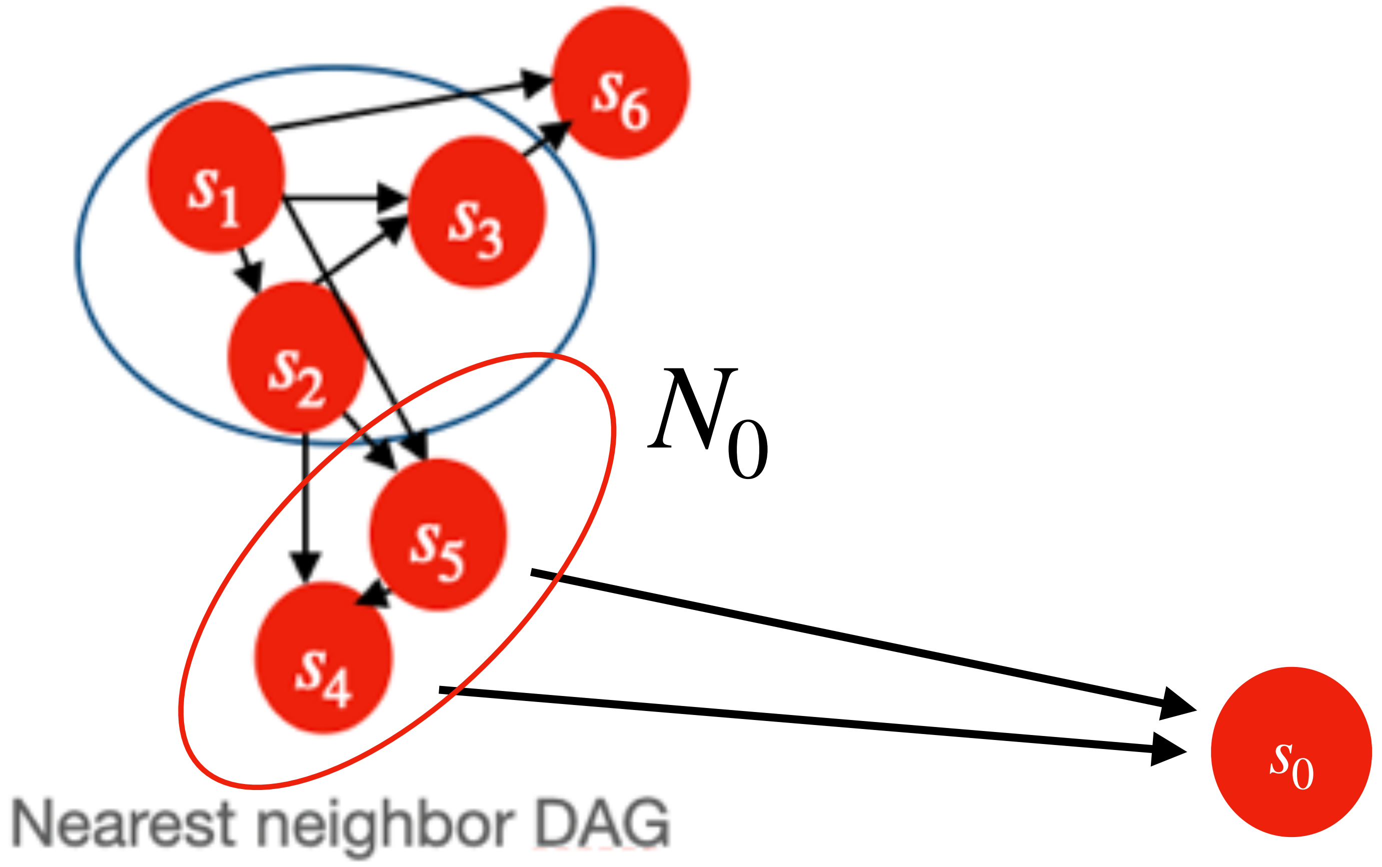


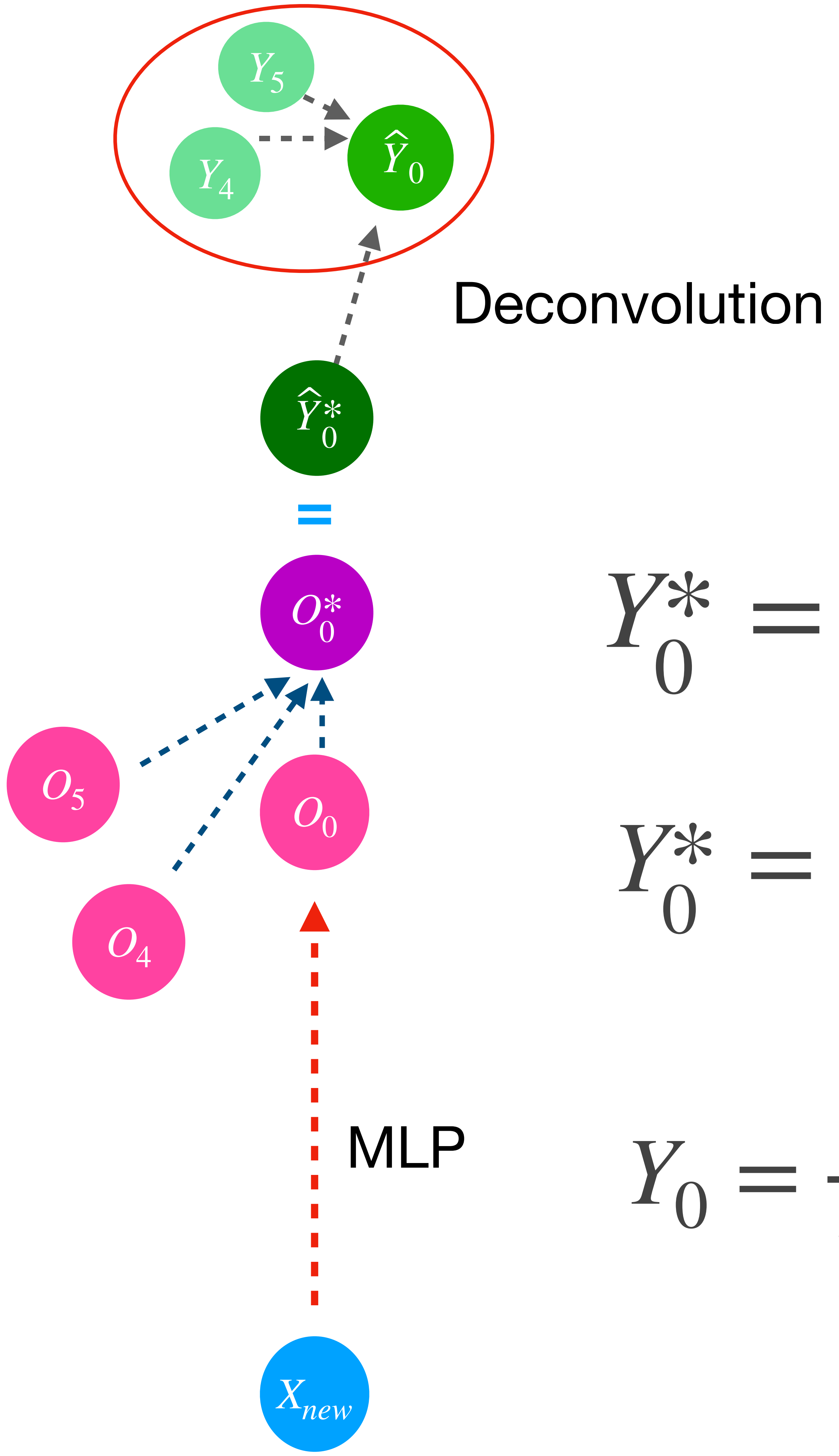
$$Y_0^* = v_0(\theta)^T Y_{N^*(0)}$$

$$Y_0^* = v_{00} Y_0 + \sum_{j \in N_0} v_{0j} Y_j$$

NN-GLS as a graph neural network (GNN)

Prediction (kriging):





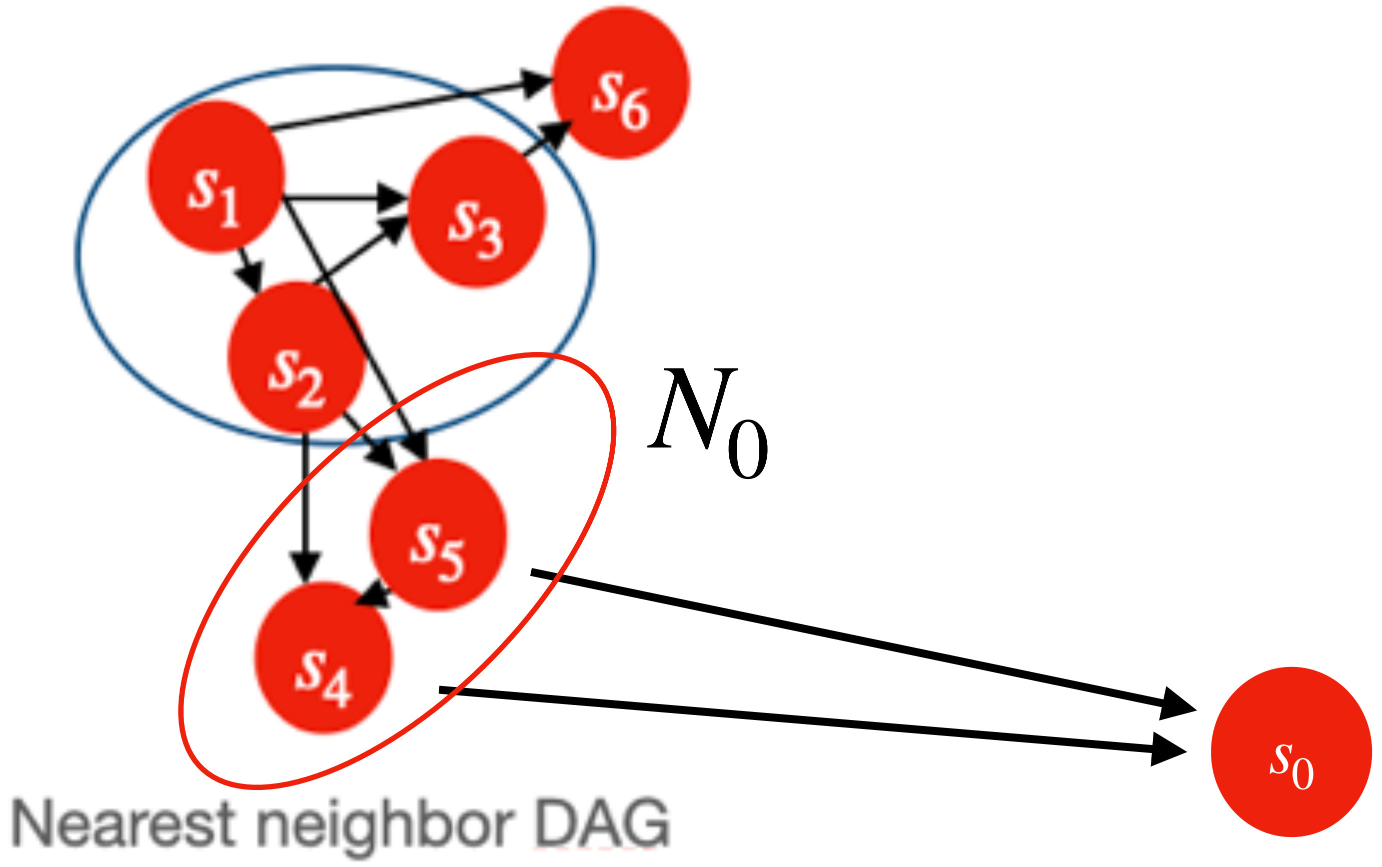
$$Y_0^* = v_0(\theta)^T Y_{N^*(0)}$$

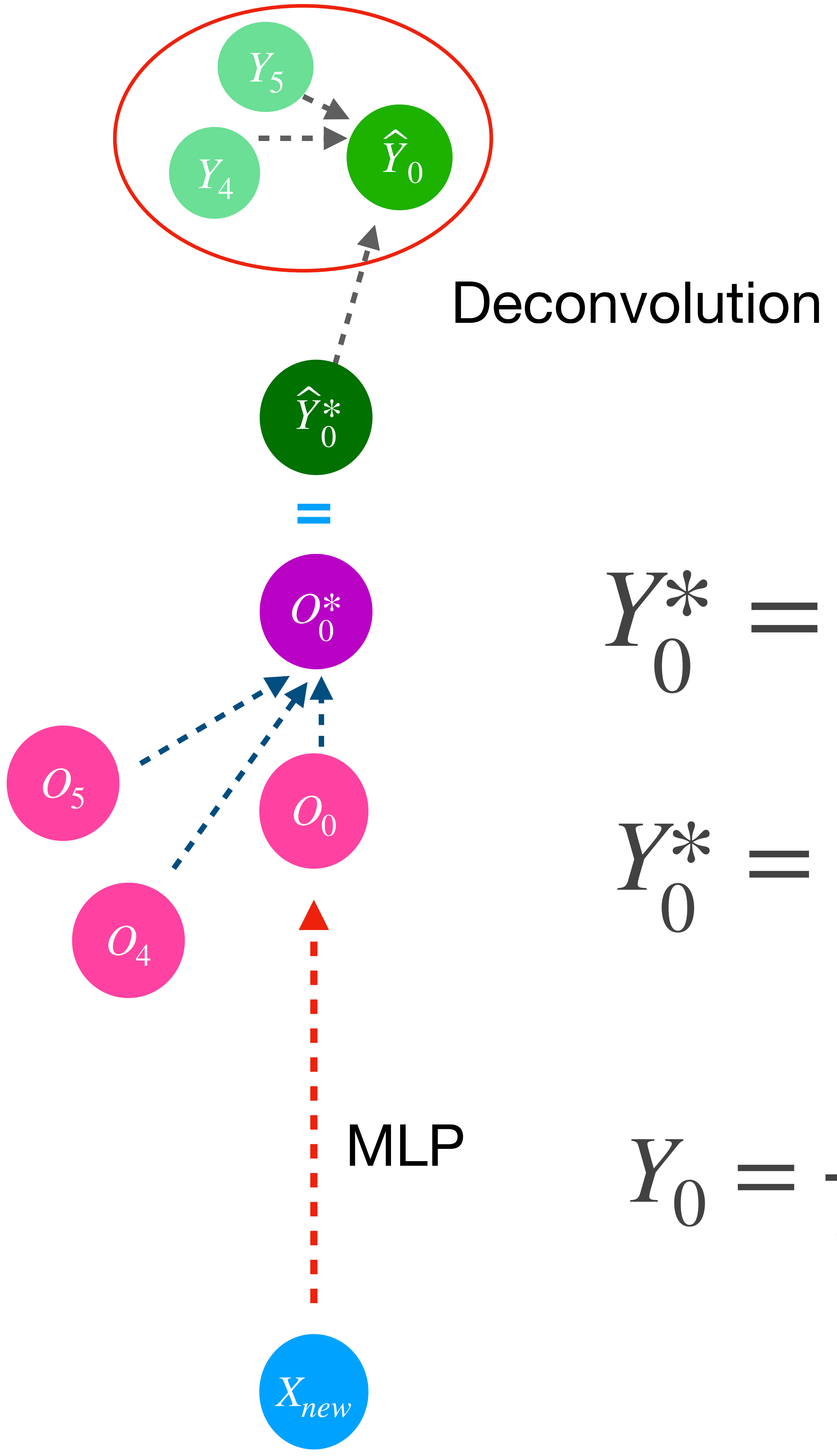
$$Y_0^* = v_{00} Y_0 + \sum_{j \in N_0} v_{0j} Y_j$$

$$Y_0 = \frac{1}{v_{00}} (Y_0^* - \sum_{j \in N_0} v_{0j} Y_j)$$

NN-GLS as a graph neural network (GNN)

Prediction (kriging):





$$Y_0^* = v_0(\theta)^T Y_{N^*(0)}$$

$$Y_0^* = v_{00} Y_0 + \sum_{j \in N_0} v_{0j} Y_j$$

$$Y_0 = \frac{1}{v_{00}} (Y_0^* - \sum_{j \in N_0} v_{0j} Y_j)$$

Prediction via the GNN is exactly equivalent to nearest-neighbor kriging mean for the model $Y \sim NNGP(m(X), \tilde{\Sigma})$

Variable importance

When the covariate X is multivariate, importance of individual covariates in a non-linear regression can be obtained using *partial dependence functions (PDF)*

PDF shows the marginal effect one covariate has on the predicted response as estimated by any machine learning model (Friedman, 2001)

PDF is obtained by integrating the remaining variables

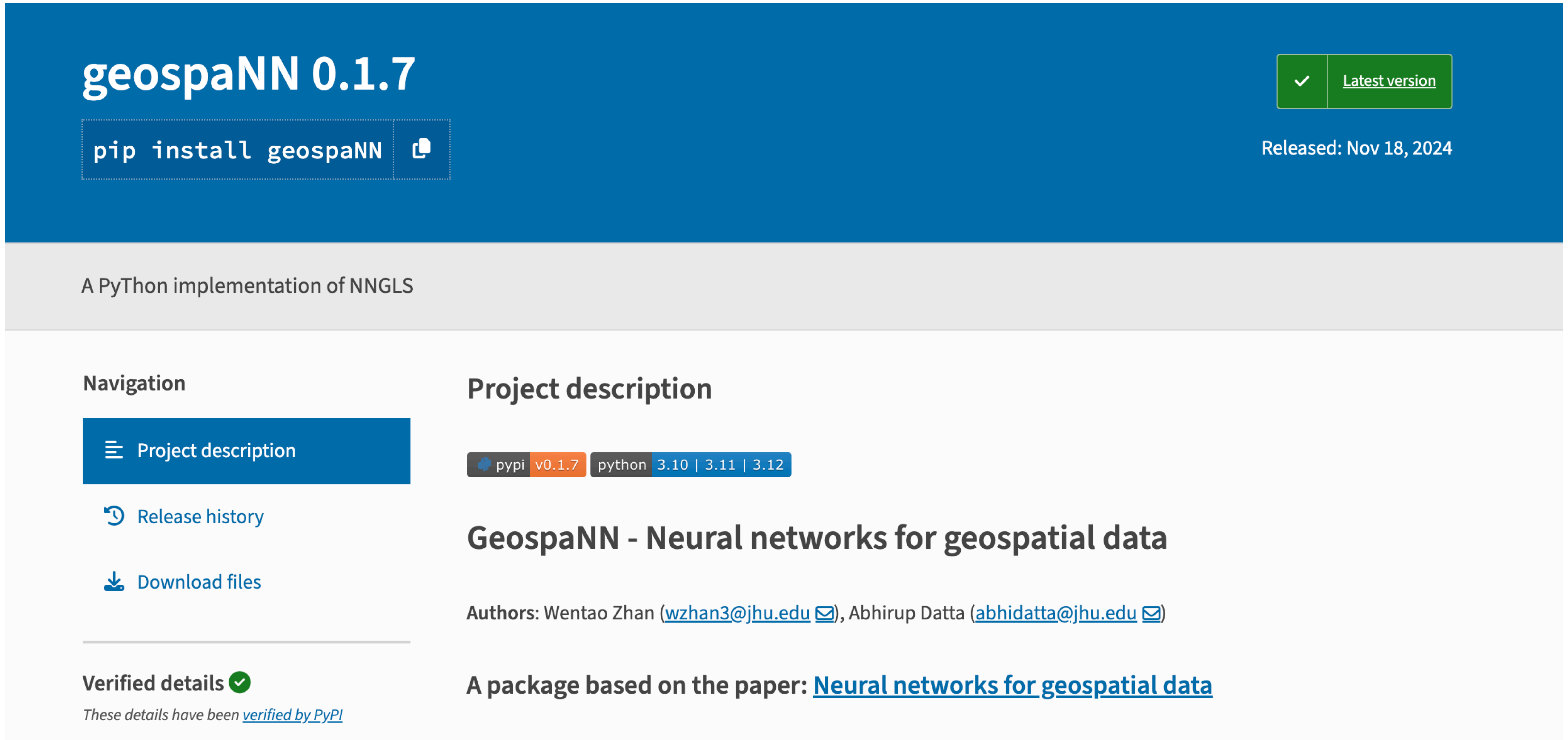
E.g., If X is two-dimensional, i.e., $X_i = (X_{i1}, X_{i2})'$, the PDF is

$$PDF(X_{.1}) = \widehat{m}_1(X_{.1}) = \frac{1}{n}\sum_{i=11}^{n} \widehat{m}_1(X_{.1}, X_{i2})$$

Partial dependence plots (PDP) are plots of PDF for each variable

geospaNN package

Python package for NN-GLS in PyPI
Available at https://pypi.org/project/geospaNN/
With real and simulated data analysis examples

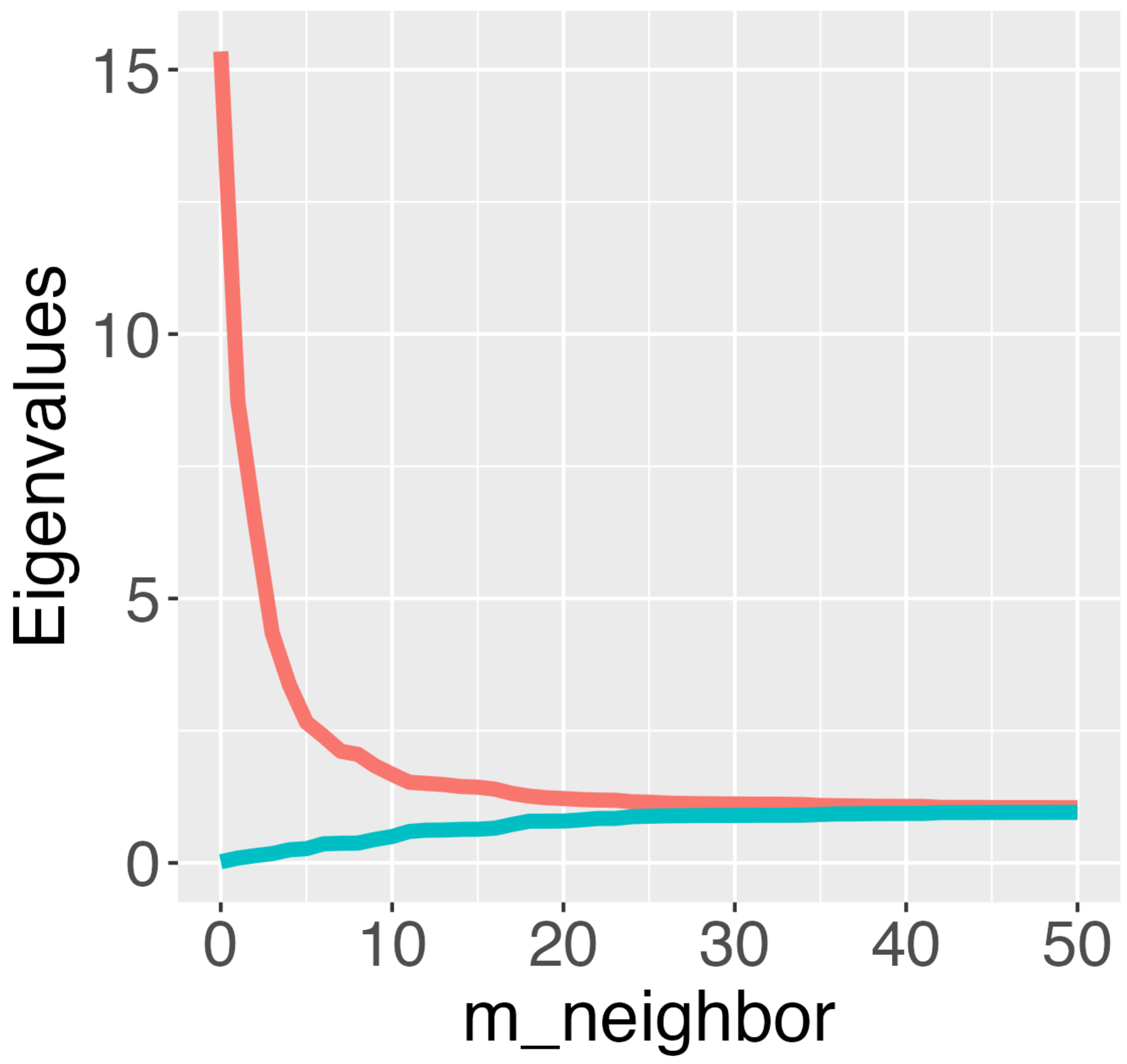


Theory

1-layer NN-GLS is consistent for estimating the non-linear mean m for irregularly observed spatially correlated data processes under increasing domain asymptotics

Finite sample error rates of NN-GLS scale by $\dfrac{\Lambda_{high}}{\Lambda_{low}}$ where Λ_{high} and Λ_{low} are upper and lower the eigenvalues of the discrepancy matrix $E = \Sigma^{T/2}\tilde{\Sigma}^{-1}\Sigma^{1/2}$

Theory



Λ_{high} and Λ_{low} of E

Error rates are better when Λ_{high} is close to Λ_{low}, i.e., when $\tilde{\Sigma} \approx \Sigma$

Worst rate when $\tilde{\Sigma} = I$, i.e., NN-GLS = NN

Shows that ignoring spatial correlation severely impacts performance of NN

Near best rate when using ≈ 15 nearest neighbors in the NNGP covariance $\tilde{\Sigma}$

Summary

NN-GLS: Neural networks within the spatial GP model $Y \sim N(m(X), \tilde{\Sigma})$

$\tilde{\Sigma}$ is the NNGP covariance matrix; m modeled as a multi-layer perceptron (MLP)

GLS loss: $(Y - O)^\top \tilde{\Sigma}^{-1}(Y - O)$, O is the output layer from the MLP

Representation as graph neural network:

MLP with two graph-convolution layers — one each for response and output

GLS loss = OLS loss between the two graph convolution layers

Novel minibatching, backpropagation, and kriging algorithms, $O(n)$ complexity

Implementation of NN-GLS in the Python package geospaNN

Theory of neural networks for spatial data showing need for modeling spatial covariance

Main References

NN-GLS paper: Zhan, W., & Datta, A. (2024). *Neural networks for geospatial data.* Journal of the American Statistical Association, (In press), 1-21.

geospaNN software for NN-GLS: https://pypi.org/project/geospaNN/

Other references

Hornik, K., Stinchcombe, M., & White, H. (1989). *Multilayer feedforward networks are universal approximators.* Neural networks, 2(5), 359-366.

Shen, X., Jiang, C., Sakhanenko, L., & Lu, Q. (2023). *Asymptotic properties of neural network sieve estimators.* Journal of nonparametric statistics, 35(4), 839-868.

Schmidt-Hieber, J. (2020). *Nonparametric regression using deep neural networks with ReLU activation function*, Annals of Statistics 1875-1897.

Farrell, M. H., Liang, T., & Misra, S. (2021). *Deep neural networks for estimation and inference.* Econometrica, 89(1), 181-213.

Fan, J., & Gu, Y. (2023). *Factor augmented sparse throughput deep relu neural networks for high dimensional regression.* Journal of the American Statistical Association, 1-15.

Demyanov, V., Kanevsky, M., Chernov, S., Savelieva, E., & Timonin, V. (1998). *Neural network residual kriging application for climatic data.* Journal of Geographic Information and Decision Analysis, 2(2), 215-232.

Seo, Y., Kim, S., & Singh, V. P. (2015). *Estimating spatial precipitation using regression kriging and artificial neural network residual kriging (RKNNRK) hybrid approach.* Water Resources Management, 29, 2189-2204.

Tarasov, D. A., Buevich, A. G., Sergeev, A. P., & Shichkin, A. V. (2018). *High variation topsoil pollution forecasting in the Russian Subarctic: Using artificial neural networks combined with residual kriging.* Applied Geochemistry, 88, 188-197.

Chen, W., Li, Y., Reich, B. J. and Sun, Y. (2024), *Deepkriging: Spatially dependent deep neural networks for spatial prediction,* Statistica Sinica 34, 291–311.

Gray, S. D., Heaton, M. J., Bolintineanu, D. S. and Olson, A. (2022), *On the use of deep neural networks for large-scale spatial prediction*, Journal of Data Science 20(4), 493–511.

Wang, H., Guan, Y. and Reich, B. (2019), *Nearest-neighbor neural networks for geostatistics*, in ‘2019 international conference on data mining workshops (ICDMW)’, IEEE, pp. 196–205.