Variable selection and architecture search for deep learning

Jean Feng
University of California, San Francisco

ASA SLDS Webinar
August 24th, 2021

Today's talk

1. Review of neural networks

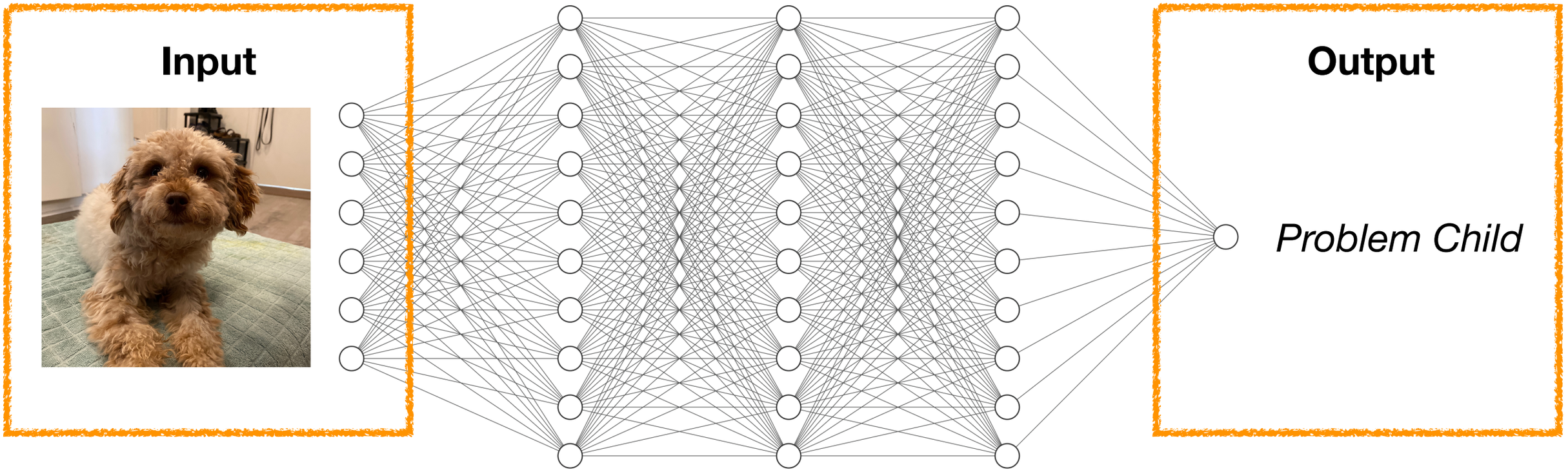
2. Neural networks with variable selection

3. Neural networks with variable *and* architecture selection

What is deep learning?

Deep learning (DL) is a non-parametric machine learning method that builds large computational graphs called neural networks to model complex nonlinearities, interactions, and hierarchies in the data.

— A statistical perspective of DL



Strengths of deep learning

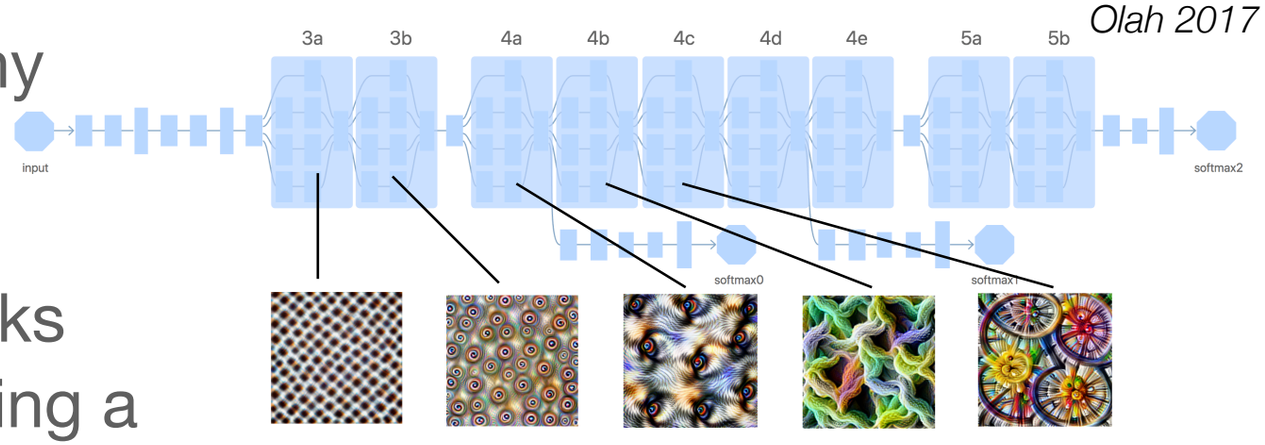
- **Universal Approximation Theorem**: We can create a sufficiently large neural network to approximate any function.

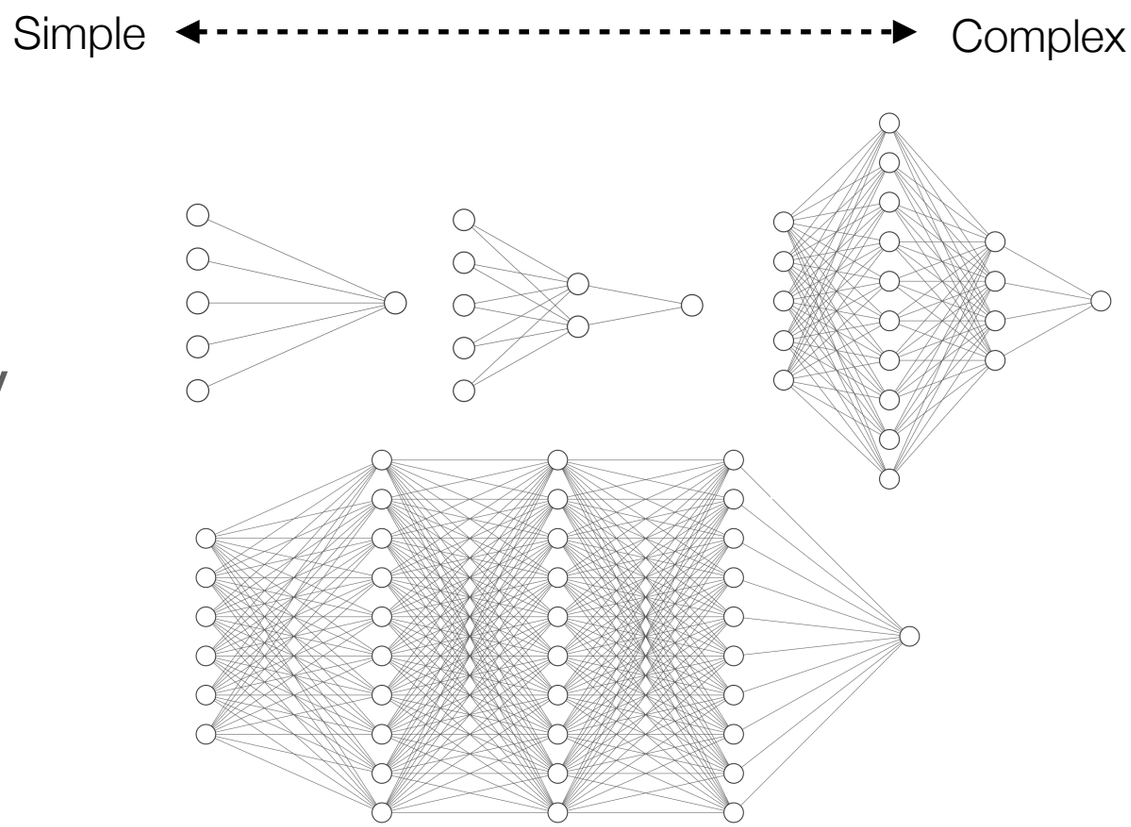
- **Memory efficient**: Neural networks can express complex function using a small number of parameters.

- **Representational learning:** Neural networks iteratively apply non-linear transformations to create increasingly abstract representations.

- **Highly modular:** It is easy to add nodes, layers, modules!

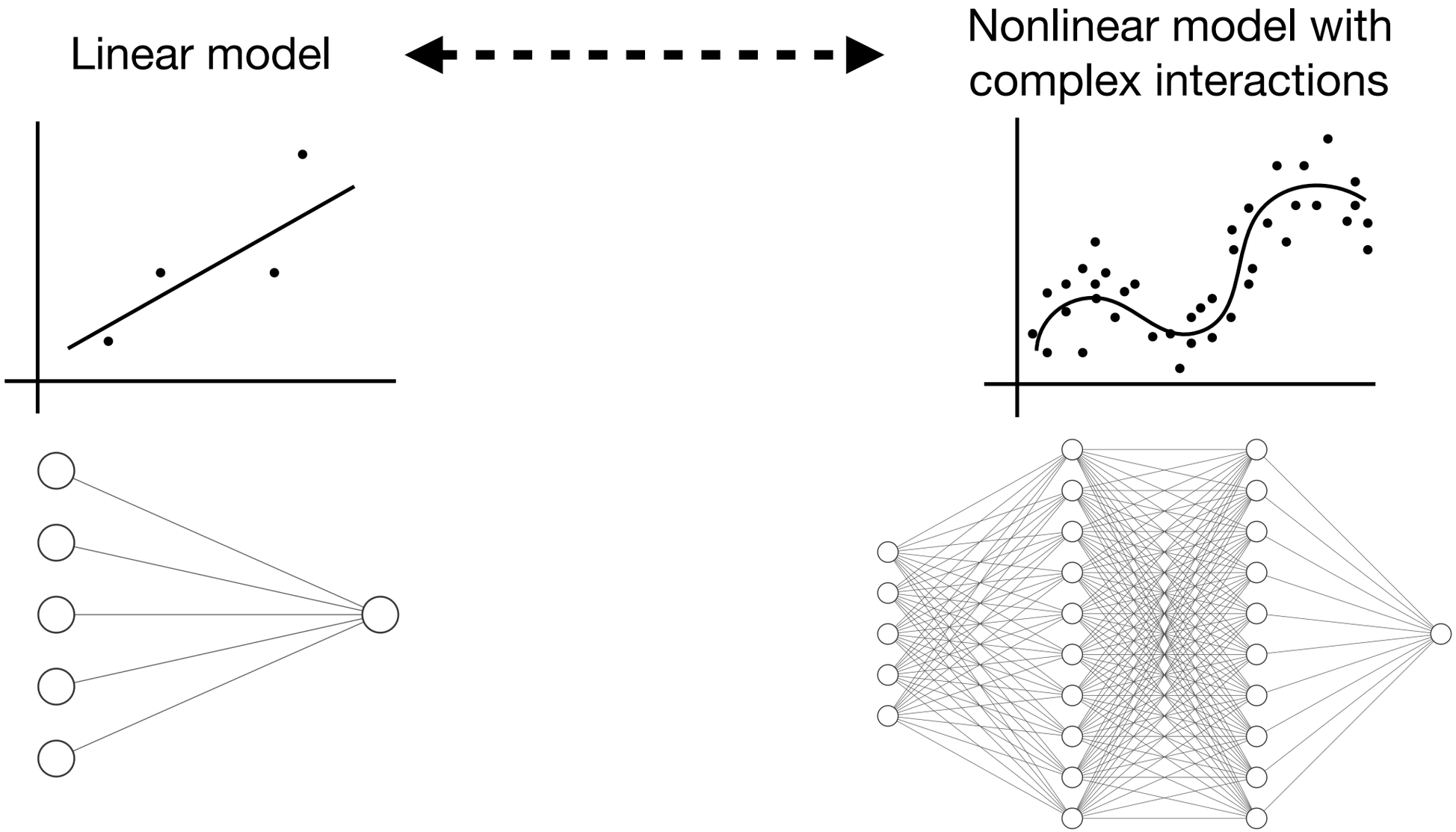
Olah 2017



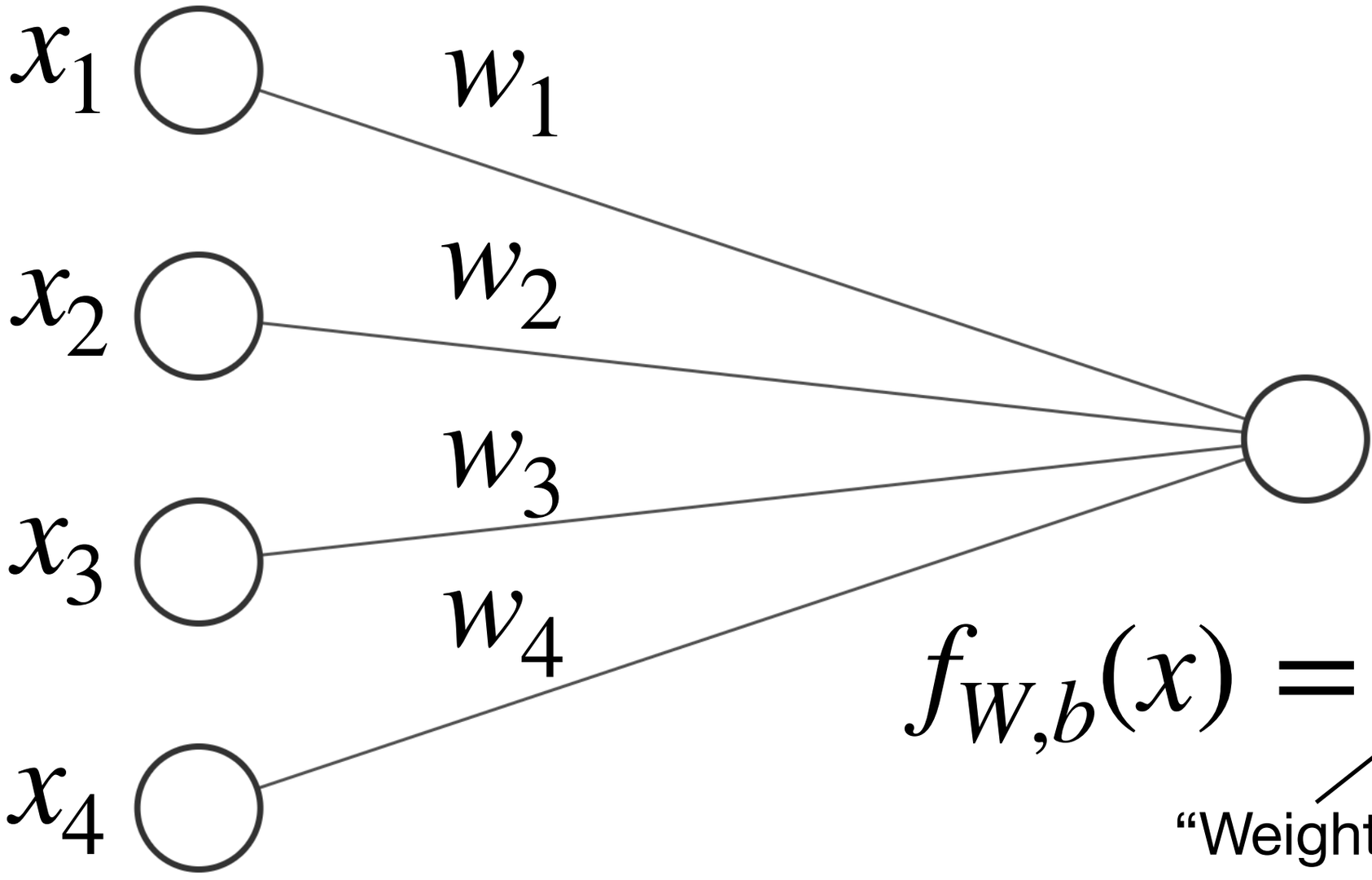


Neural networks

- Neural networks span a wide range of models:



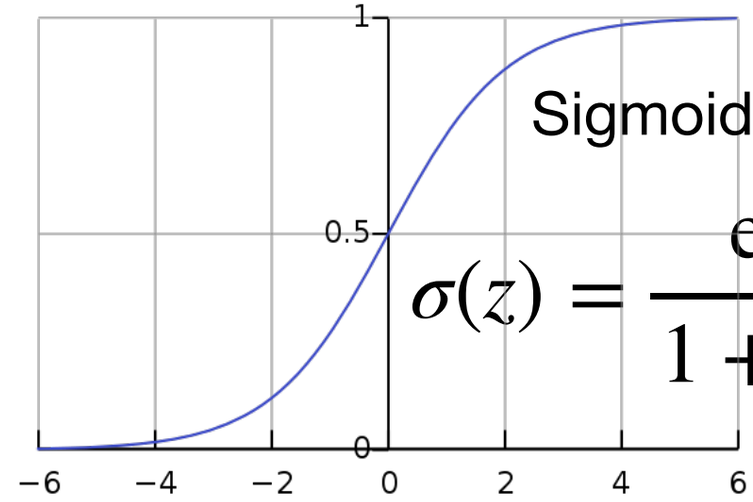
Linear regression as a neural network



Input Layer $\in \mathbb{R}^4$

Output Layer $\in \mathbb{R}^1$

Logistic regression as a neural network



Sigmoid function

$$\sigma(z) = \frac{\exp(z)}{1 + \exp(z)}$$

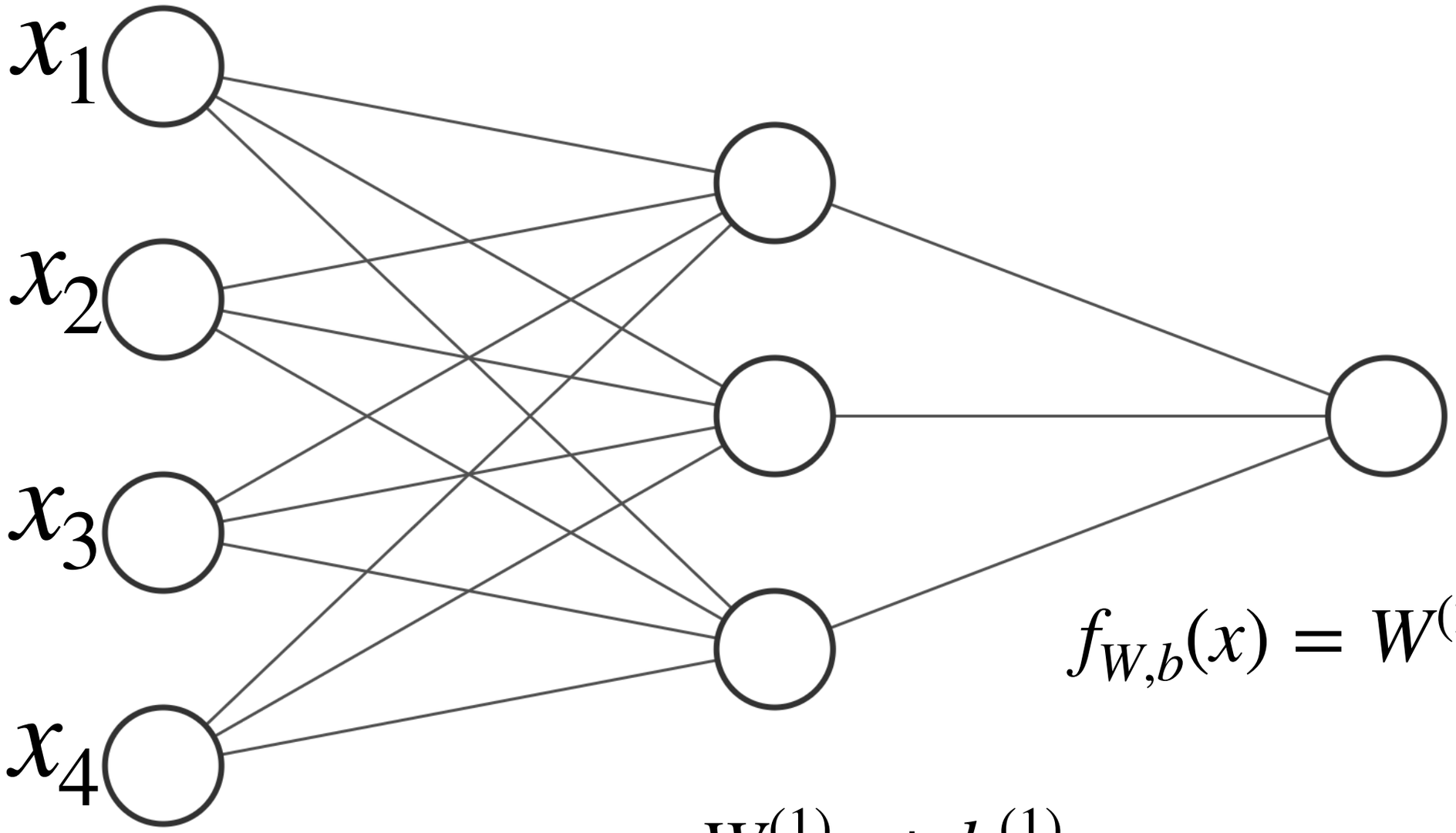
$$f_{W,b}(x) = \sigma(Wx + b)$$

"Activation function"

Input Layer $\in \mathbb{R}^4$

Output Layer $\in \mathbb{R}^1$

A linear neural network



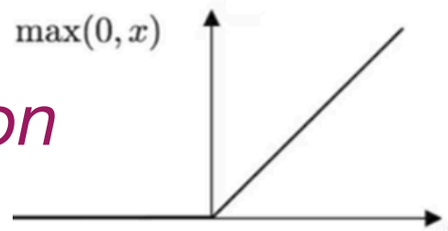
$$f_{W,b}(x) = W^{(2)} z_1 + b^{(2)}$$

$$z_1 = W^{(1)} x + b^{(1)}$$

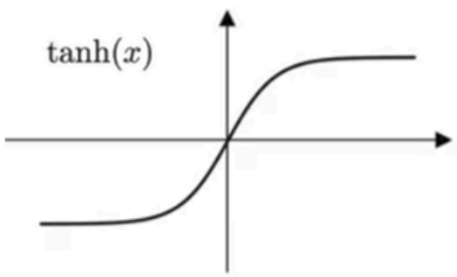
Input *Hidden layer* *Output*

A dense neural network

ReLU Function



Hyper Tangent Function



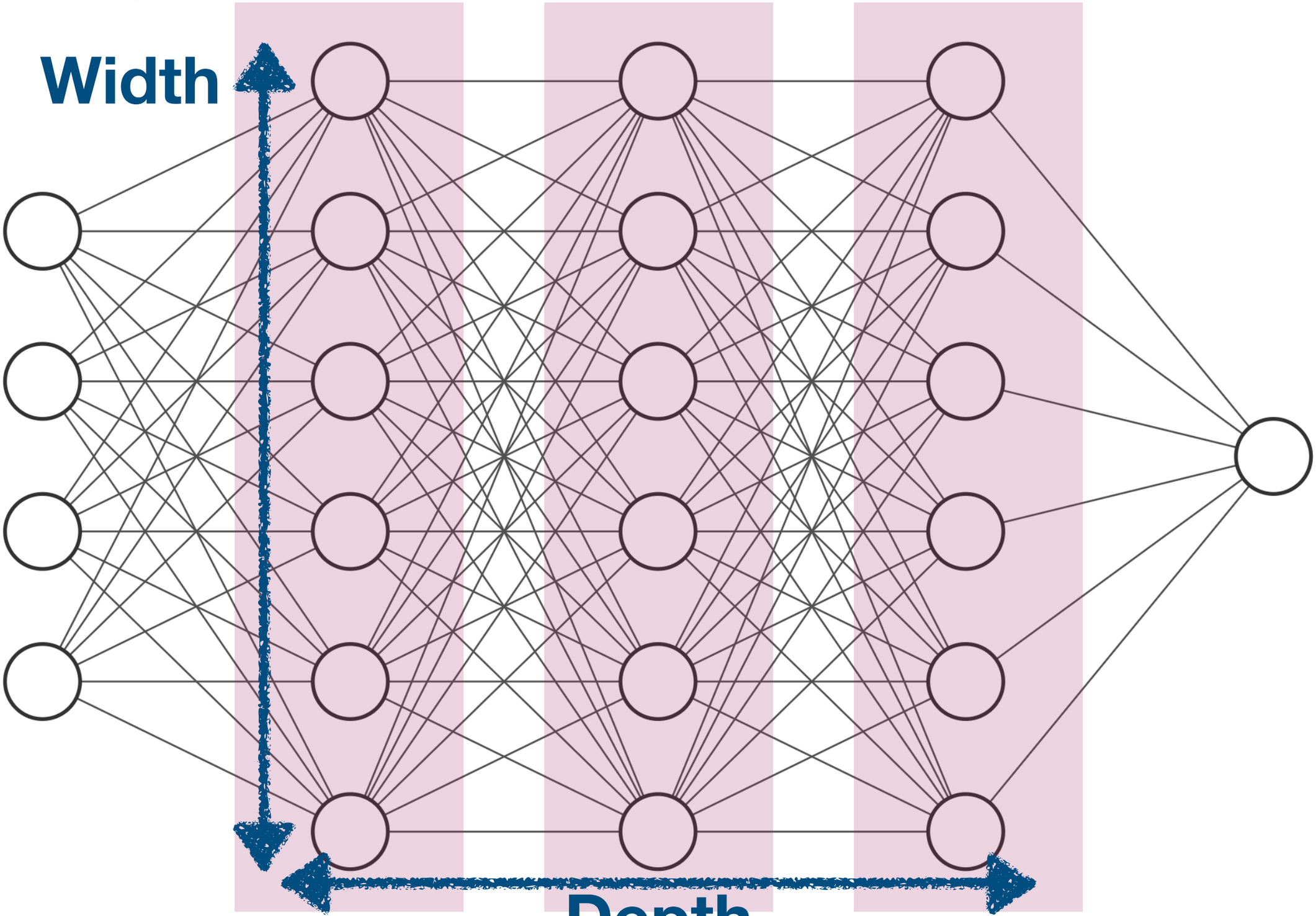
Nonlinear activation function

$$f_{W,b}(x) = \sigma_2(W^{(2)}z_1 + b^{(2)})$$

$$z_1 = \sigma_1(W^{(1)}x + b^{(1)})$$

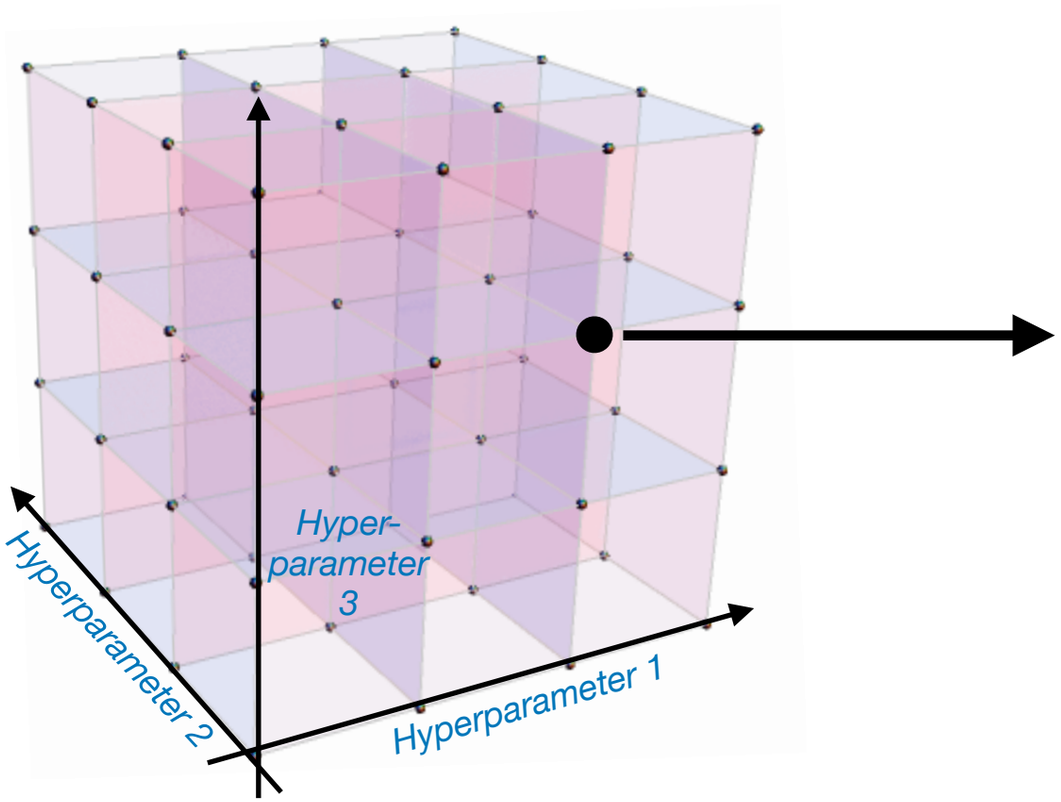
Input *Hidden layer* *Output*

Deep neural networks



Full procedure for training a neural network

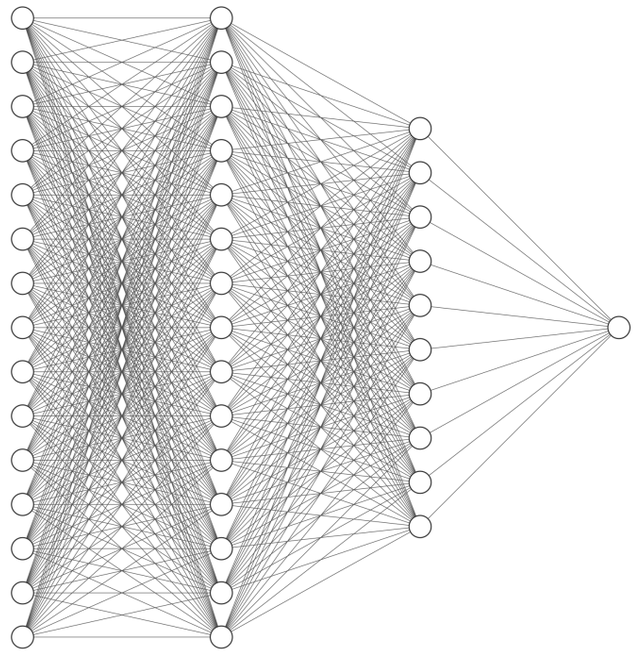
"Outer" optimization procedure:
Hyperparameter optimization



Hyperparameters to tune:

- Number of layers
- Number of hidden nodes per layer
- Which variables to include
- Penalty parameters
- …

"Inner" optimization procedure:
Neural network training

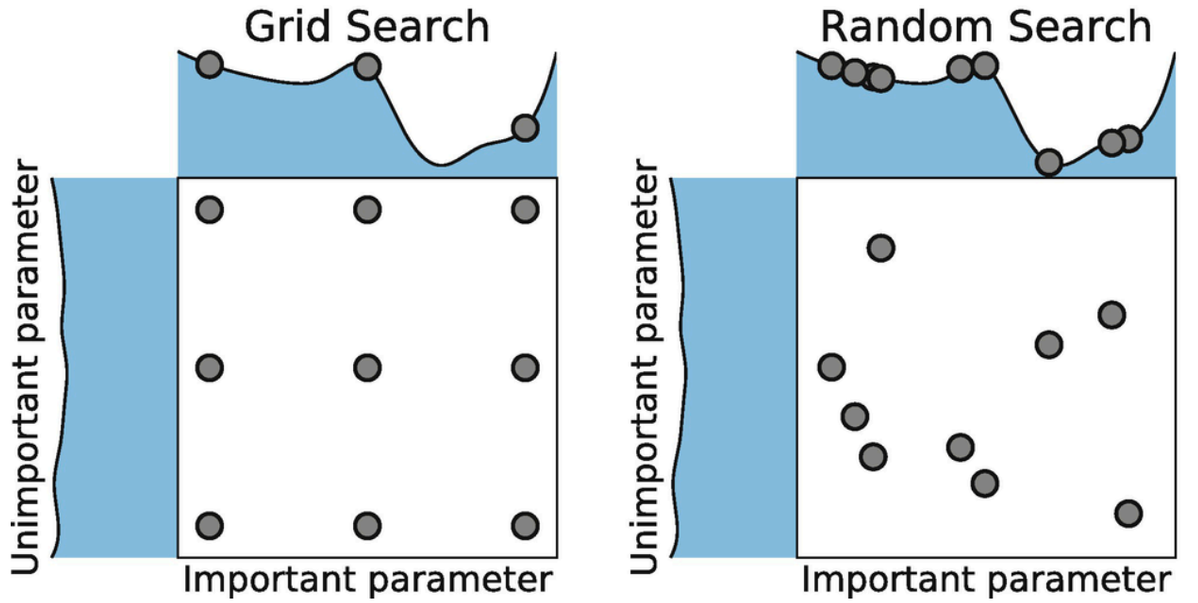


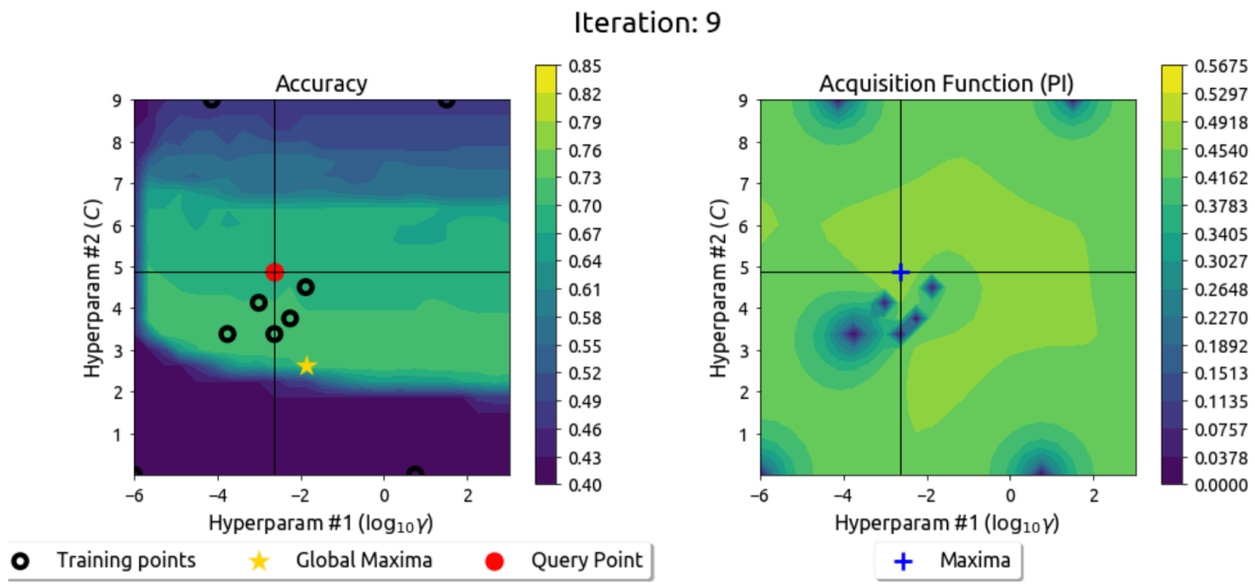
Minimize objective function, e.g.

$$\min_{W,b} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_{W,b}(x_i)\right)^2 + \lambda \|W\|^2$$

Hyperparameter optimization

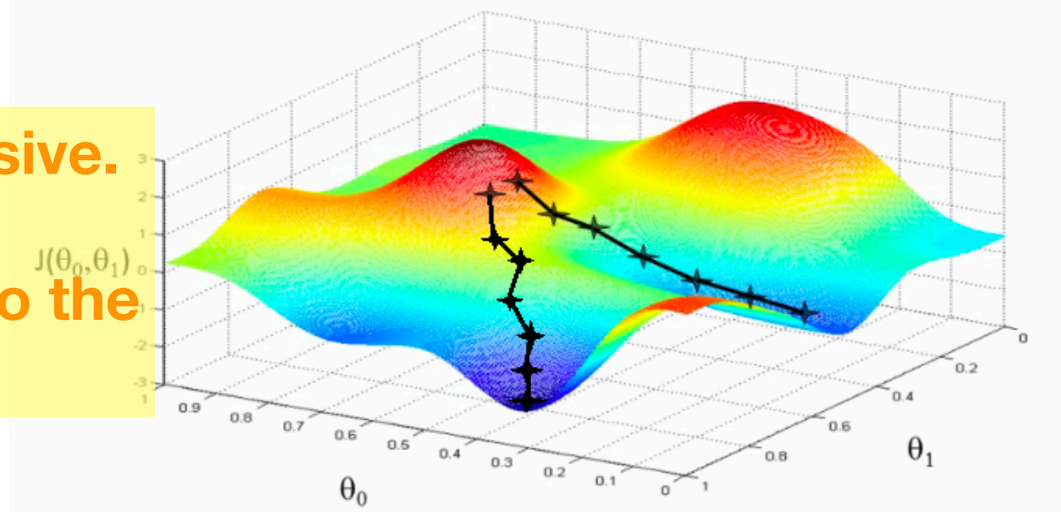
- Grid search
- Random search
- Gradient-free optimization
- Gradient-based optimization





All of these methods are computationally expensive.

Is the added computation time worth it, relative to the performance gain?



Will a neural network really help?

Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?

Manuel Fernández-Delgado MANUEL.FERNANDEZ.DELGADO@USC.ES
Eva Cernadas EVA.CERNADAS@USC.ES
Senén Barro SENEN.BARRO@USC.ES
CITIUS: Centro de Investigación en Tecnoloxías da Información da USC
University of Santiago de Compostela
Campus Vida, 15872, Santiago de Compostela, Spain

Dinani Amorim DINANIAMORIM@GMAIL.COM
Departamento de Tecnologia e Ciências Sociais- DTCS
Universidade do Estado da Bahia
Av. Edgard Chastinet S/N - São Geraldo - Juazeiro-BA, CEP: 48.305-680, Brasil

Data set	#pat.	#inp.	#cl.	%Maj.
abalone	4177	8	3	34.6
ac-inflam	120	6	2	50.8
acute-nephritis	120	6	2	58.3
adult	48842	14	2	75.9
annealing	798	38	6	76.2
arrhythmia	452	262	13	54.2

Rank	Acc.	κ	Classifier
32.9	82.0	63.5	parRF_t (RF)
33.1	**82.3**	**63.6**	rf_t (RF)
36.8	81.8	62.2	svm_C (SVM)
38.0	81.2	60.1	svmPoly_t (SVM)
39.4	81.9	62.5	rforest_R (RF)
39.6	82.0	62.0	elm_kernel_m (NNET)
40.3	81.4	61.1	svmRadialCost_t (SVM)
42.5	81.0	60.0	svmRadial_t (SVM)
42.9	80.6	61.0	C5.0_t (BST)
44.1	79.4	60.5	avNNet_t (NNET)

- Neural networks don't often outperform other off-the-shelf ML methods on tabular datasets.
- Neural networks often overfit on high-dimensional datasets.

*Is there really no **simple** procedure for fitting a neural network that works well for **tabular** and/or **high-dimensional** data?*

Today's talk

1. Review of neural networks

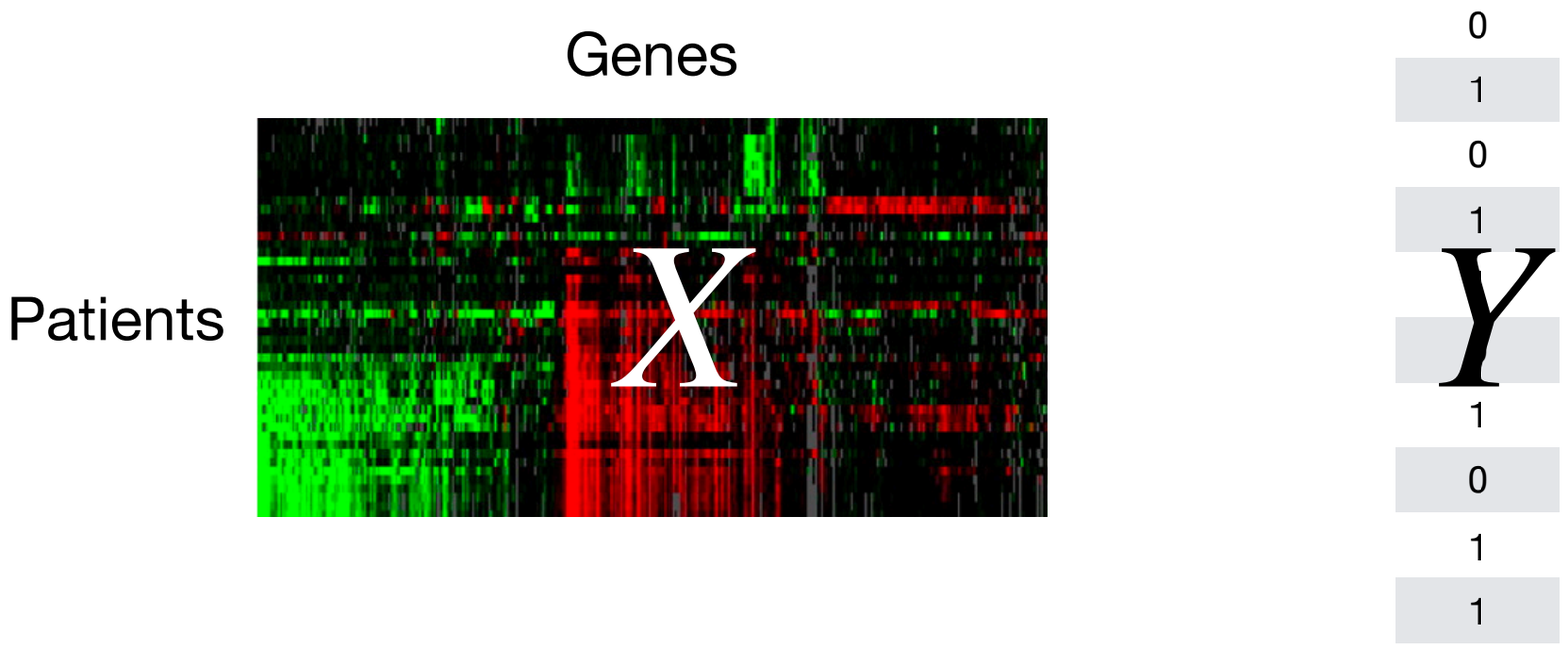
2. Neural networks with variable selection

3. Neural networks with variable *and* architecture selection

Nonparametric learning in high dimensions

Predicting Y given X when:

- The number of variables p is large compared to the number of observations n (**high-dimensional**)
- The conditional relationship is unknown and we'd like to place minimal assumptions on its structure **(nonparametric)**
- Our prior knowledge tells us that only a subset of variables are important **(sparsity)**



Recall: sparse linear models

- Ordinary least squares is likely to overfit in high-dimensional settings.

- Sparsity-inducing penalties like the lasso and the group lasso are commonly used in such settings to regularize the model:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \theta^\top x_i\right)^2 + \lambda \|\theta\|_1$$

Lasso (Tibshirani 1996)

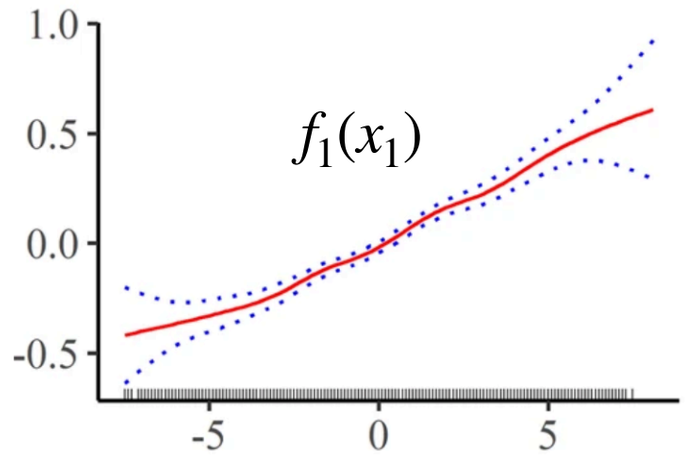
$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \theta^\top x_i\right)^2 + \lambda_1 \|\theta\|_1 + \lambda_2 \sum_{j=1}^{m} \|\theta_{(j)}\|_2$$

Sparse Group Lasso (Simon 2013)

Recall: sparse additive models

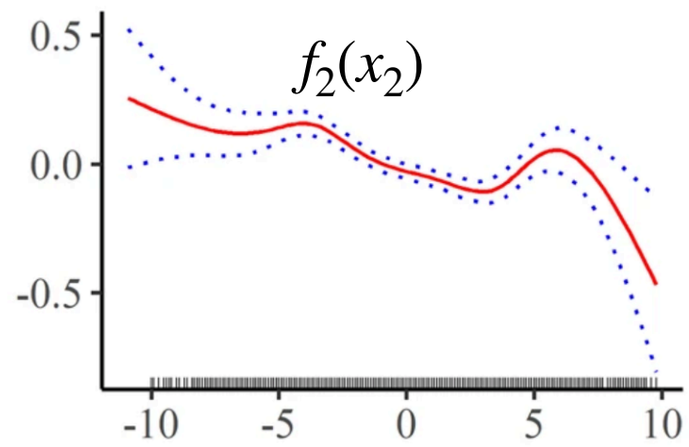
- Relaxing the assumption of linearity, we can also fit additive models of the form:

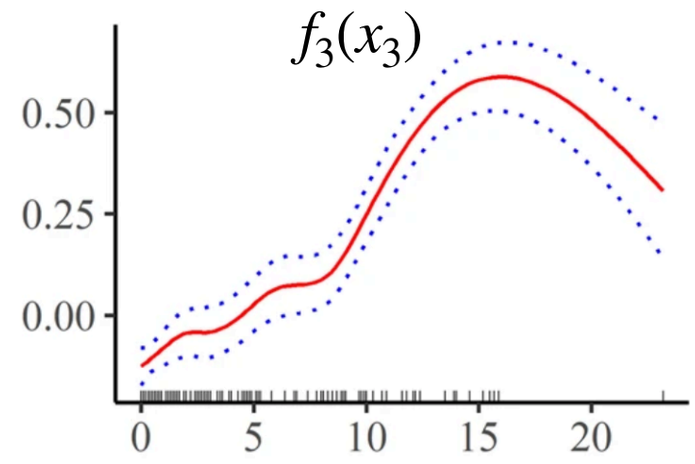
$$f(x) = \sum_{j=1}^{p} f_j(x_j)$$



- To reduce overfitting, we can fit sparse additive models (SpAM) by penalizing the L2 norm of the univariate functions (Ravikumar 2009)

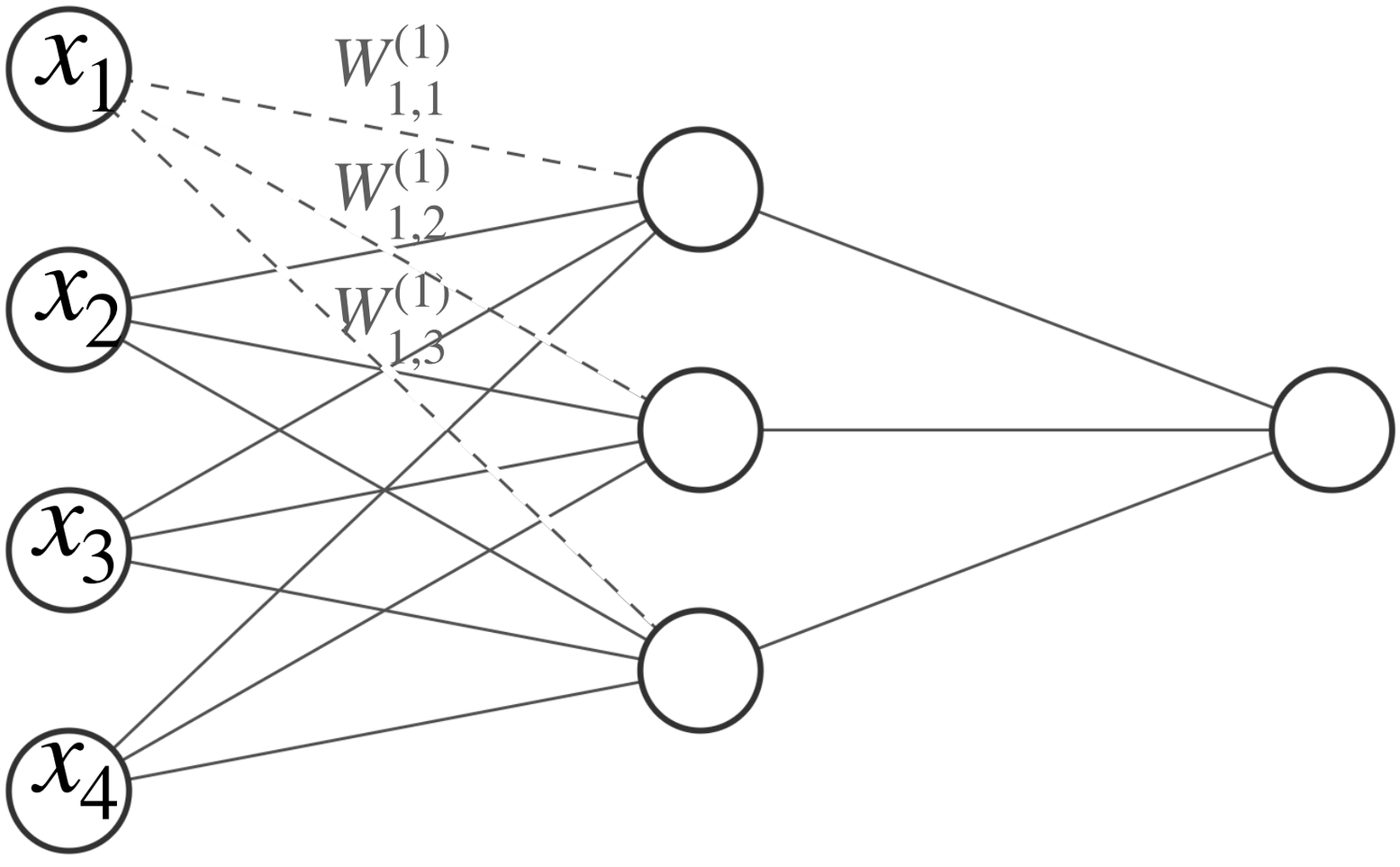
$$\min_{f_j : j=1,\ldots,p} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f(x_i)\right)^2 + \lambda \sum_{j=1}^{p} \| f_j \|_2$$





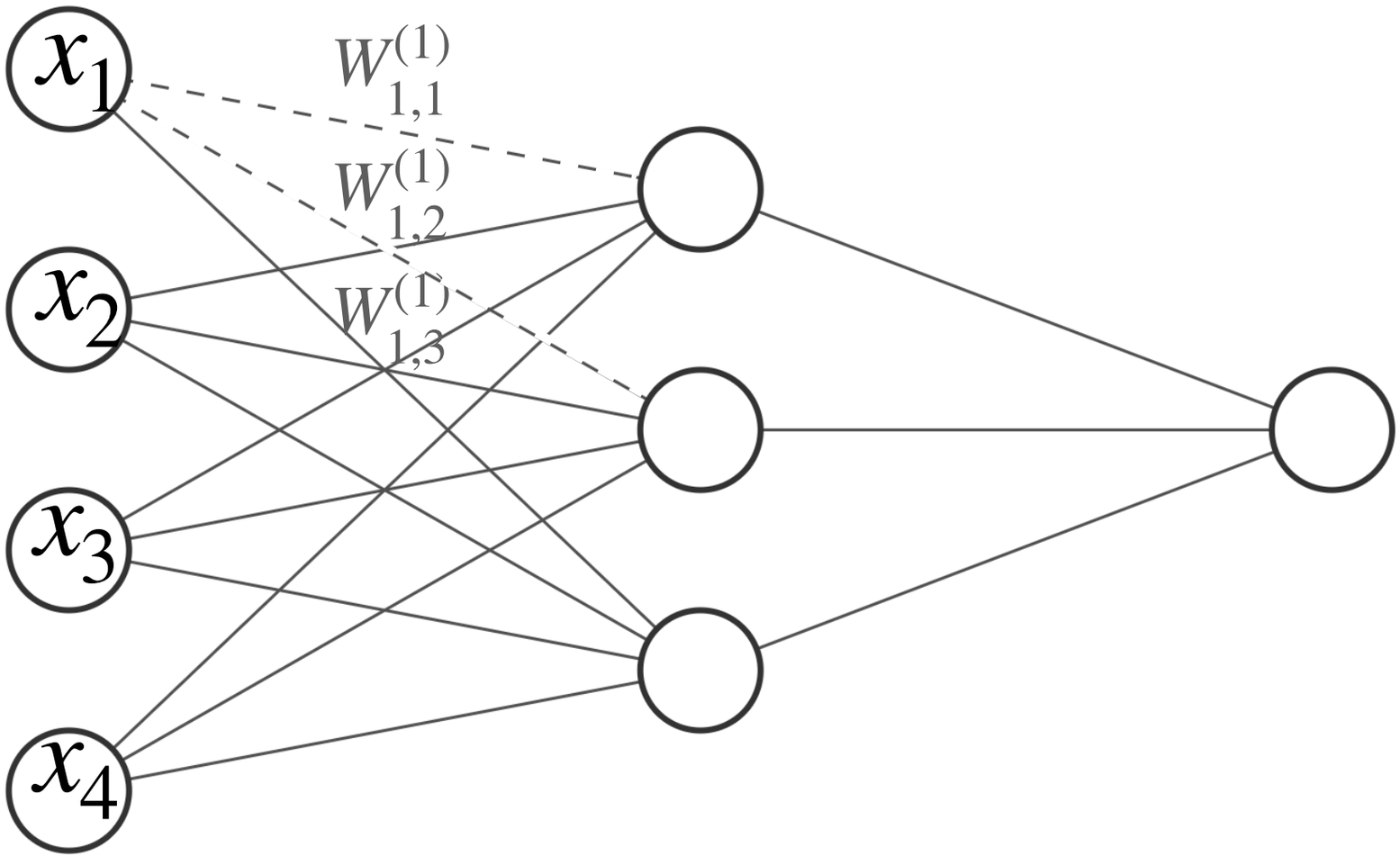
Sparsity + Neural networks?

- We can take a similar approach for neural networks.
- Example 1: If $W^{(1)}_{1,1} = W^{(1)}_{1,2} = W^{(1)}_{1,3} = 0$, then the neural network will not depend on x_1.



Sparsity + Neural networks?

- We can take a similar approach for neural networks.
- Example 2: If $W^{(1)}_{1,1} = W^{(1)}_{1,2} = 0$, then only one of the hidden nodes will depend on x_1.



Sparse-**I**nput **N**eural **N**etworks (SPINN)

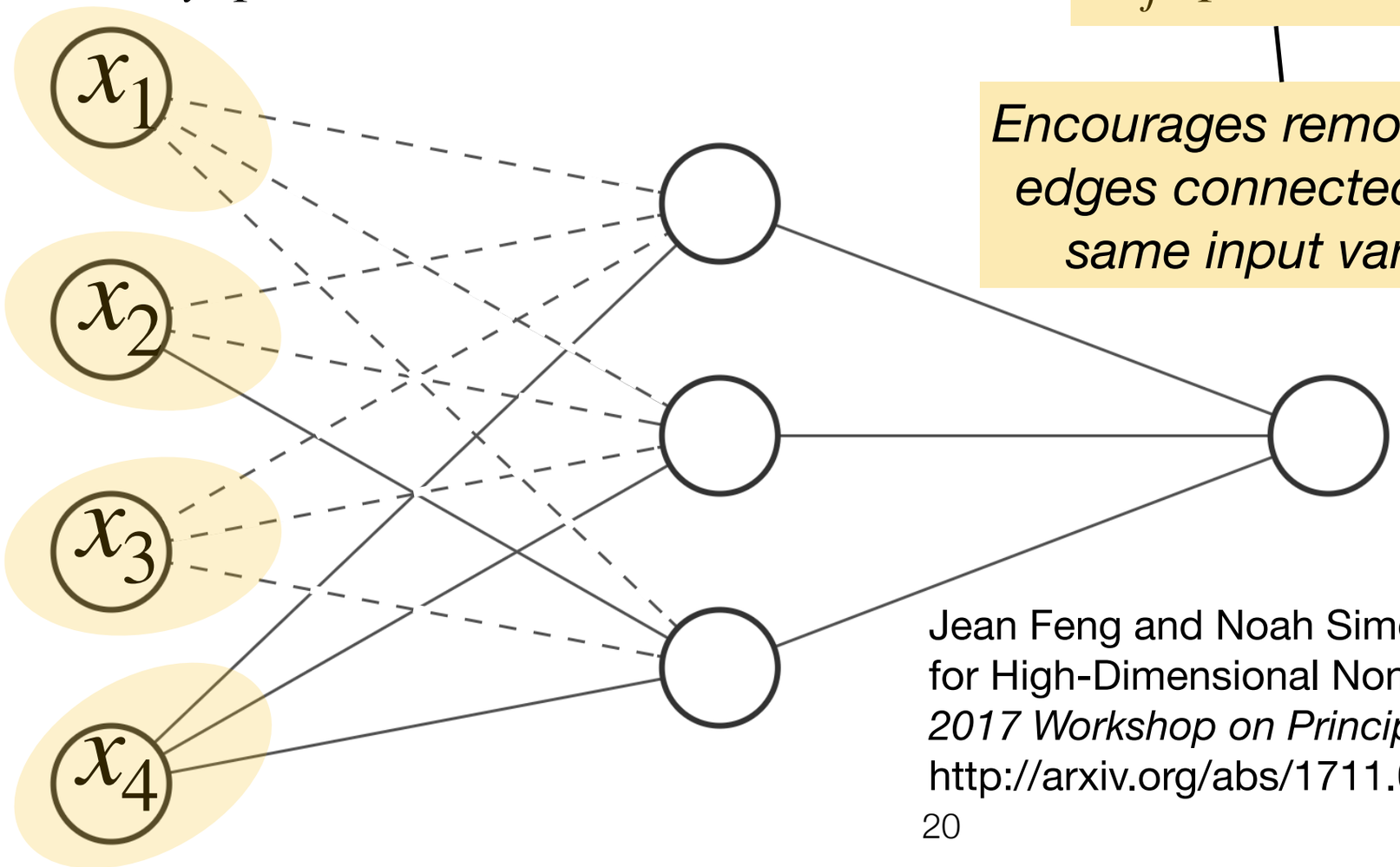
- We can use a sparse group lasso penalty to train a neural network that depends on a small number of variables:

Encourages sparsity of edge weights in the first layer

$$\min_{W,b} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_{W,b}(x_i)\right)^2 + \lambda_1 \|W^{(1)}\|_1 + \lambda_2 \sum_{j=1}^{p} \|W^{(1)}_{j,\cdot}\|_2 + \lambda_3 \sum_{l=2}^{L} \|W^{(l)}\|_2^2$$

Encourages removal of all edges connected to the same input variable

Encourages weight matrices of later layers to be small



Jean Feng and Noah Simon. "Sparse-Input Neural Networks for High-Dimensional Nonparametric Regression." *ICML 2017 Workshop on Principled Approaches to Deep Learning.* http://arxiv.org/abs/1711.07592.

Training sparse-input neural networks

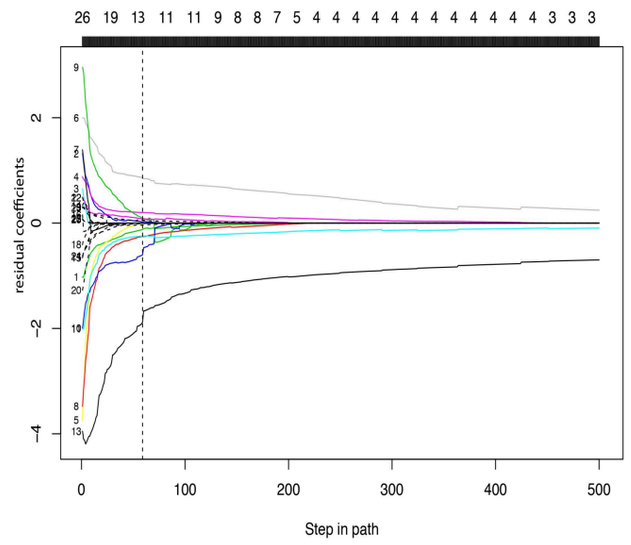
- We minimize the penalized empirical loss via ***generalized*** gradient descent:

At each iteration, we:

1. Take a gradient step with respect to the smooth terms (empirical risk + ridge penalty)

2. Shrink and threshold the parameters in the sparse group lasso ← ***New step***

- To test different values of the penalty parameter, you can consider **warm-starts** to accelerate training.
  - Recent works have explored analogs of the Lasso paths for neural networks (Lemhadri 2021).



Simulation study

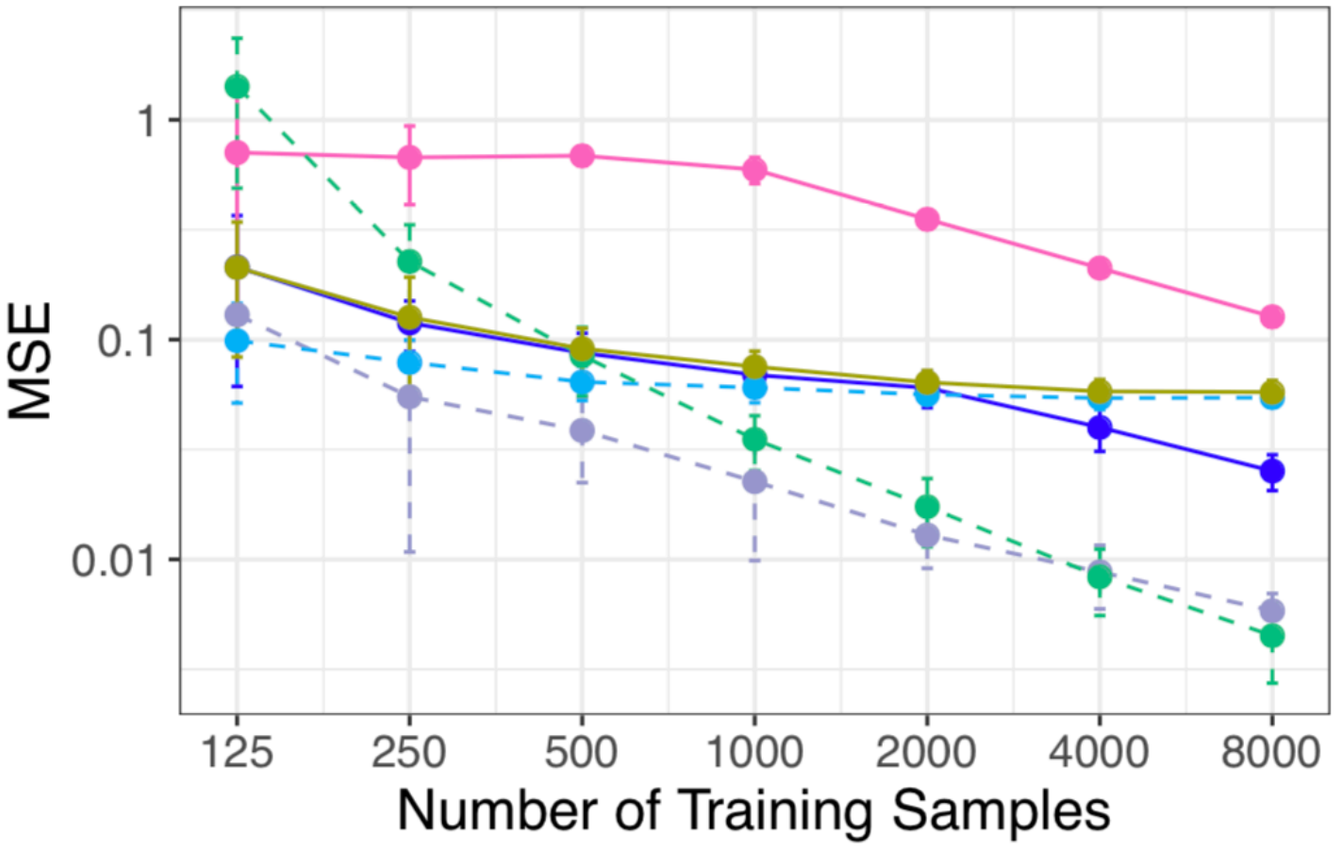
Additive multivariate model, $p = 1000$

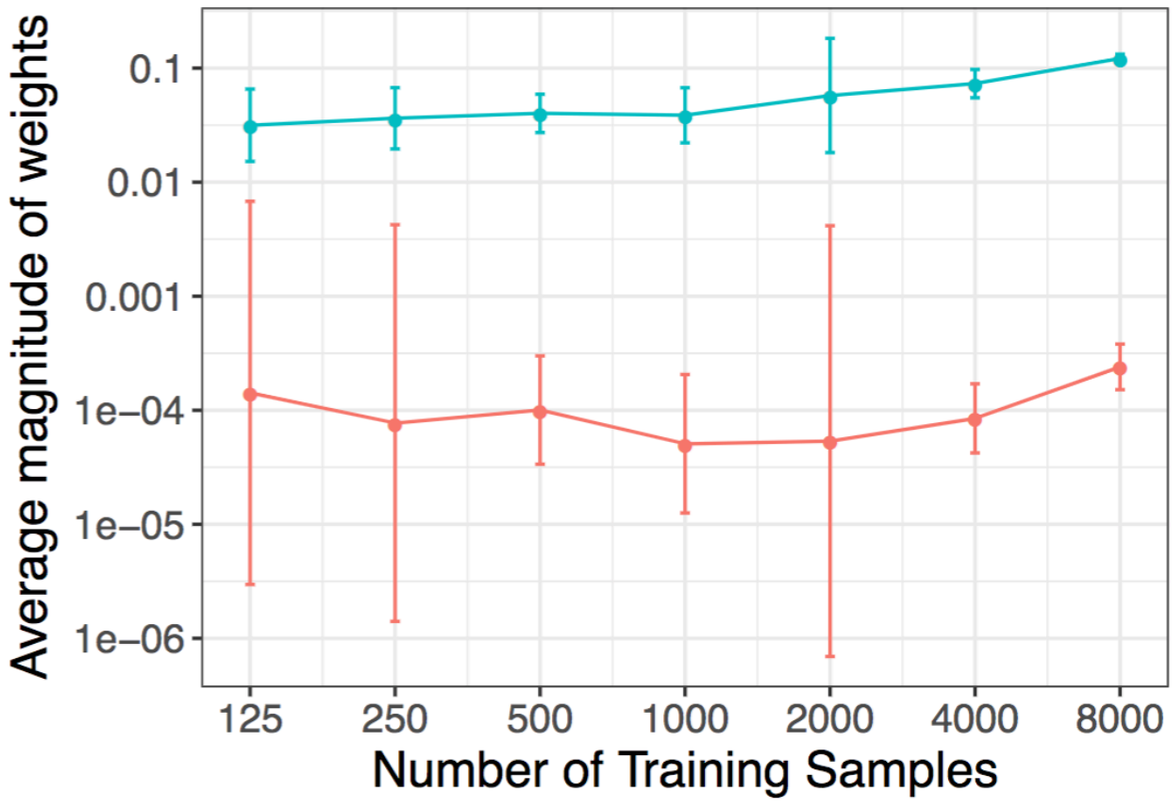
$$E[Y|X] = (x_1 \wedge x_2)\cos(1.5x_3 + 2x_4) + e^{x_5+\sin(x_4)}x_2 + \sin(x_6 \vee x_3)(x_5 - x_1)$$

Methods

- Sparse input NN
- Ridge-penalized NN
- Sparse Additive Model (SpAM)
- Oracle NN using only relevant covariates
- Oracle additive model using univariate smoothing splines
- Oracle 6-variate smoothing spline

Simulation study





Models

- Sparse-input NN
- Ridge-penalized NN
- Oracle NN
- Multivariate Oracle
- Univariate Oracle
- SpAM

Weights

- Irrelevant
- Relevant

Application: Predicting riboflavin production from gene expression data

- High-throughput genomic data set for the production of riboflavin in Bacillus subtilis[4]
  - $n = 71$ experimental settings with gene expression profiles of $p = 4088$ genes
- Randomly split the data with 4/5 training data and 1/5 test data. Results for 30 random splits. Standard errors in parentheses.

Method	MSE on test set	Num. of nonzero features
Sparse-input NN	**0.124** (0.010)	52.7 (6.1)
Ridge-penalized NN	0.237 (0.018)	4088 (0)
SpAM	0.145 (0.014)	34.5 (1.3)
Linear model with Lasso	0.145 (0.017)	39.1 (3.3)

Summary

- We can use the lasso and the group lasso to train neural networks that depend on a small number of variables.

- SPINN performs well in high-dimensional settings and *significantly* better than ridge-penalized NNs.

- However, the performance of SPINN depends on performing architecture search, which is computationally expensive.

Today's talk

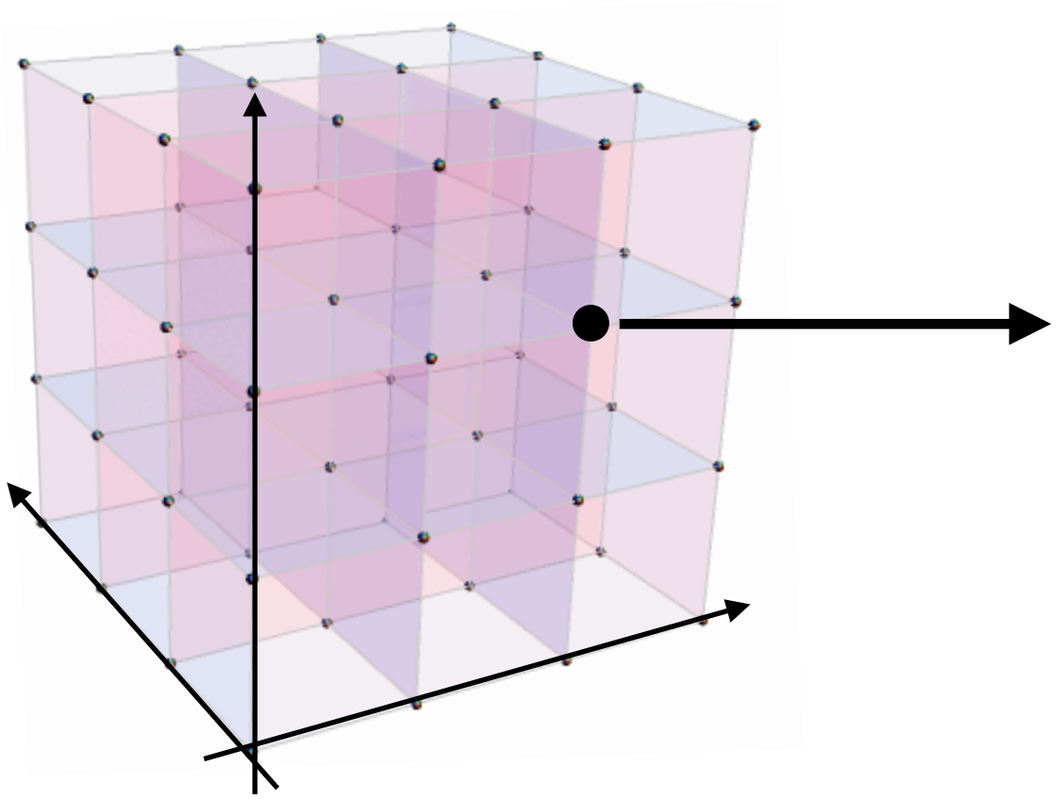
1. Review of neural networks

2. Neural networks with variable selection

3. Neural networks with variable *and* architecture selection

We have shifted the burden of variable selection to the "inner" optimization procedure, but the problem of architecture selection remains…

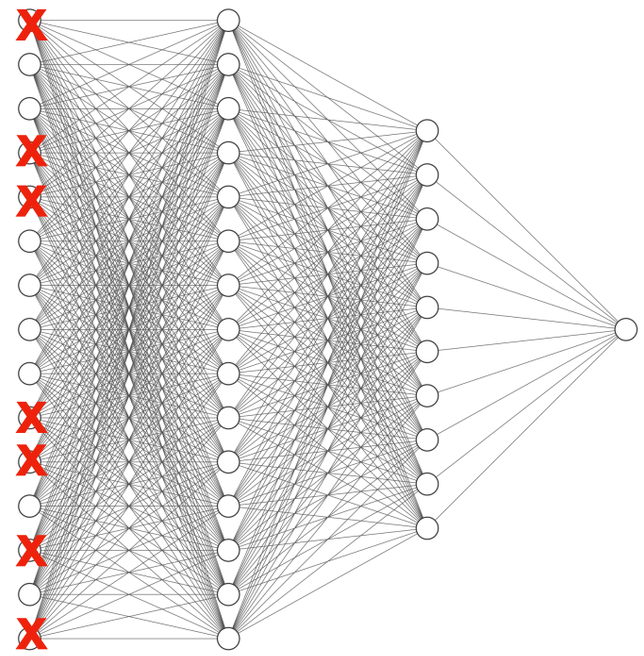
"Outer" optimization procedure:
Hyperparameter optimization



Hyperparameters to tune:

- ~~Which variables to include~~
- Number of layers
- Number of hidden nodes in each layer
- Penalty parameters
- …

"Inner" optimization procedure:
Neural network training



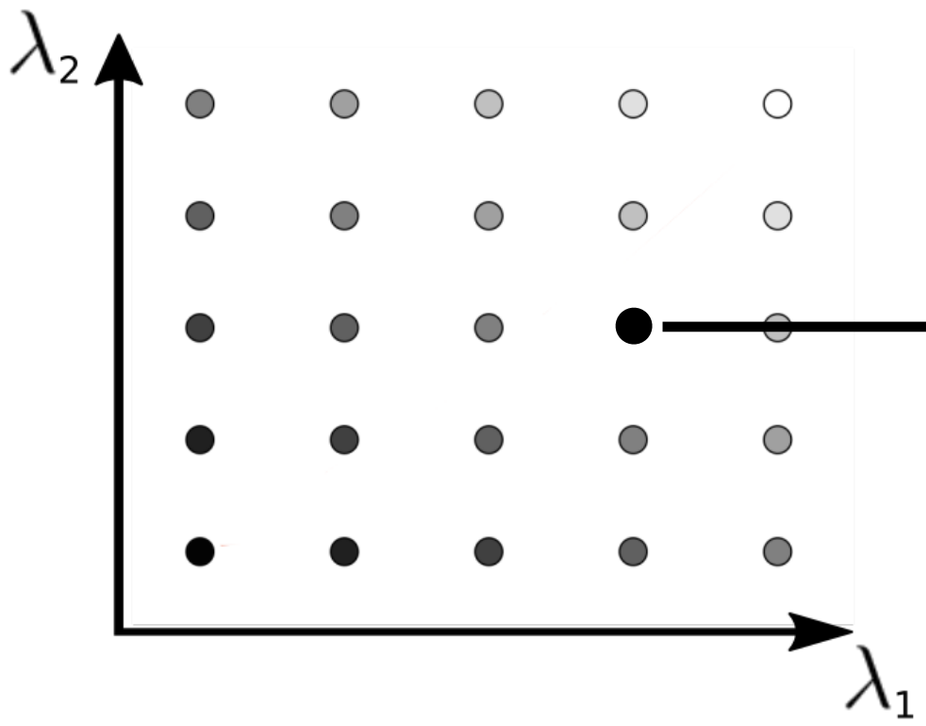
$$\min_{W,b} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_{W,b}(x_i)\right)^2 + \sum_{j=1}^{3} \lambda_j P_j(W)$$

Penalization performs:

- Variable selection

Can we also shift the burden of architecture search?

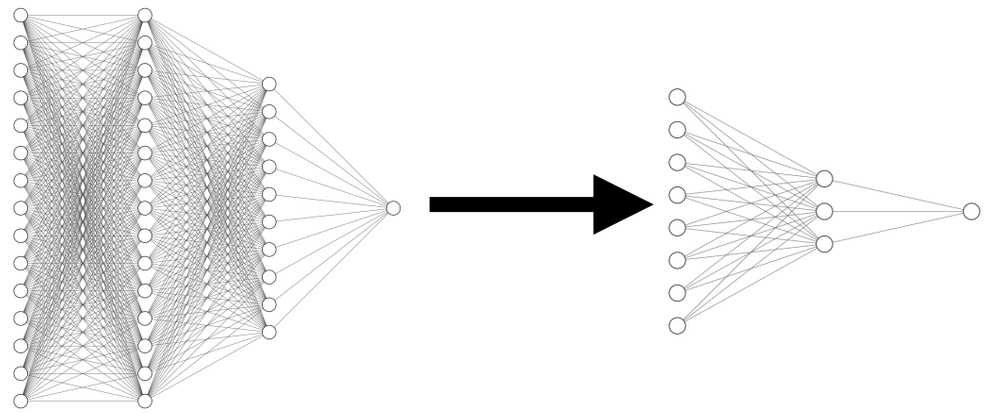
"Outer" optimization procedure:
Hyperparameter optimization



Hyperparameters to tune:

- ~~Which variables to include~~
- ~~Number of layers~~
- ~~Number of hidden nodes in each layer~~
- Penalty parameters
- …

"Inner" optimization procedure:
Neural network ***pruning*** + training



$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_\theta(x_i)\right)^2 + \lambda_1 P_1(\theta) + \lambda_2 P_2(\theta)$$

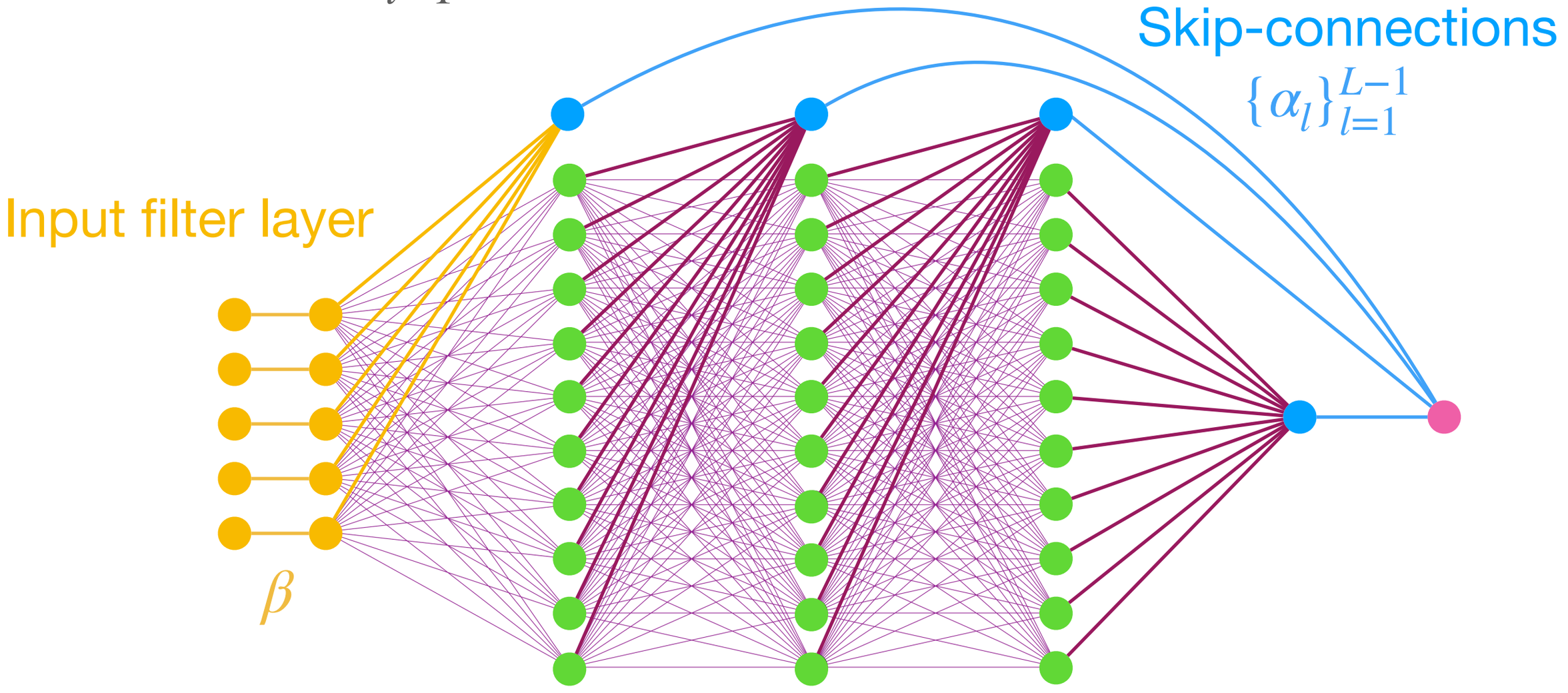
Two!

Penalization performs:

- Variable selection
- Architecture selection

Sparse-**I**nput hi**ER**archical networks (SIER-net)

$$\min_{\beta,W,\alpha} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_{\beta,W,\alpha}(x_i) \right)^2 + \lambda_1 \|\beta\|_1 + \lambda_2 \|W\|_1$$

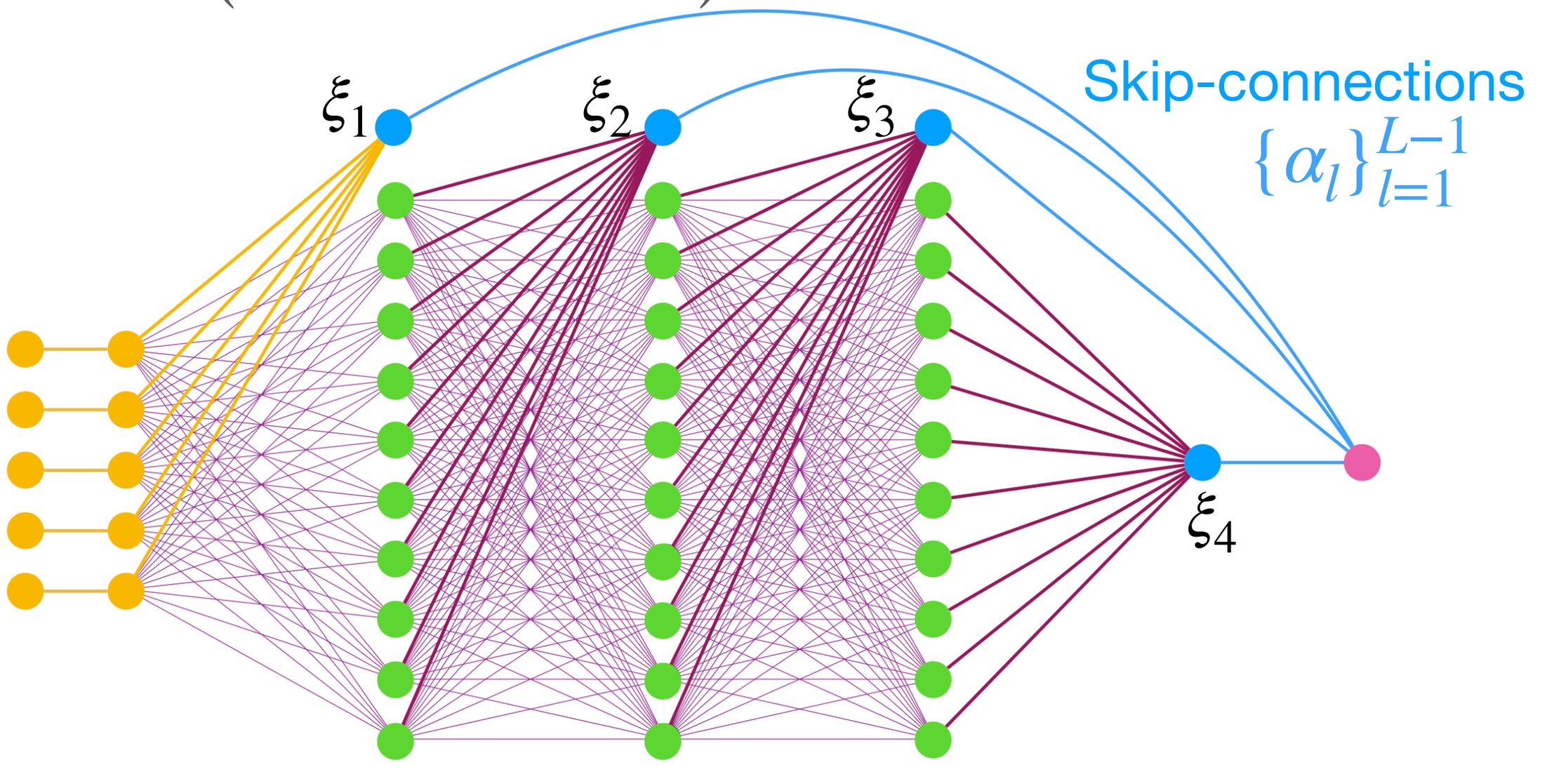


Jean Feng and Noah Simon. “Ensembled Sparse-Input Hierarchical Networks for High-Dimensional Datasets.” *Under review*. http://arxiv.org/abs/2005.04834.

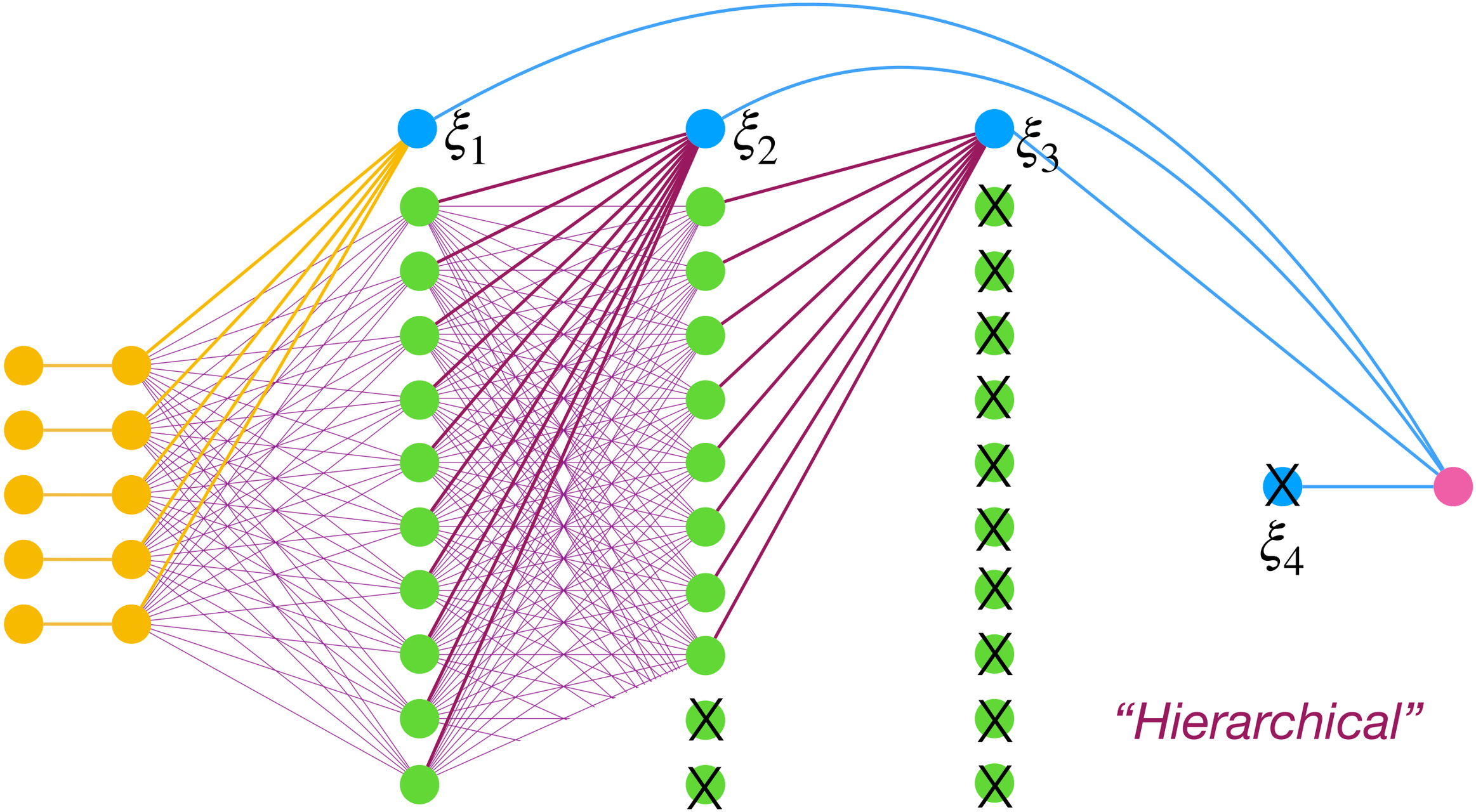
$$\min_{\beta,W,\alpha} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_{\beta,W,\alpha}(x_i) \right)^2 + \lambda_1 \|\beta\|_1 + \lambda_2 \|W\|_1$$

$$f_{\beta,W,\alpha}(x) = \psi \left(\sum_{l=1}^{L-1} \frac{|\alpha_l|}{\sum_{l'=1}^{L} |\alpha_{l'}|} \xi_l \right)$$

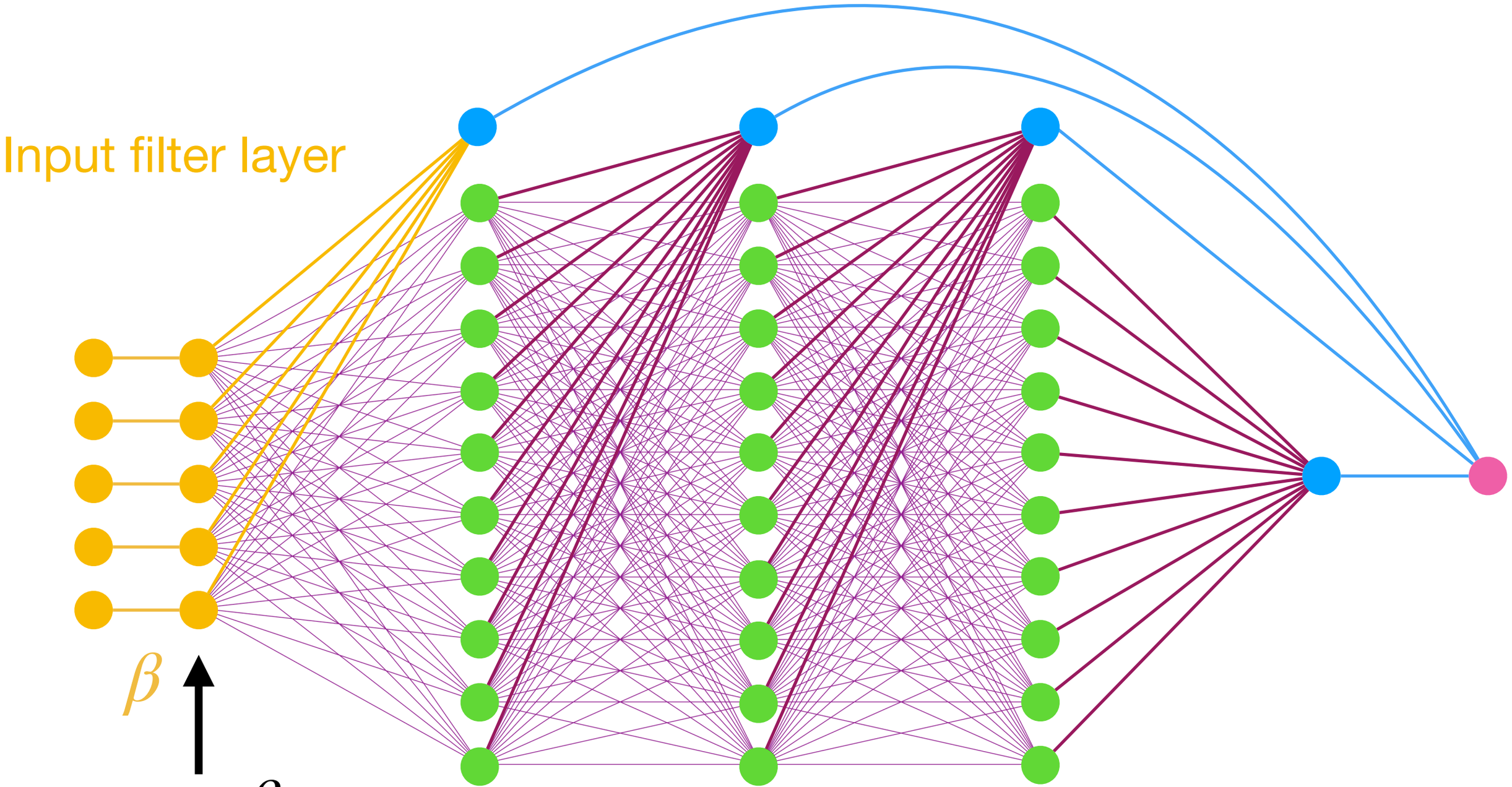
where ξ_l are the values of the additional hidden nodes at layer l



$$\min_{\beta,W,\alpha} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_{\beta,W,\alpha}(x_i) \right)^2 + \lambda_1 \|\beta\|_1 + \lambda_2 \|W\|_1$$

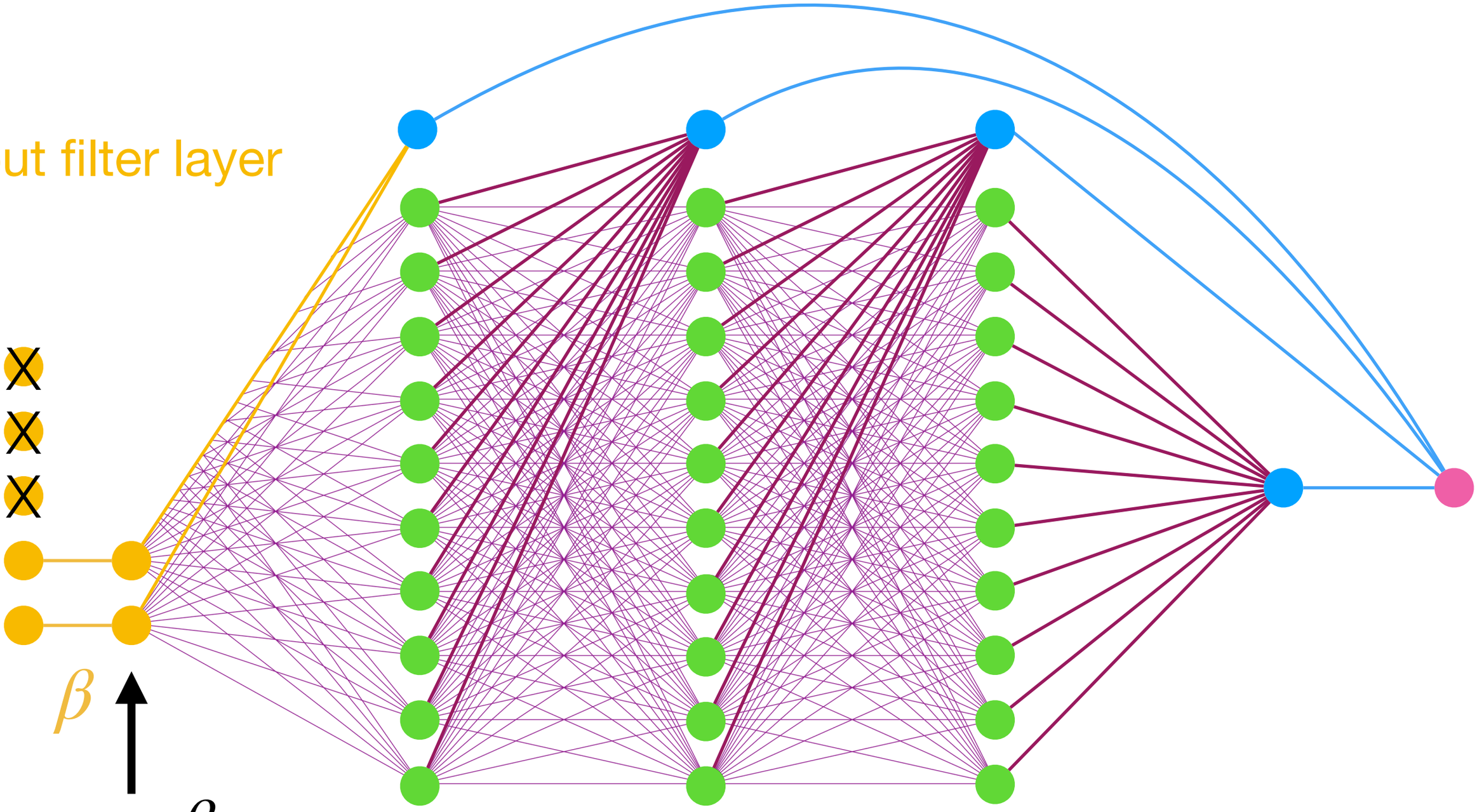


$$\min_{\beta,W,\alpha} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_{\beta,W,\alpha}(x_i) \right)^2 + \lambda_1 \|\beta\|_1 + \lambda_2 \|W\|_1$$



$$\min_{\beta,W,\alpha} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_{\beta,W,\alpha}(x_i) \right)^2 + \lambda_1 \|\beta\|_1 + \lambda_2 \|W\|_1$$

Input filter layer

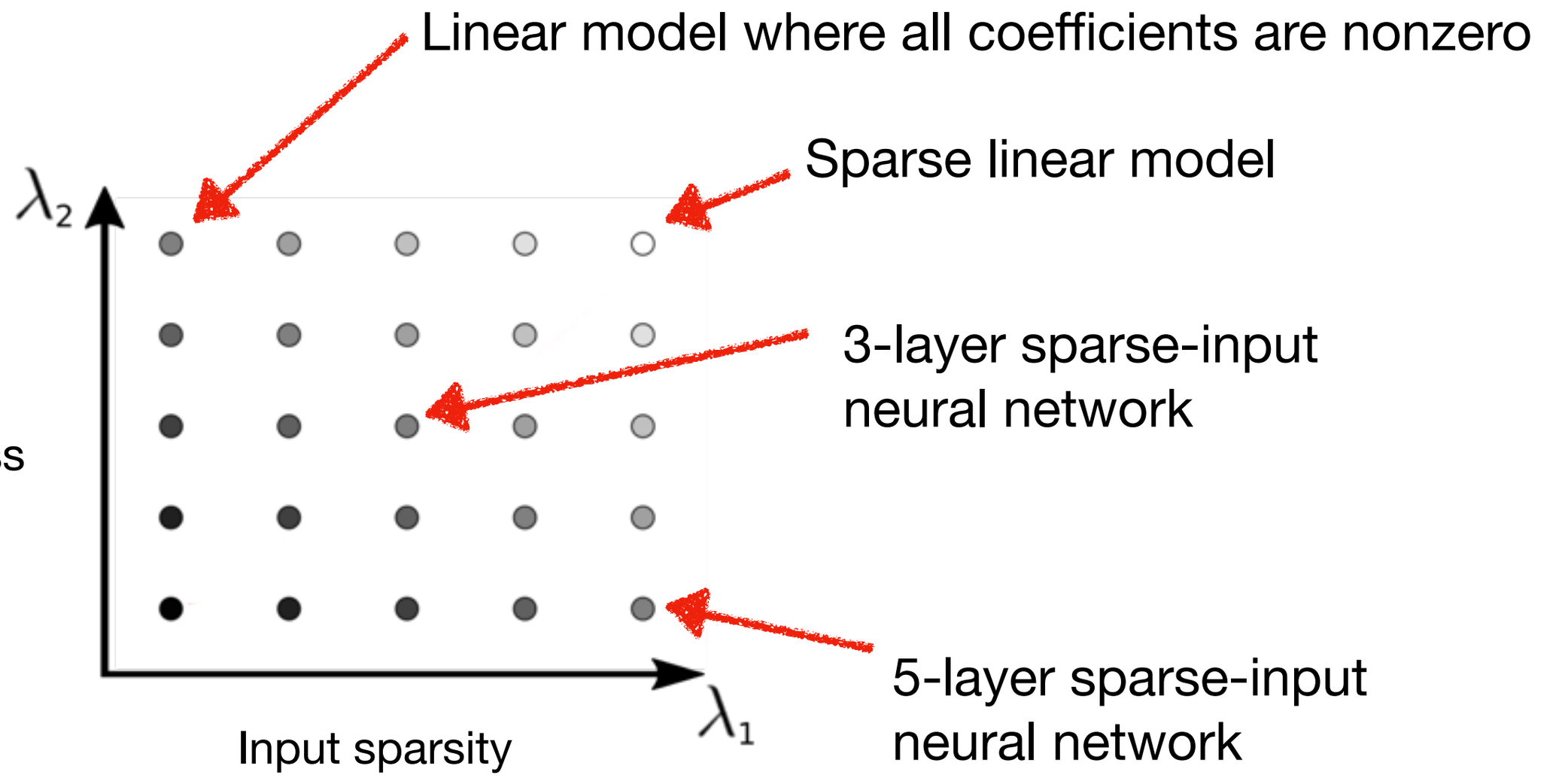


β

$$z_{1,j} = \beta_j x_j$$

for $j = 1,\ldots,d$

$$\min_{\beta,W,\alpha} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_{\beta,W,\alpha}(x_i) \right)^2 + \lambda_1 \|\beta\|_1 + \lambda_2 \|W\|_1$$



Variable selection

- Establishing variable selection guarantees is generally a difficult endeavor that requires strong assumptions, even for linear models.

- Variable selection for neural networks is even harder because the objective is non-convex with respect to the model parameters.
    - Under certain conditions, the total weight assigned to irrelevant input variables by SPINN with a single hidden layer scales at a rate of $O(\sqrt{\log p})$ (Feng and Simon 2017).

- Rather than focusing on variable selection, can we say something about ***variable screening*** under less stringent conditions?

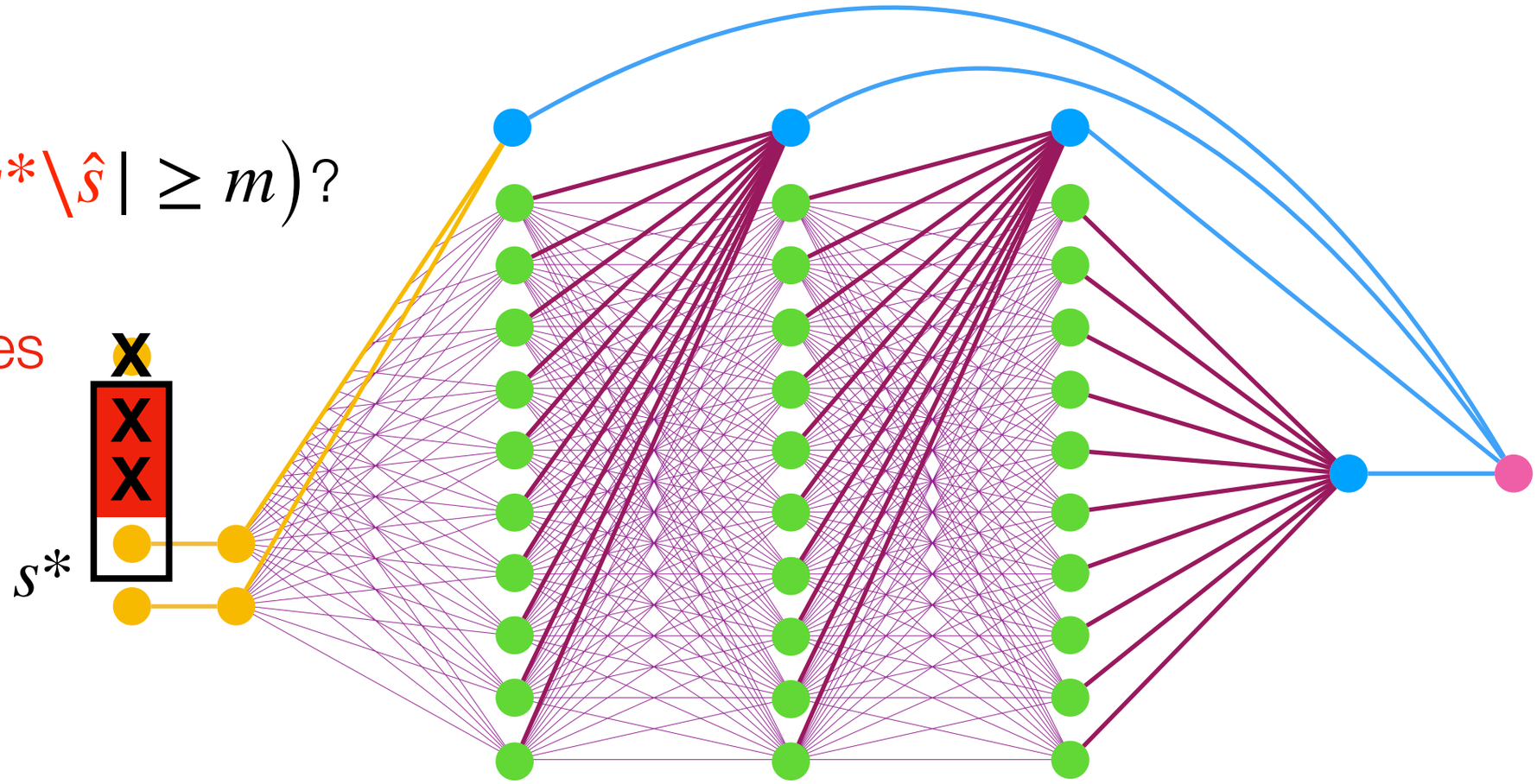
Can SIER-net be used for support screening?

- For true model f^*, suppose s^* is its true support, e.g. $E[Y|X=x] = f^*(x_1, \ldots, x_{|s^*|})$.

- Let $\hat{s}$ be the estimated support by SIER-net.

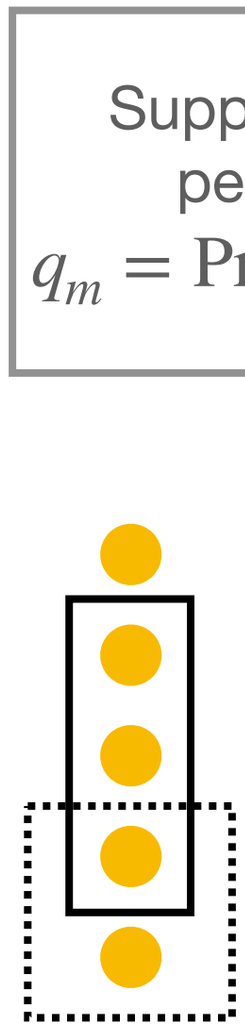
What is $\mathbf{Pr}\left(|s^* \backslash \hat{s}| \geq m\right)$?

False negatives
$= s^* \backslash \hat{s}$



Predictive value of the true support

Support screening performance:
$q_m = \Pr\left(|s^* \backslash \hat{s}| \geq m\right)$



Predictive value of variables *inside* s^* relative to variables *outside:*
$\gamma(m)$

True support $s^* = \{X_2, X_3, X_4\}$ explains 95% of the variance.

Candidate support $\{X_1, X_2\}$ can explain 92% of the variance.

Support screening: Results

Support screening performance:
$q_m = \Pr\left(|s^* \backslash \hat{s}| \geq m\right)$

Predictive value of variables *inside* s^* relative to variables *outside:*
$\gamma(m)$

- **Upper bound:** For sparse-input hierarchical networks, q_m is small if n is at least $n \gtrsim 1/\gamma^2(m)$.

➡Sparse-input hierarchical networks can perform support screening effectively *if the covariates are independent*.

- **Lower bound:** *Any* estimation procedure requires at least $n \gtrsim 1/\gamma(2m)$ for q_m to be small.

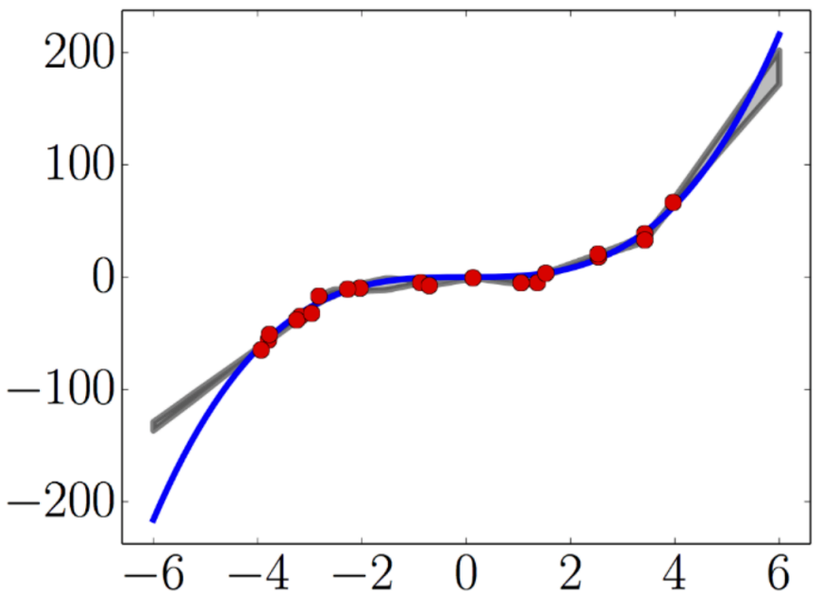
➡However, accurate support screening is unrealistic if the covariates are highly correlated.

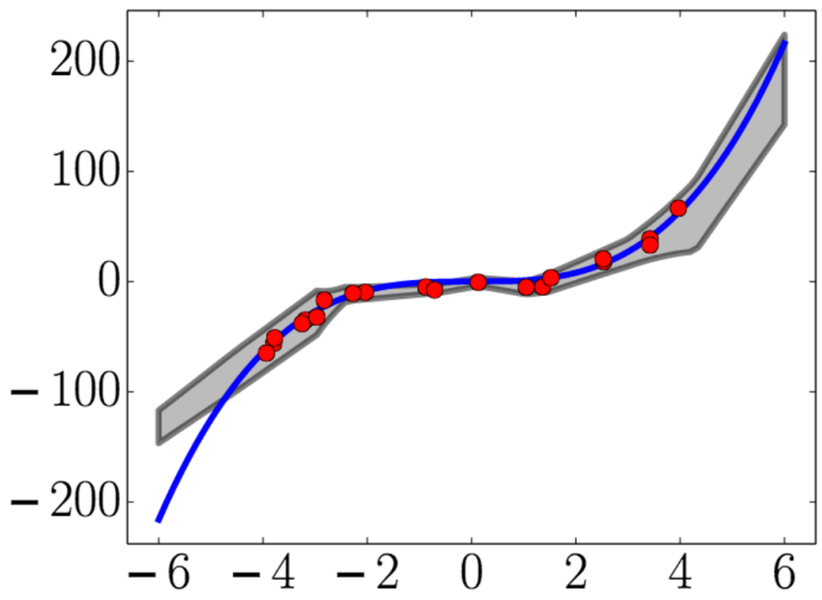
Ensembling!

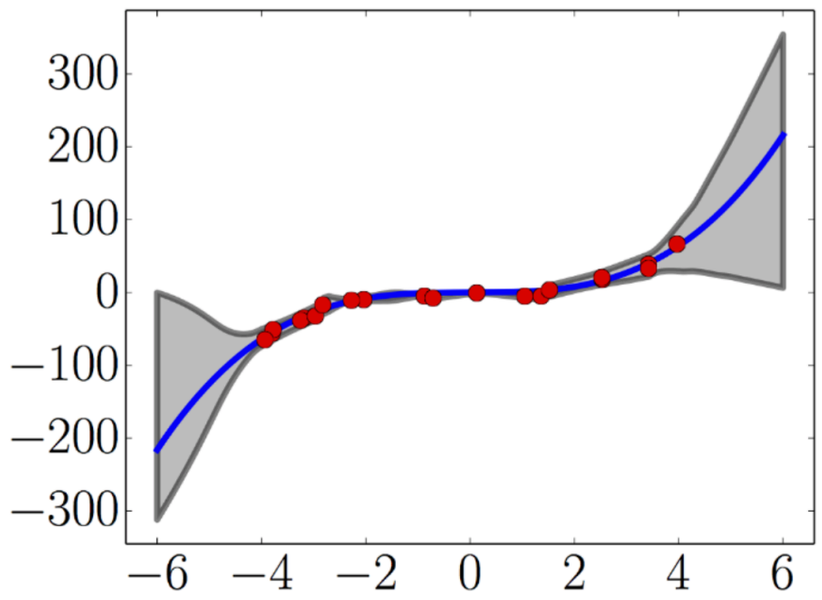
Recall the power of ensembling…

- Ensembling tends to improve performance for high-variance, low-bias methods, e.g. decision trees and neural networks (Lakshminarayanan 2017).

- Ensembling can be viewed as an approximation of Bayesian model averaging (BMA) (Wilson 2020, Pearce 2020).

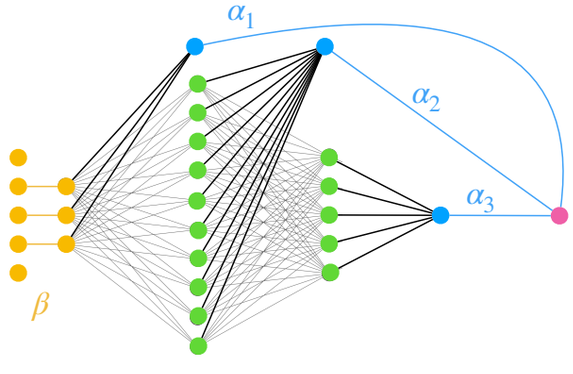




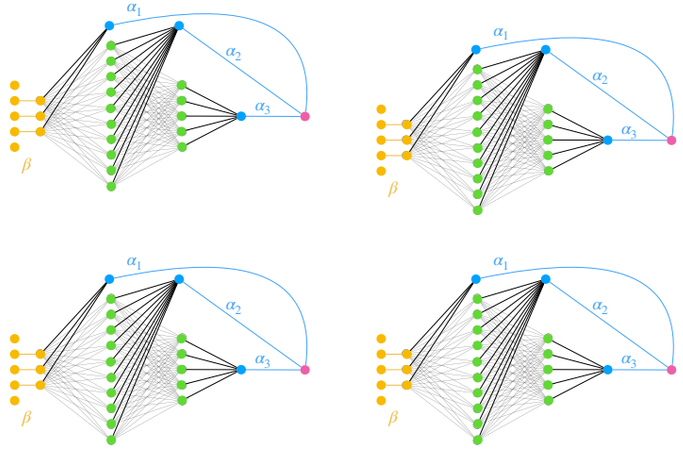


Ensemble by **A**veraging **S**parse-**I**nput hi**ER**archical networks (EASIER-net)

SIER-net



EASIER-net



- By viewing ensembling as an approximation of Bayesian model averaging, ensembling SIER-net gives us two major benefits with respect to variable selection:
  - The ensemble quantifies the uncertainty of the true support: the probability that a member in EASIER-net selects support $\tilde{s}$ is approximately equal to the posterior probability that the true support is $\tilde{s}$
  - Ensembling induces a “grouping effect,” where strongly correlated variables are selected at similar rates.

Simulation study: Variable screening

$$Y = X_1 * X_2 + \sin(X_3 + X_4) + \epsilon$$

$$X_i = \tilde{X}_i \text{ and } \quad X_{i+4} = \rho\tilde{X}_i + (1-\rho)\tilde{X}_{i+4} \text{ for } i = 1, ..., 4$$

ρ	Covariate index							
	1	2	3	4	5	6	7	8
0.00	1.000	1.000	1.000	1.000	0.000	0.000	0.000	0.000
0.50	1.000	1.000	1.000	1.000	0.000	0.000	0.000	0.000
0.80	0.980	1.000	1.000	1.000	0.020	0.000	0.000	0.000
0.90	0.860	0.890	0.870	0.990	0.160	0.120	0.280	0.010
0.95	0.630	0.730	0.540	0.940	0.380	0.300	0.570	0.270
1.00	0.550	0.480	0.470	0.570	0.460	0.540	0.600	0.640

Application: Gene expression data analysis

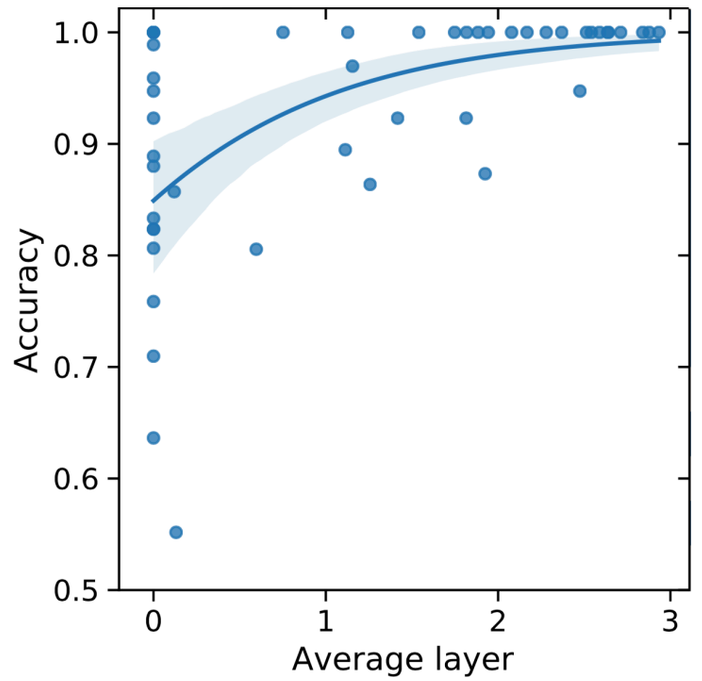
- We study the performance of SIER-net and EASIER-net on 46 gene expression datasets from the Curated Microarray Database (CuMiDa)
    - Number of categories: two to seven
    - Number of measured genes: ten to fifty thousand
    - Number of observations: ten to three hundred
- Initialize SIER-net and EASIER-net with five hidden layers and 100 hidden nodes per layer

Application: Gene expression data analysis

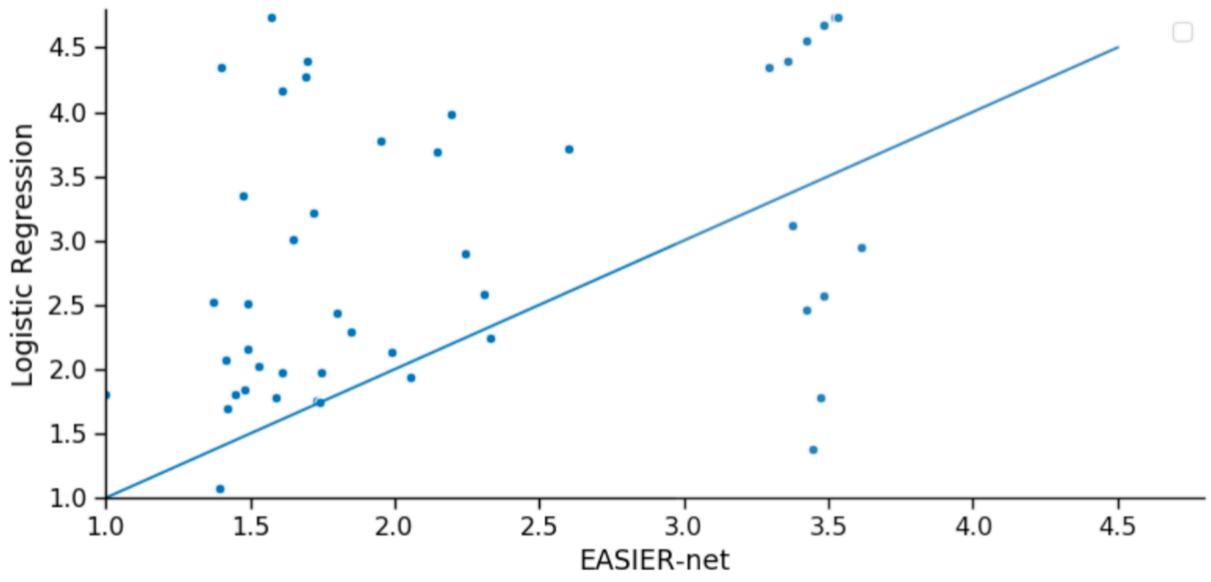
Method	Mean accuracy (SE)	# of times achieved top accuracy	Negative log likelihood
Logistic Regression	0.888 (0.022)	26	0.340 (0.040)
Random Forest	0.883 (0.022)	22	0.370 (0.037)
Gradient Boosted Trees	0.886 (0.021)	24	–
SIER-net	0.898 (0.020)	32	0.278 (0.048)
EASIER-net	**0.904 (0.019)**	**36**	**0.257 (0.043)**

Application: Gene expression data analysis

- How deep were the neural networks?



- How many variables were selected?



Conclusion

- Neural network can be useful for tabular, high-dimensional datasets… But to make them worth the computation time, you'll need some extra tricks!

- EASIER-nets rely on *two* carefully-designed lasso penalties.

- The ensembling step in EASIER-net helps quantify the uncertainty of the variable selection procedure.

References

- Jean Feng and Noah Simon. “Ensembled Sparse-Input Hierarchical Networks for High-Dimensional Datasets.” *Under review*. http://arxiv.org/abs/2005.04834.
  - Python code: https://github.com/jjfeng/easier_net
  - R package: https://github.com/jjfeng/easier_net_R
- Jean Feng and Noah Simon. “Sparse-Input Neural Networks for High-Dimensional Nonparametric Regression.” *ICML 2017 Workshop on Principled Approaches to Deep Learning.* http://arxiv.org/abs/1711.07592.

Thank you!

- Joint with work Noah Simon

- ASA SLDS and Jaime Speiser