Machine Learning I
80-629A

Apprentissage Automatique I
80-629

Neural Networks
— Week #5

This lecture

- **Neural Networks**

  A. Modeling

  B. Fitting

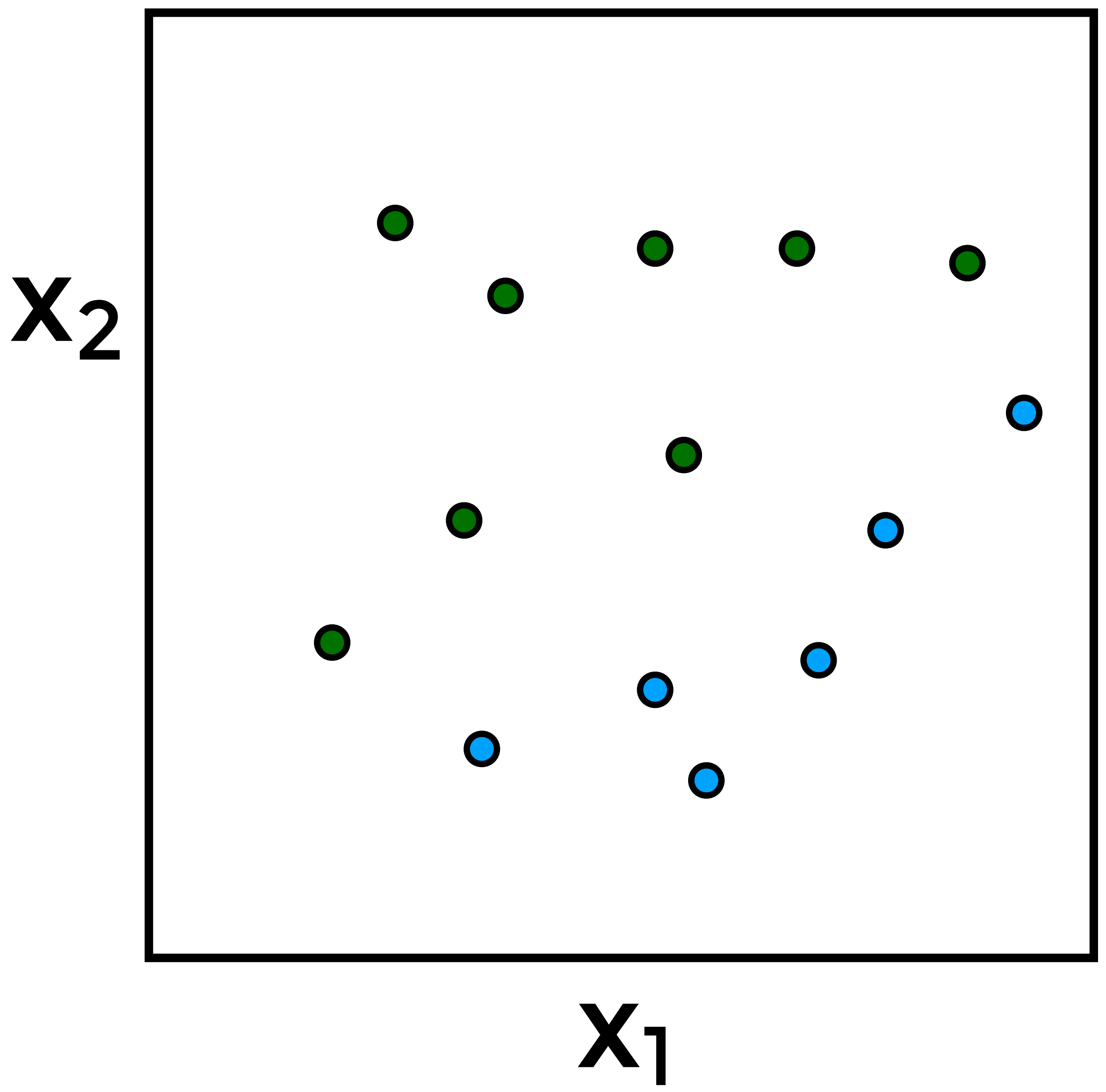
  C. Deep neural networks

  D. In practice

Some of today's material is (adapted) from Joelle Pineau's slides

From Linear Classification to Neural Networks

Recall Linear Classification



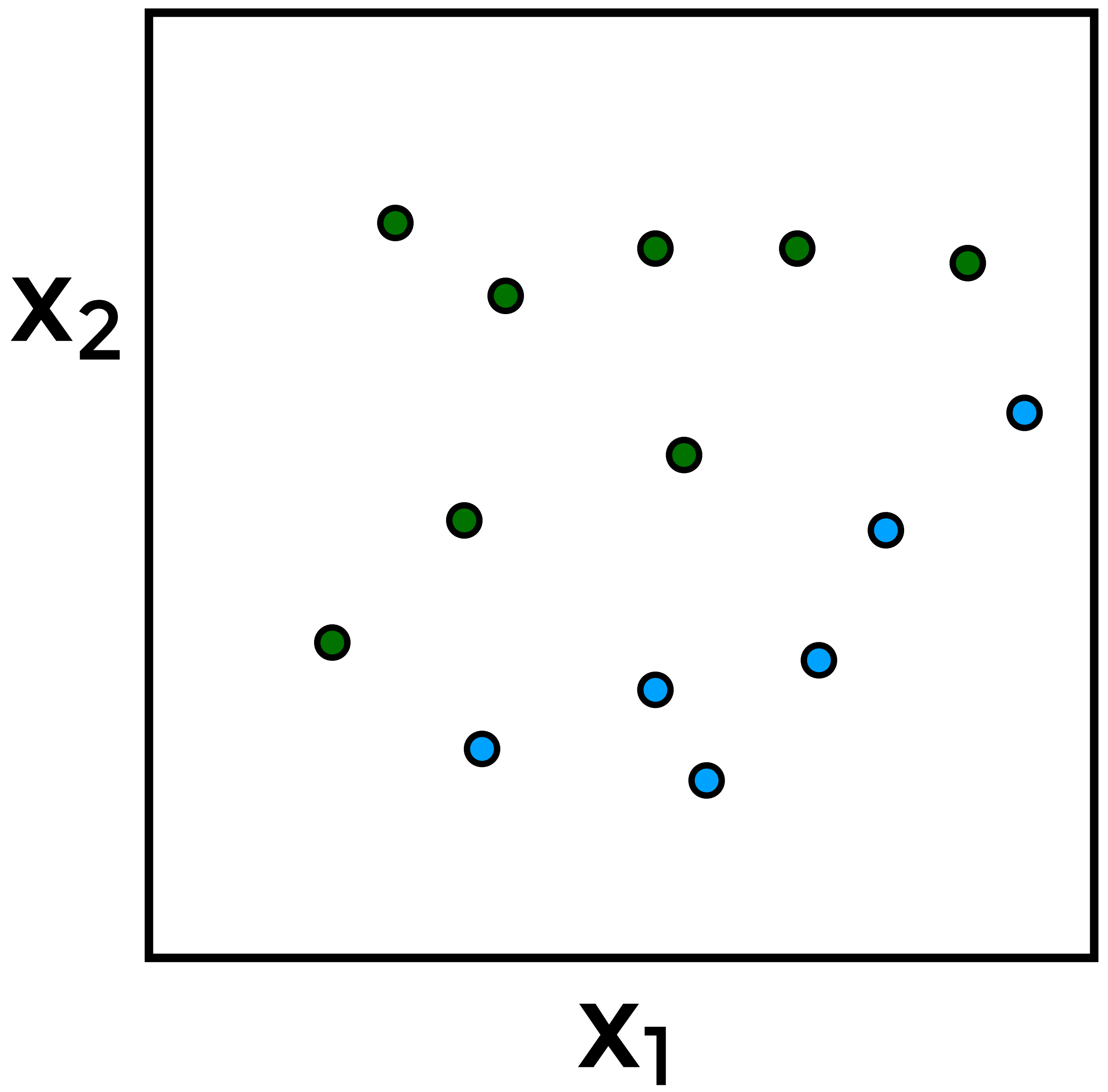
Recall Linear Classification

$$y(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0$$

Decision

$$(\mathbf{w}^\top \mathbf{x} + w_0) > 0 \implies \text{green}$$

$$(\mathbf{w}^\top \mathbf{x} + w_0) < 0 \implies \text{blue}$$



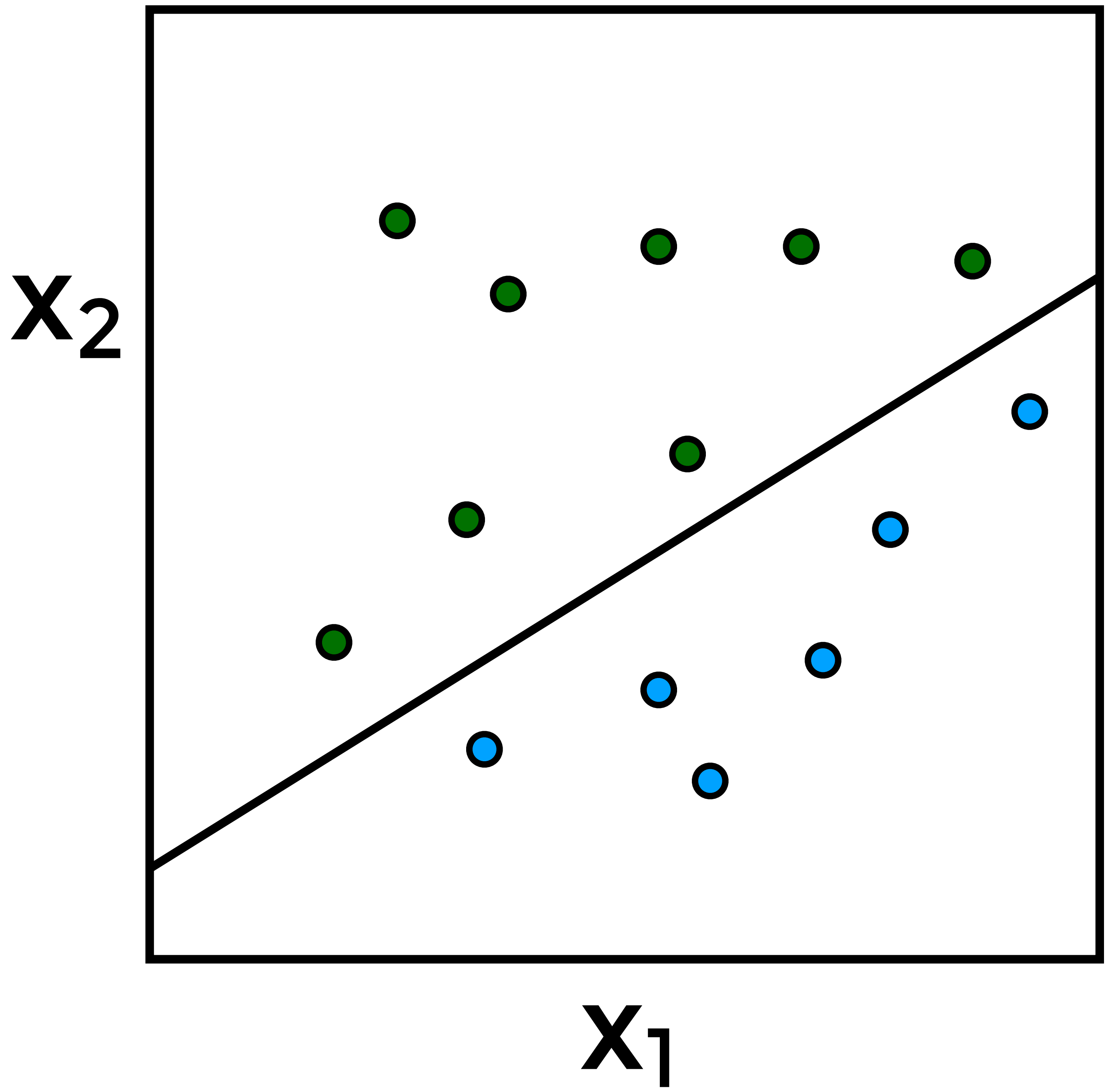
Recall Linear Classification

$$y(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0$$

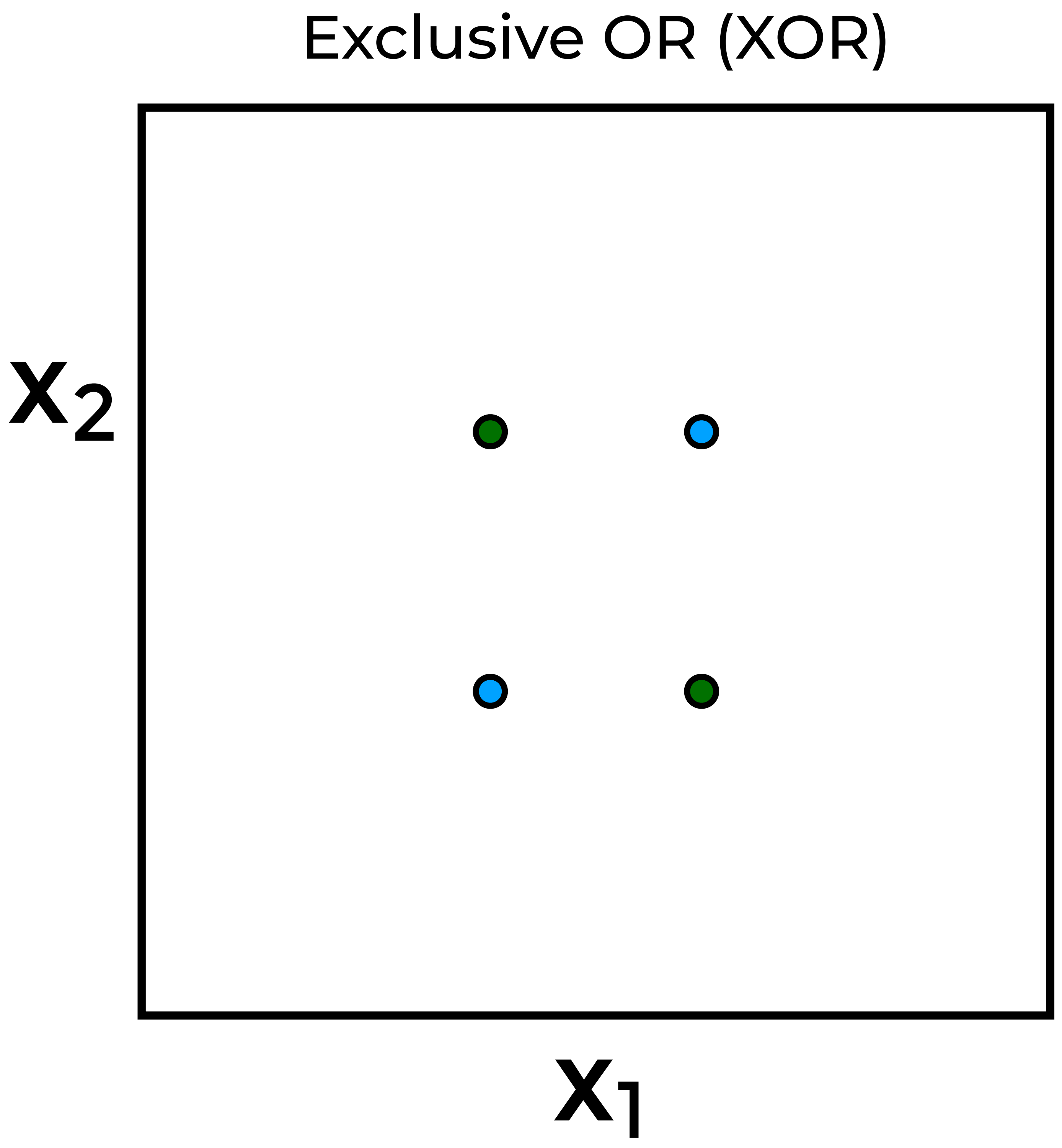
Decision

$$(\mathbf{w}^\top \mathbf{x} + w_0) > 0 \implies$$

$$(\mathbf{w}^\top \mathbf{x} + w_0) < 0 \implies$$

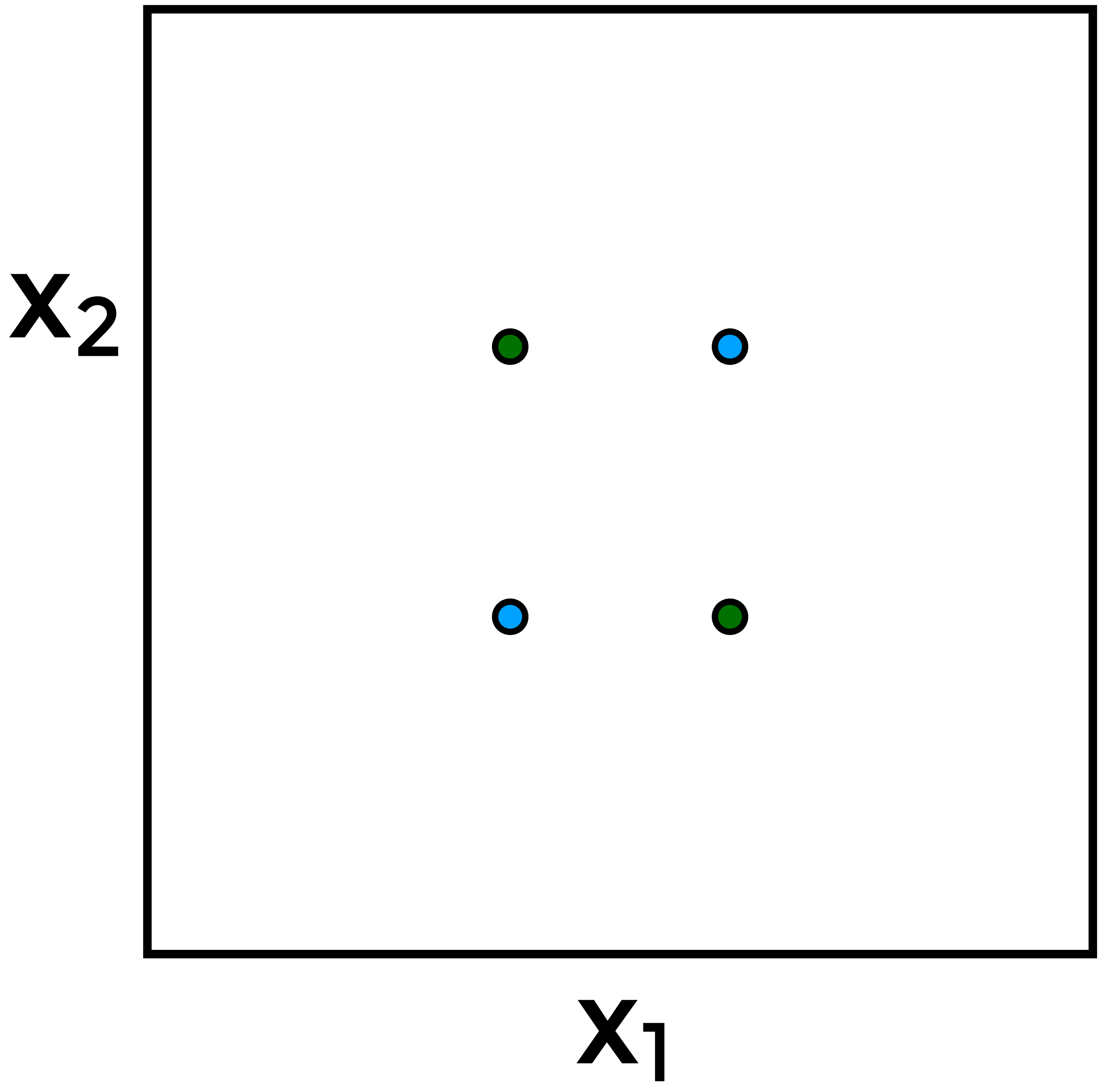


What if data is not linearly separable?



What if data is not linearly separable?

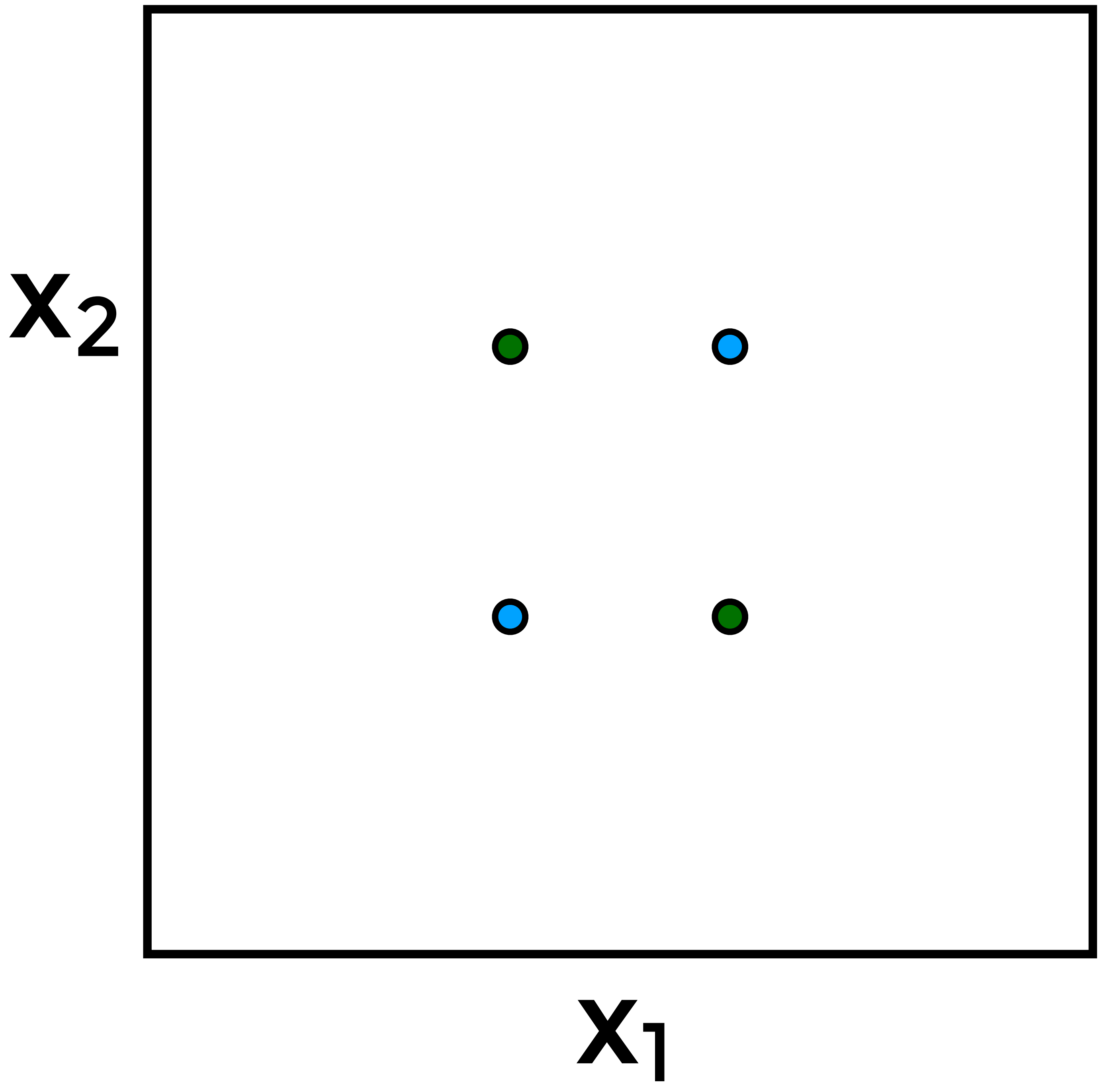
Exclusive OR (XOR)



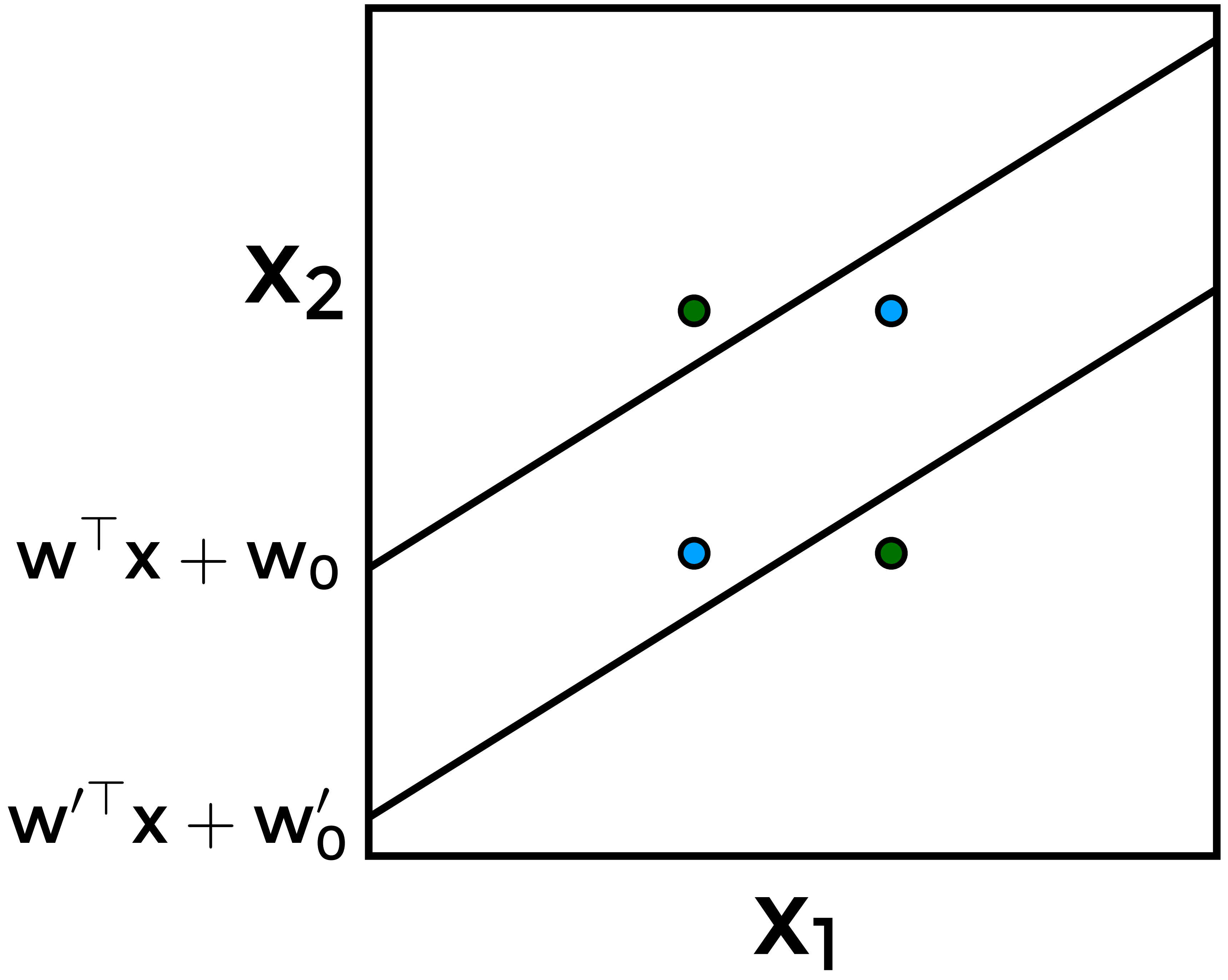
Use the joint decision of several linear classifier?

What if data is not linearly separable?

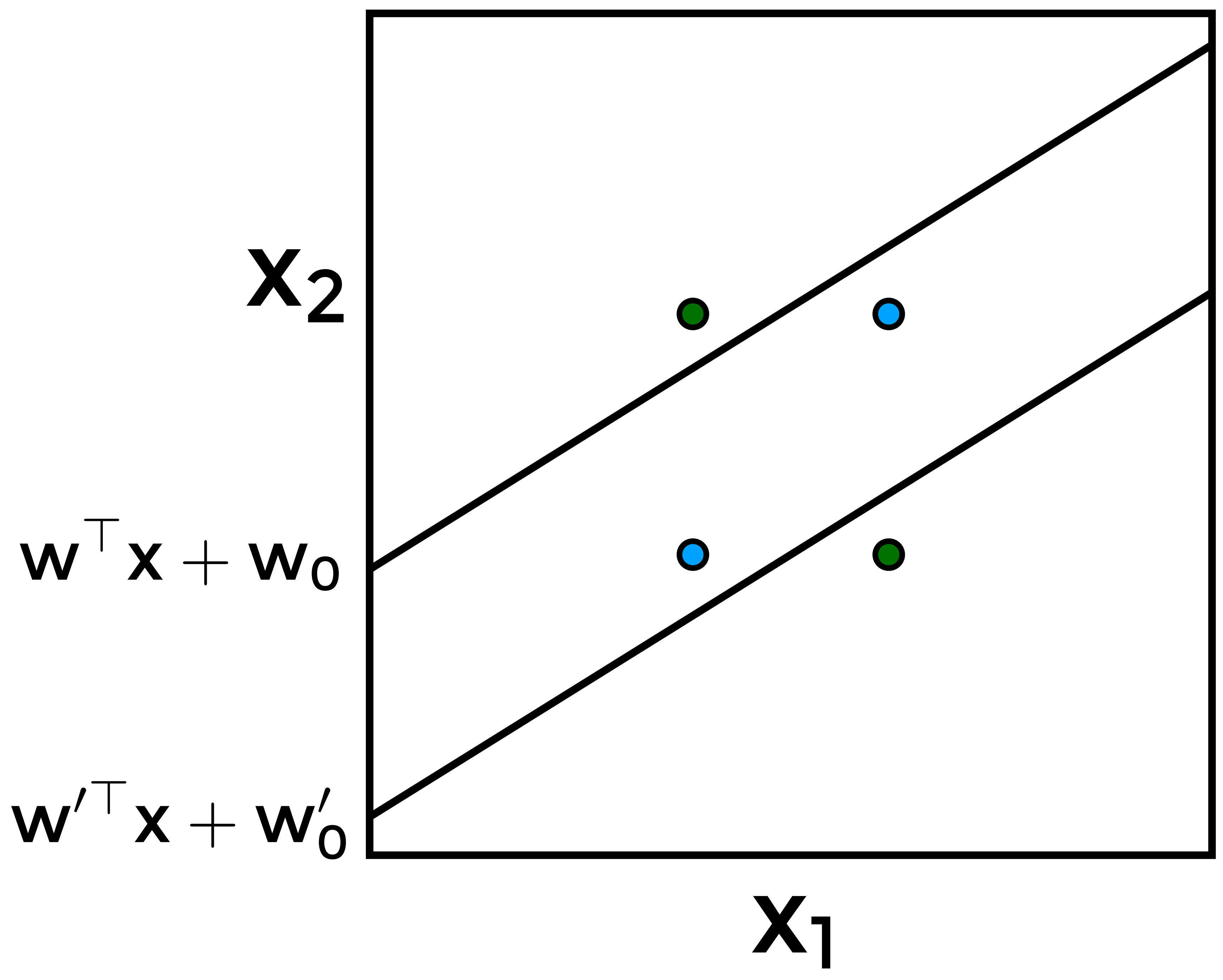
Exclusive OR (XOR)



Use the joint decision of several linear classifier?



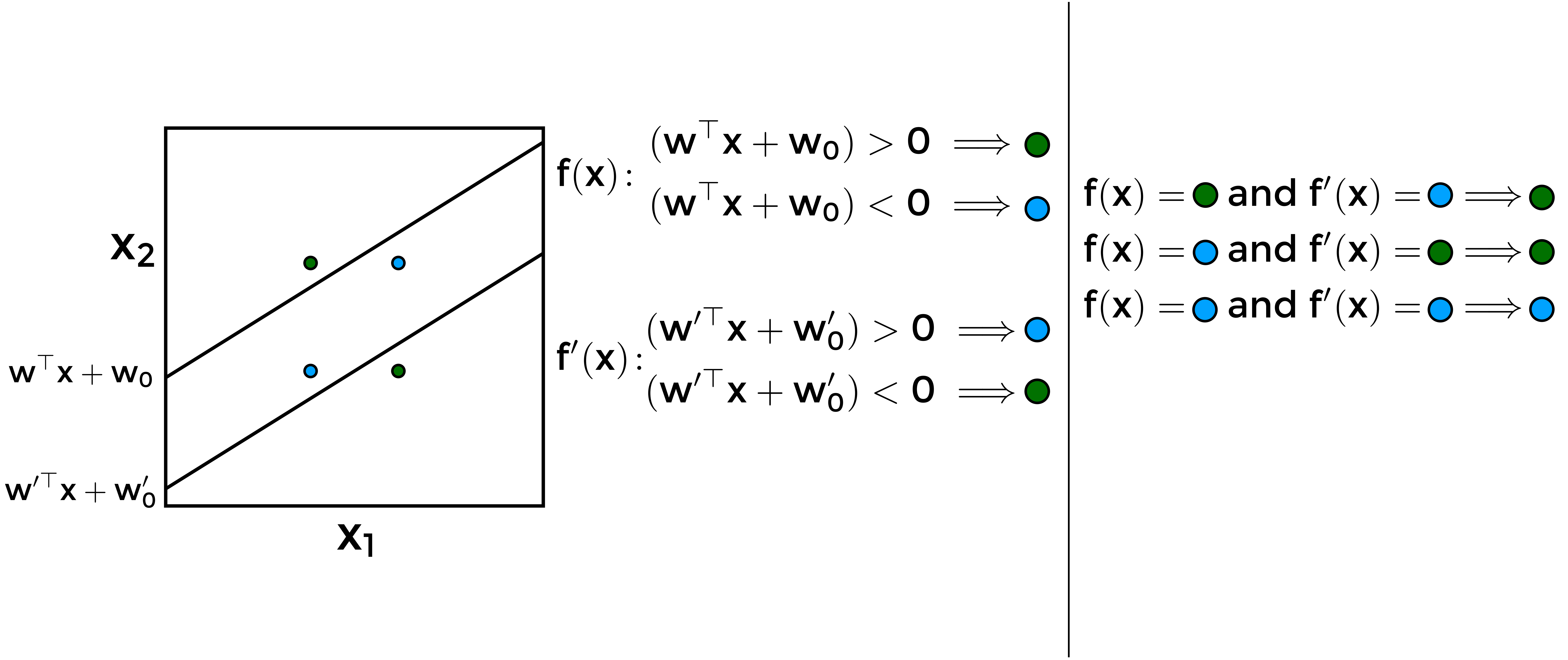
Combining models



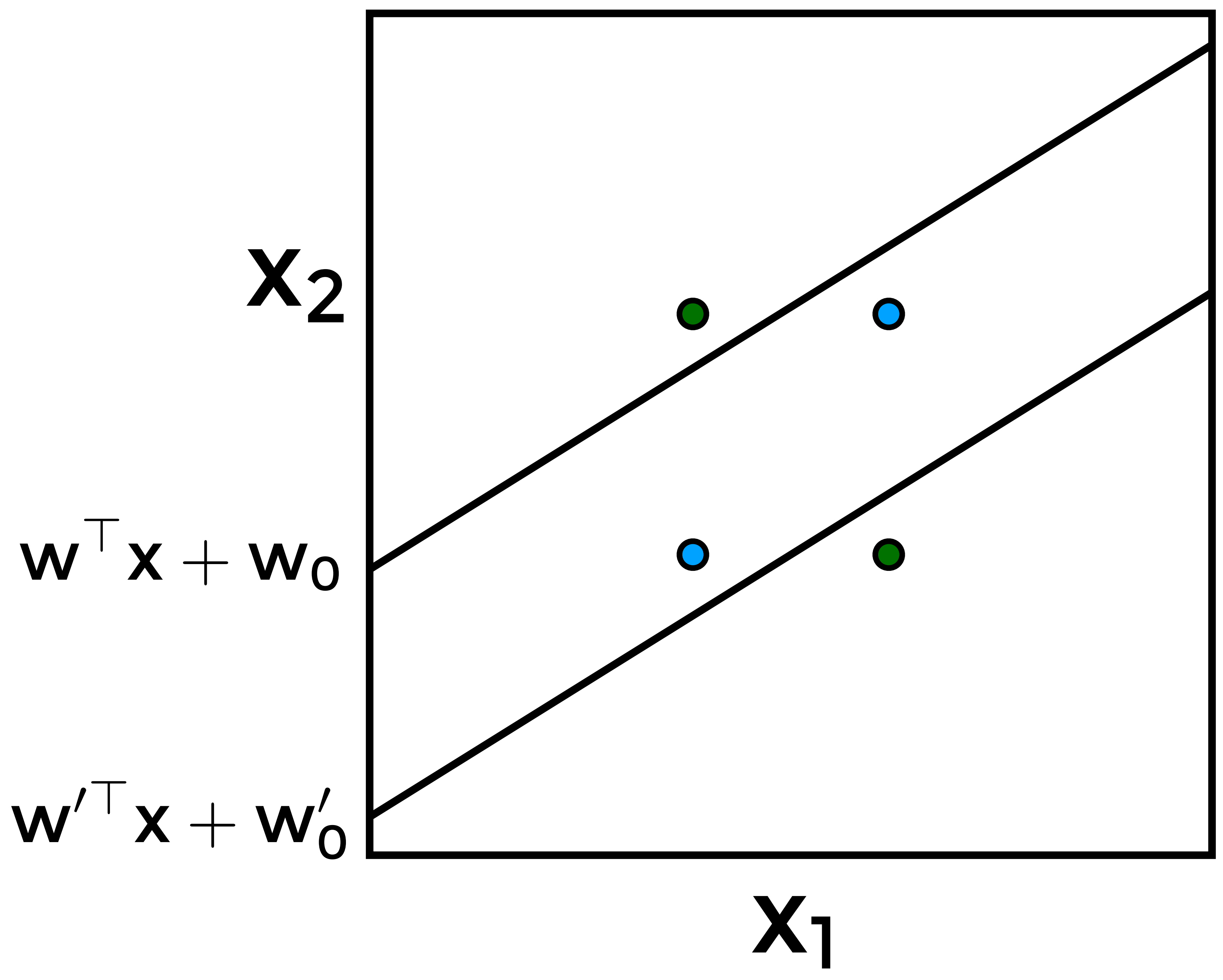
$$f(x): \begin{array}{l} (w^\top x + w_0) > 0 \implies \\ (w^\top x + w_0) < 0 \implies \end{array}$$

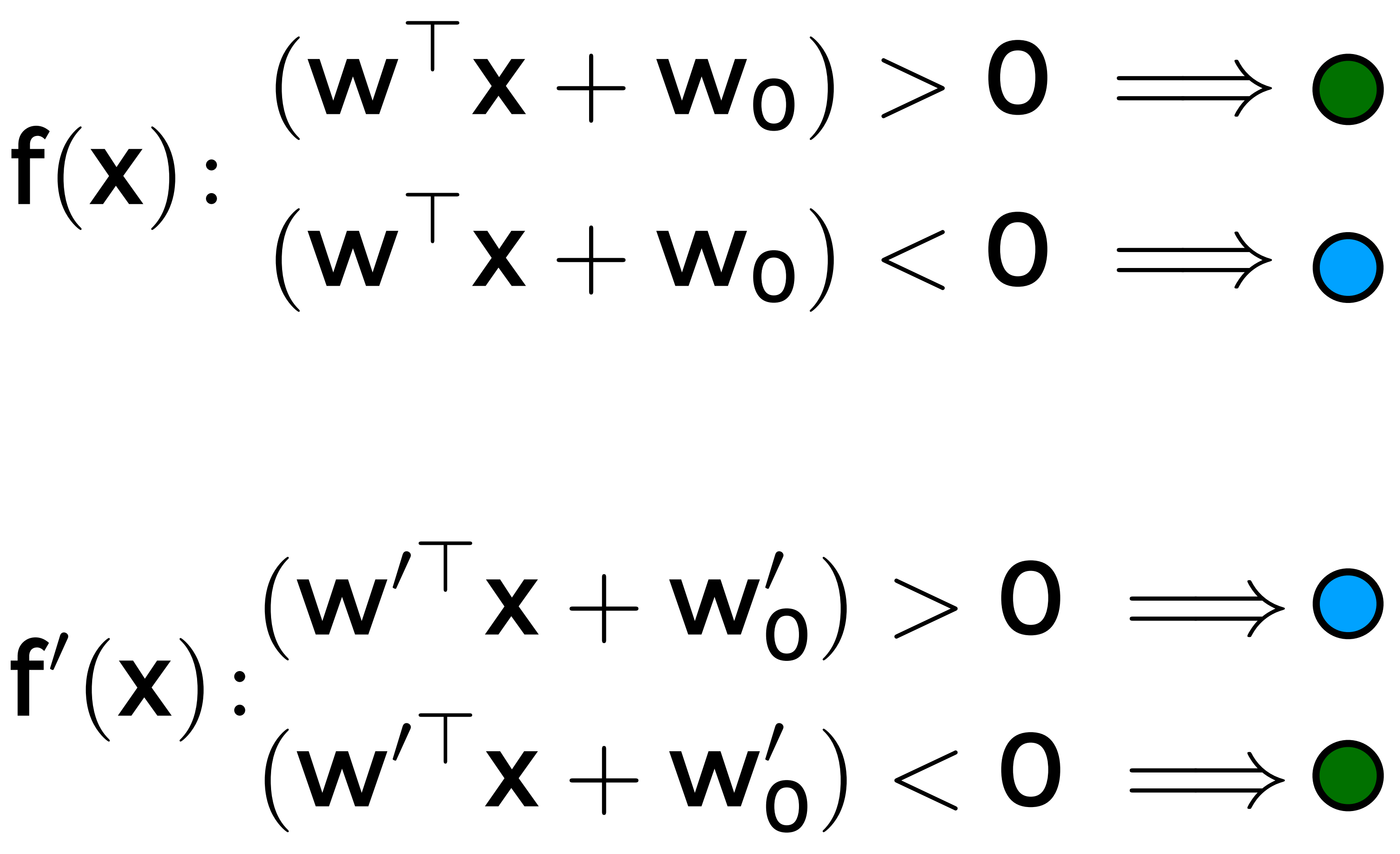
$$f'(x): \begin{array}{l} (w'^\top x + w'_0) > 0 \implies \\ (w'^\top x + w'_0) < 0 \implies \end{array}$$

Combining models

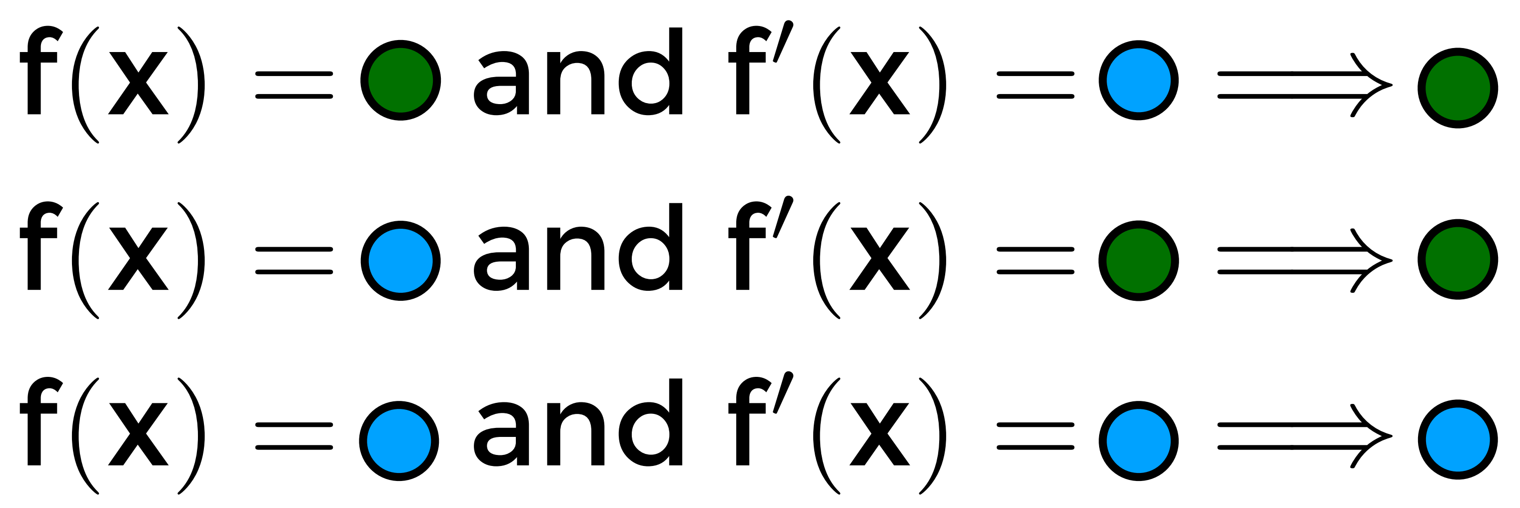


Combining models



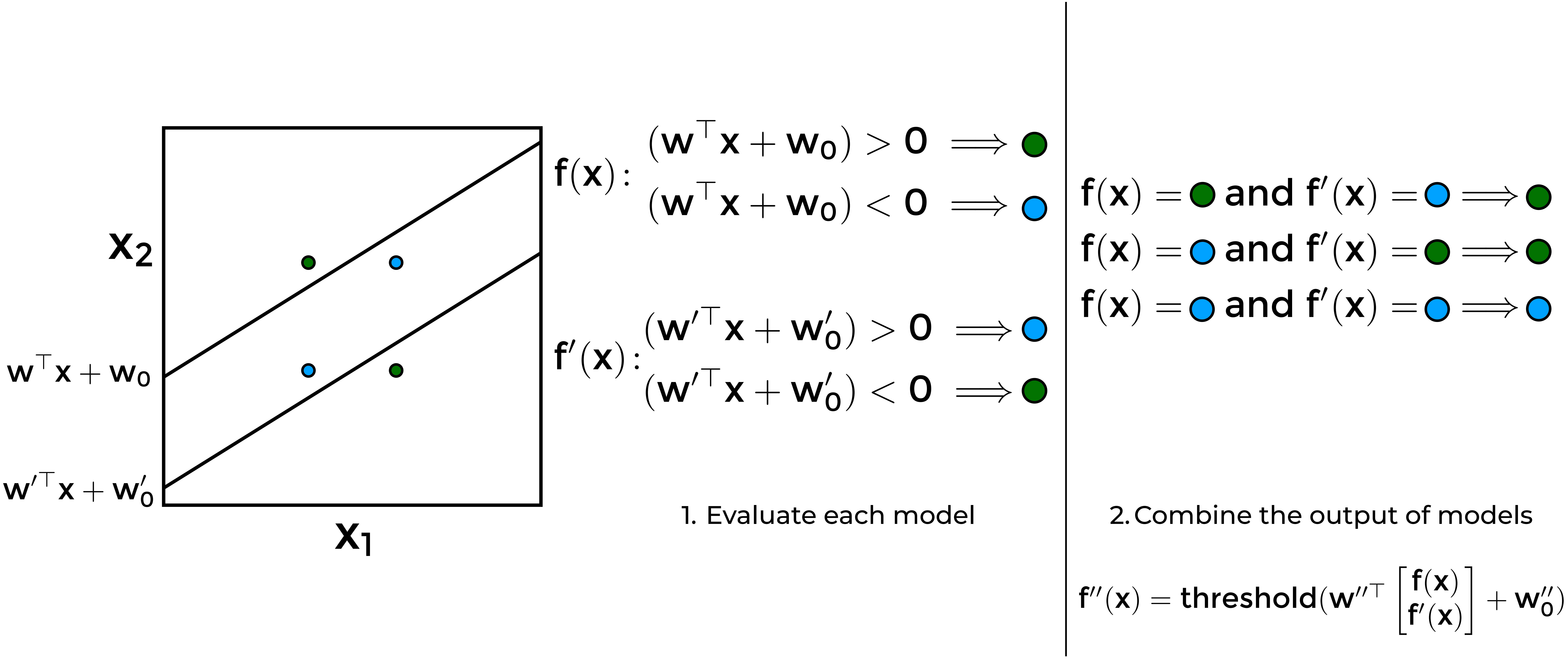


1. Evaluate each model



2. Combine the output of models

Combining models



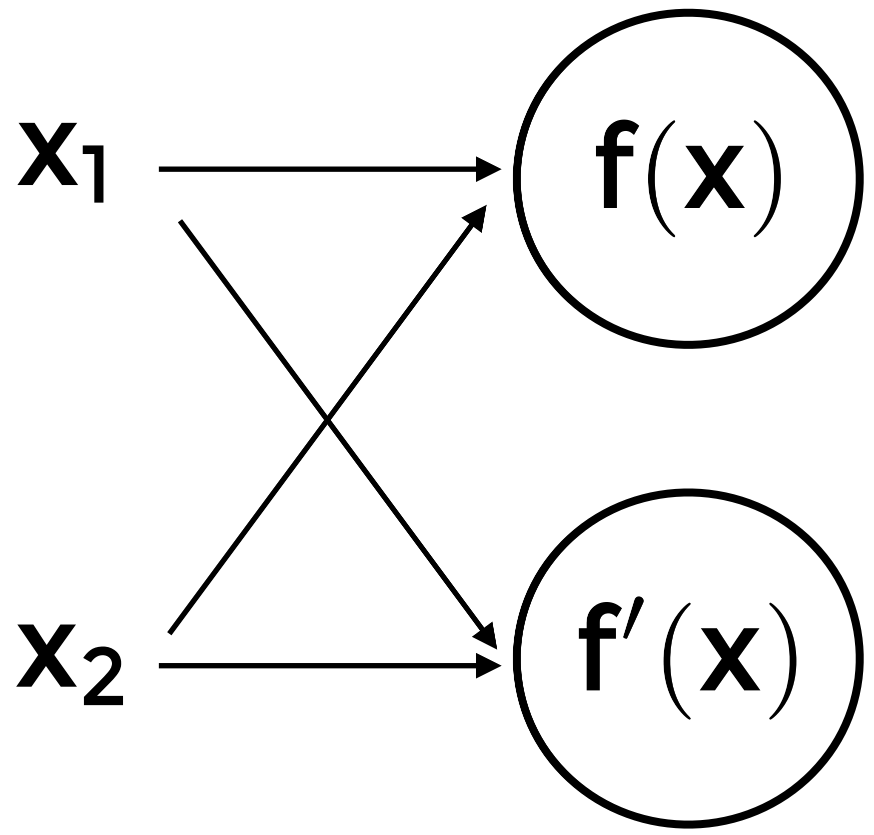
Combining model (graphical view)

Combining model (graphical view)

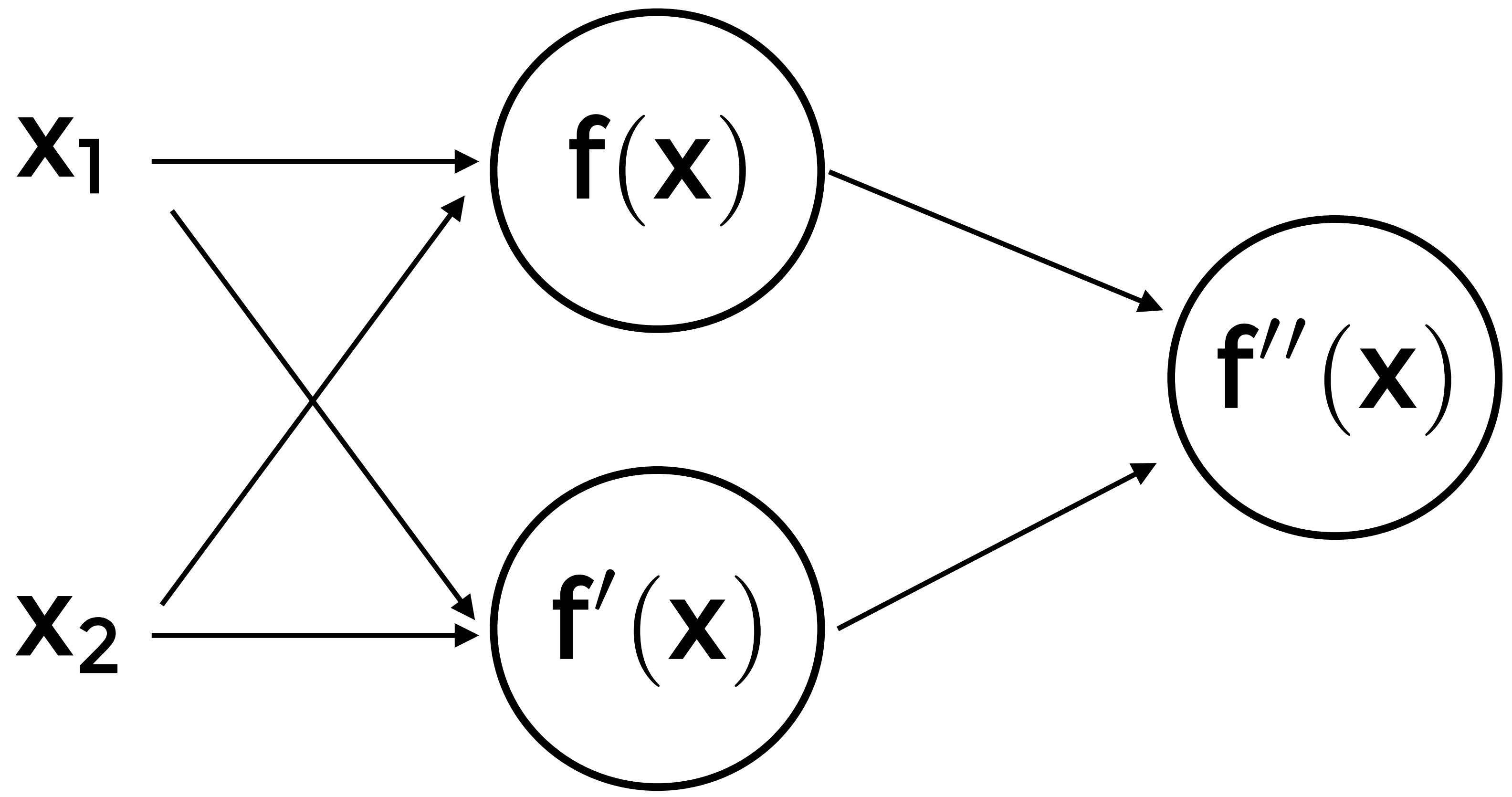
x_1

x_2

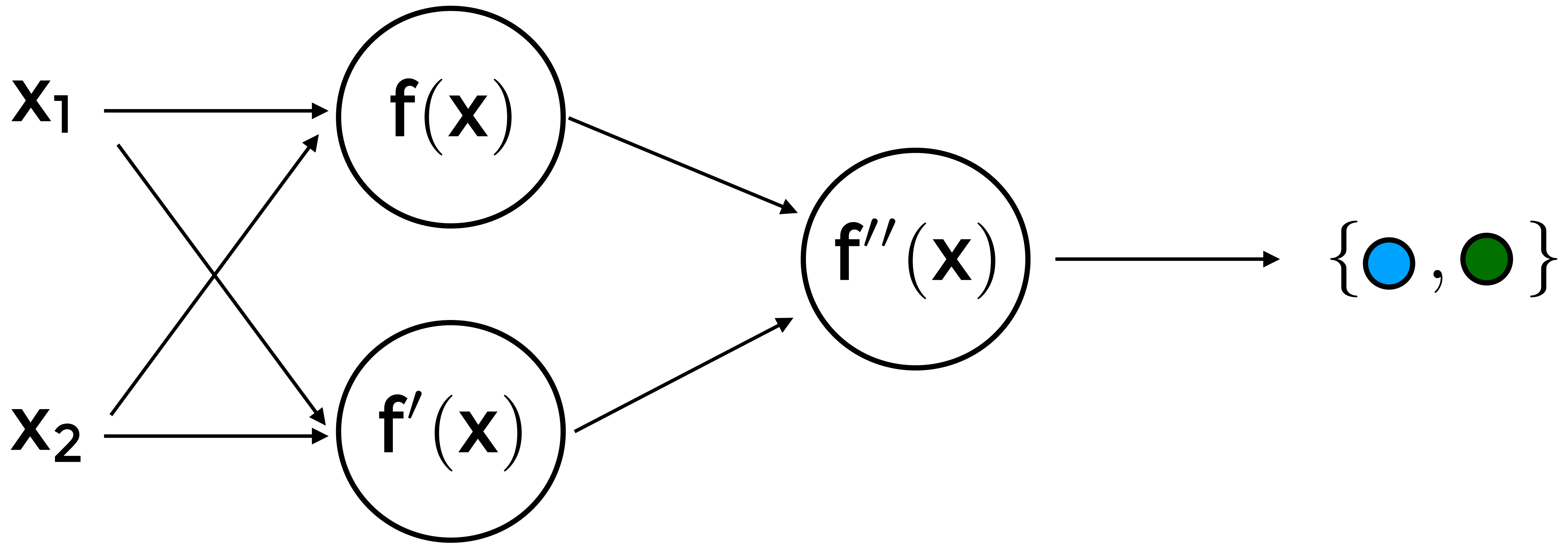
Combining model (graphical view)



Combining model (graphical view)

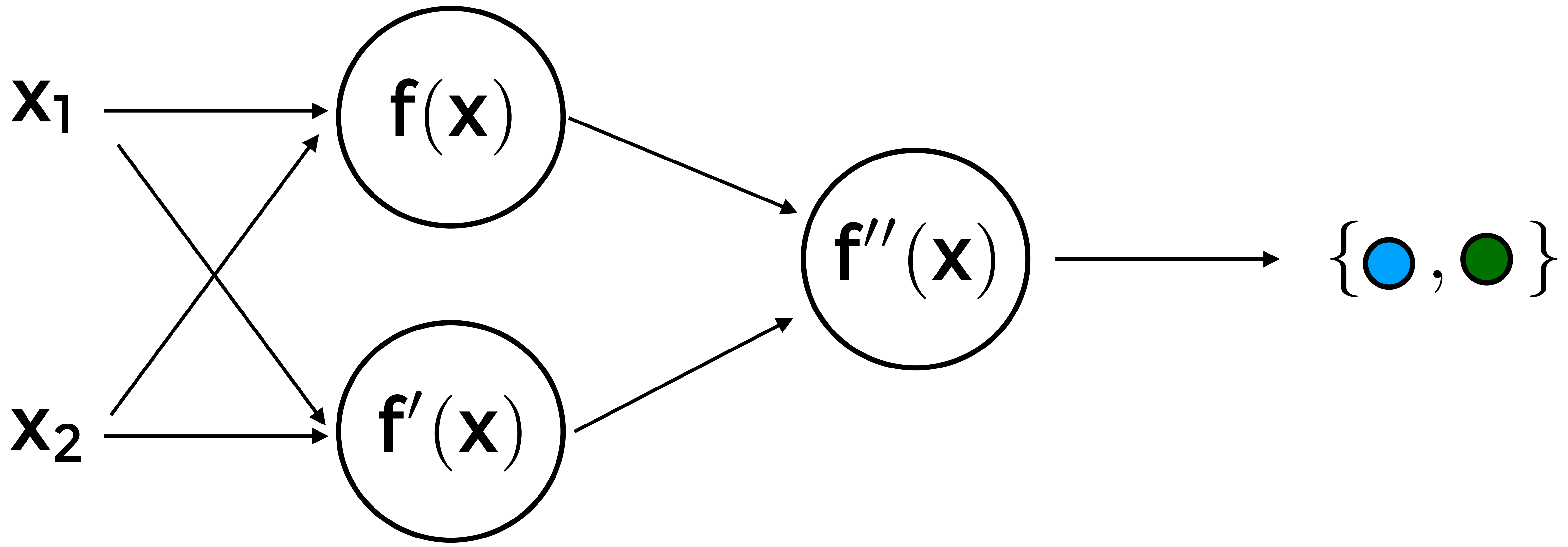


Combining model (graphical view)



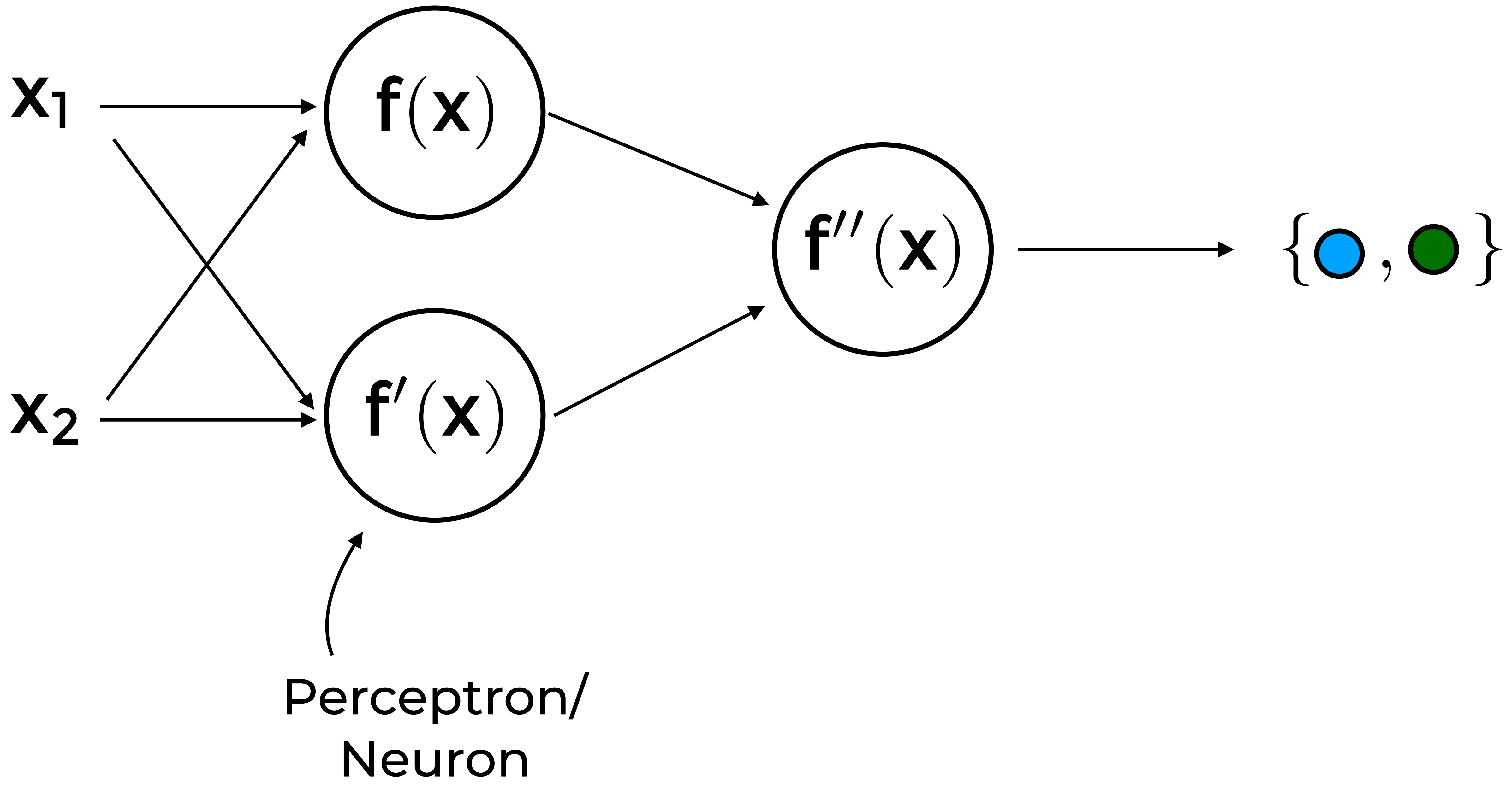
Combining model (graphical view)

Neural Network

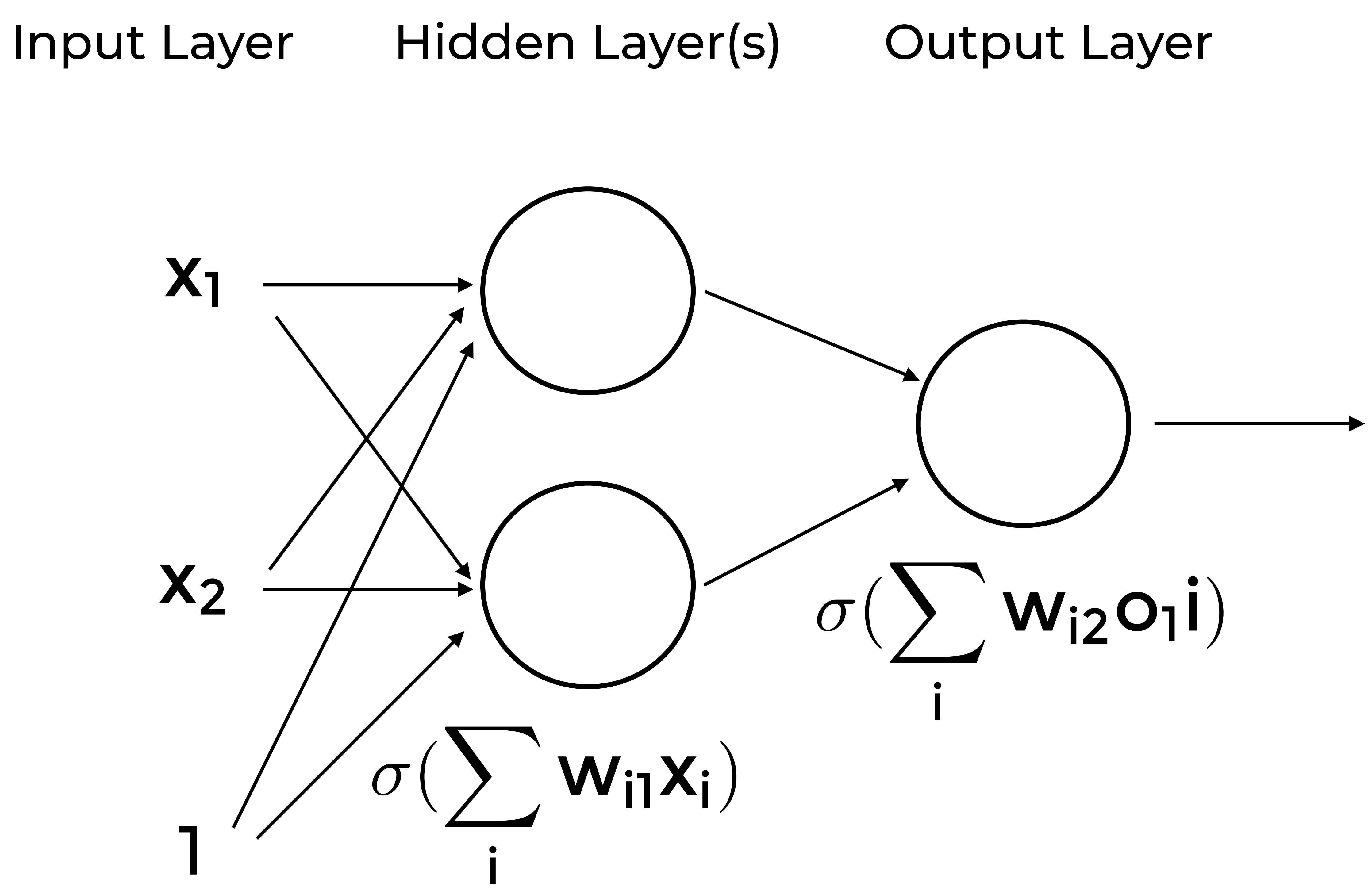


Combining model (graphical view)

Neural Network

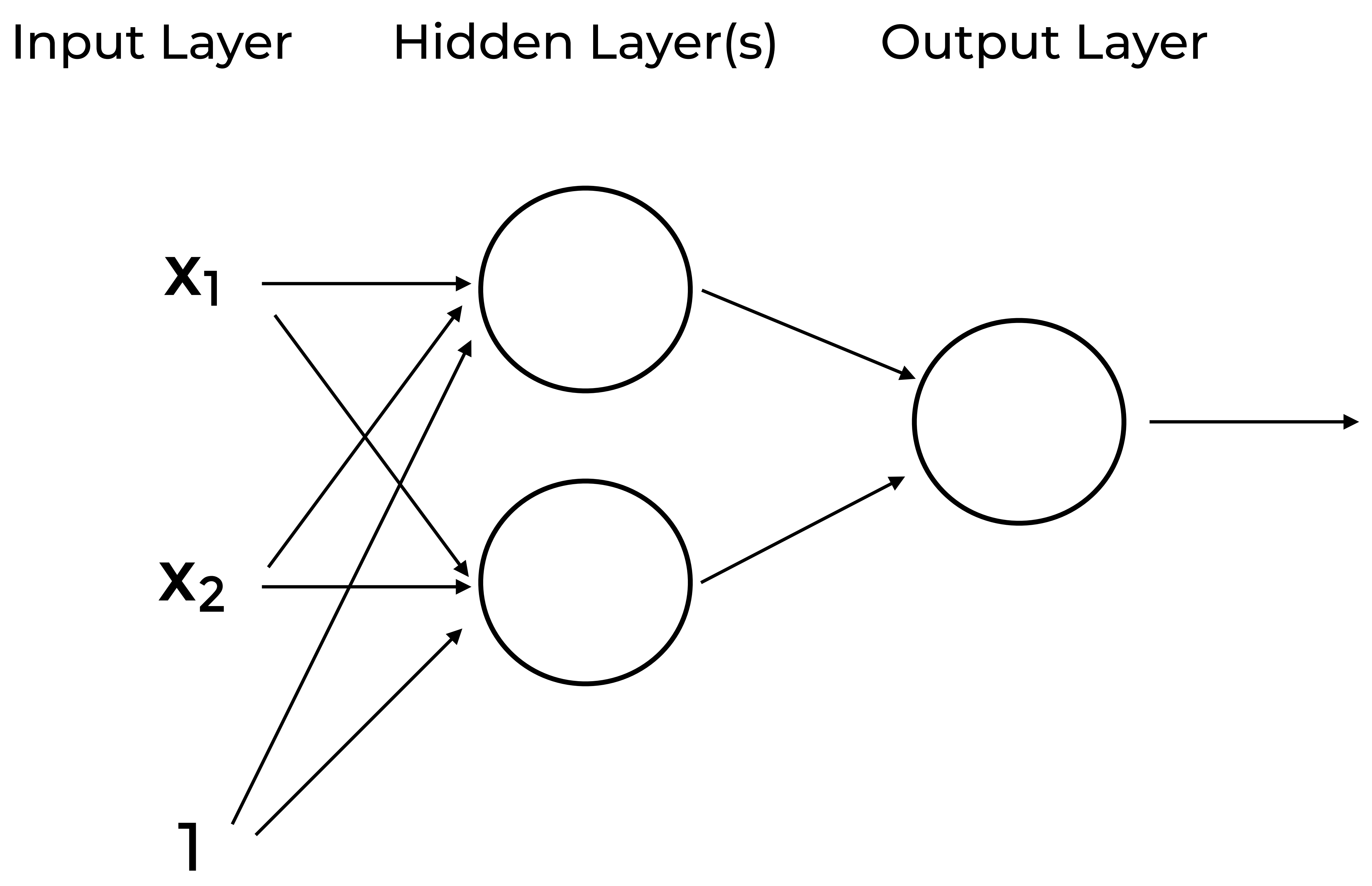


Feed-forward neural network



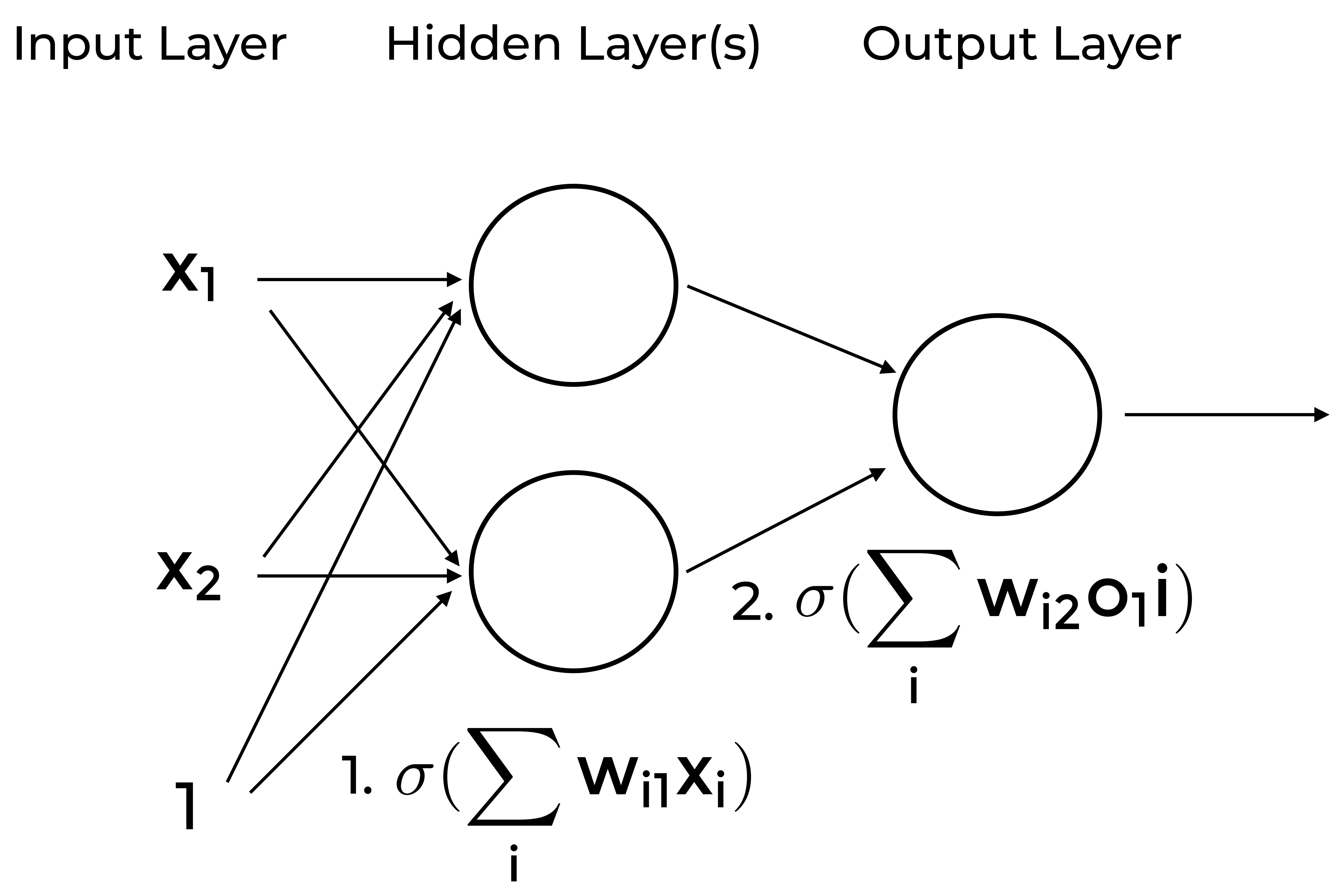
- Each arrow denotes a connection
  - A signal associated with a weight
- Each node is the weighted sum of its input followed by a non-linear activation
- Connections go left to right
  - No connections within a layer
  - No backward connections (recurrent)

Feed-forward neural network



1. **An input layer**
   - **Its size is the number of inputs + 1**
2. **One or more hidden layer(s)**
   - **Their size is a hyper-parameter**
3. **An output layer**
   - **Its size is the number of outputs**

Compute a prediction (forward pass)



Neural Networks

- Flexible model class
  - Highly-non linear models
  - Good for regression/classification/density estimation
- Models behind “Deep Learning”
  - Historical aspects

Learning the Parameters of a Neural Network

Fitting a neural network

Fitting a neural network

How do we estimate the model's parameters?

Fitting a neural network

How do we estimate the model's parameters?

- **No-closed form solution**

Fitting a neural network

How do we estimate the model's parameters?

- No-closed form solution
- Gradient-based optimization

Fitting a neural network

How do we estimate the model's parameters?

- No-closed form solution
- Gradient-based optimization
  - Threshold functions are not differentiable

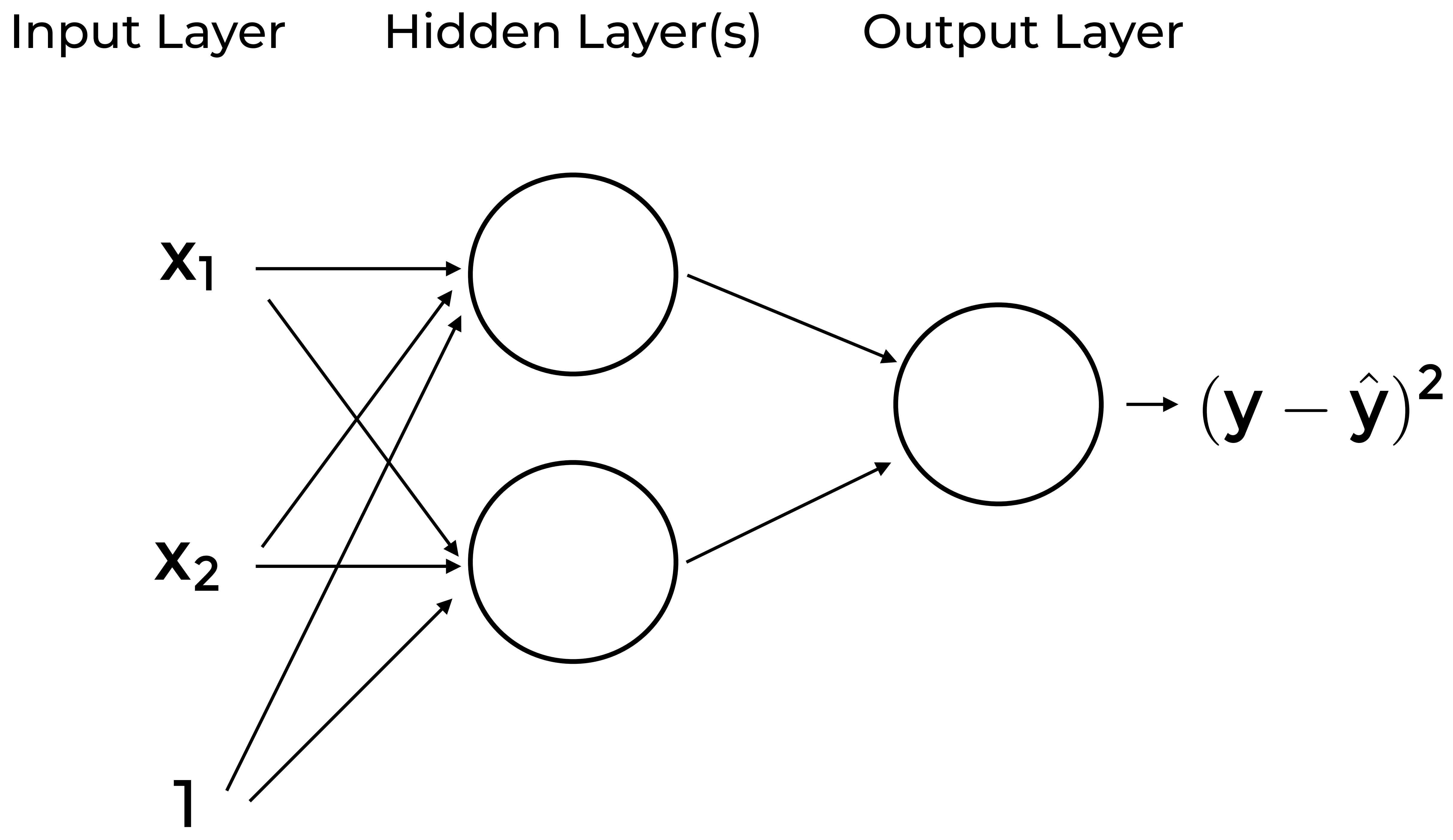
Fitting a neural network

How do we estimate the model's parameters?

- No-closed form solution
- Gradient-based optimization
  - Threshold functions are not differentiable
    - Replace by sigmoid (inverse logit). A soft threshold.

$$\text{sigmoid}(\mathbf{a}) := \left(\frac{1}{1 + \exp(-\mathbf{a})}\right)$$

Fit the parameters (w) (backward pass)



- **Derive a gradient wrt the parameters (w)**

$$\frac{\partial (y-\hat{y})^2}{\partial w_j} = \frac{\partial (y - f(\sum_i w_i o_i))^2}{\partial w_j}$$
$$= \frac{\partial (y - f(\sum_{i'} w_{i'} f(\sum_j w_j x_j)_{i'})^2}{\partial w_j}$$

- **The back-propagation starts from the output node(s) and heads toward the input(s)**

- **In practice, the order of the computation is important**

Gradient descent

- No closed-form formula
- Repeat the following steps (for t=0,1,2,... until convergence):
  1. Calculate a gradient $\nabla \mathbf{w}_{ij}^{t}$
  2. Apply the update $\mathbf{w}_{ij}^{t+1} = \mathbf{w}_{ij}^{t} - \alpha \nabla \mathbf{w}_{ij}^{t}$
- Stochastic gradient descent
  - One example at a time
- Batch gradient descent
  - All examples at a time

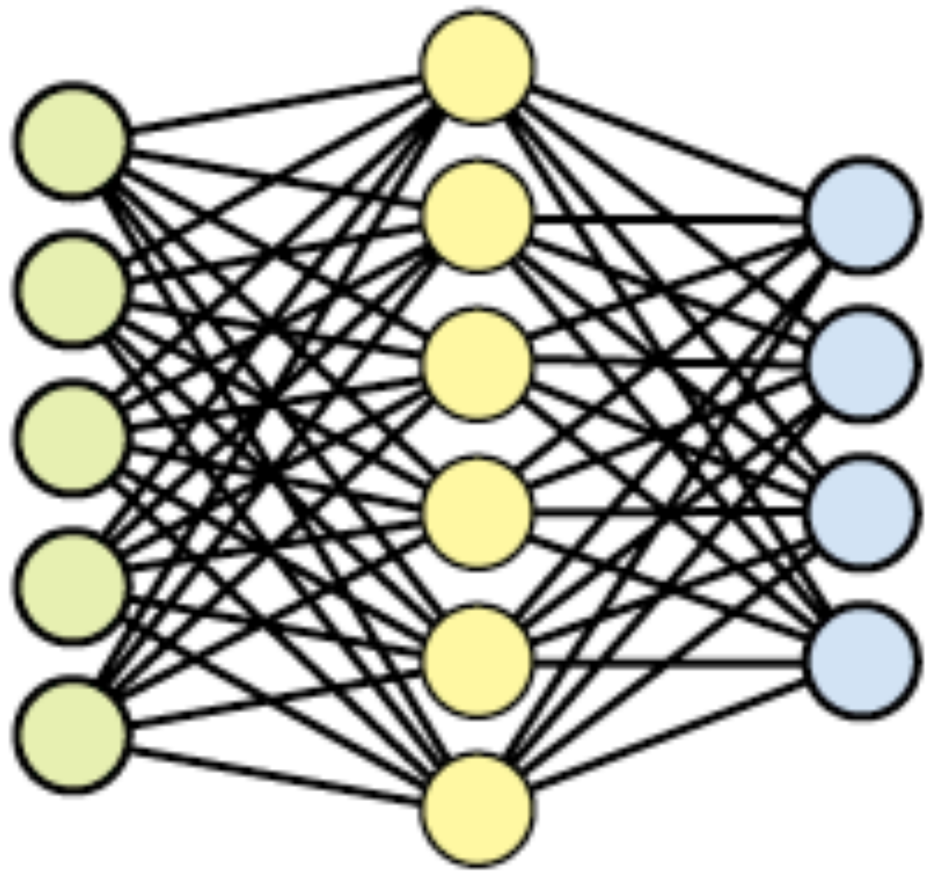
What can an MLP learn?

1. A single unit (neuron)
   - Linear classifier + non-linear output
2. A network with a single hidden layer
   - Any continuous function (but may require exponentially many hidden units as a function of the inputs)
3. A network with two (or more) hidden layers
   - Any function can be approximated with arbitrary accuracy.

The Importance of Representations

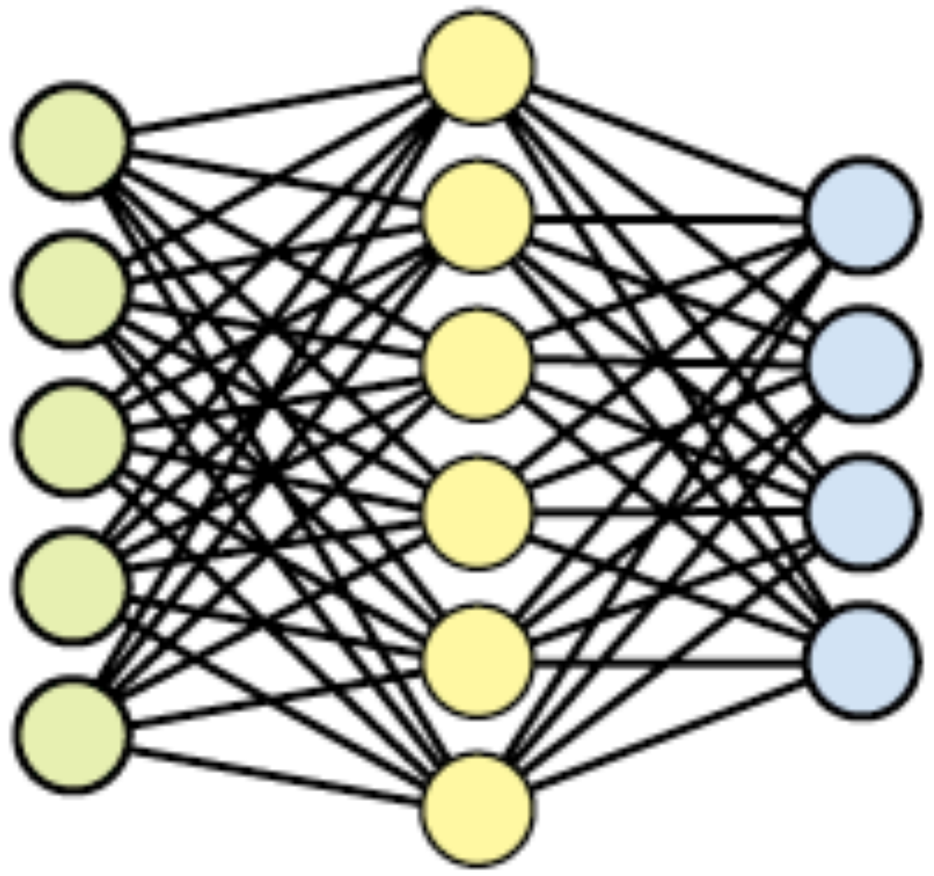
From Neural Networks to **Deep Neural Networks**

A neural Network

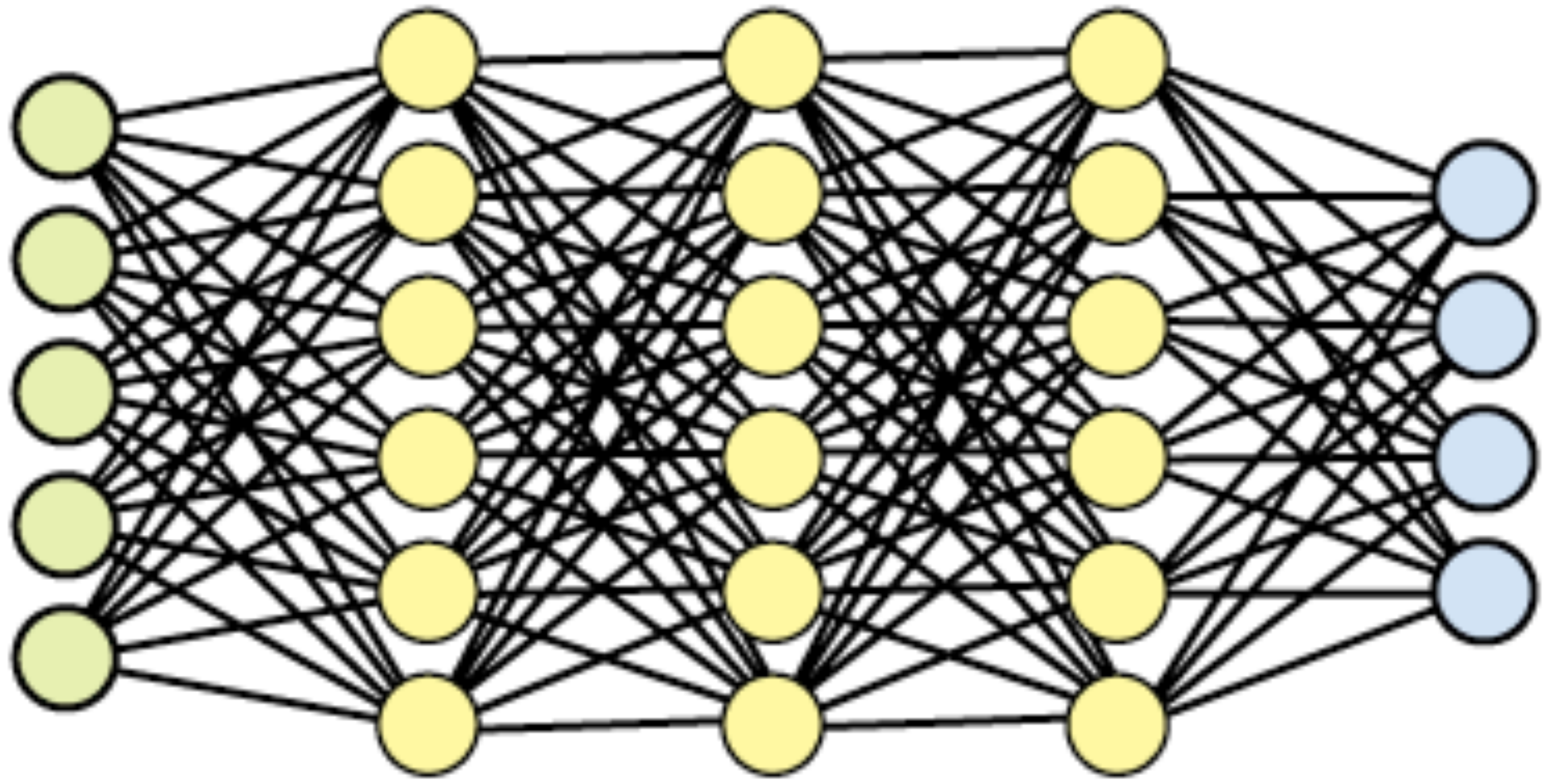


From Neural Networks to Deep Neural Networks

A neural Network

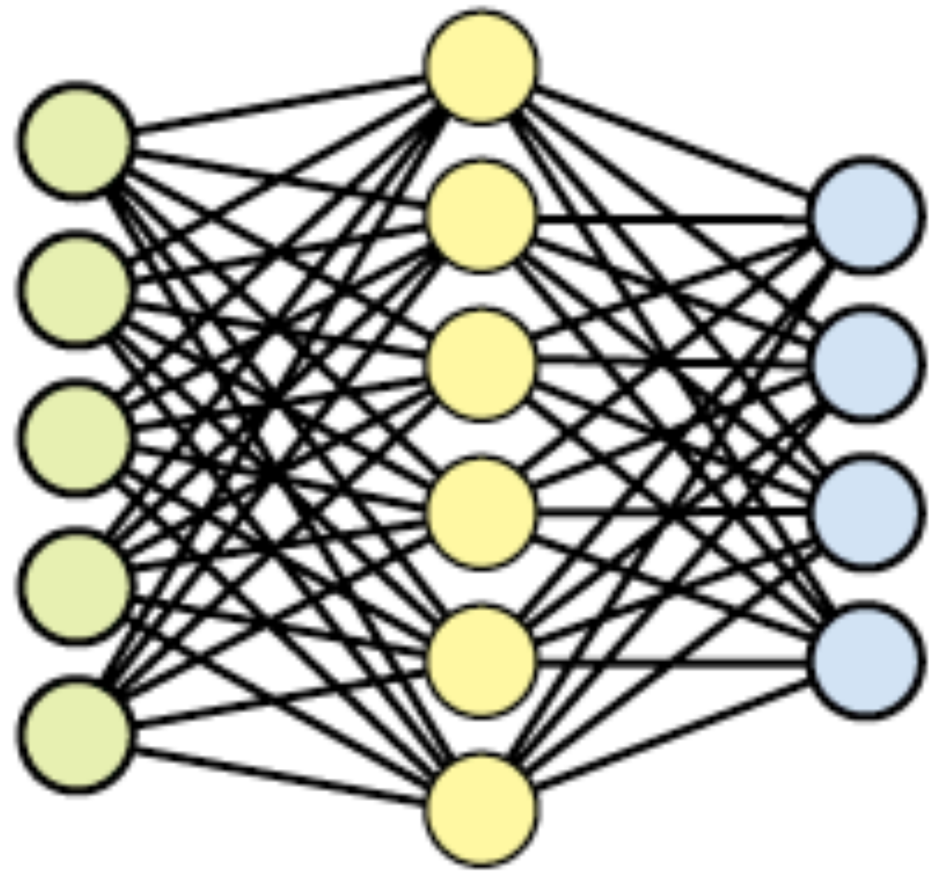


A deep neural Network

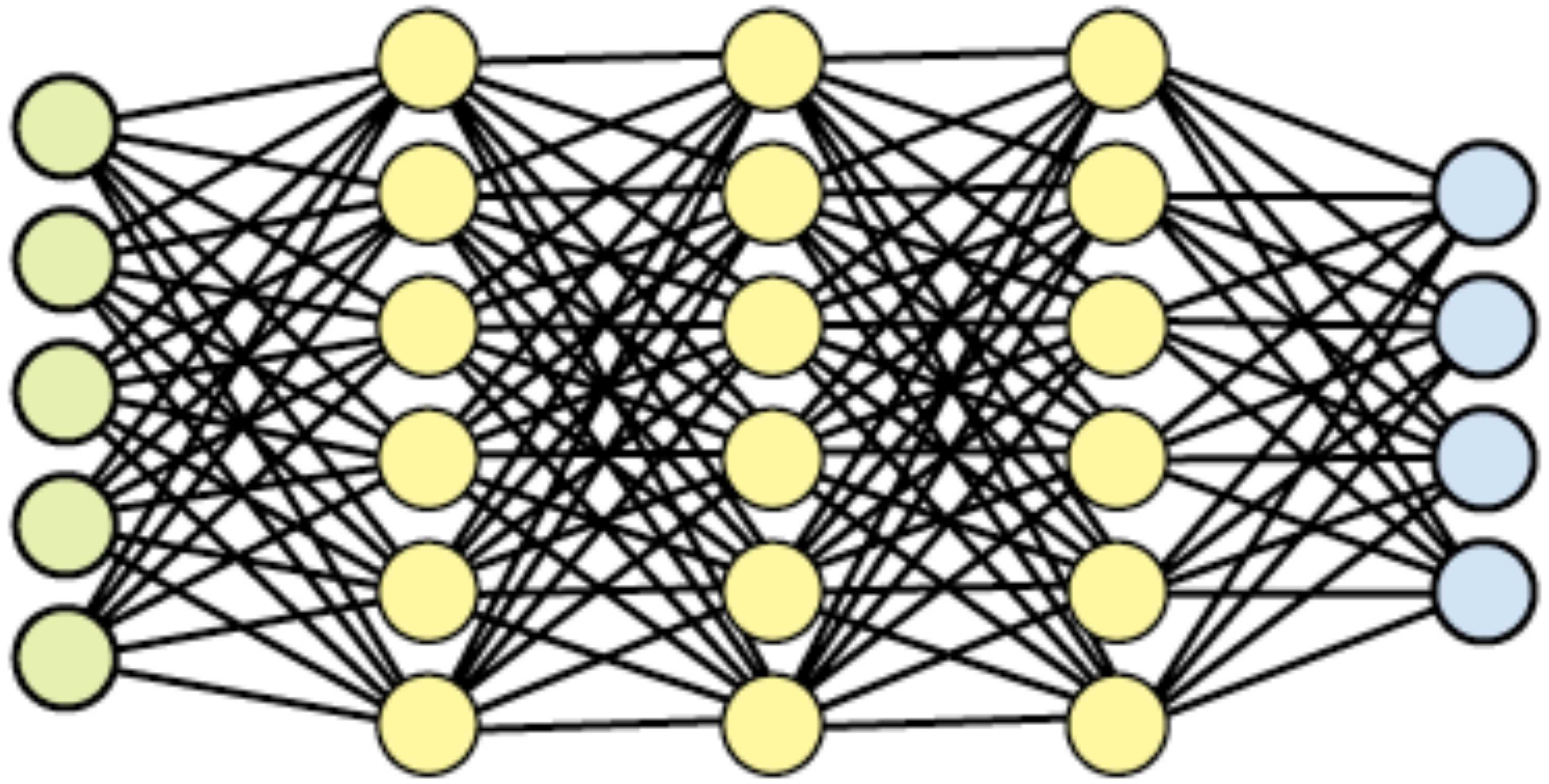


From Neural Networks to Deep Neural Networks

A neural Network



A deep neural Network

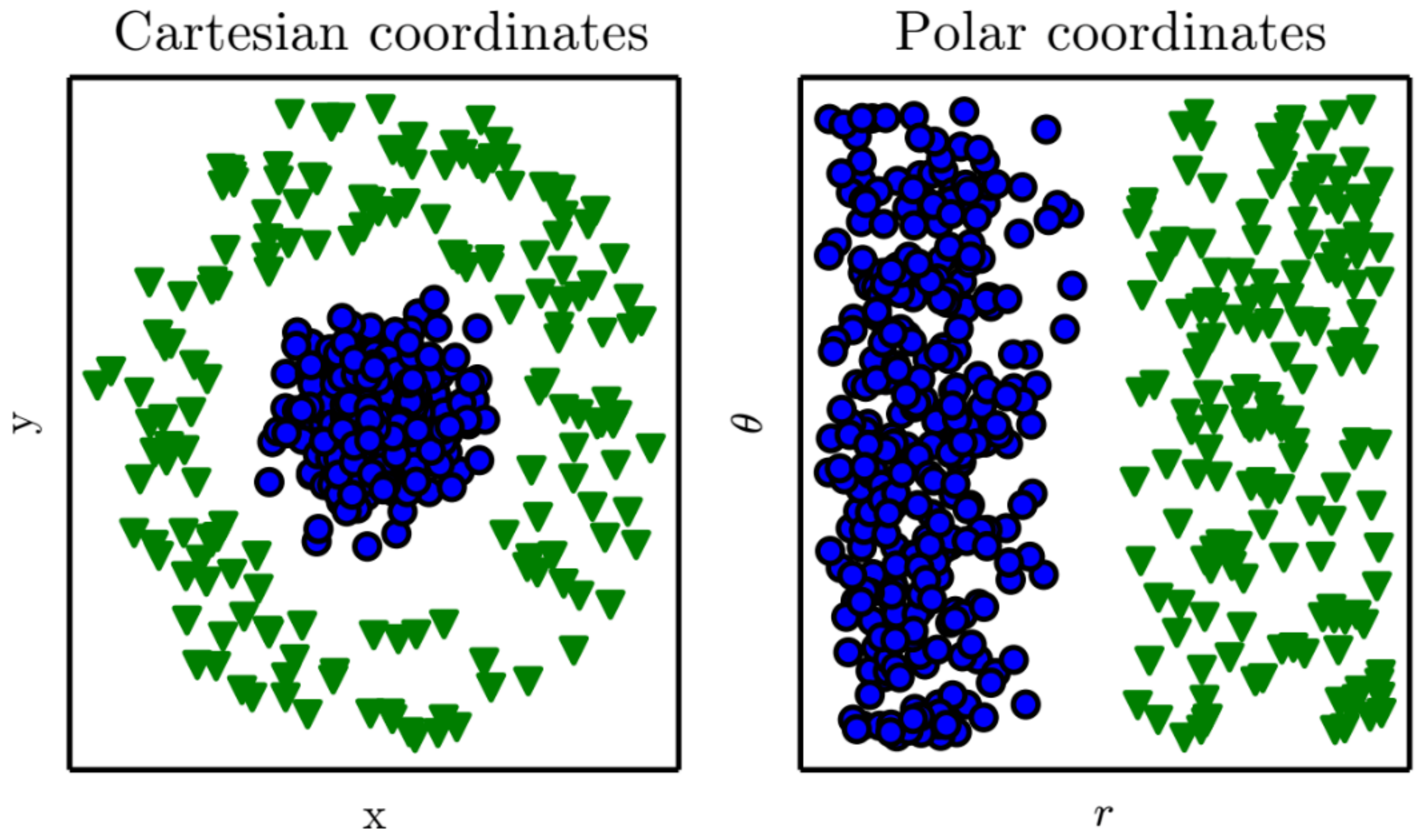


Modern deep learning provides a powerful framework for supervised learning. By adding more layers and more units within a layer, a deep network can represent functions of increasing complexity.

Deep Learning — Part II, p.163
http://www.deeplearningbook.org/contents/part_practical.html

Another View of deep learning

- Representations are important



[From: Honglak Lee (and Graham Taylor)]

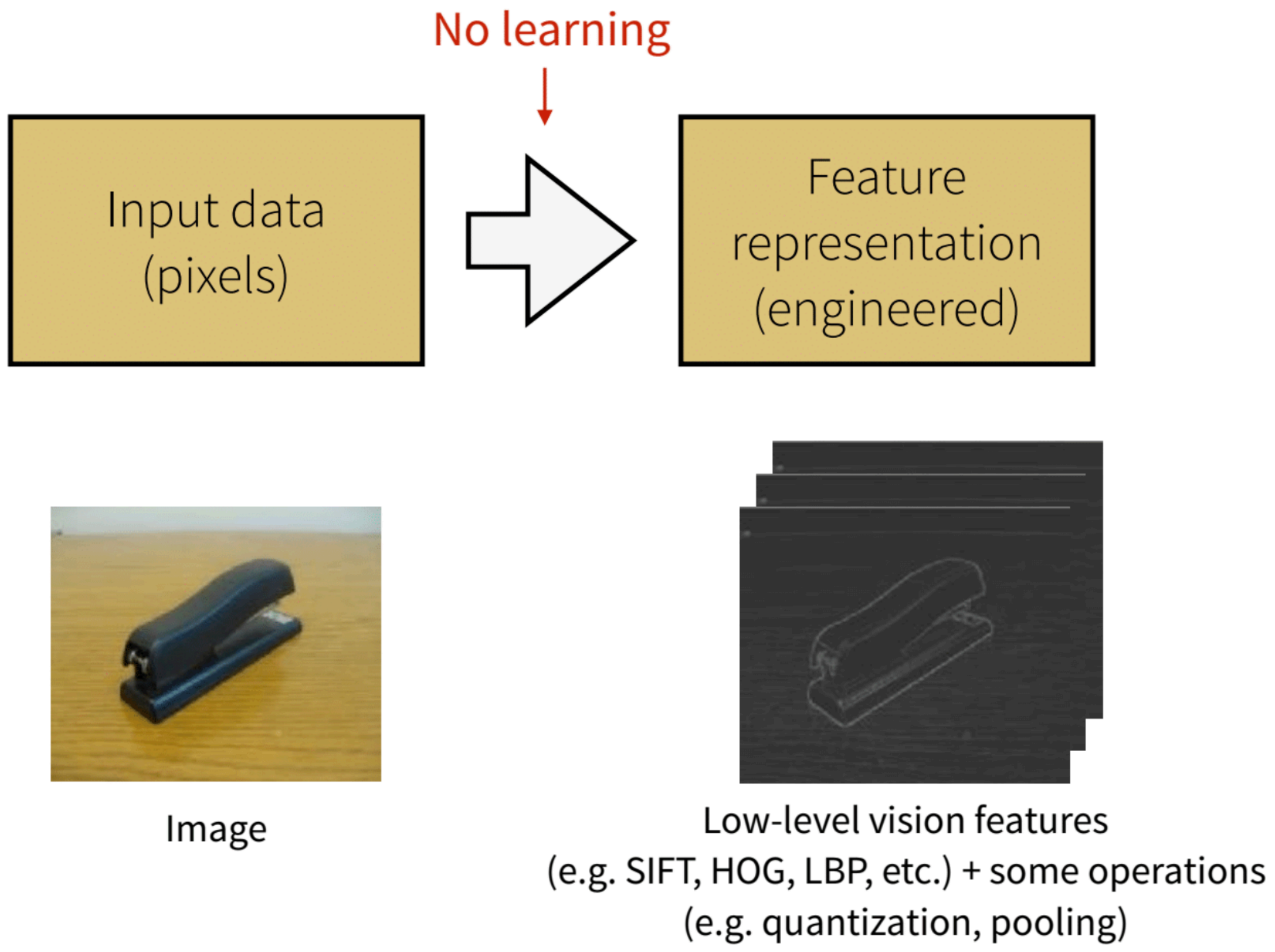
Representations

Input data
(pixels)

Image

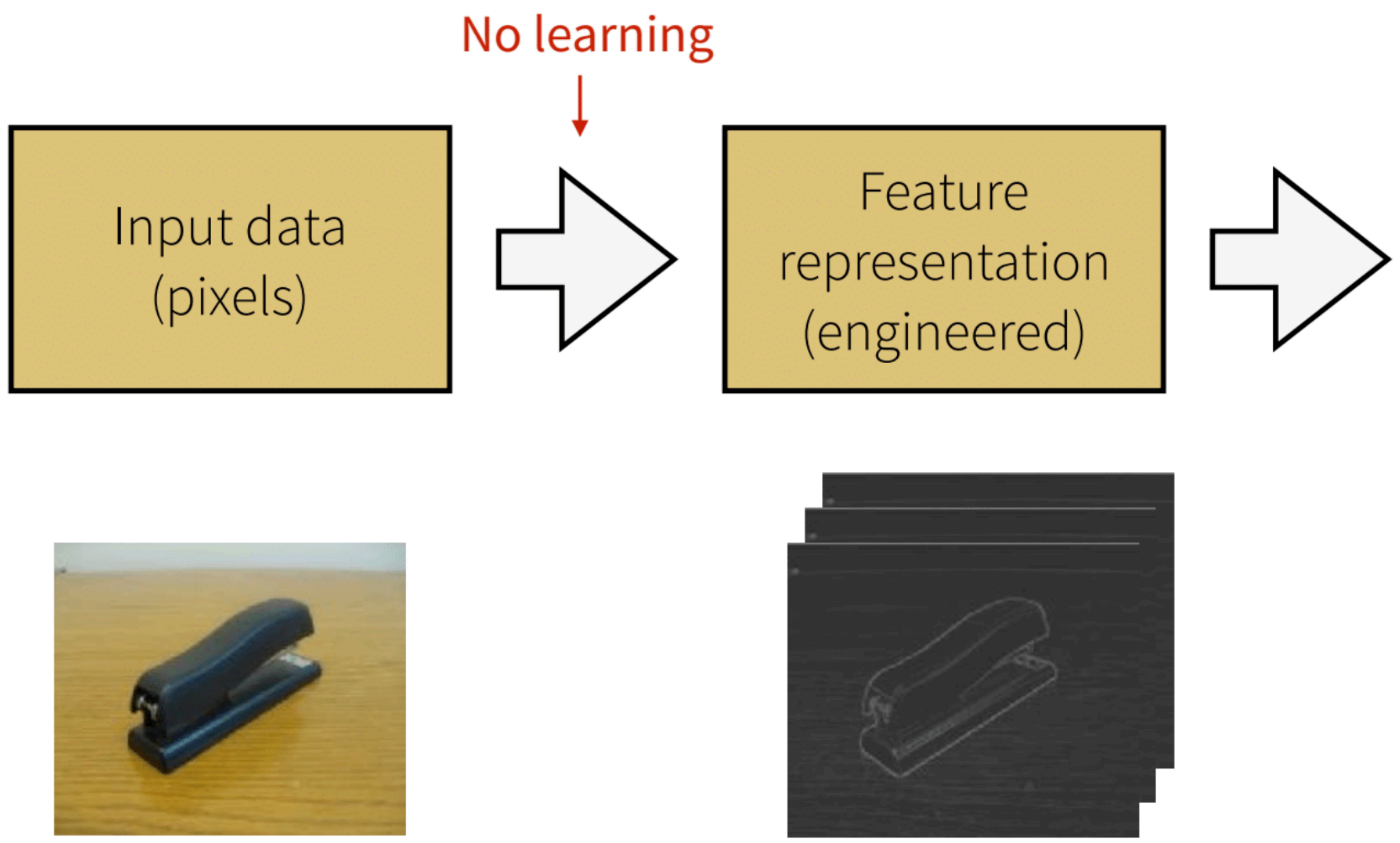
[From: Honglak Lee (and Graham Taylor]

Representations



[From: Honglak Lee (and Graham Taylor]

Representations

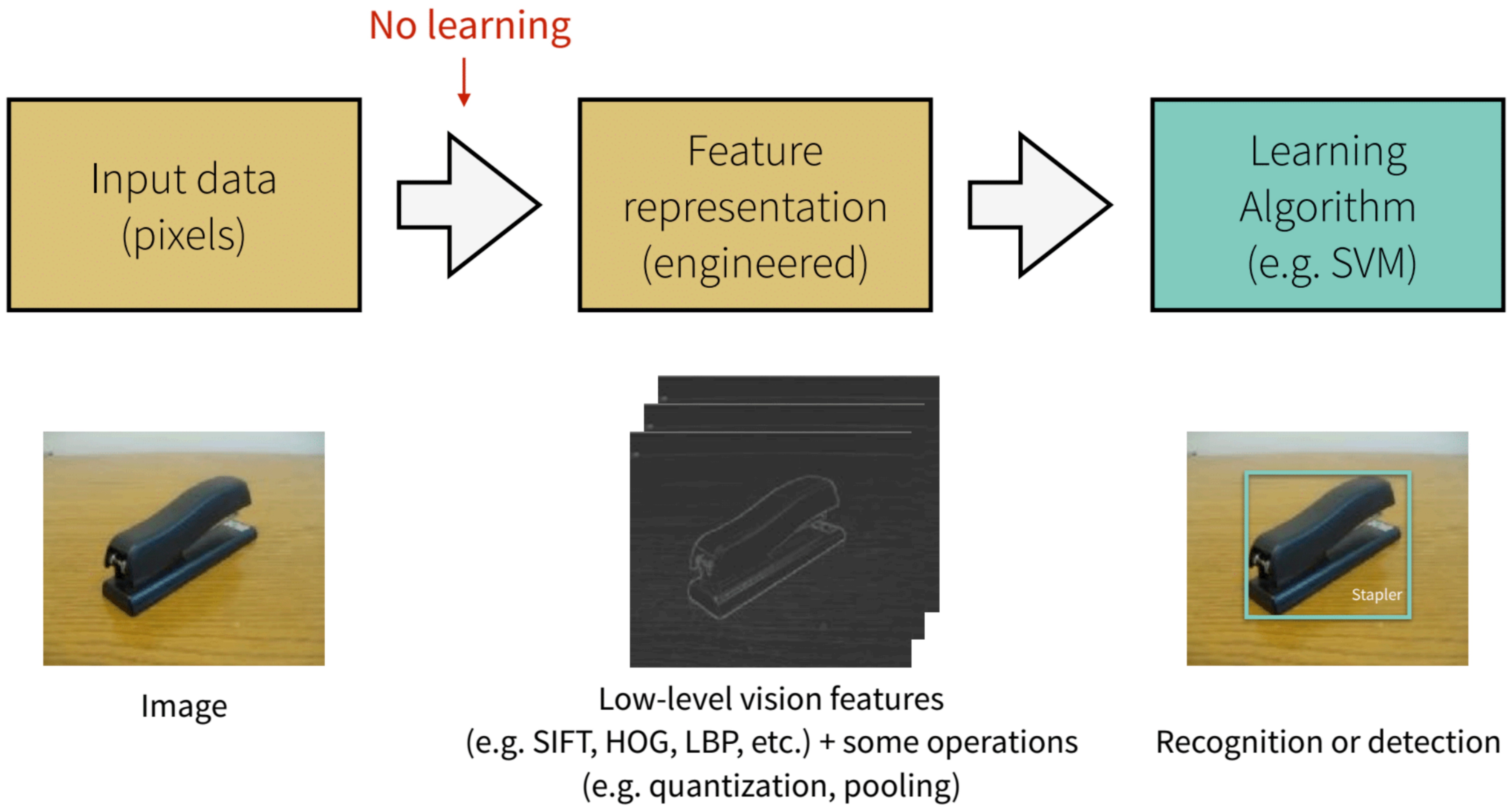


Image

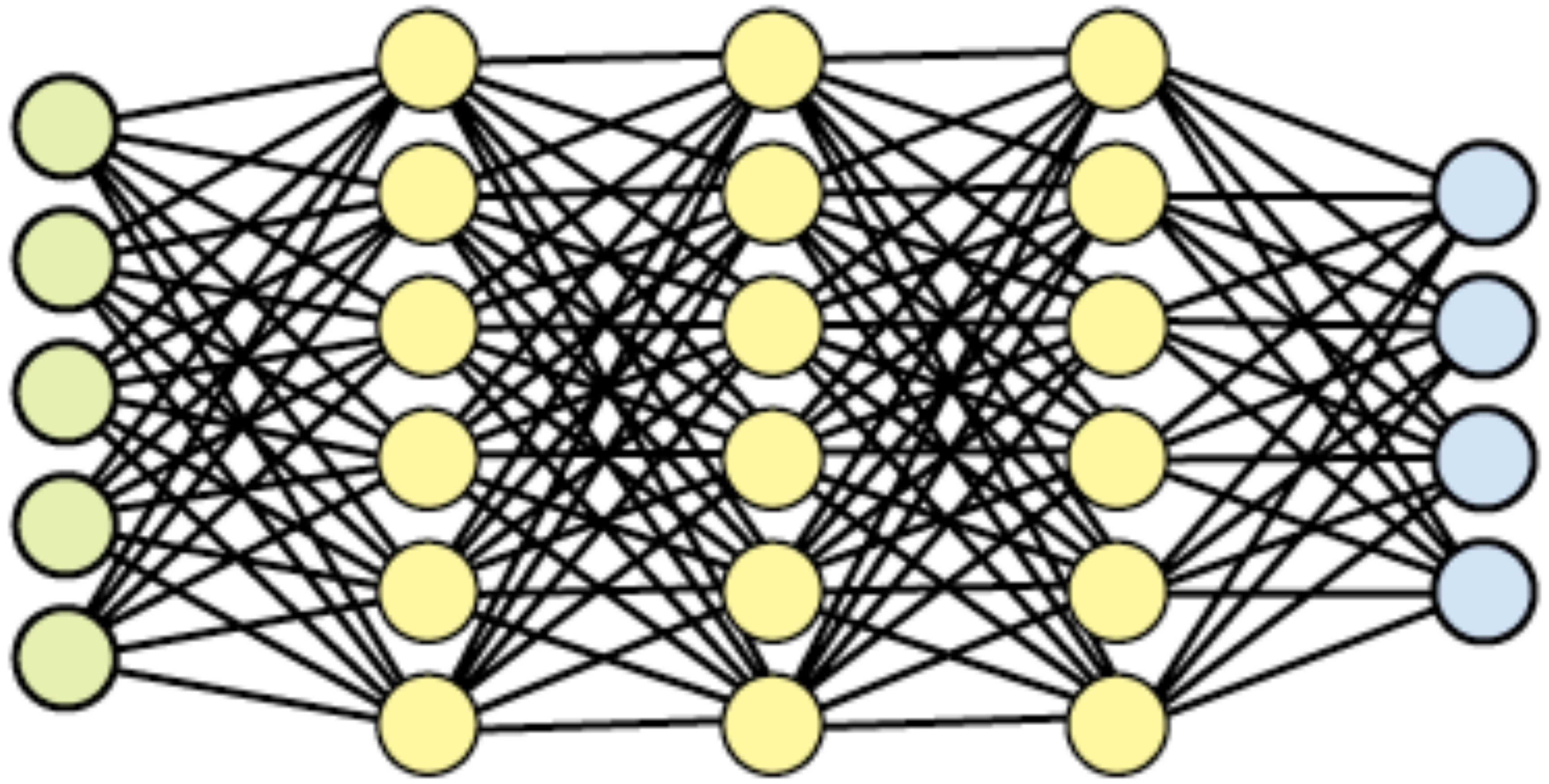
Low-level vision features
(e.g. SIFT, HOG, LBP, etc.) + some operations
(e.g. quantization, pooling)

[From: Honglak Lee (and Graham Taylor]

Representations

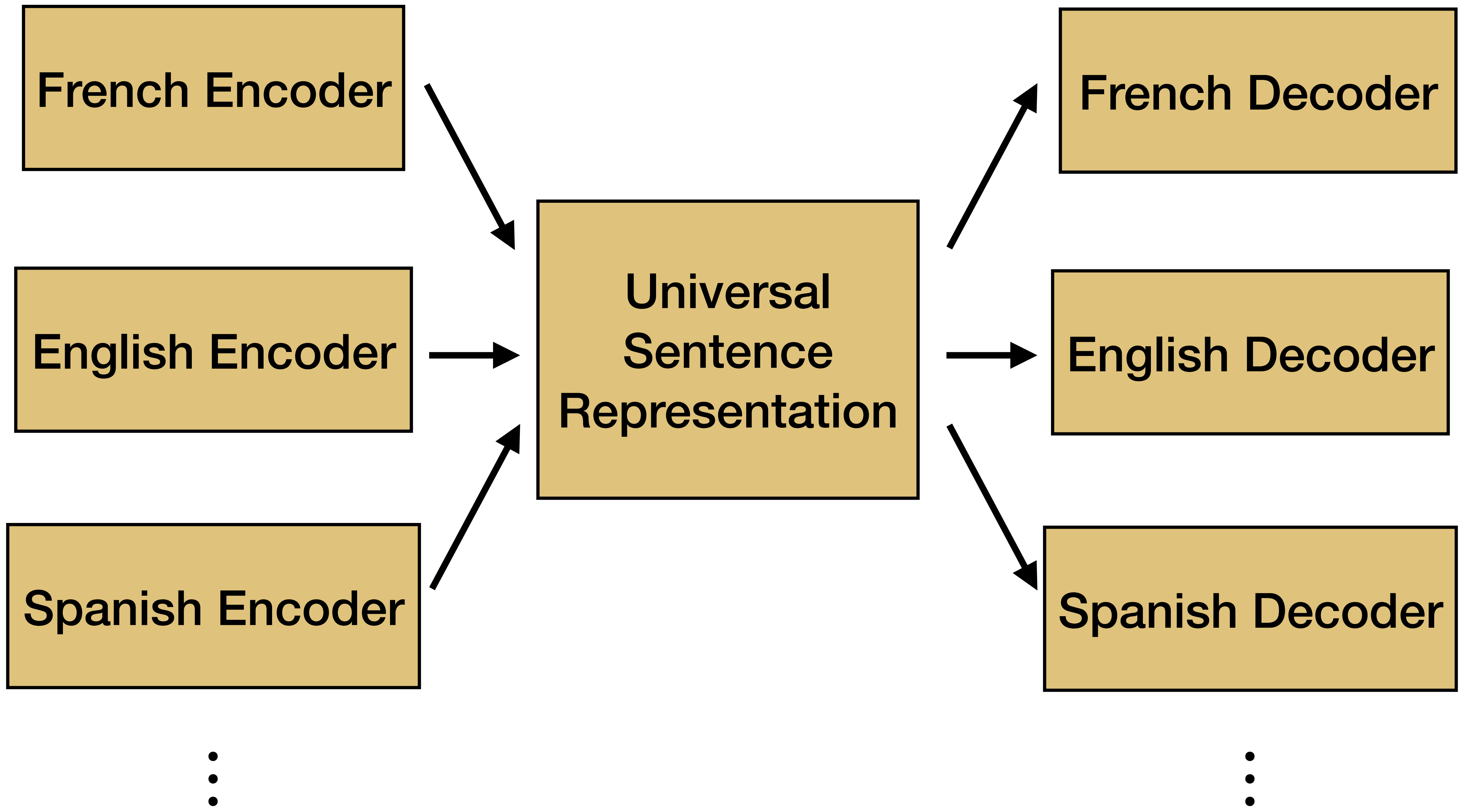


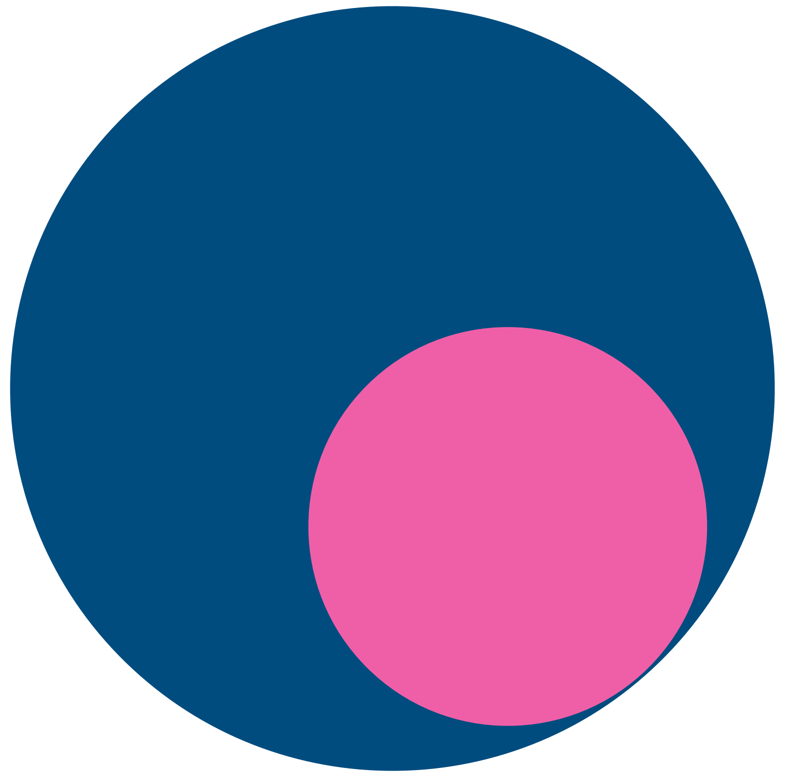
[From: Honglak Lee (and Graham Taylor]



Data → Layer 1 → Layer 2 … Classifier → Output

Machine Translation





Machine learning

Make machines that can learn

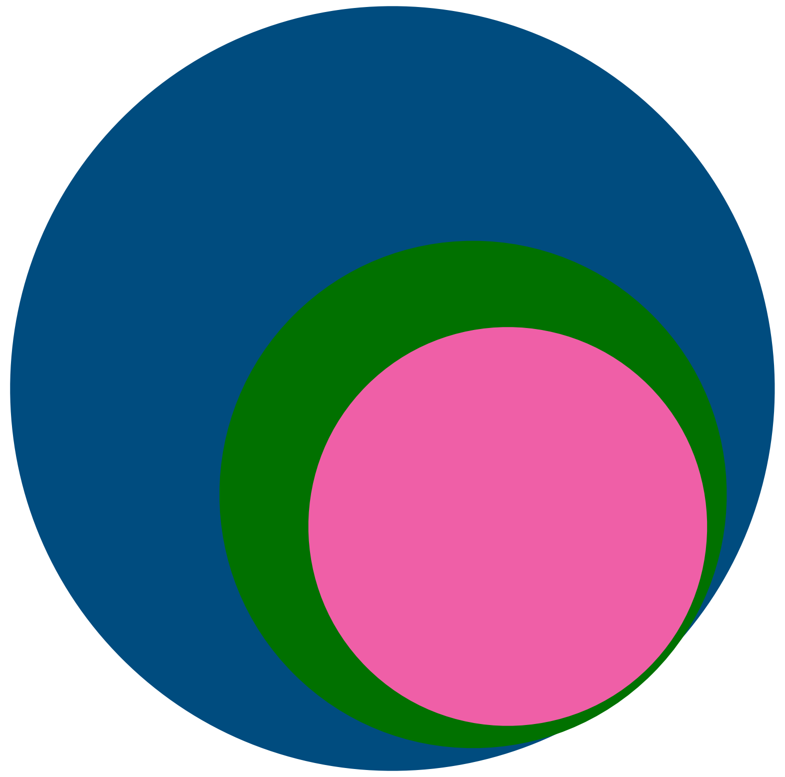
Deep learning

A set of machine learning techniques
based on neural networks

Idea: Hugo Larochelle

Representation learning

Machine learning paradigm to
discover data representations



Machine learning

Make machines that can learn

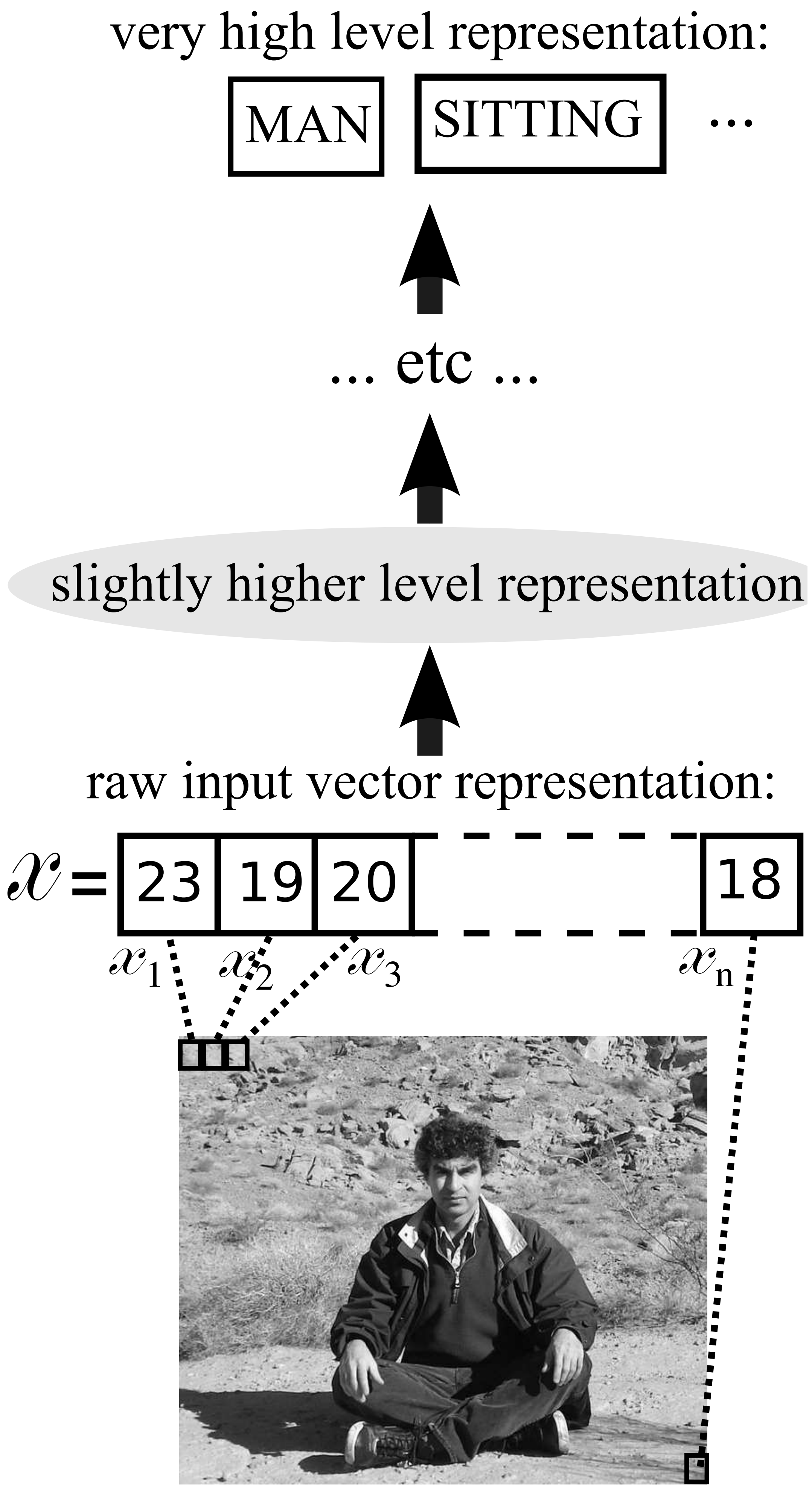
Deep learning

A set of machine learning techniques
based on neural networks

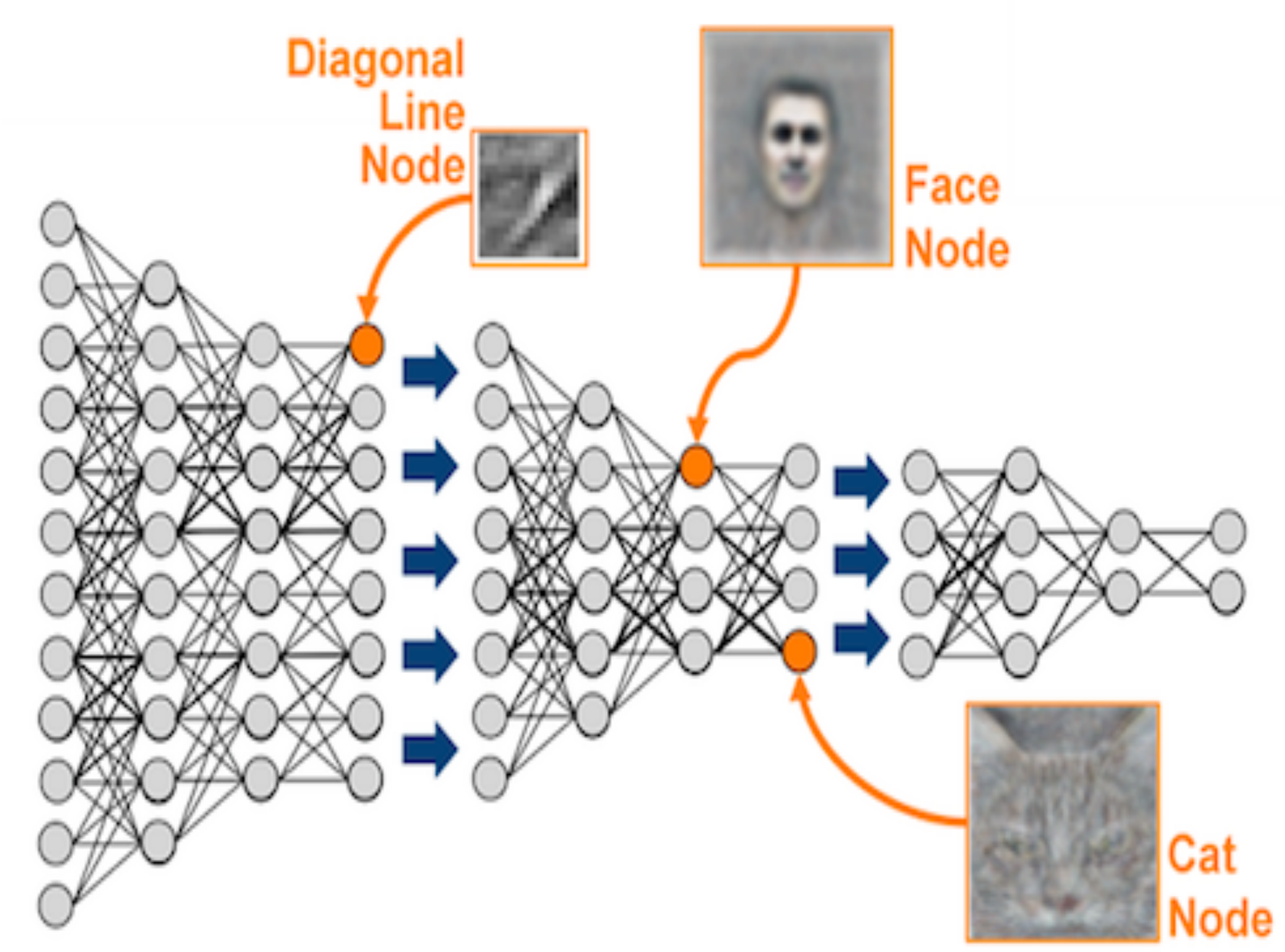
Idea: Hugo Larochelle

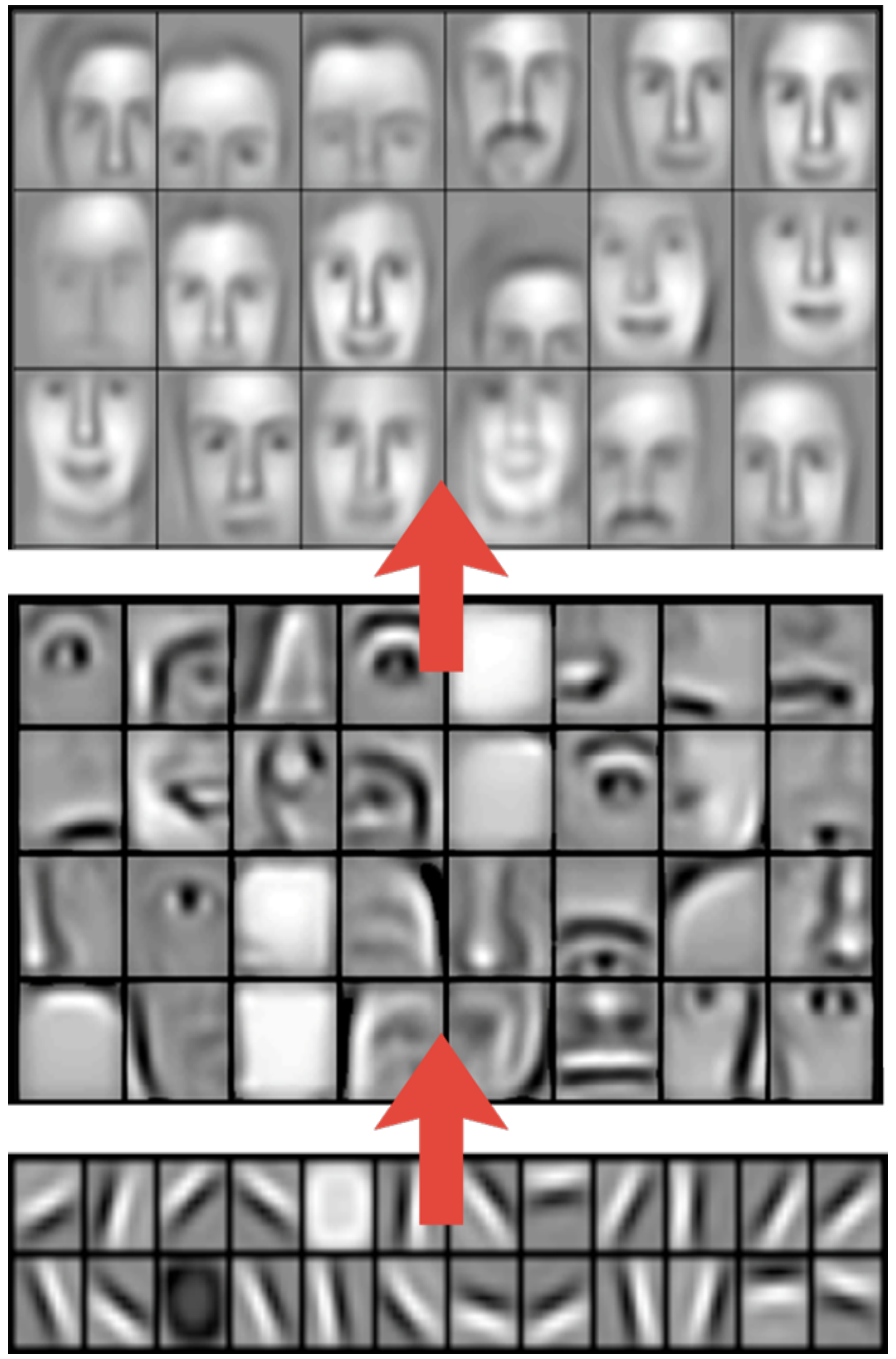
Deep neural networks

- Several layers of hidden nodes
- Parameters at different levels of representation



[Figure from Yoshua Bengio]





[Figures from Yoshua Bengio]

Neural Network Hyper-parameters

Hyperparameters

1. Model specific
    - Activation functions (output & hidden), Network size
2. Optimisation Objective
    - Regularization, Early-stopping, Dropout
3. Optimization procedure
    - Momentum, Adaptive learning rates

Activation Functions

- Non-linear functions that transform the weighted sum of the inputs, e.g.:

$$f(\sum_i w_i x_i)$$

Activation Functions

- Non-linear functions that transform the weighted sum of the inputs, e.g.:

$$f(\sum_i w_i x_i)$$

Activation Functions

- Non-linear functions that transform the weighted sum of the inputs, e.g.:

$$f(\sum_i w_i x_i)$$

- Non-linearities increase model representation power

Activation Functions

- Non-linear functions that transform the weighted sum of the inputs, e.g.:

$$f(\sum_i w_i x_i)$$

- Non-linearities increase model representation power

- Non-linearities increase the difficult of optimization

Activation Functions

- Non-linear functions that transform the weighted sum of the inputs, e.g.:

$$f(\sum_i w_i x_i)$$

  - Non-linearities increase model representation power

  - Non-linearities increase the difficult of optimization

- Different functions for hidden units and output units

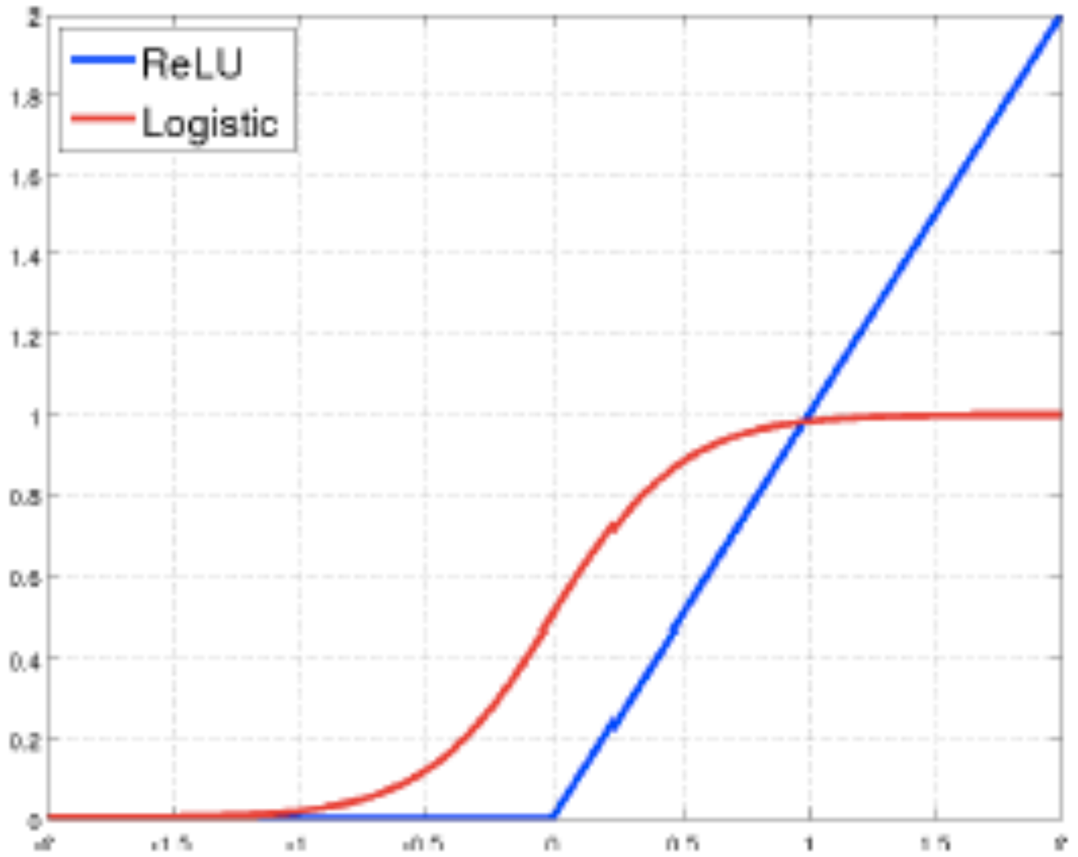
Activation functions — hidden units

- Traditional
  - Logistic-like units
    - $f(z) = \text{logit}^{-1}(z) = \frac{1}{1 + \exp(-z)}$
    - $f(z) = \tanh(z)$
  - Saturate on both sides
  - Derivable everywhere

 [https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/40811.pdf]

Activation functions — hidden units

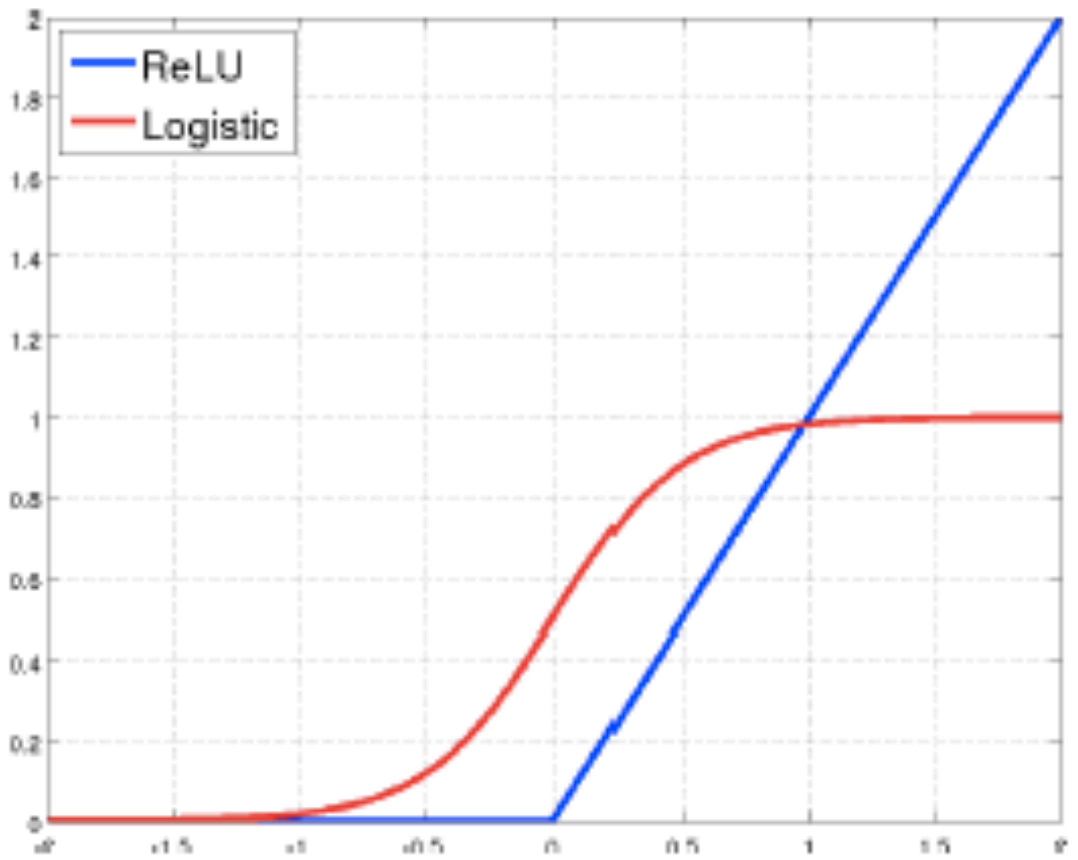
- Traditional
  - Logistic-like units
    - $f(z) = \text{logit}^{-1}(z) = \dfrac{1}{1 + \exp(-z)}$
    - $f(z) = \tanh(z)$
  - Saturate on both sides
  - Derivable everywhere



[https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/40811.pdf]

Activation functions — hidden units

- Traditional
  - Logistic-like units
    - $f(z) = \text{logit}^{-1}(z) = \frac{1}{1+\exp(-z)}$
    - $f(z) = \tanh(z)$
  - Saturate on both sides
  - Derivable everywhere



- Rectified linear units (Relu)

  $f(z) = \max\{0, z\}$

  - Non-derivable at a single point
- Now Standard
  - Better results / faster training
  - Shuts off units
- Leaky Relu

  $g(z) = \max\{0, z\} + \alpha \min\{0, z_i\}$

[https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/40811.pdf]

Activation functions — Output units

Output type	Output Unit	Equivalent Statistical Distribution
Binary (0,1)	$\text{sigmoid}(z) = \frac{1}{1 + \exp(-z)}$	Bernoulli
Categorical (0,1,2,3,k)	$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{i'} \exp(z_{i'})}$	Multinoulli
Continuous	Identity(z) = z	Gaussian
Multi-modal	mean, (co-)variance, components	Mixture of Gaussians

Regularization

- Weight decay on the parameters
  - L2 penalty on the parameters
- Early stopping of the optimization procedure
  - Monitor the validation loss and terminate when it stops improving
- Number of hidden layers and hidden units per layer

Momentum

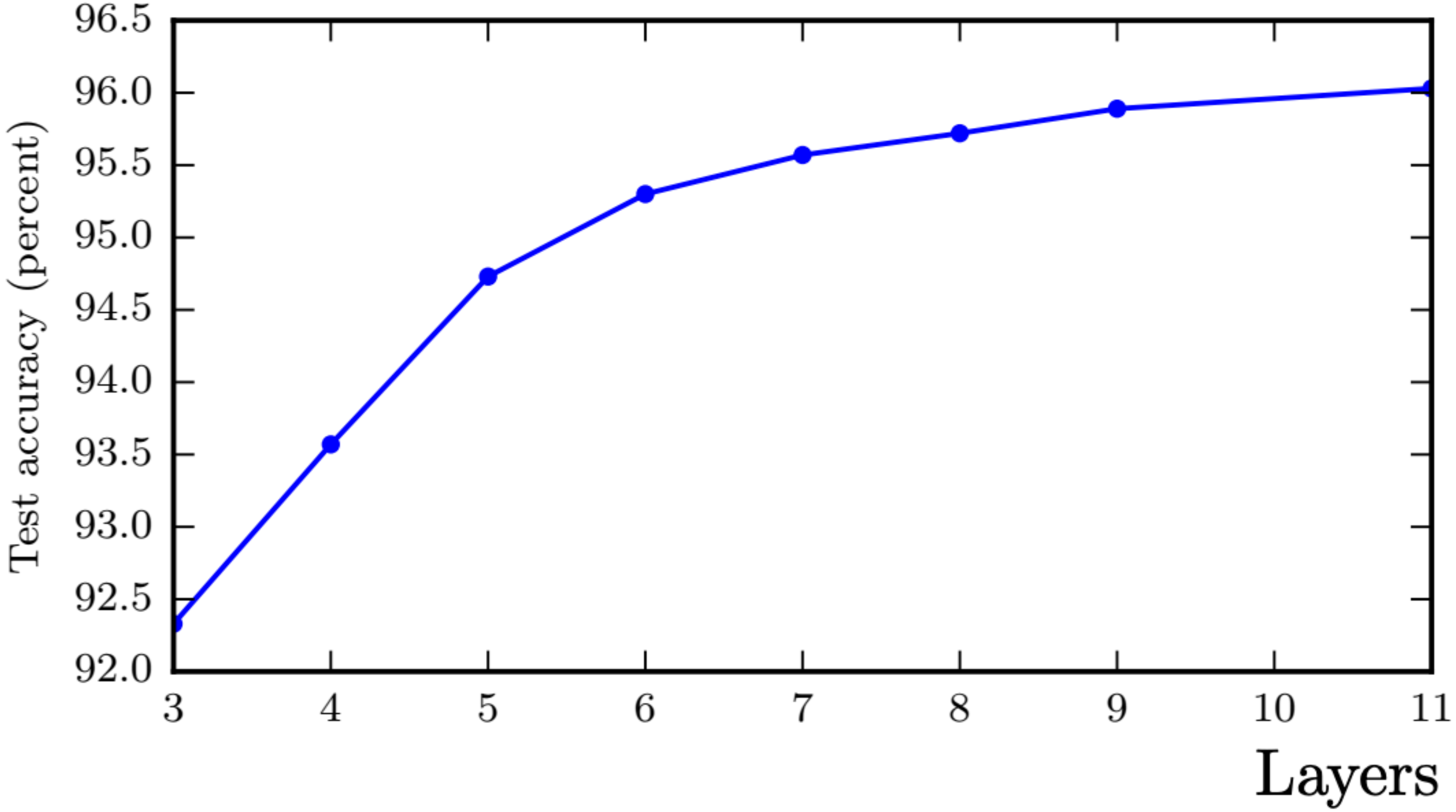
$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \alpha \nabla \mathbf{w}^{t} \quad \text{(Gradient Descent)}$$

$$\mathbf{v} = \beta \mathbf{v} - \alpha \nabla \mathbf{w}^{t} \quad \text{(Gradient Descent w. momentum)}$$
$$\mathbf{w}^{t+1} = \mathbf{w}^{t} + \mathbf{v}$$

- Pro: Can allow you to jump over small local optima
- Pro: Goes faster through flat areas by using acquired speed
- Con: Can also jump over global optima
- Con: One more hyper-parameter to tune
- More advanced adaptive steps: adagrad, adam

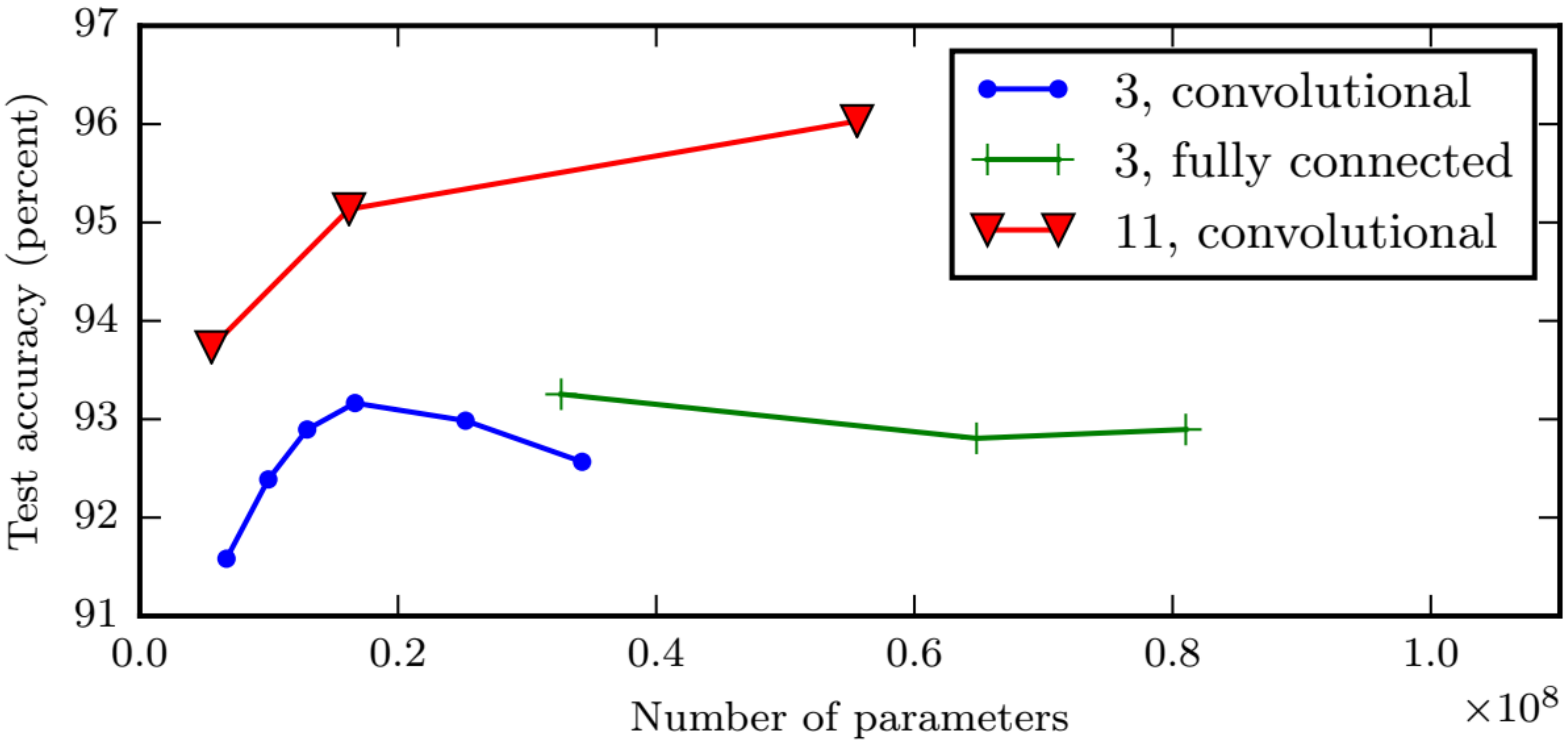
Wide or Deep?

Wide or Deep?



[Figure 6.6, Deep Learning, book]

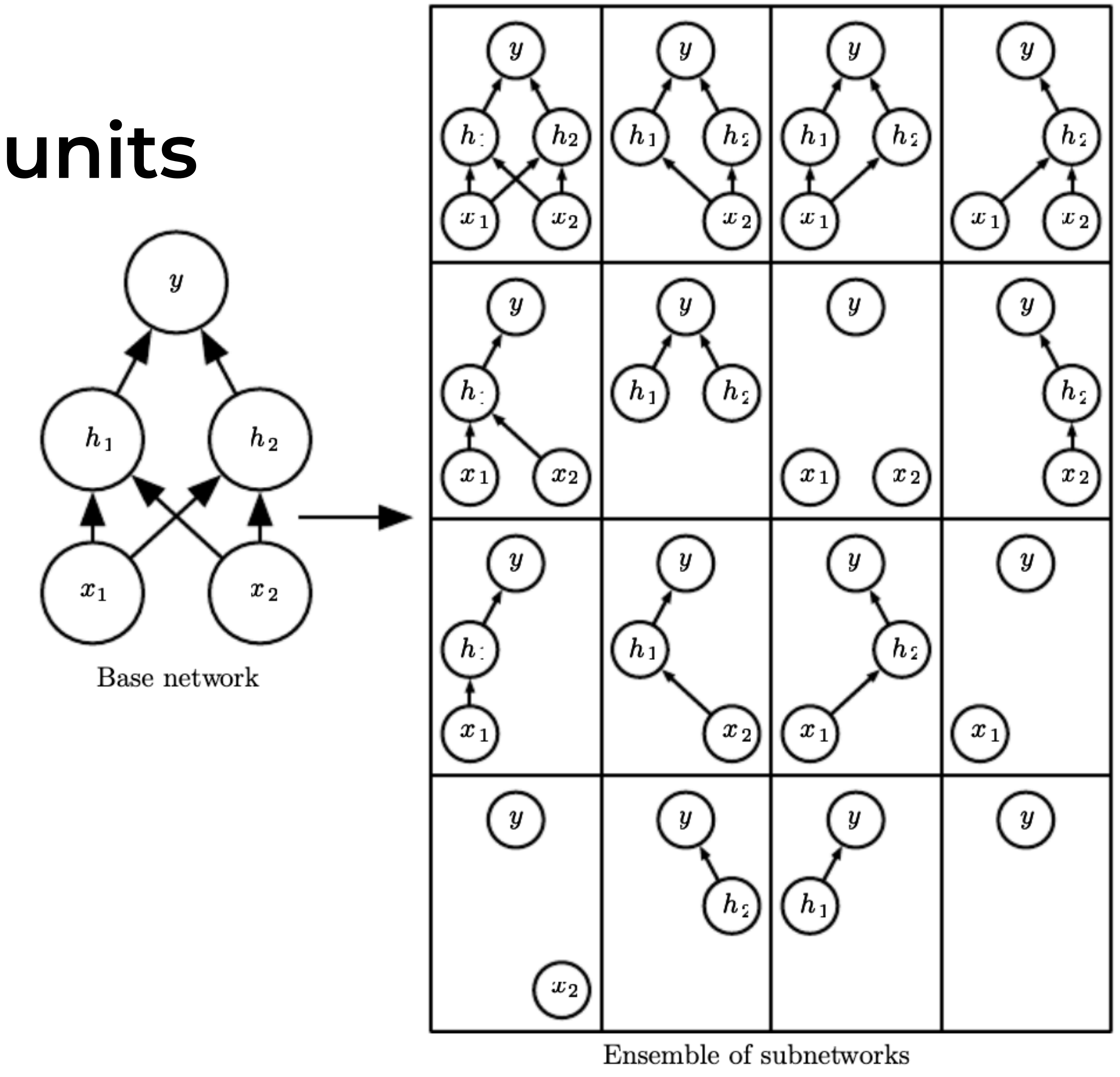
Wide or Deep?



[Figure 6.7, Deep Learning, book]

Dropout

- Standard regularization technique
- At training drop a percentage of the units
  - Used for non-output layer
  - Prevents co-adaptation / Bagging
- At test: use the full network
  - Normalize the weights



[Figure 7.6, Deep Learning]

Neural Network Takeaways

Neural Networks takeaways

- Very flexible models
  - Composed of simple units (neurons)
  - Adapt to different types of data
- (Highly) non-linear models
  - E.g., Can learn to order/rank inputs easily
- Scale to very large datasets
- May require "fiddling" with model architecture + optimization hyper-parameters
  - Standardizing data can be very important

Where do NNs shine

- Input is high-dimensional discrete or real-valued
- Output is discrete or real valued, or a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human interpretability is not important
- The computation of the output based on the input has to be fast

Most tasks that consist of mapping an input vector to an output vector, and that are easy for a person to do rapidly, can be accomplished via deep learning, given sufficiently large models and sufficiently large datasets of labeled training examples.

Other tasks, that cannot be described as associating one vector to another, or that are difficult enough that a person would require time to think and reflect in order to accomplish the task, remain beyond the scope of deep learning for now.

Deep Learning — Part II, p.163
http://www.deeplearningbook.org/contents/part_practical.html

Neural Networks in Practice

In practice

- Software now derives gradients automatically
  - You specify the architecture of the network
    - Connection pattern
    - Number of hidden layers
    - Number of layers
    - Activation functions
    - Learning rate (learning rate updates)
    - Dropout
- For intuitions: https://playground.tensorflow.org

A selection of standard tools (in python)

- Scikit-learn
    - Machine learning toolbox
        - Feed-forward neural networks
- Neural network specific tools
    - PyTorch, Tensorflow
        - Keras
- More specific tools for specific tasks:
    - caffe for computer vision, pySpark for distributed computations, NLTK for natural language processing