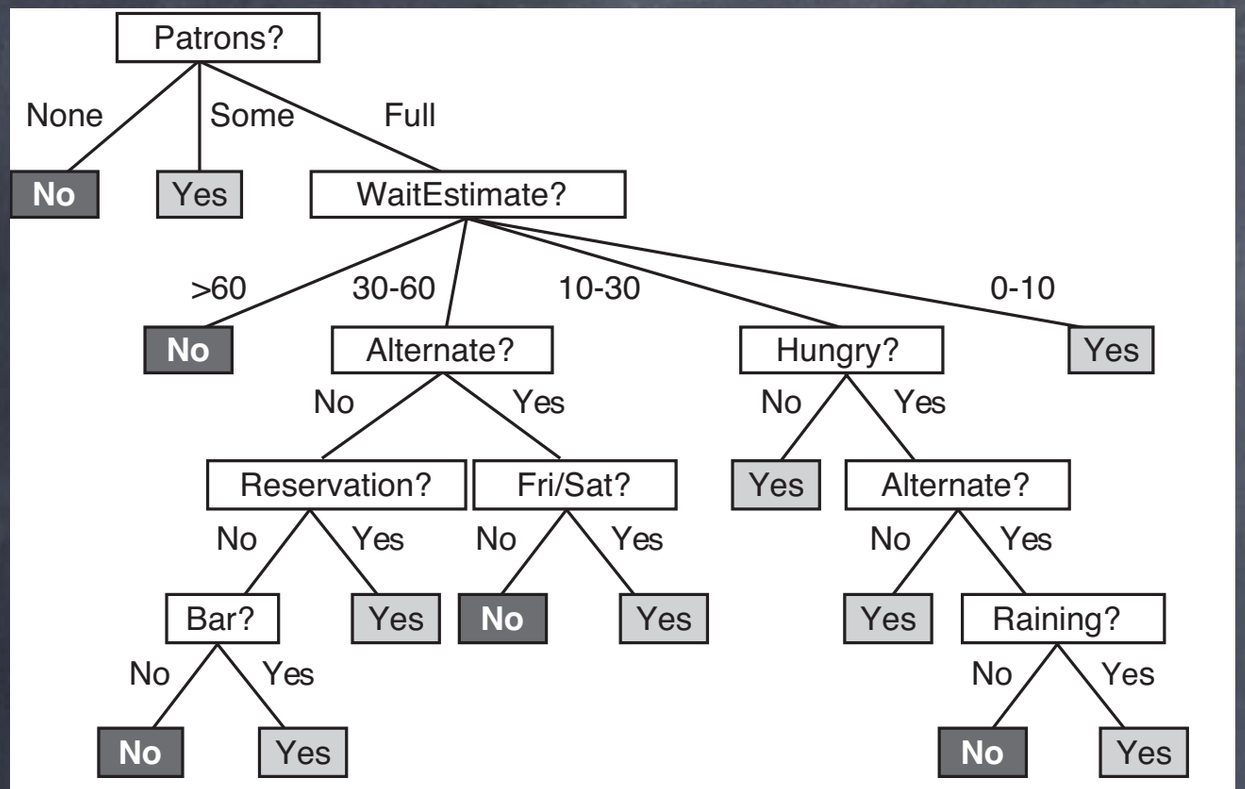
CSC242: Introduction to Artificial Intelligence

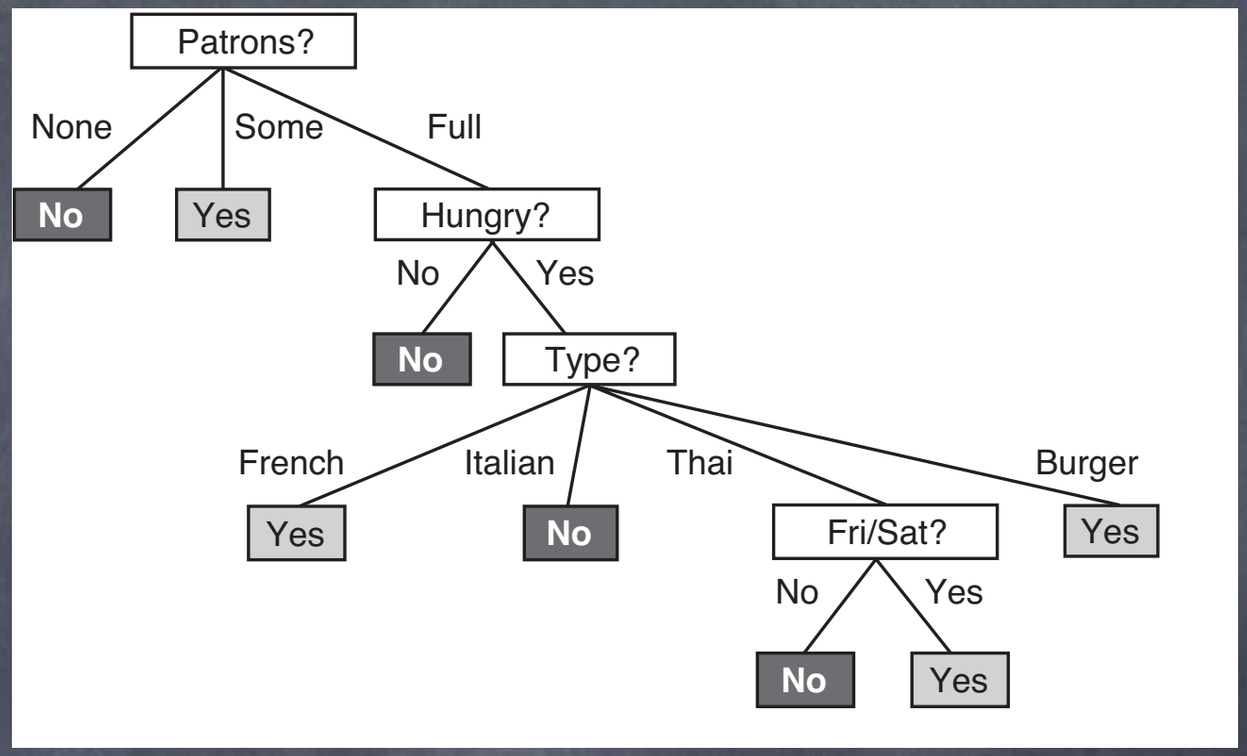
Lecture 4.3

Please put away all electronic devices

Decision Trees

- Represent sequence of tests that lead to a decision
- Compact representation of how to make the decision
- Can be learned from examples
- Decision trees have explanatory power!





Learning lets the data speak for itself

Supervised Learning

- Given a training set of N example input-output pairs:

$$(\mathbf{x}_1,\ y_1),\ (\mathbf{x}_2,\ y_2),\ \ldots,\ (\mathbf{x}_\mathrm{N},\ y_\mathrm{N})$$

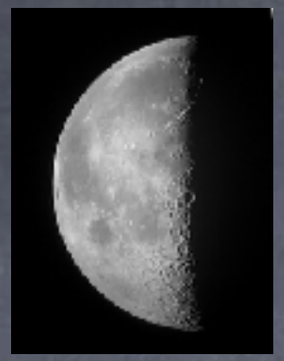
  where each $y_j = f(\mathbf{x}_j)$
- Discover function h that approximates f
- Search through the space of possible hypotheses for one that will perform well

Hypothesis Space

- Decision trees (Boolean formulas)
- Linear functions $y = mx + b$
- Polynomials (of some degree)
- Java programs?
- Turing machines?

Evaluating Hypotheses

- Accuracy: fits the data
  - Test on some examples
  - Generalization: predicts outputs for unseen inputs
    - Test with cross-validation
- Simplicity and "searchability"



red	3:25	full	yes	6
red	17:31	3/4	yes	3
blue	21:09	full	no	6
red	11:15	1/8	no	1
green	14:28	1/4	yes	2

Overfitting

- A learned model adjusts to random features of the input that have no causal relation to the target function
- Usually happens when a model has too many parameters relative to the number of training examples

Overfitting

- Becomes more likely as the hypothesis space and number of input attributes grows
- Becomes less likely as the number of training examples increases
- Overfitting => Poor generalization

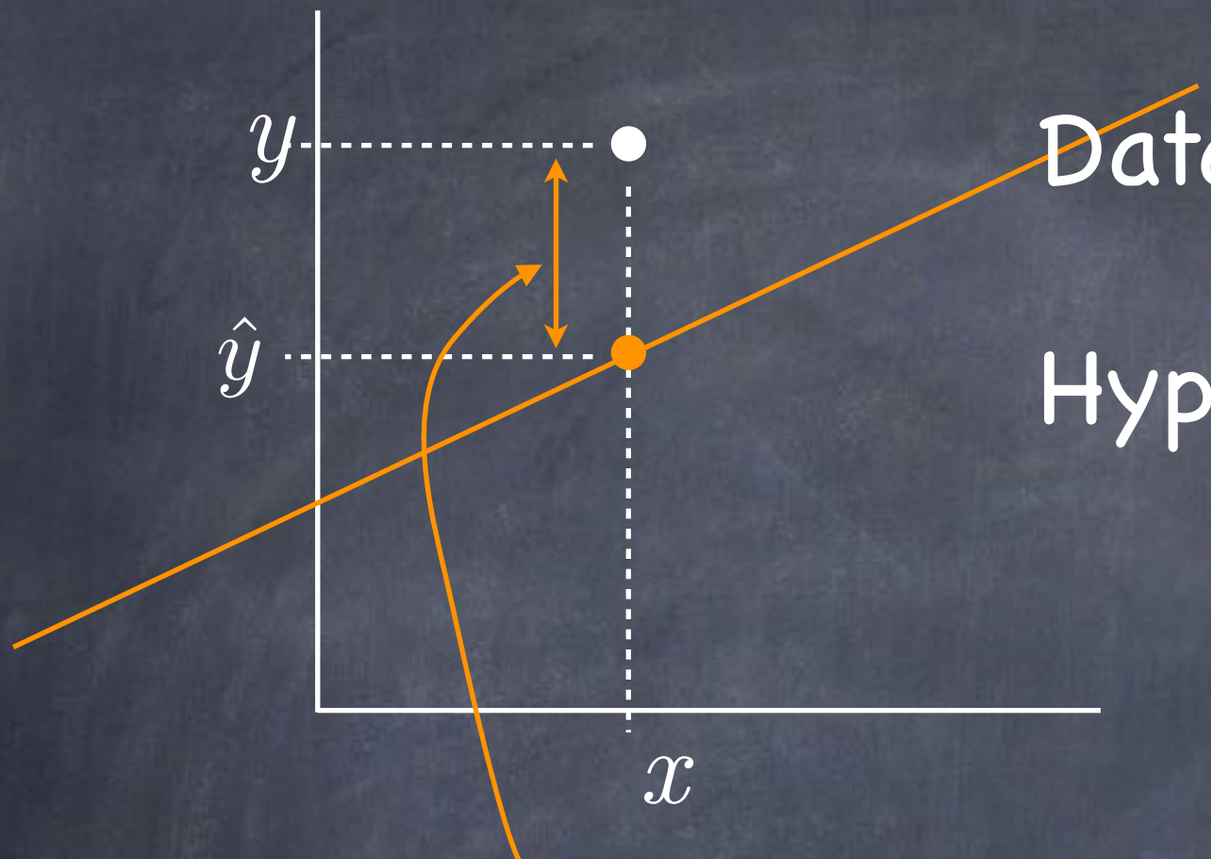
More Data More Data More Data
(or simpler hypotheses)

Univariate Linear Regression

- Given a set of N data points (x, y)
- Find linear function (line)

$$h_{\mathrm{w}}(x) = \mathbf{w} \cdot \mathbf{x} = w_0 + w_1 x$$

that best fits the data



Data point (x,y) from $y = f(x)$

Hypothesis $h_{\mathrm{w}}(x) = w_1 x + w_0$

$$\hat{y} = h_{\mathrm{w}}(x) = w_1 x + w_0$$

$$L(y, \hat{y}) = |y - \hat{y}| = L_1(y, \hat{y})$$

$$= (y - \hat{y})^2 = L_2(y, \hat{y})$$

$$= 0 \text{ if } y = \hat{y} \text{ else } 1 = L_{0/1}(y, \hat{y})$$

Univariate Linear Regression

Find $\mathbf{w} = [w_0, w_1]$ that minimizes $L(h_\mathrm{w})$

$$\mathrm{w}^* = \operatorname*{argmin}_\mathrm{w} L(h_\mathrm{w})$$

$$= \operatorname*{argmin}_\mathrm{w} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

Gradient Descent

$\mathbf{w} \leftarrow$ any point in parameter space

loop until convergence do

for each w_i in $\mathbf{w}$ do

Gradient of loss function along w_i axis

Update rule

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} L(\mathbf{w})$$

Learning rate

Gradient Descent for Univar. Linear Regression

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j))$$

$$w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \times x_j$$

Multivariate Linear Regression

- Hypothesis space:

$$h_{\mathbf{w}}(x) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

$$= w_0 + \sum_i w_i x_i$$

$$\mathbf{w} = [w_0, w_1, w_2, \ldots, w_n]$$

$$\mathbf{x} = [1, x_1, x_2, \ldots, x_n]$$

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \sum_i w_i x_i$$

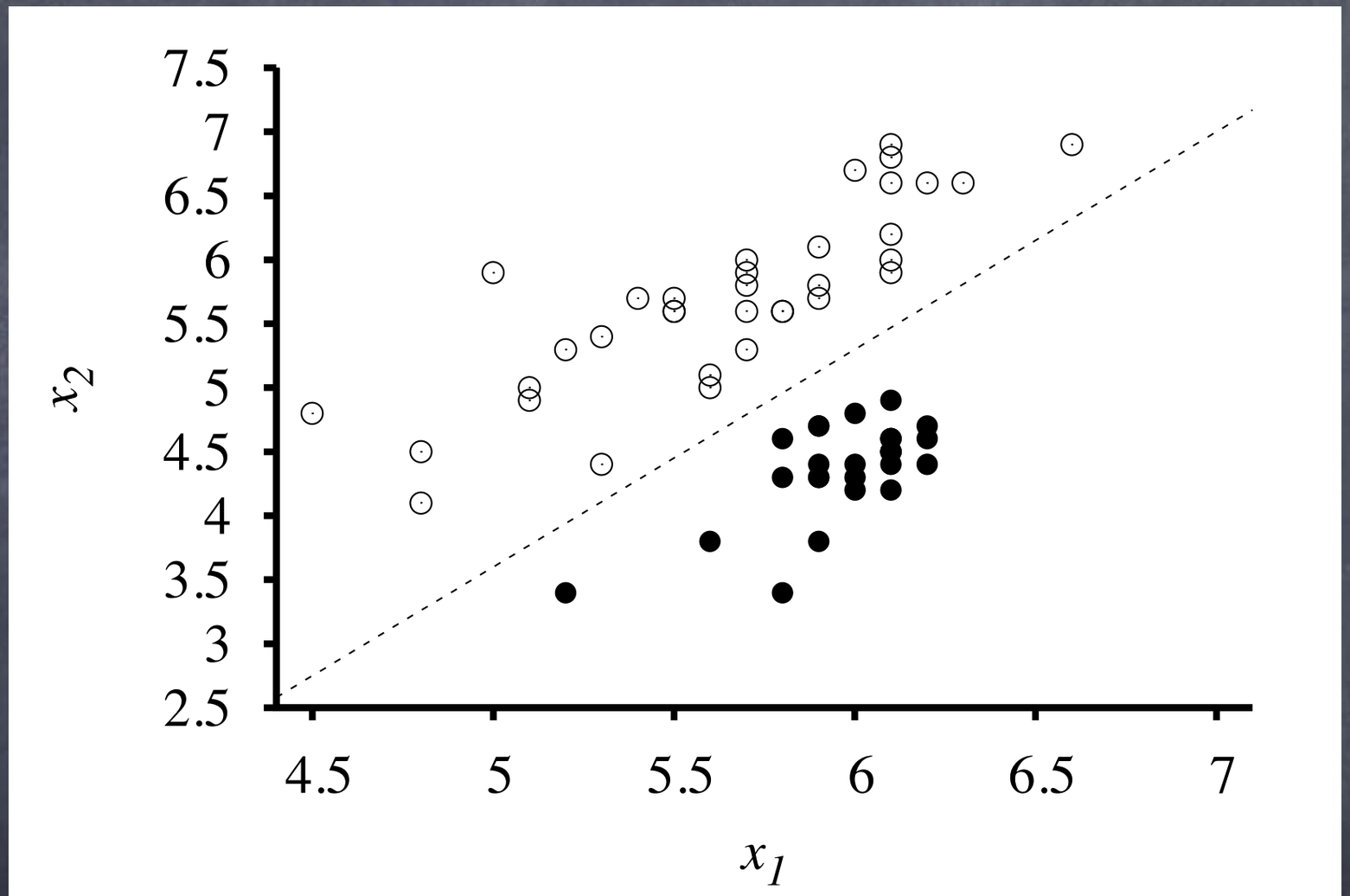
Linear Regression using Gradient Descent

- Hypothesis space: $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$
- Goal: $\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} L(h_{\mathbf{w}})$
- Gradient descent
  - Update rule:

$$w_i \leftarrow w_i + \alpha \sum_j x_{j,i}(y_j - h_{\mathbf{w}}(\mathbf{x}_j))$$

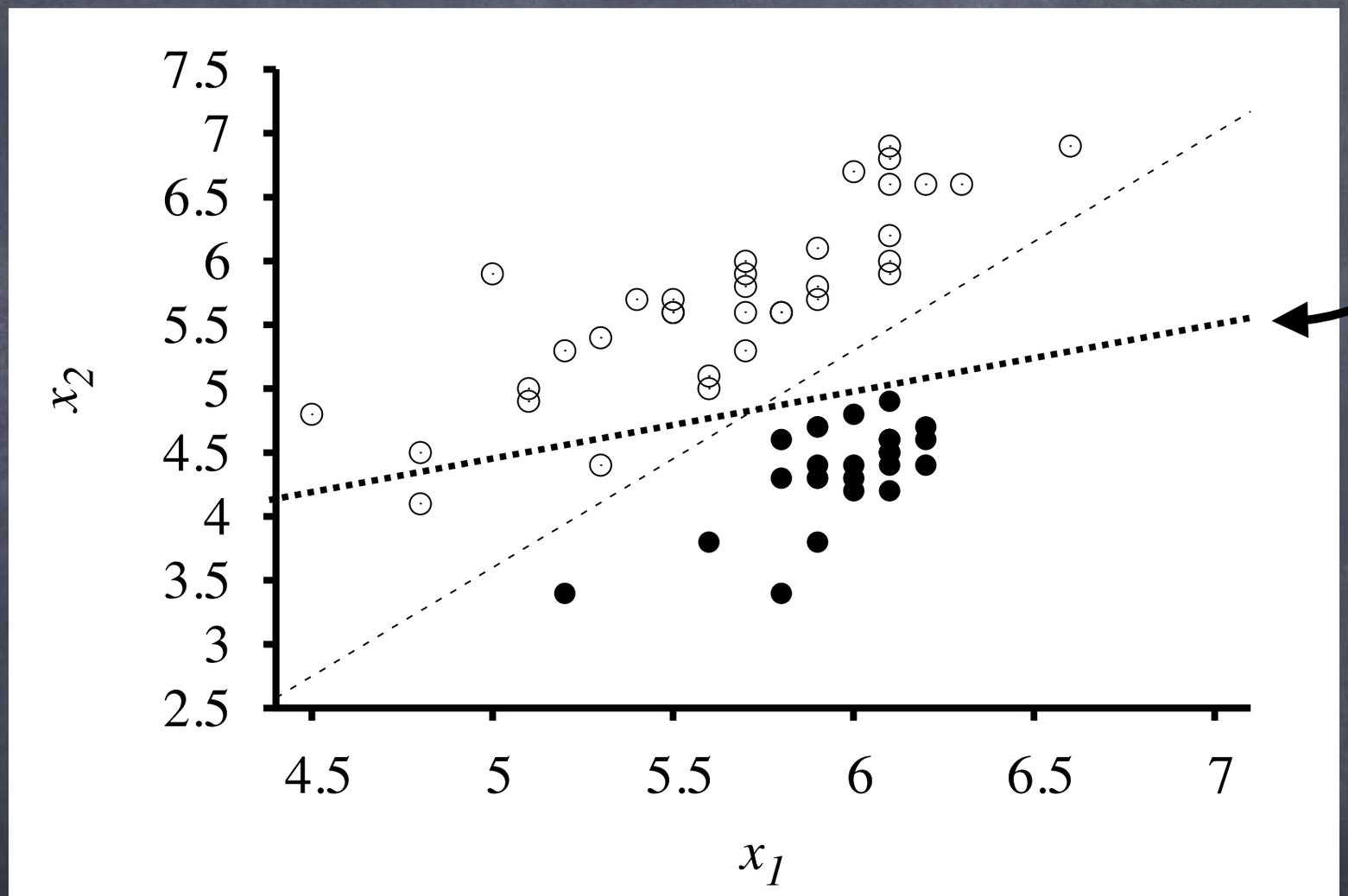
Linear Regression using Gradient Descent

- Learning (estimating) a function from examples: supervised learning
- Minimizing loss between hypothesis and actual values (typically L_2 loss)
- Solve using gradient descent (iterative, local search method) if no closed-form solution



$$x_2 = 1.7x_1 - 4.9$$

$$x_2 = 0.52x_1 + 2.11$$



$$x_2 = 1.7x_1 - 4.9$$

Linear Separator

- Decision boundary is a line
- Line in 2D, plane in 3D, hyperplane in nD
- Data that admit a linear separator are said to be linearly separable

Linear Classifier

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

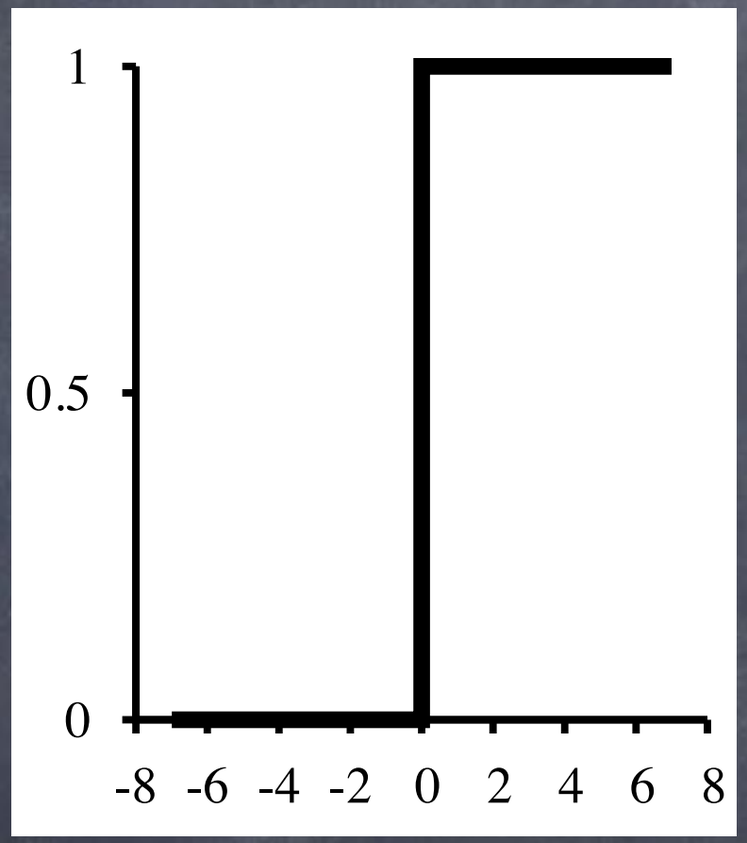
$$\mathbf{w} \cdot \mathbf{x} = 0$$

All instances of one class are above the line: $\mathbf{w} \cdot \mathbf{x} > 0$

All instances of one class are below the line: $\mathbf{w} \cdot \mathbf{x} < 0$

$$h_{\mathbf{w}}(\mathbf{x}) = Threshold(\mathbf{w} \cdot \mathbf{x})$$

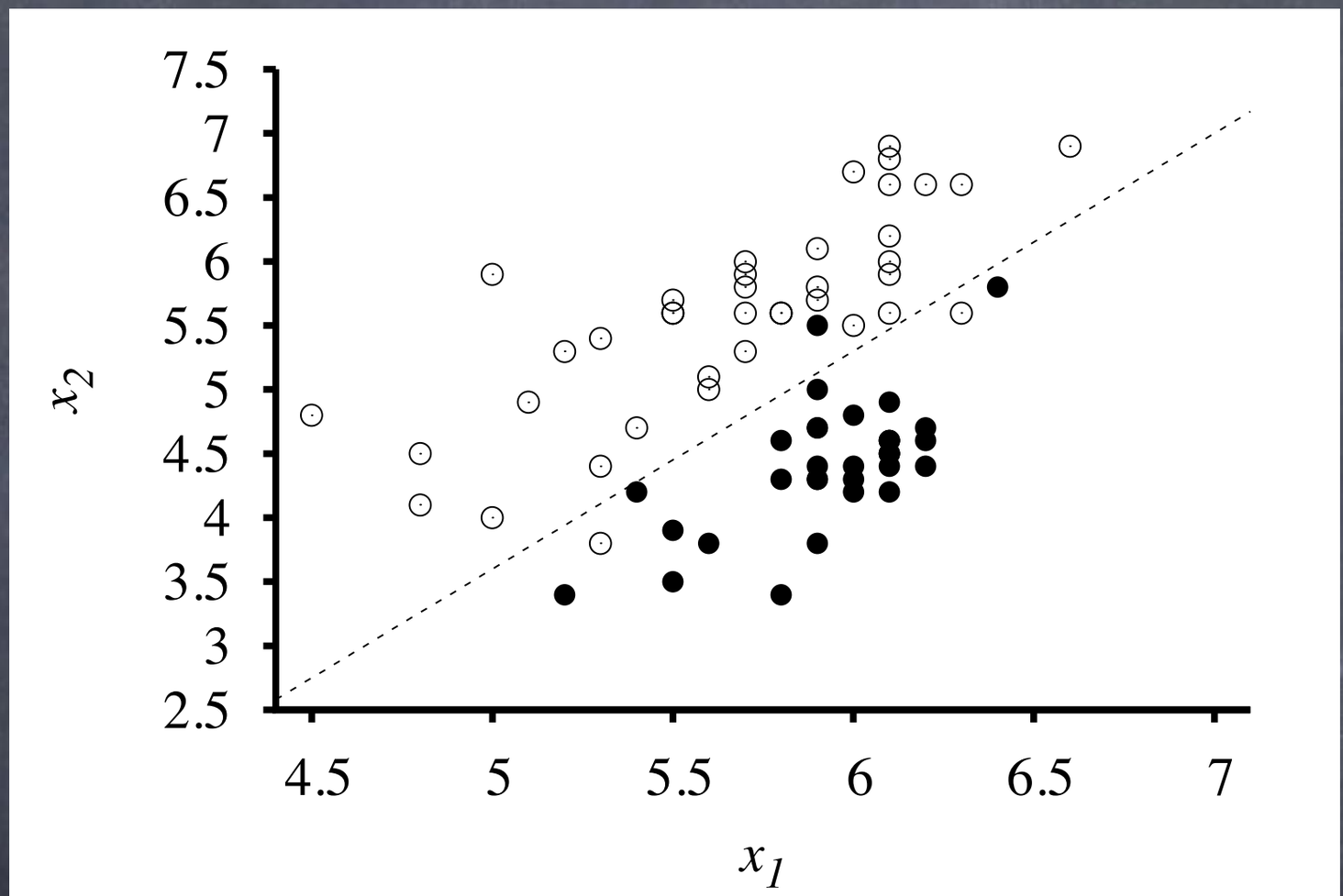
Hard Threshold



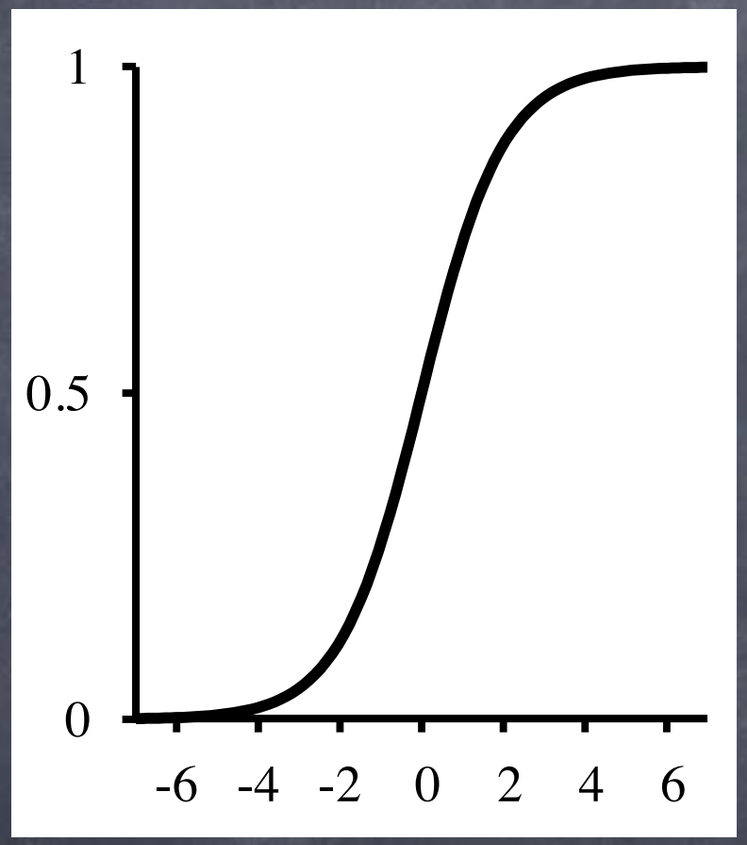
$$Threshold(z) = 1 \text{ if } z \geq 0$$
$$= 0 \text{ otherwise}$$

Linear Classifier with Hard Threshold

- Learn with gradient descent
  - Perceptron learning rule just like linear regression
- Convergence "isn't pretty but it works"


7.5
7
6.5
6
5.5
5
4.5
4
3.5
3
2.5
x_2
4.5
5
5.5
6
6.5
7
x_1

Soft Threshold



$$Logistic(z) = \frac{1}{1 + e^{-z}}$$

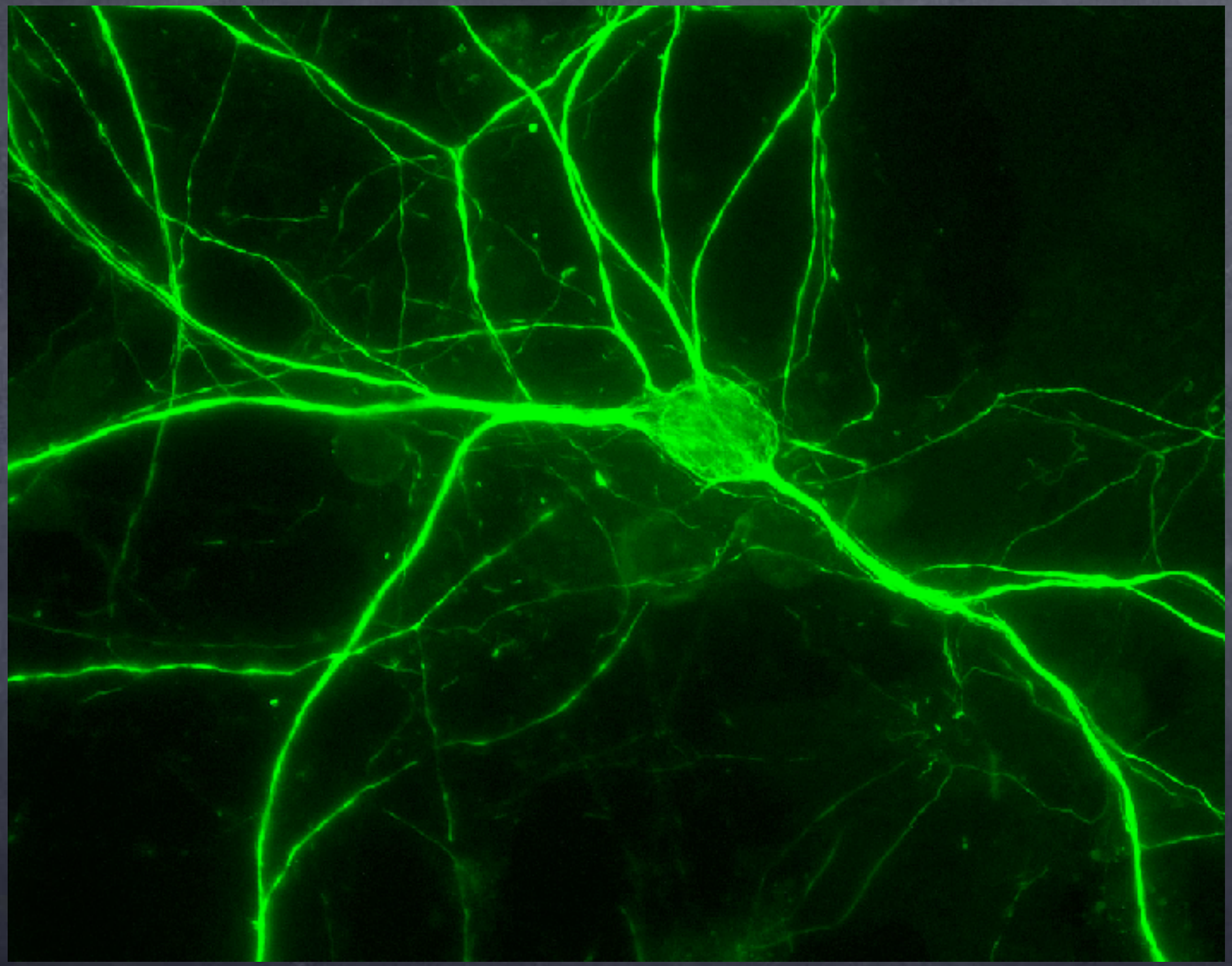
Linear Classifier with Logistic Threshold

$$h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

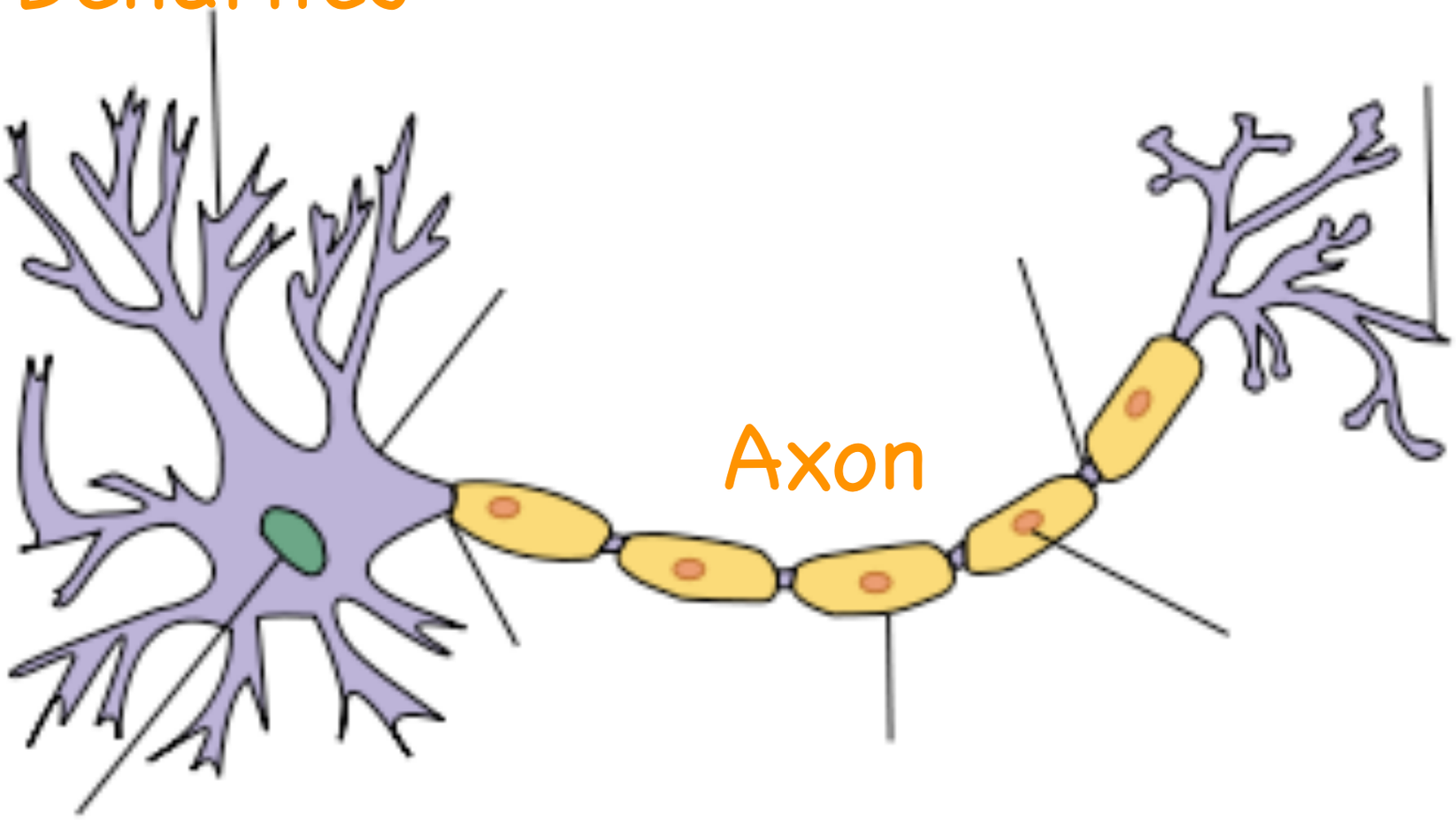
$$\begin{aligned}
\mathbf{w}^* &= \operatorname*{argmin}_{\mathbf{w}} L(h_{\mathbf{w}}) \\
&= \operatorname*{argmin}_{\mathbf{w}} L(Logistic(\mathbf{w} \cdot \mathbf{x})) \\
&= \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{N} L_2(y_i, Logistic(\mathbf{w} \cdot \mathbf{x})) \\
&= \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{N} (y_i - \frac{1}{1 - e^{-\mathbf{w} \cdot \mathbf{x}_i}})^2
\end{aligned}$$

Learning Linear Classifiers

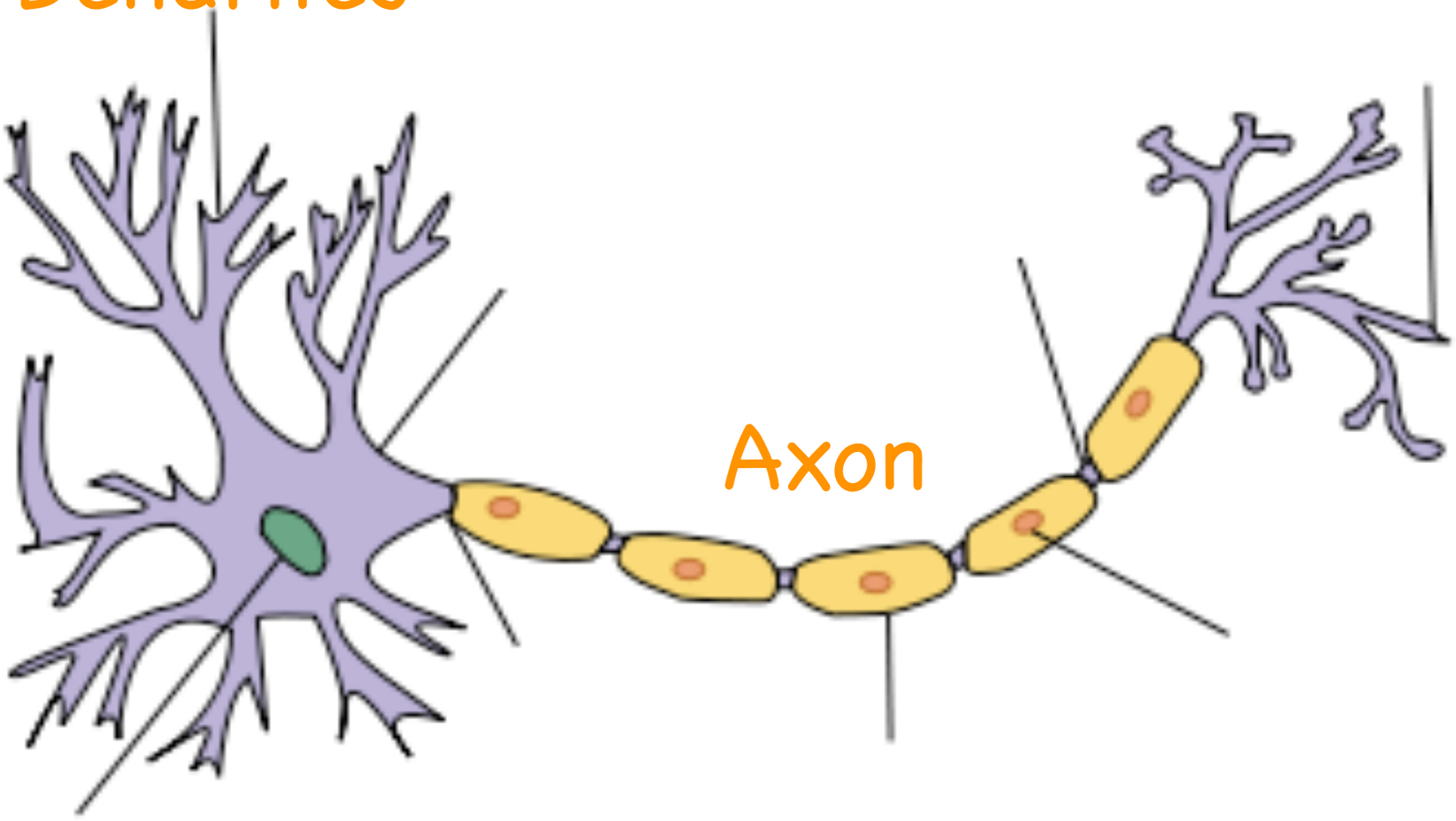
- Can learn linear classifiers from data
  - Based on linear regression
- Hard threshold
  - Unpredictable, not robust to noise
- Soft threshold: logistic regression
  - Slower, but more predictable
  - Robust to noisy data



Dendrites

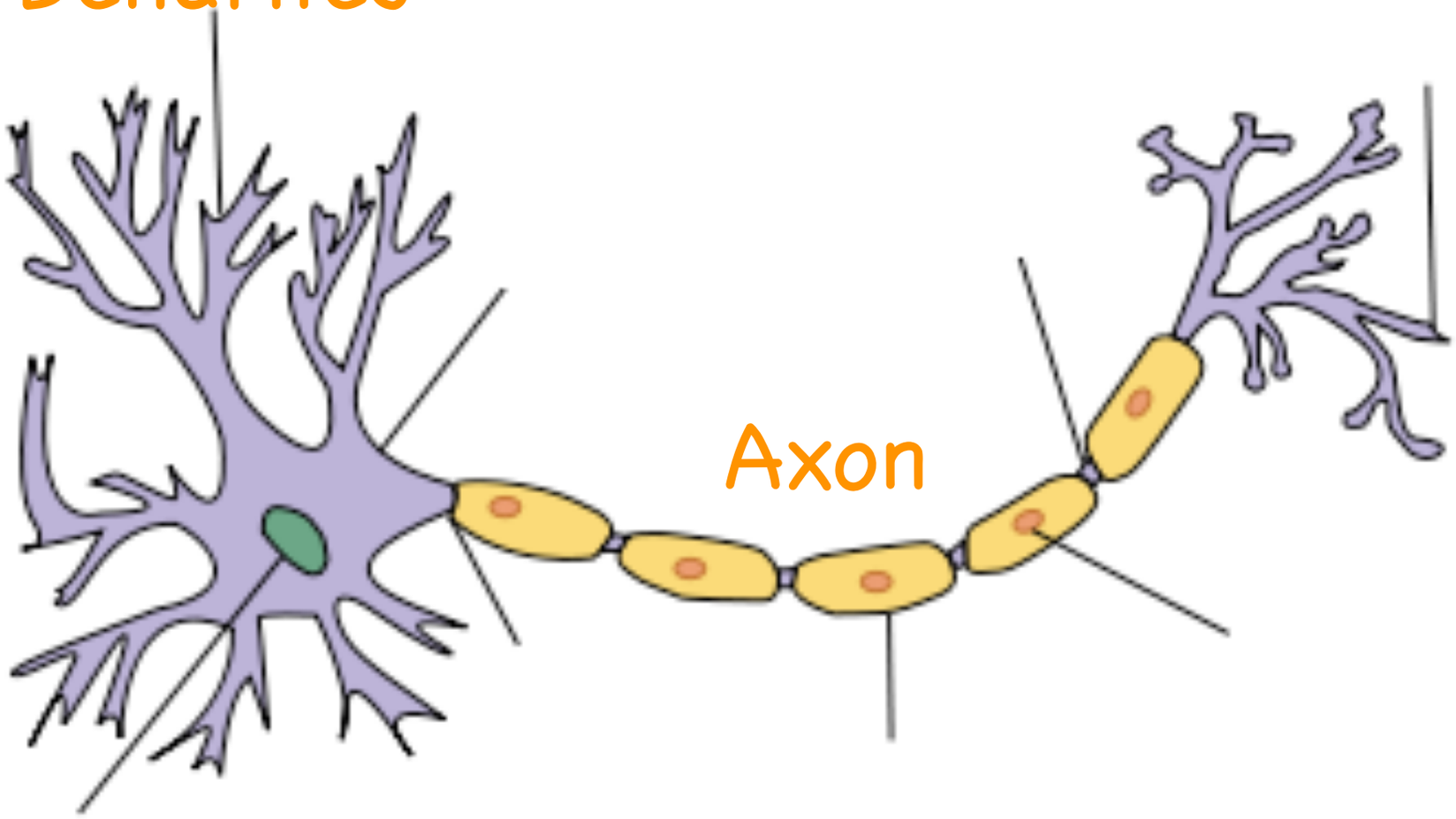


Dendrites



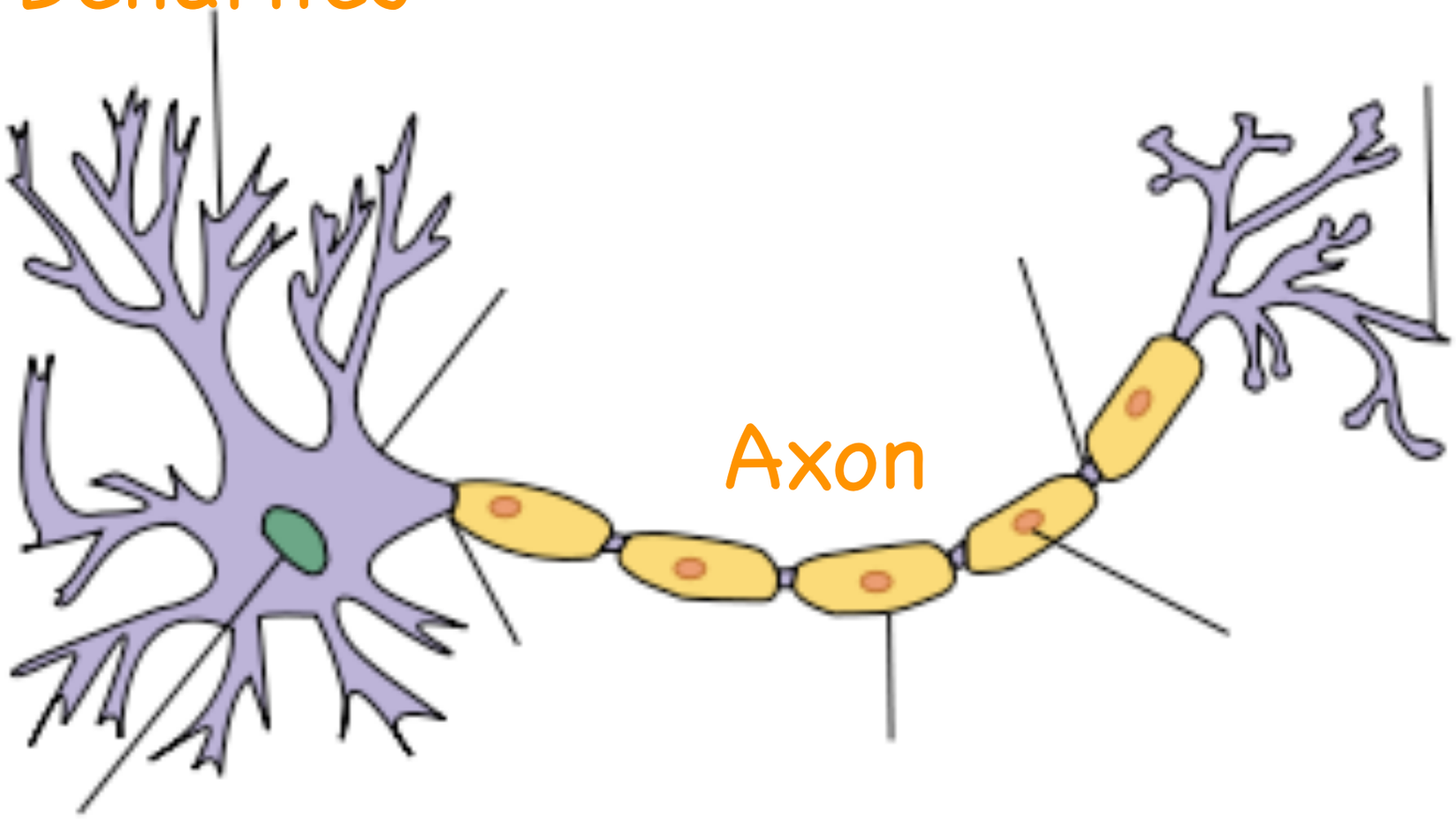
$\sum i n_i$

Dendrites



$$\Sigma in_i > threshold$$

Dendrites



$$\Sigma in_i > threshold \Rightarrow output = 1$$

Linear Classifier

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

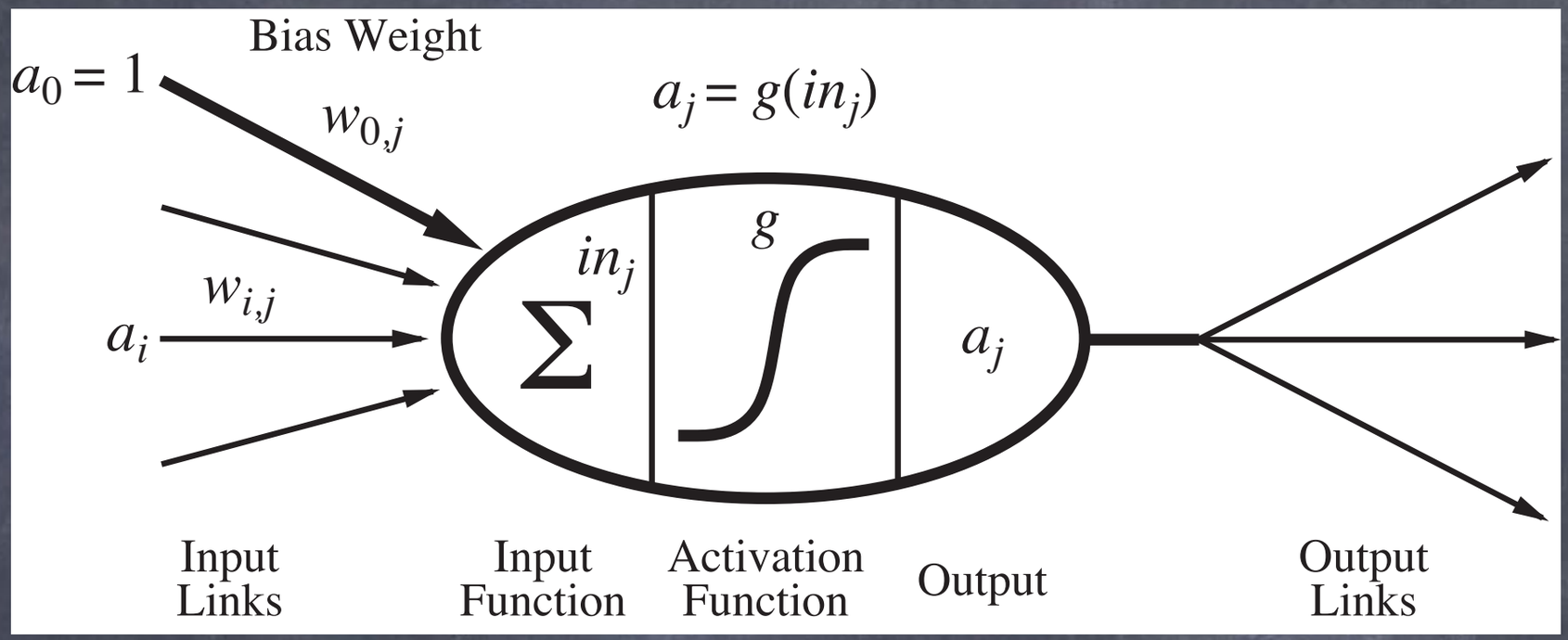
$$\mathbf{w} \cdot \mathbf{x} = 0$$

All instances of one class are above the line: $\mathbf{w} \cdot \mathbf{x} > 0$

All instances of one class are below the line: $\mathbf{w} \cdot \mathbf{x} < 0$

$$h_{\mathbf{w}}(\mathbf{x}) = \text{Threshold}(\mathbf{w} \cdot \mathbf{x})$$

"Neuron"

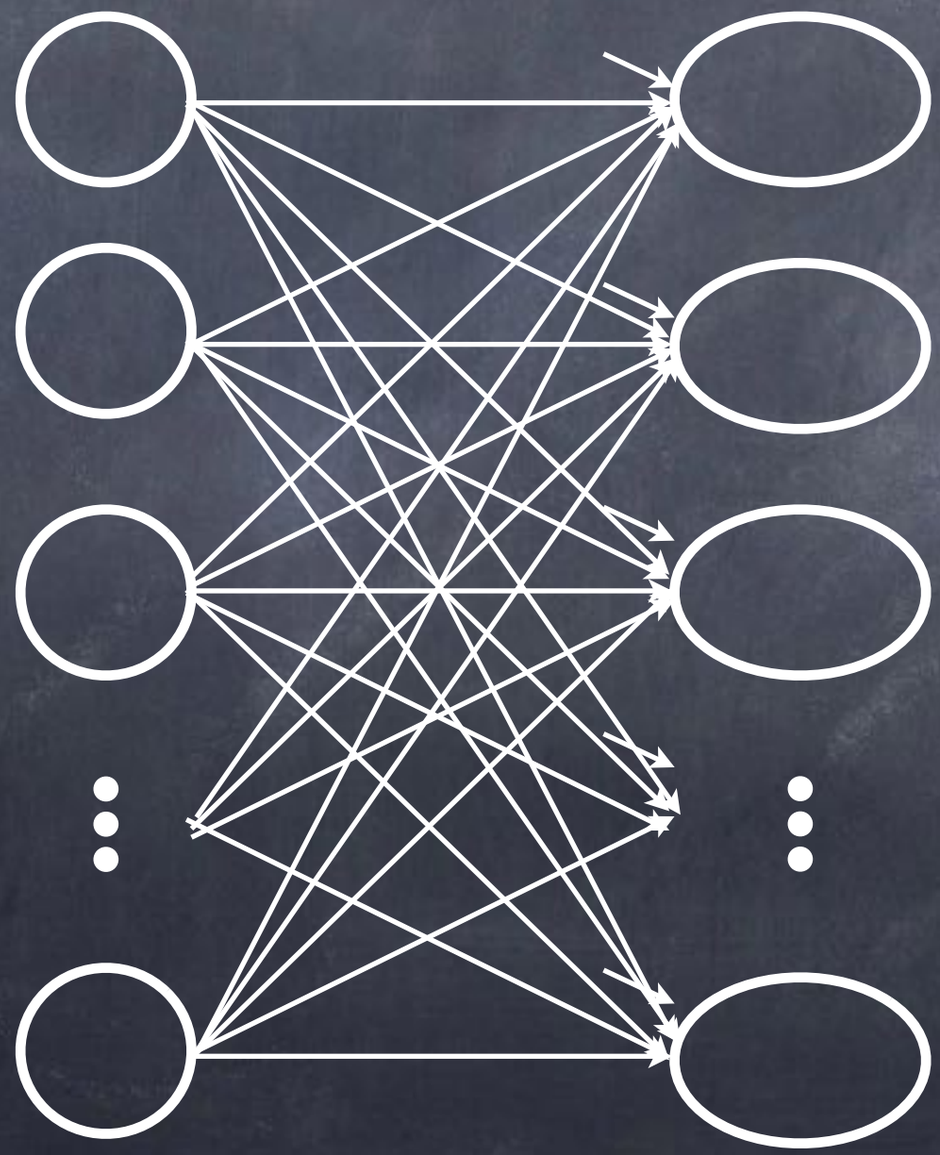


$$a_j = g(\sum_i w_{i,j} a_i)$$

Neural Network

- Graph of linear classifiers with logistic thresholds ("units")
- Outputs connected to inputs

Single-Layer Feed-Forward NNs

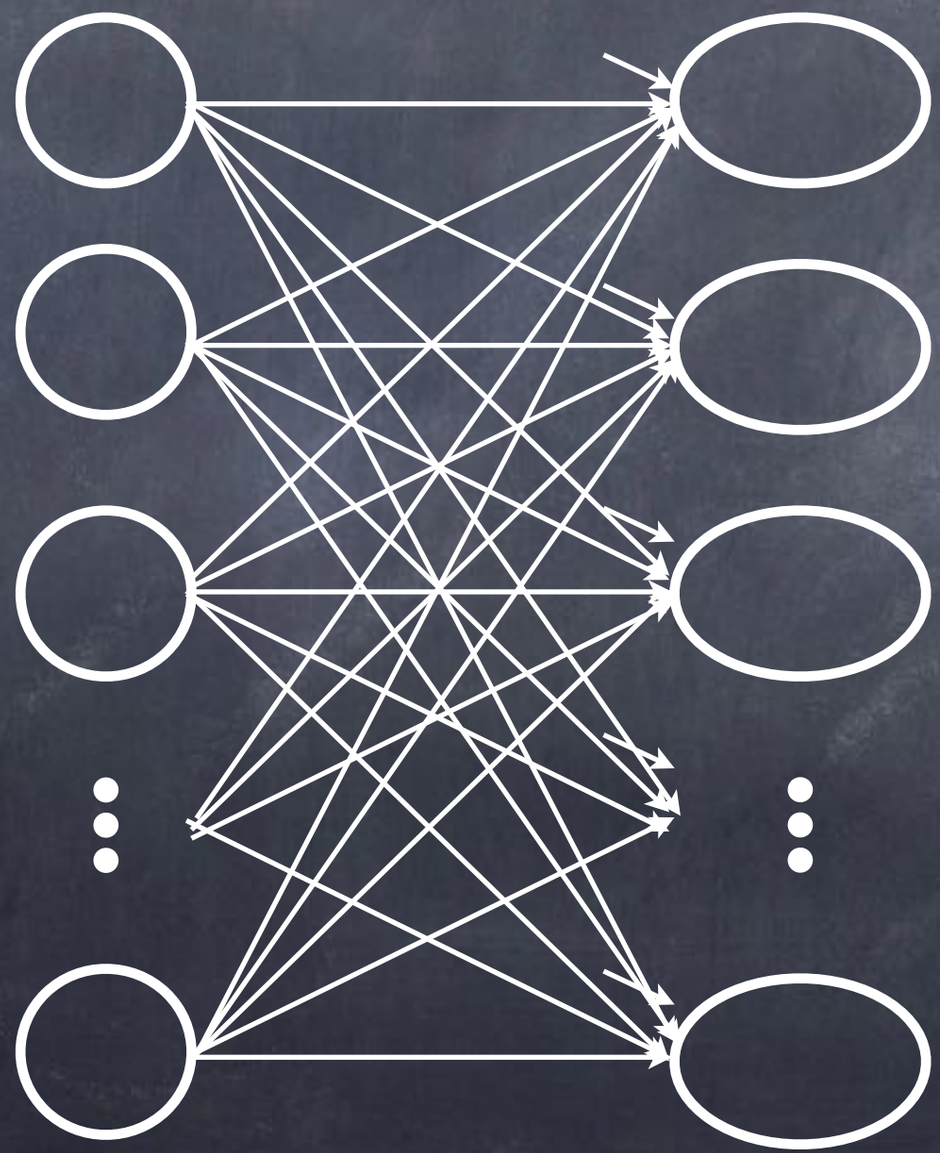


Single-Layer Feed-Forward NNs

Input

Attributes

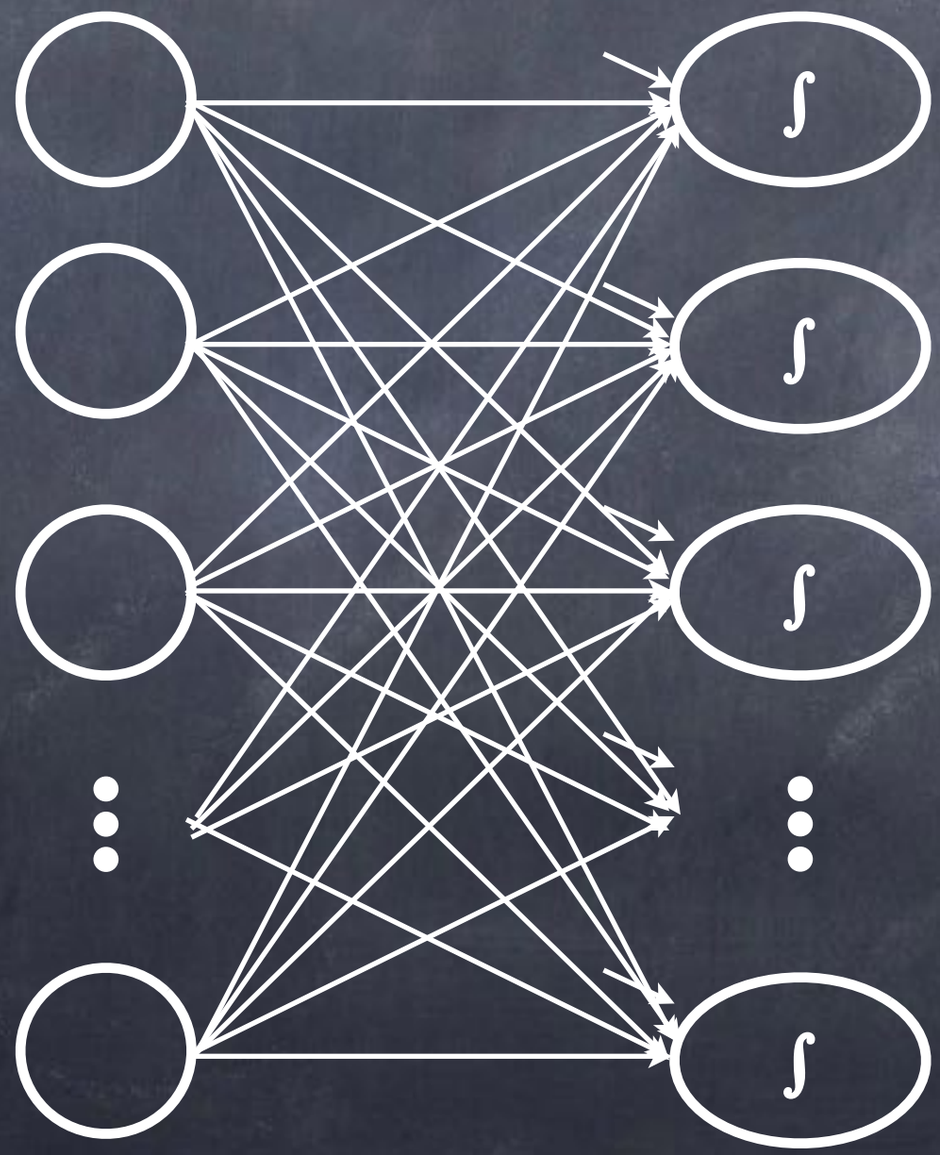
x_i



Single-Layer Feed-Forward NNs

Input

Output

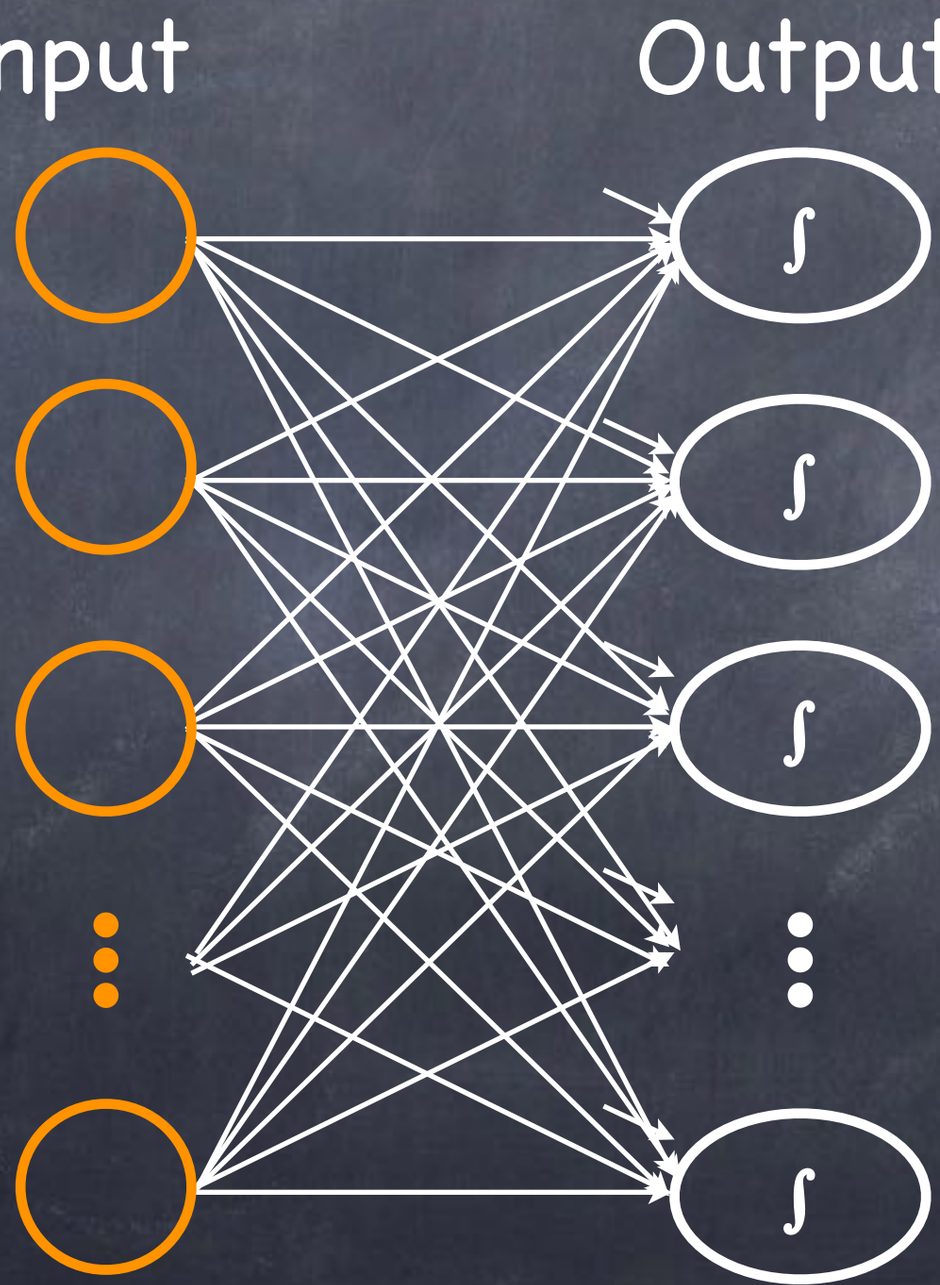


Attributes

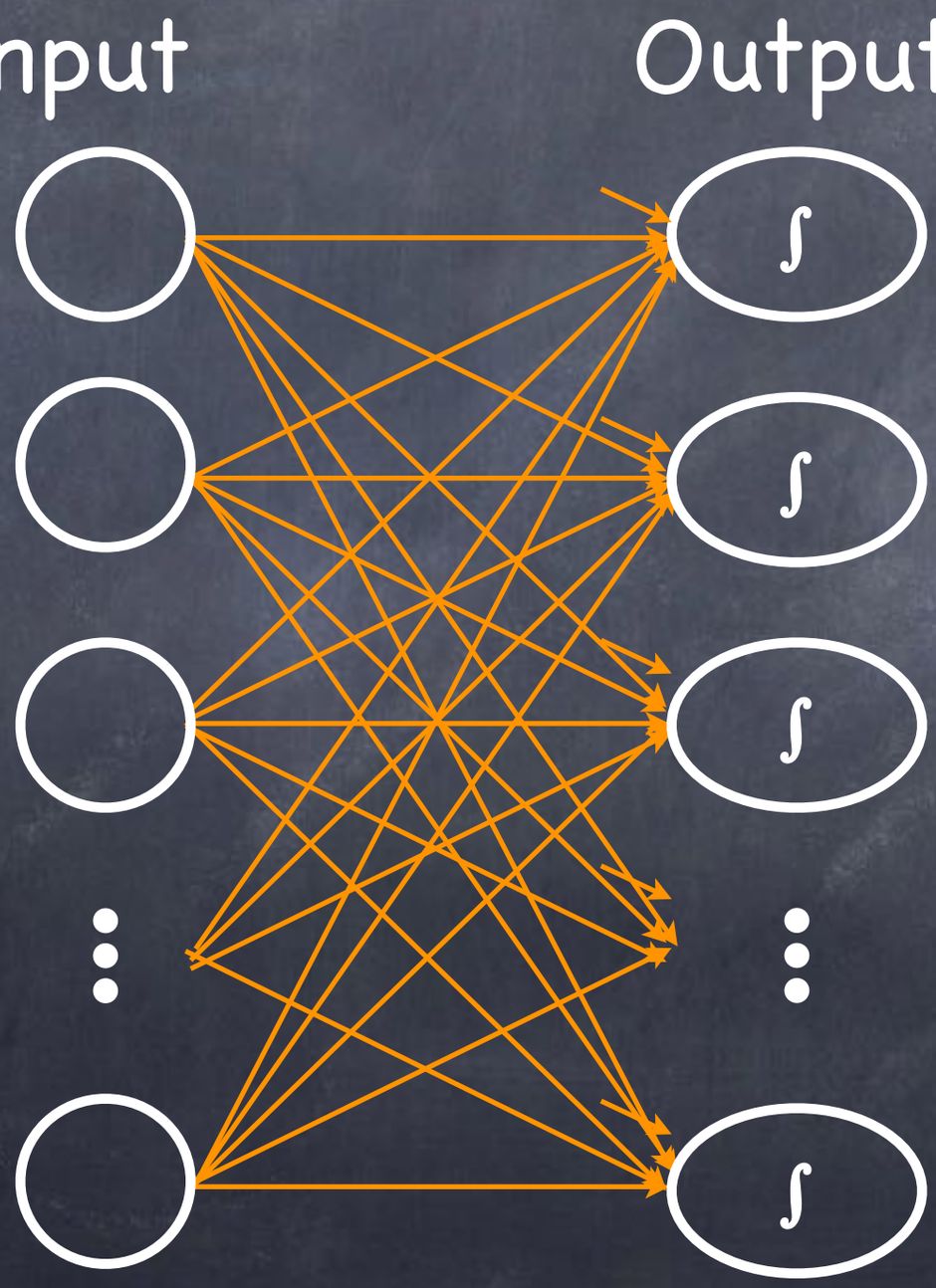
x_i

Classes

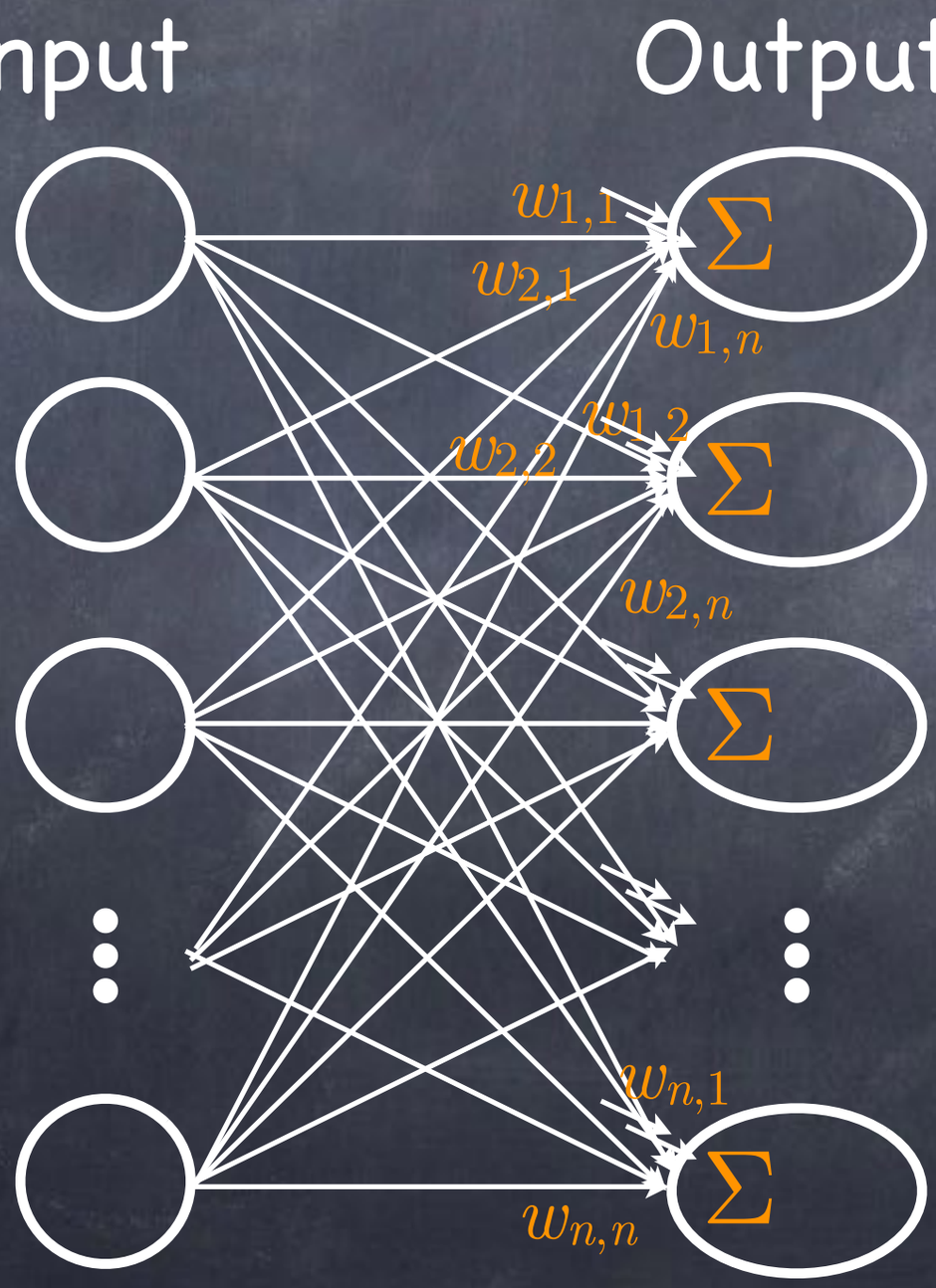
Single-Layer Feed-Forward NNs



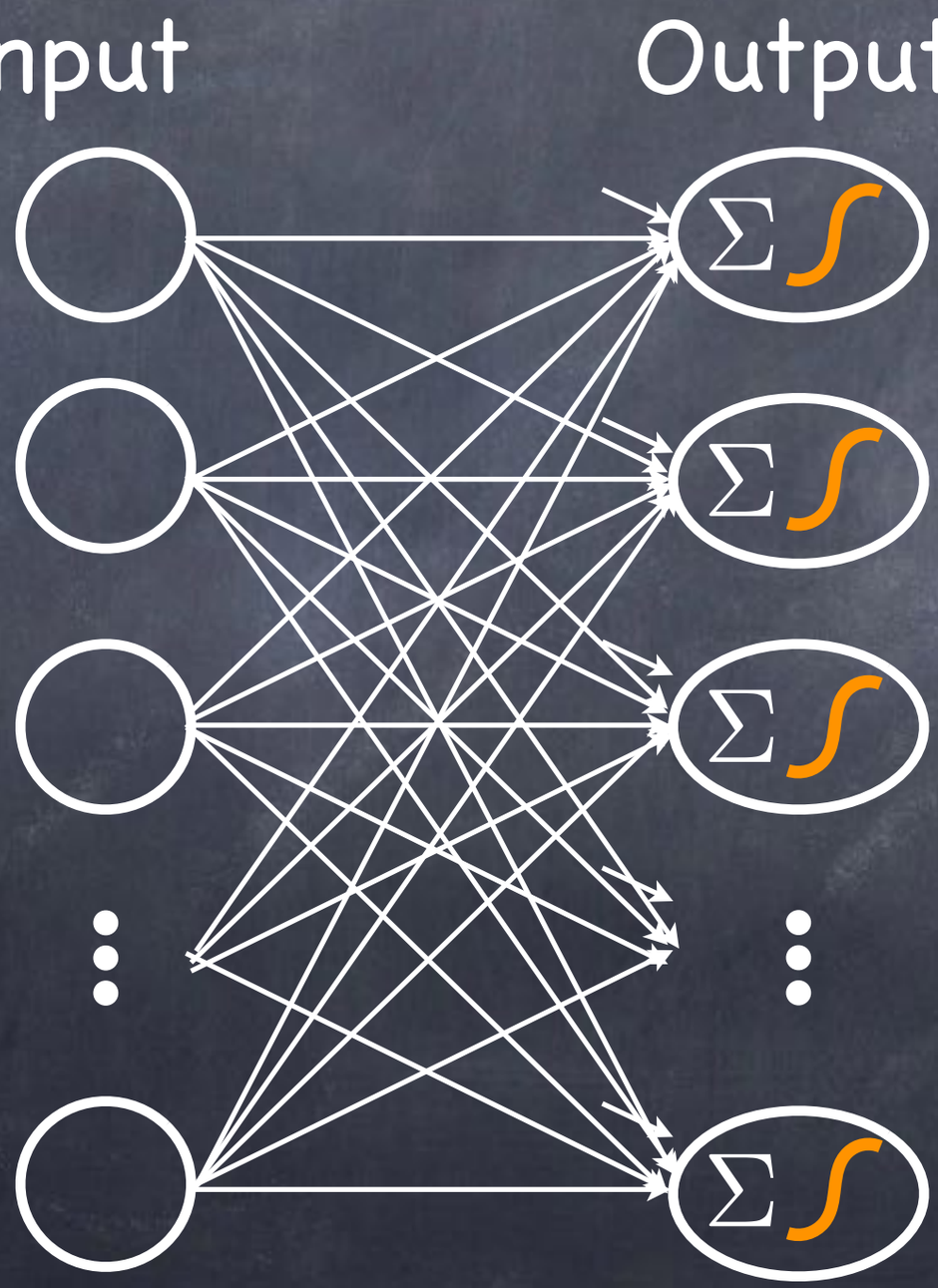
Single-Layer Feed-Forward NNs



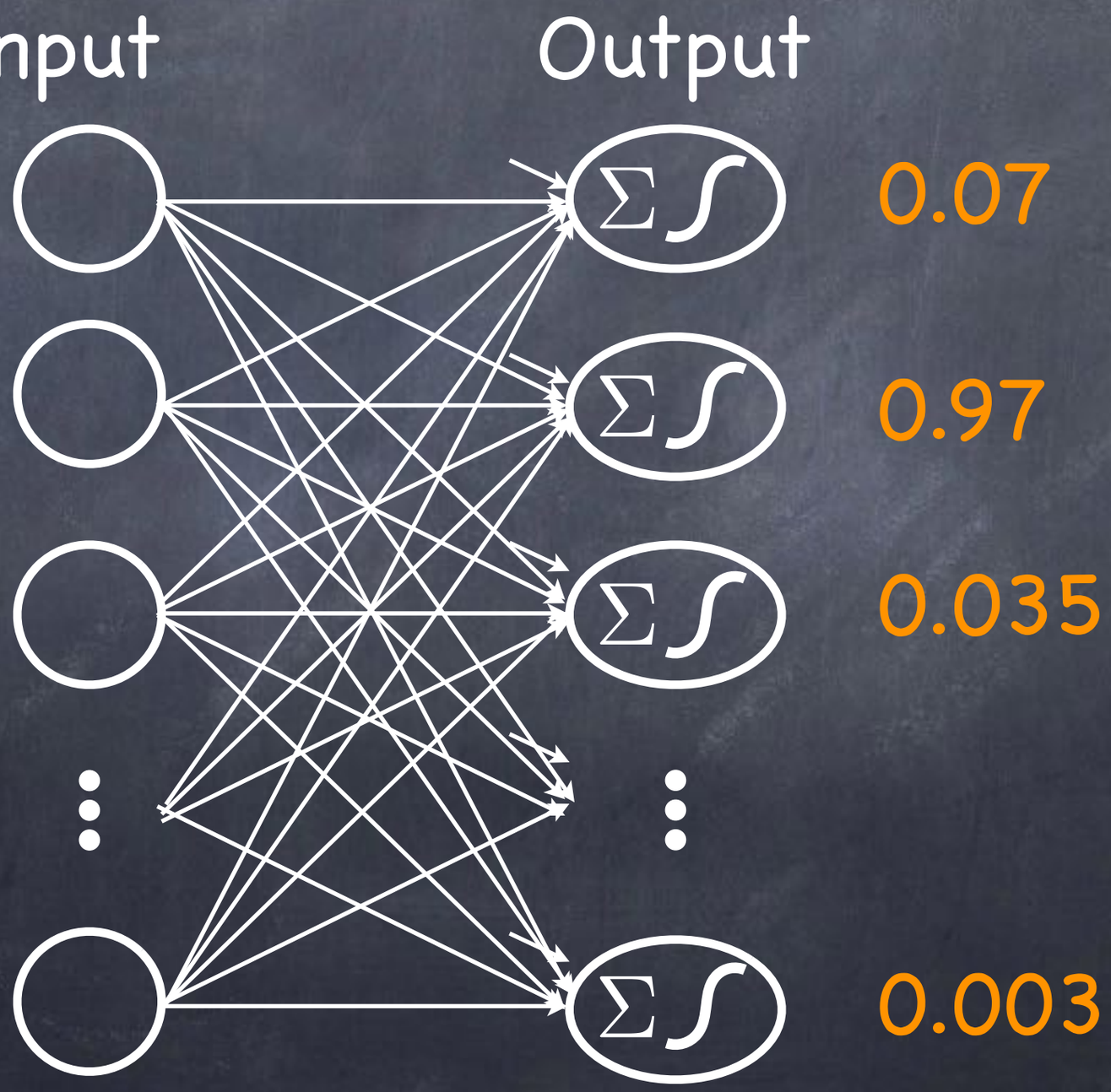
Single-Layer Feed-Forward NNs



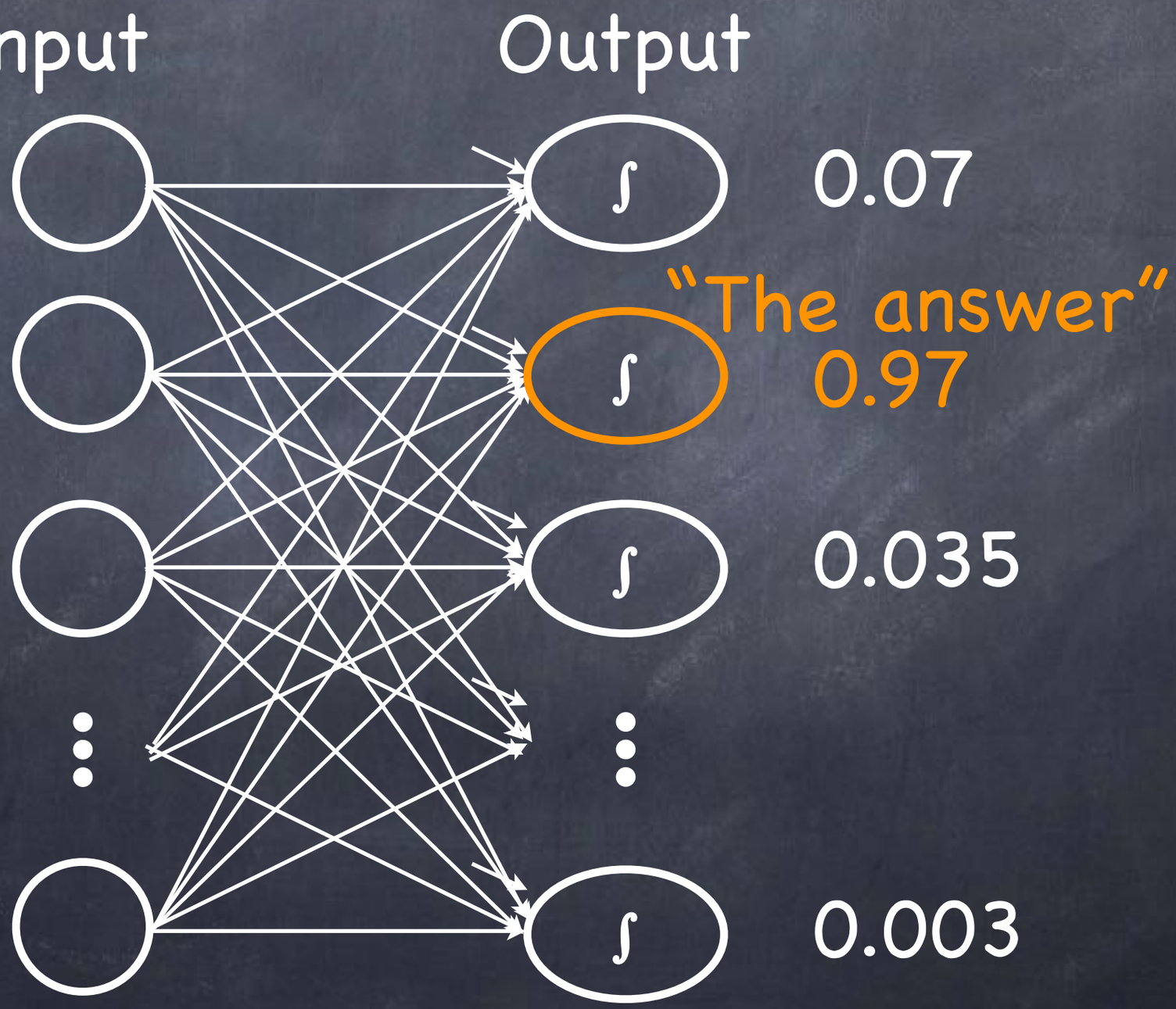
Single-Layer Feed-Forward NNs



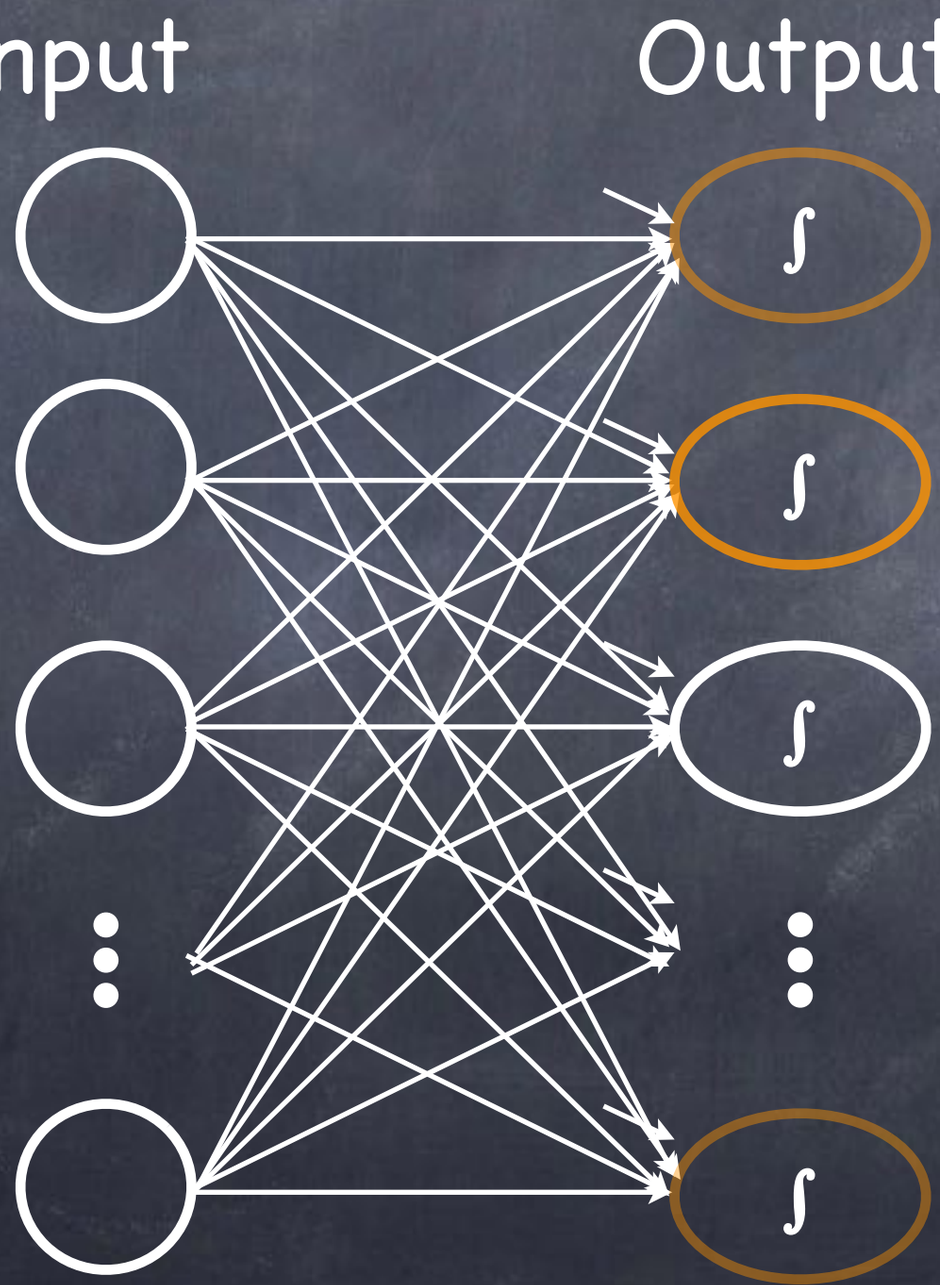
Single-Layer Feed-Forward NNs



Single-Layer Feed-Forward NNs

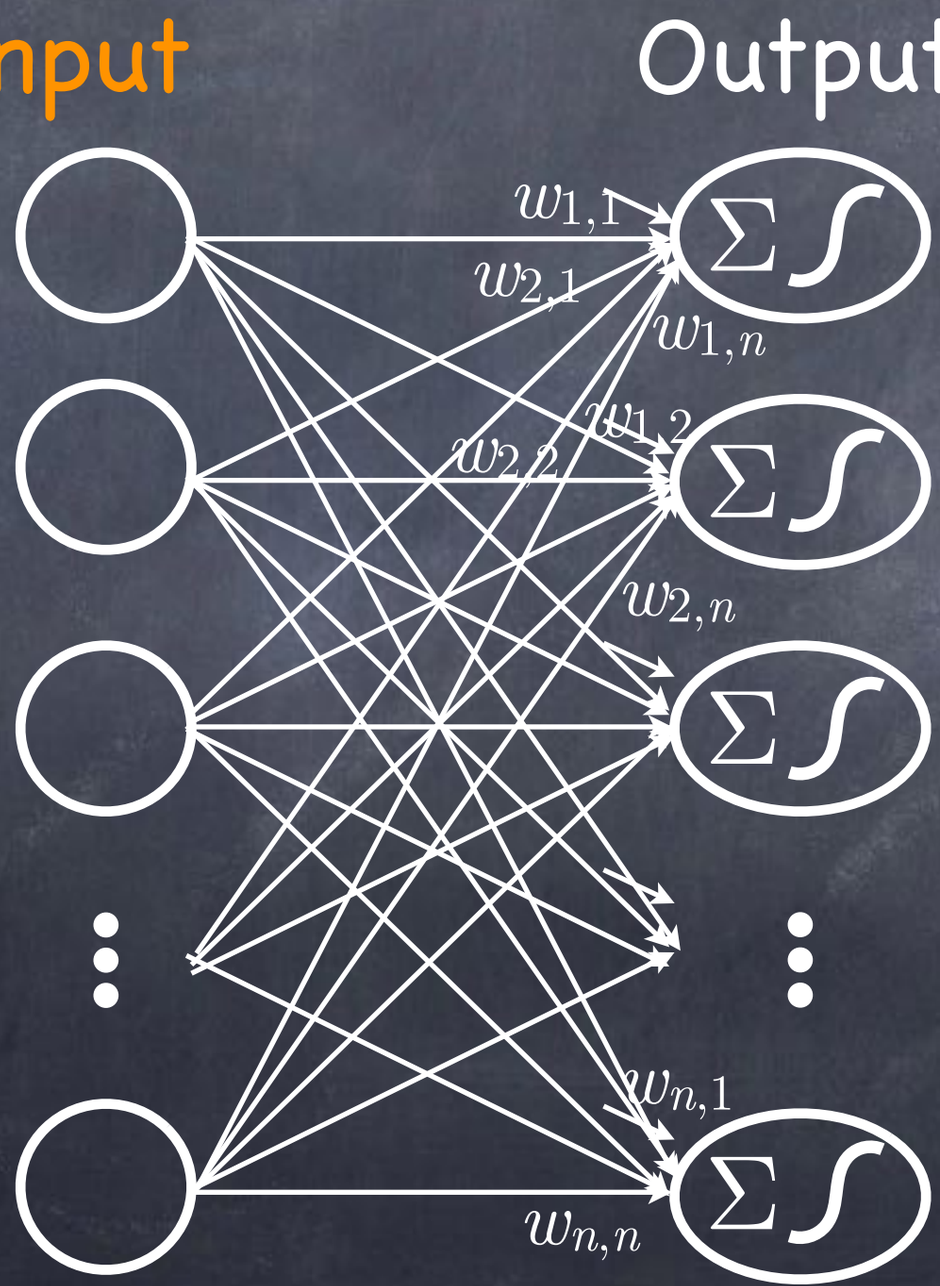


Single-Layer Feed-Forward NNs

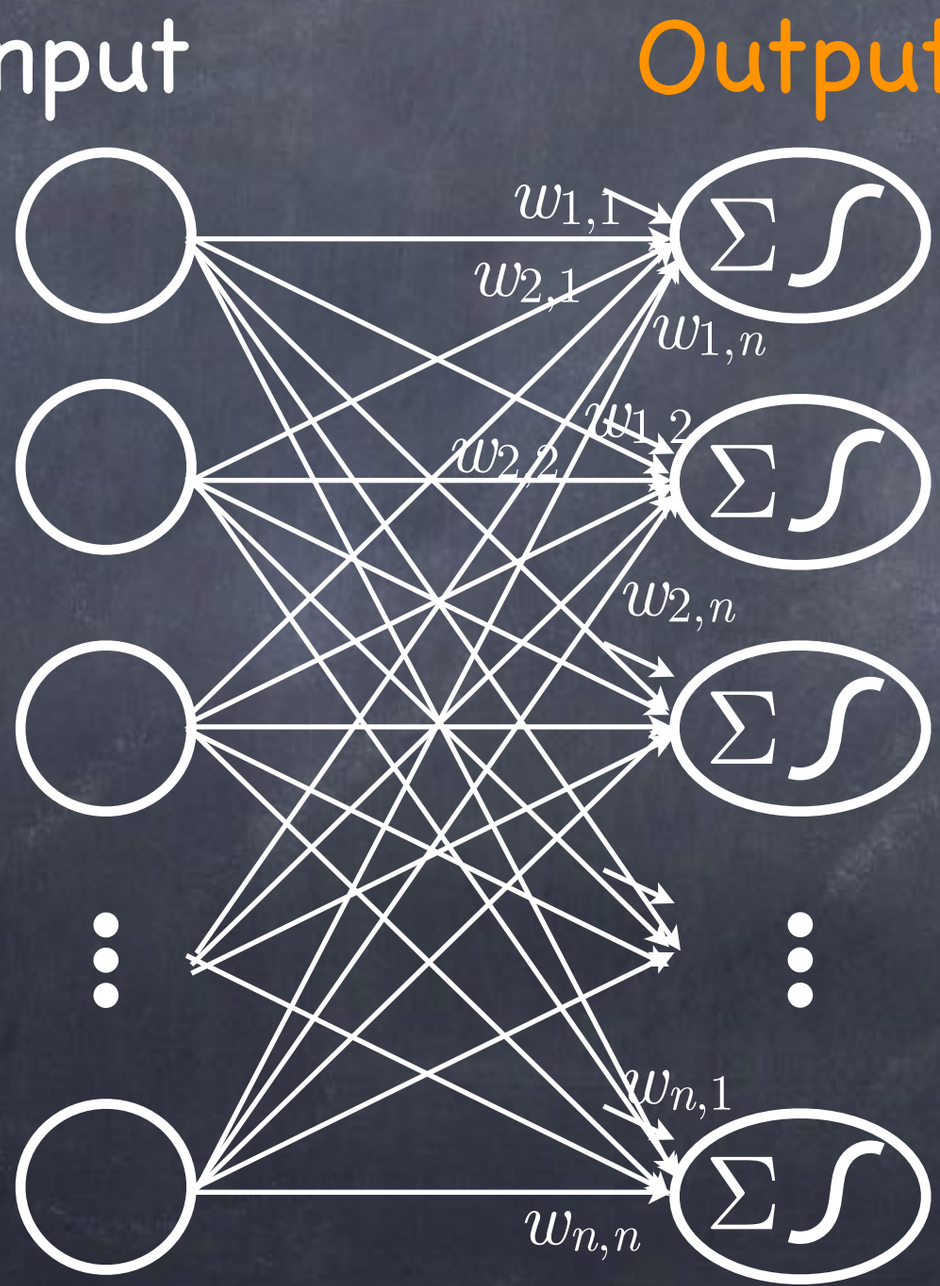


???

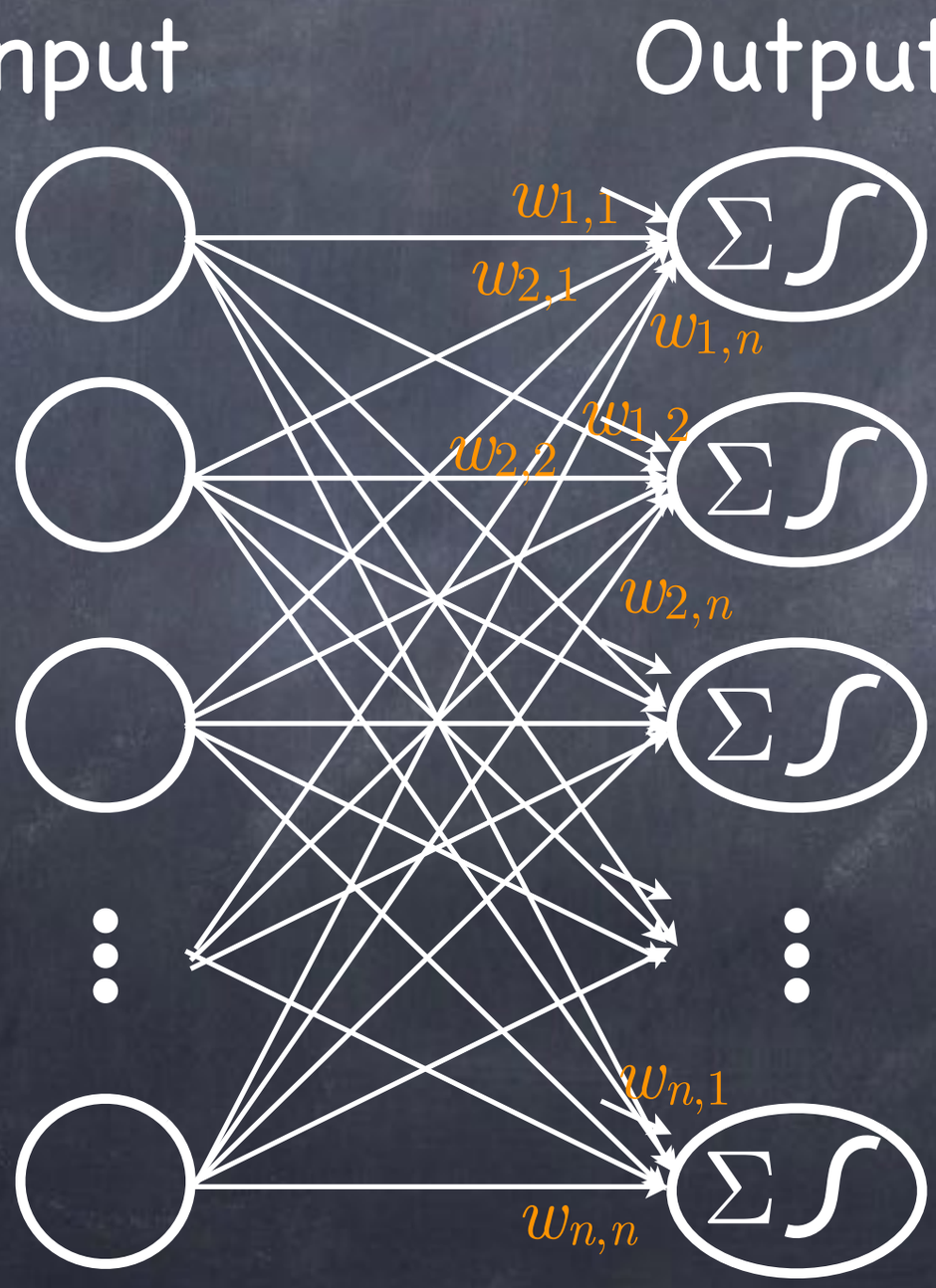
Single-Layer Feed-Forward NNs



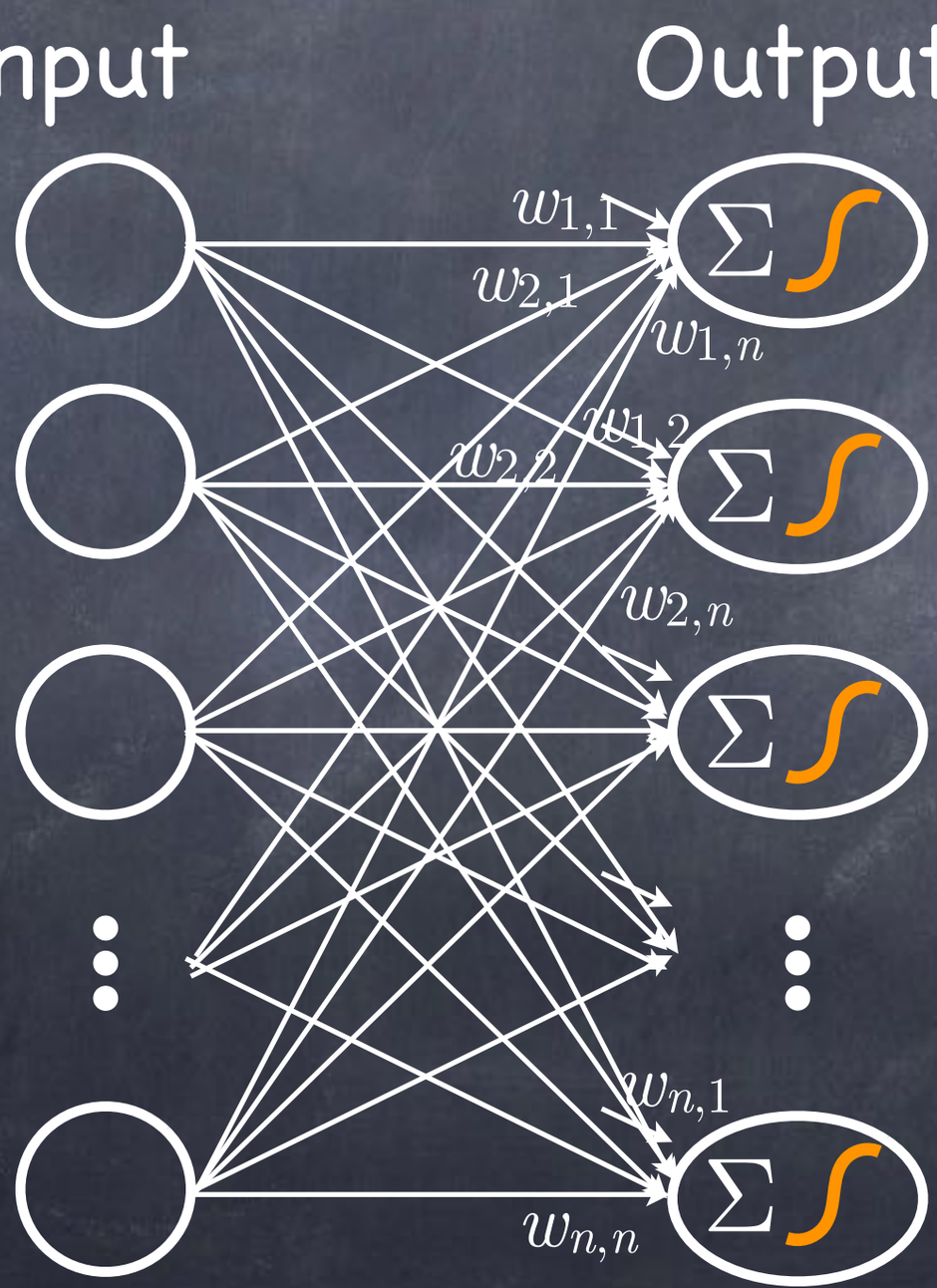
Single-Layer Feed-Forward NNs



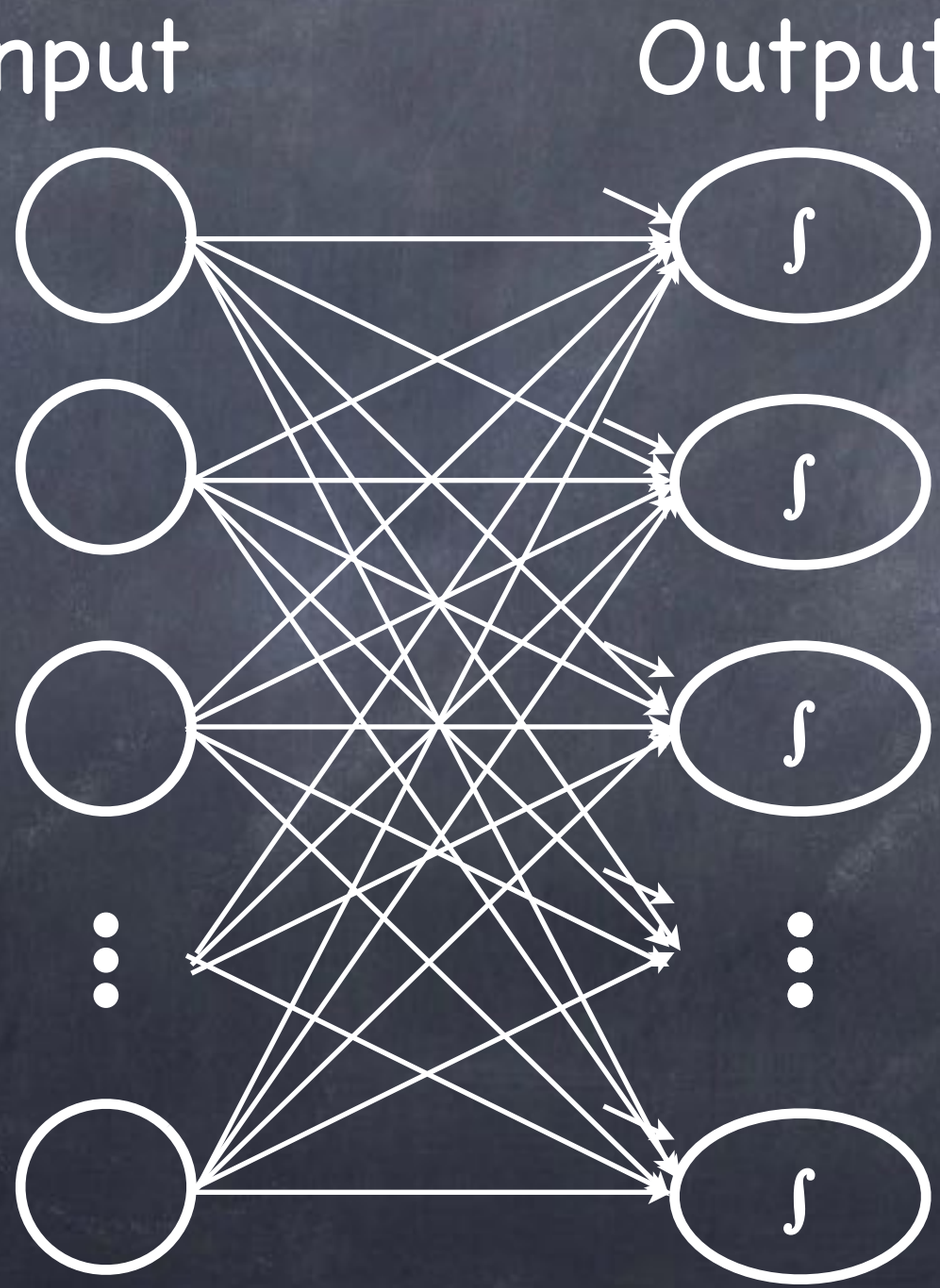
Single-Layer Feed-Forward NNs



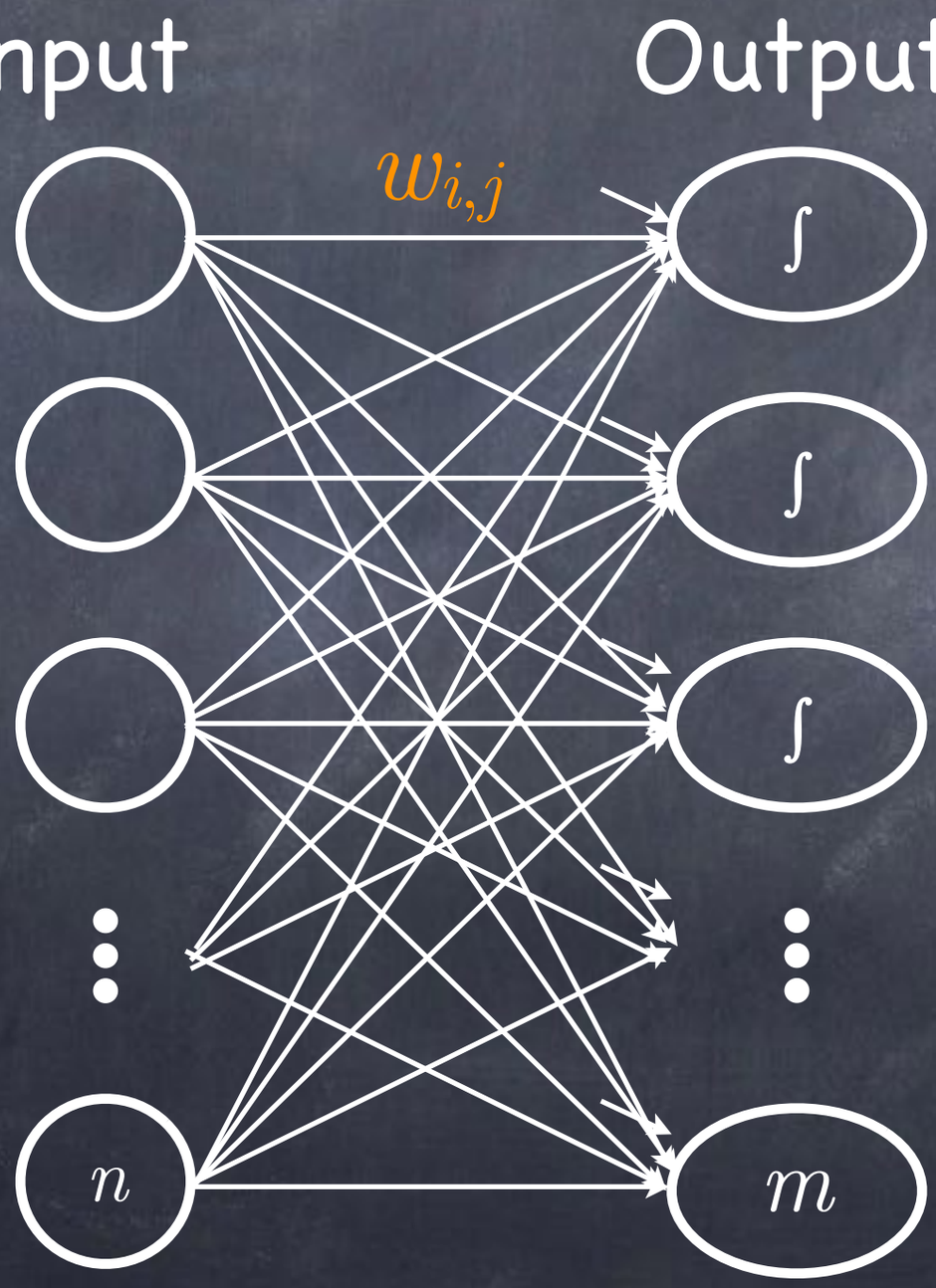
Single-Layer Feed-Forward NNs



Single-Layer Feed-Forward NNs



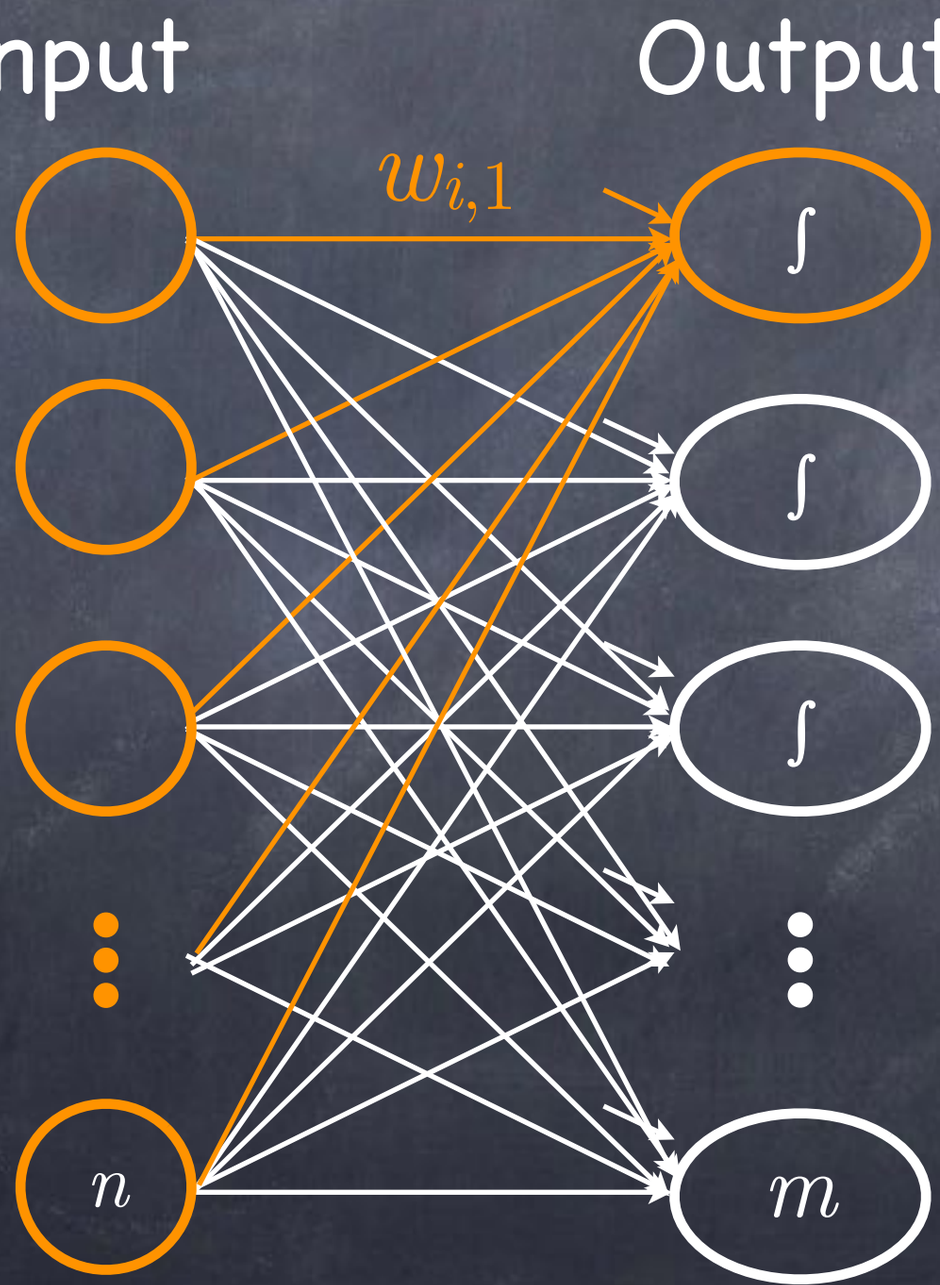
Single-Layer Feed-Forward NNs



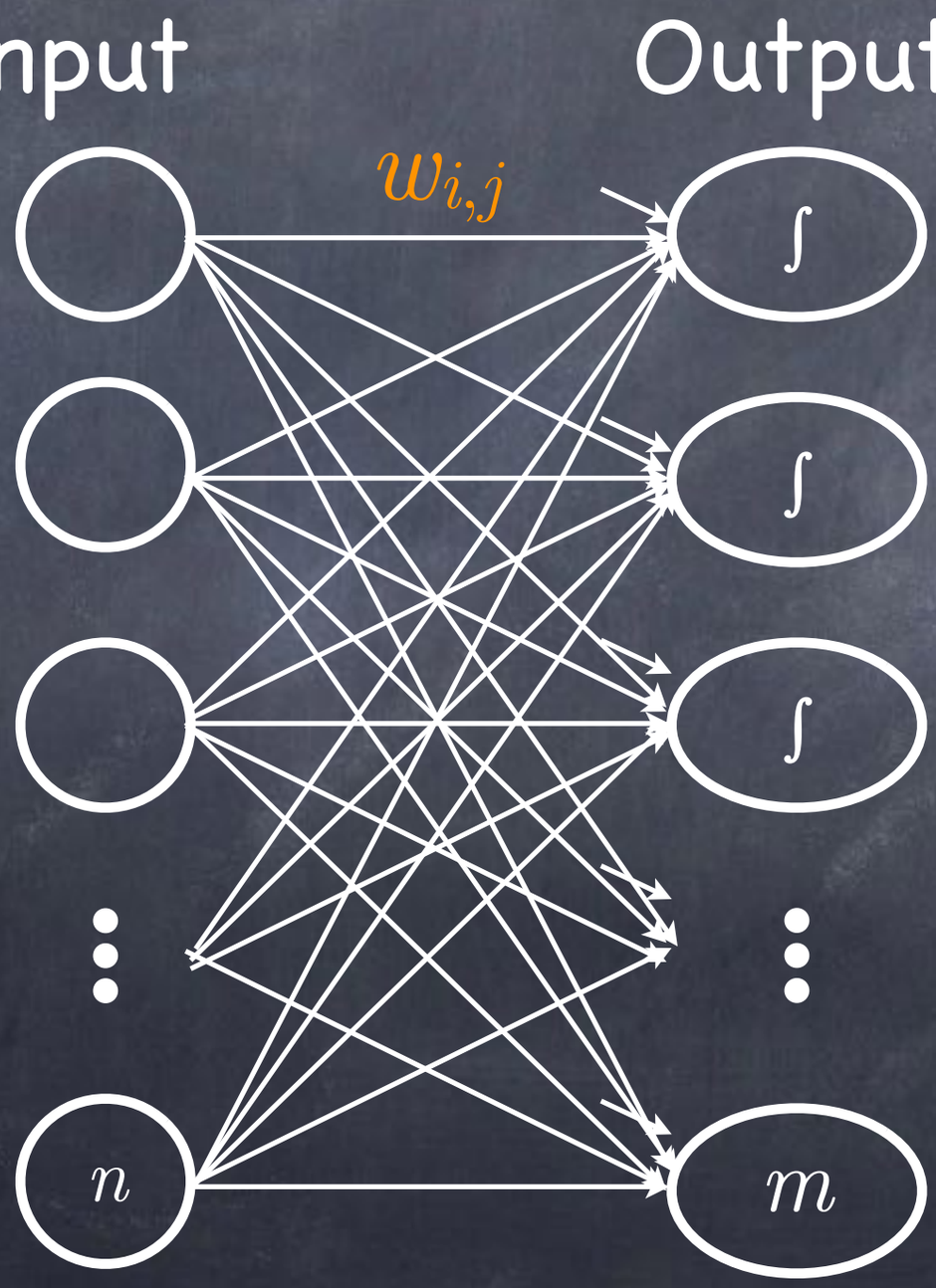
$m*(n+1)$

weights

Learning in Single-Layer Feed-Forward NNs



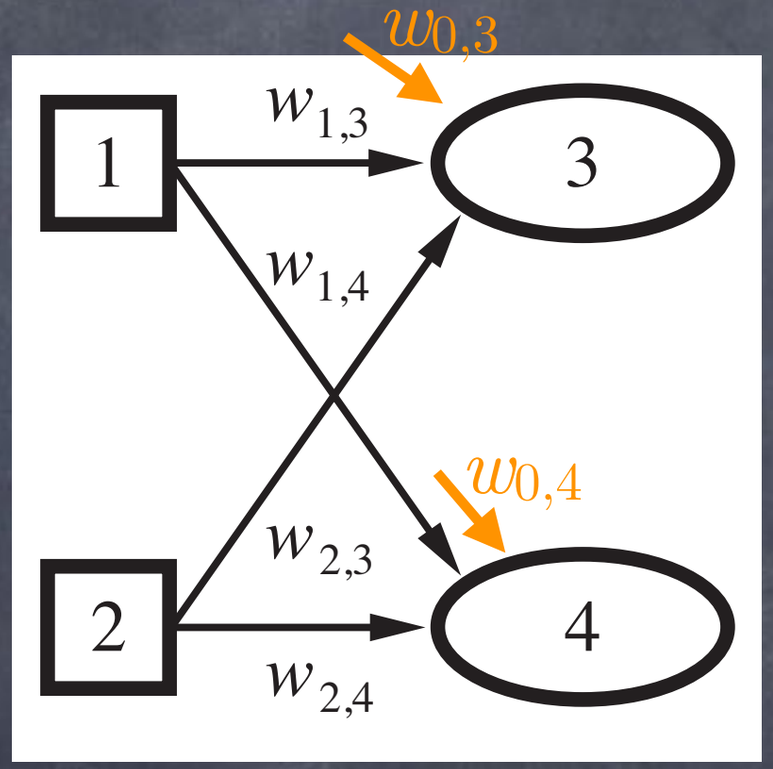
Learning in Single-Layer Feed-Forward NNs



Learn m linear classifiers over n features

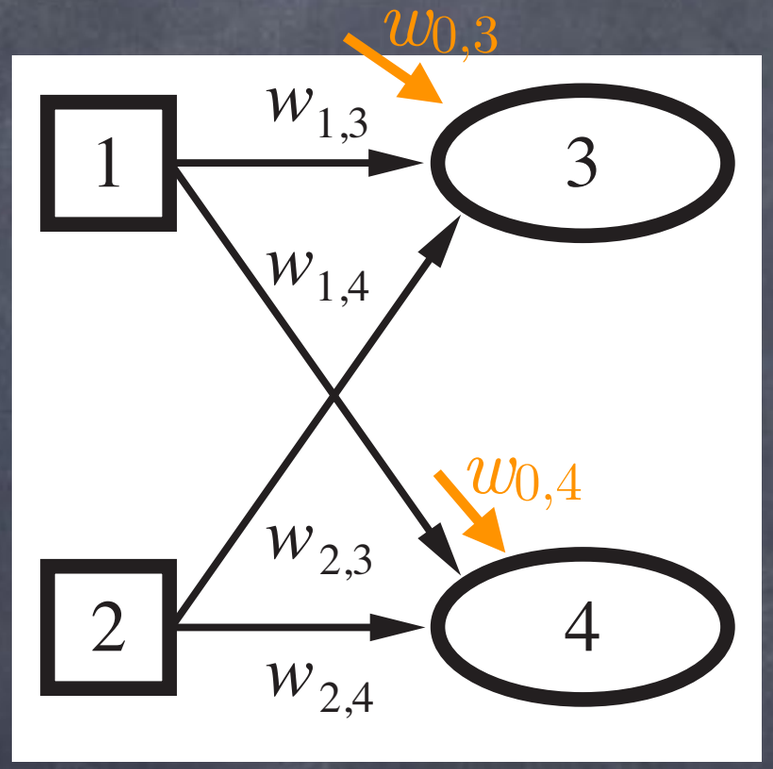
Learn using gradient descent

Learning in Single-Layer Feed-Forward NN



$$a_j = g(\sum_i w_{i,j} a_i) = g(w_{0,j} + w_{1,j} a_1 + w_{2,j} a_2)$$

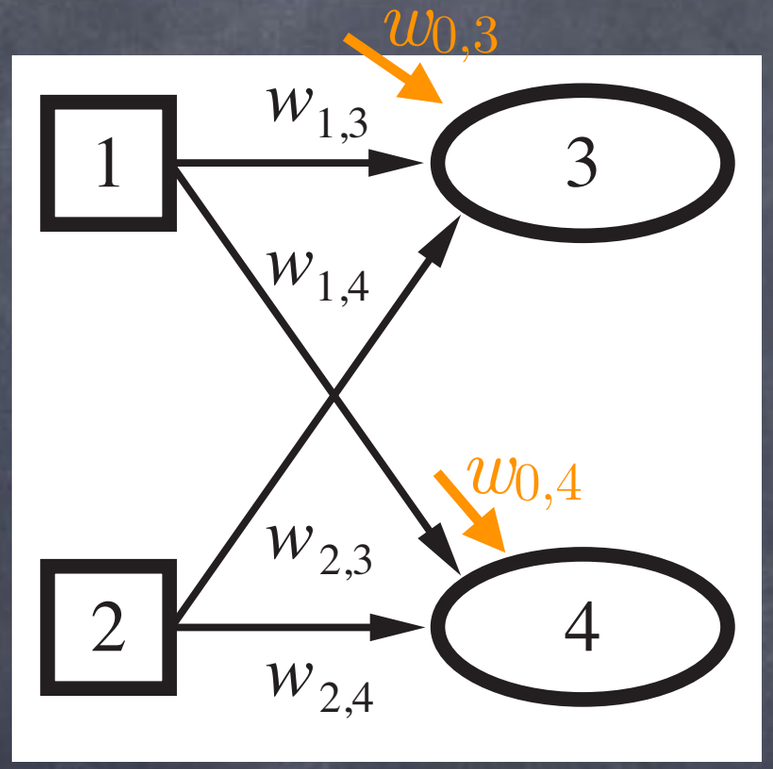
Learning in Single-Layer Feed-Forward NN



$$a_3 = g(\sum_i w_{i,3} a_i) = g(w_{0,3} + w_{1,3} a_1 + w_{2,3} a_2)$$

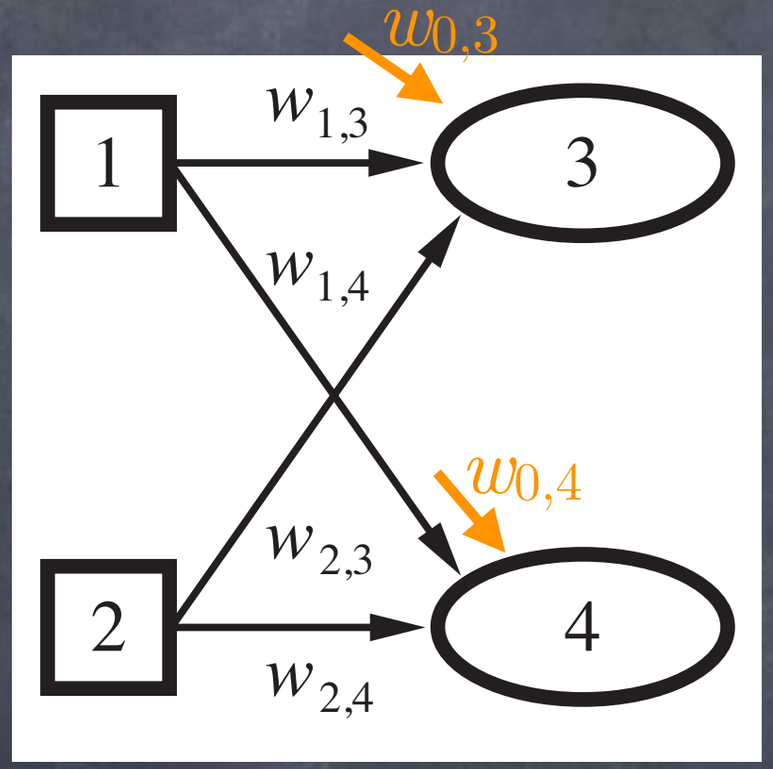
$$= g(w_{0,3} + w_{1,3} x_1 + w_{2,3} x_2)$$

Learning in Single-Layer Feed-Forward NN



$$a_j = g(\sum_i w_{i,j} a_i) = g(w_{0,j} + w_{1,j} a_1 + w_{2,j} a_2)$$

Learning in Single-Layer Feed-Forward NN



$$output_j = g(\sum_i w_{i,j} a_i) = g(w_{0,j} + w_{1,j} a_1 + w_{2,j} a_2)$$

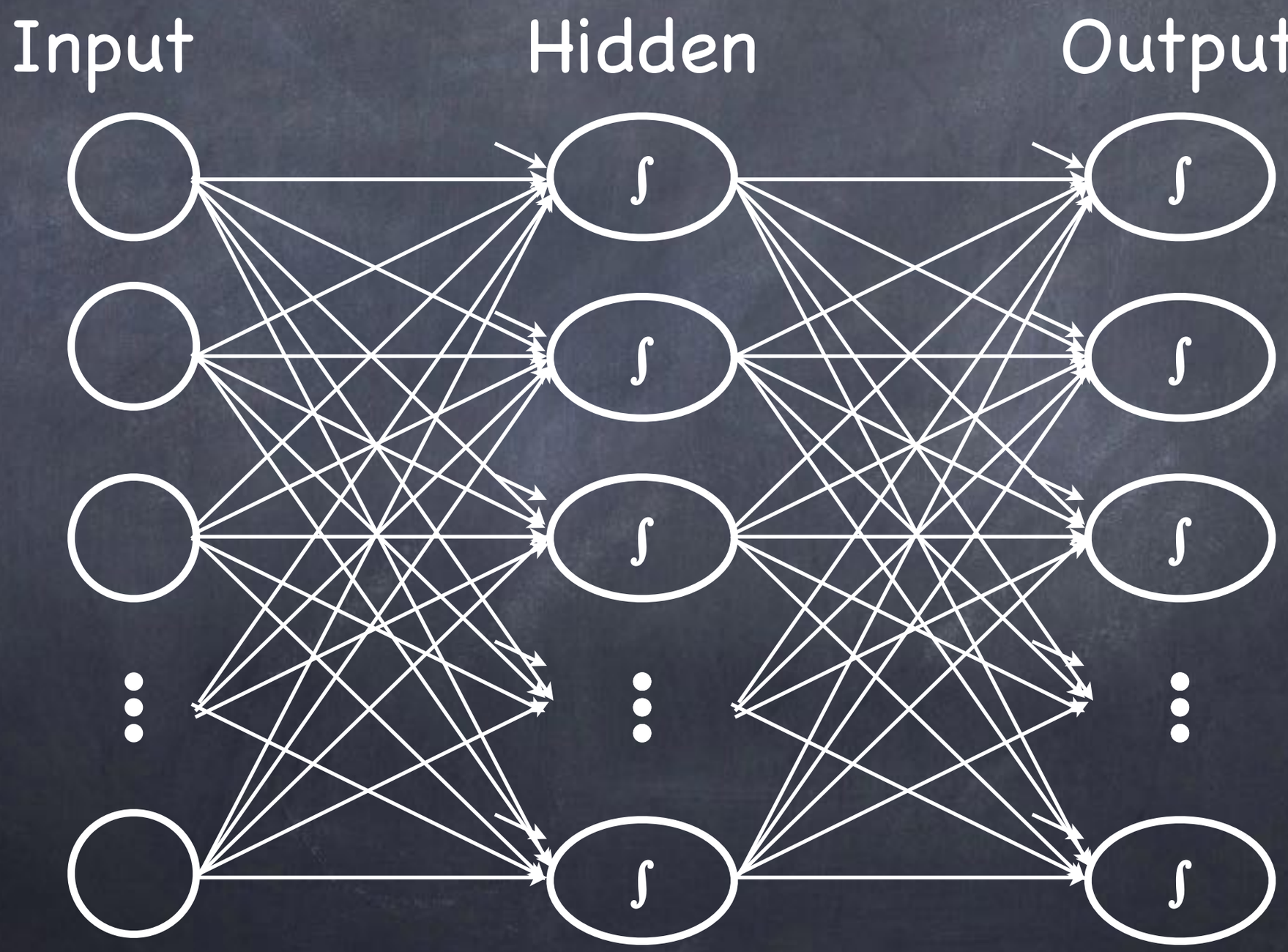
There are simple functions that cannot be learned by single-layer feed-forward networks

General Neural Networks

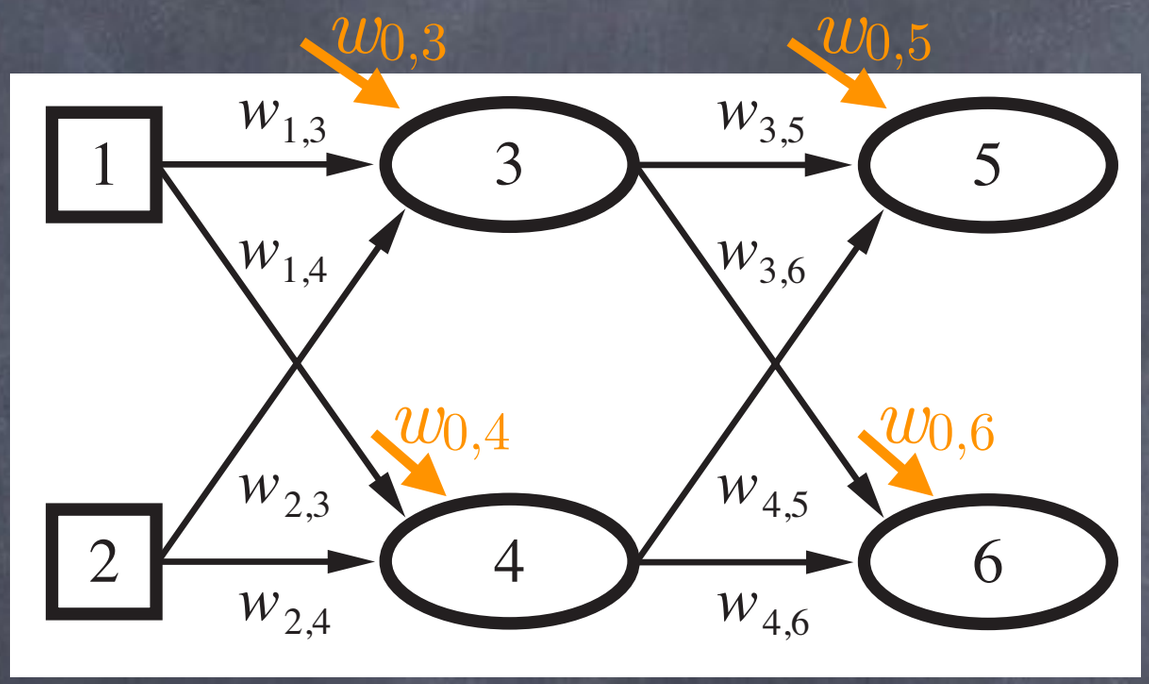
Input

Output

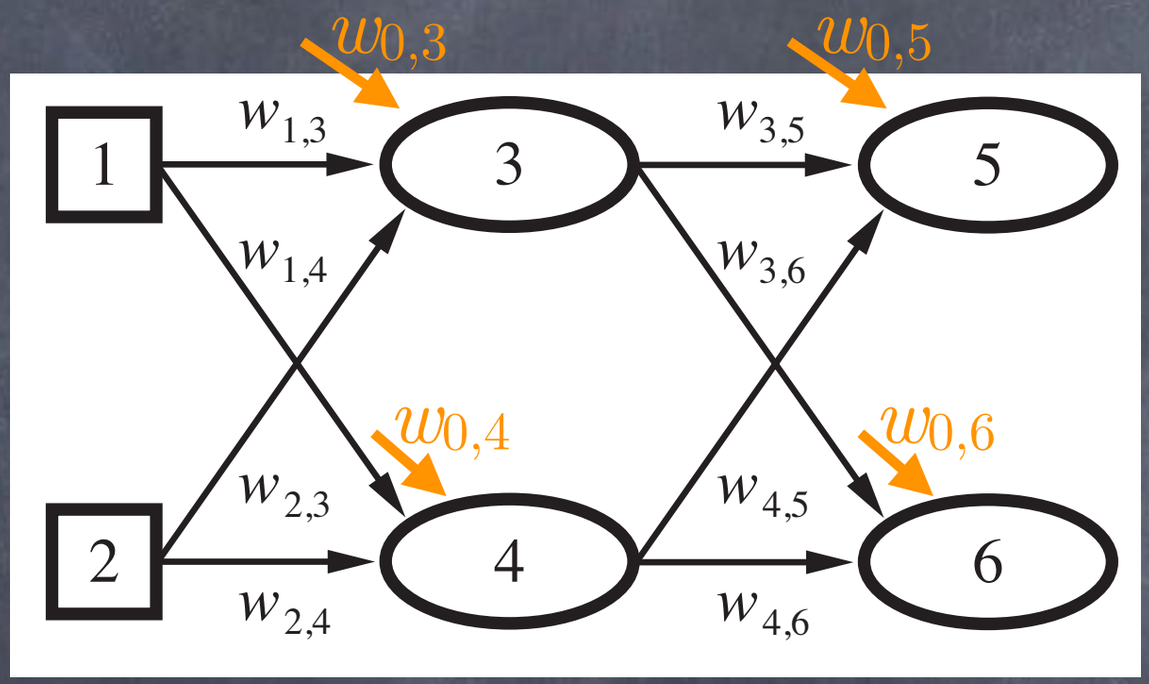
Multi-Layer Feed-Forward NNs



Multi-Layer Feed-Forward NNs

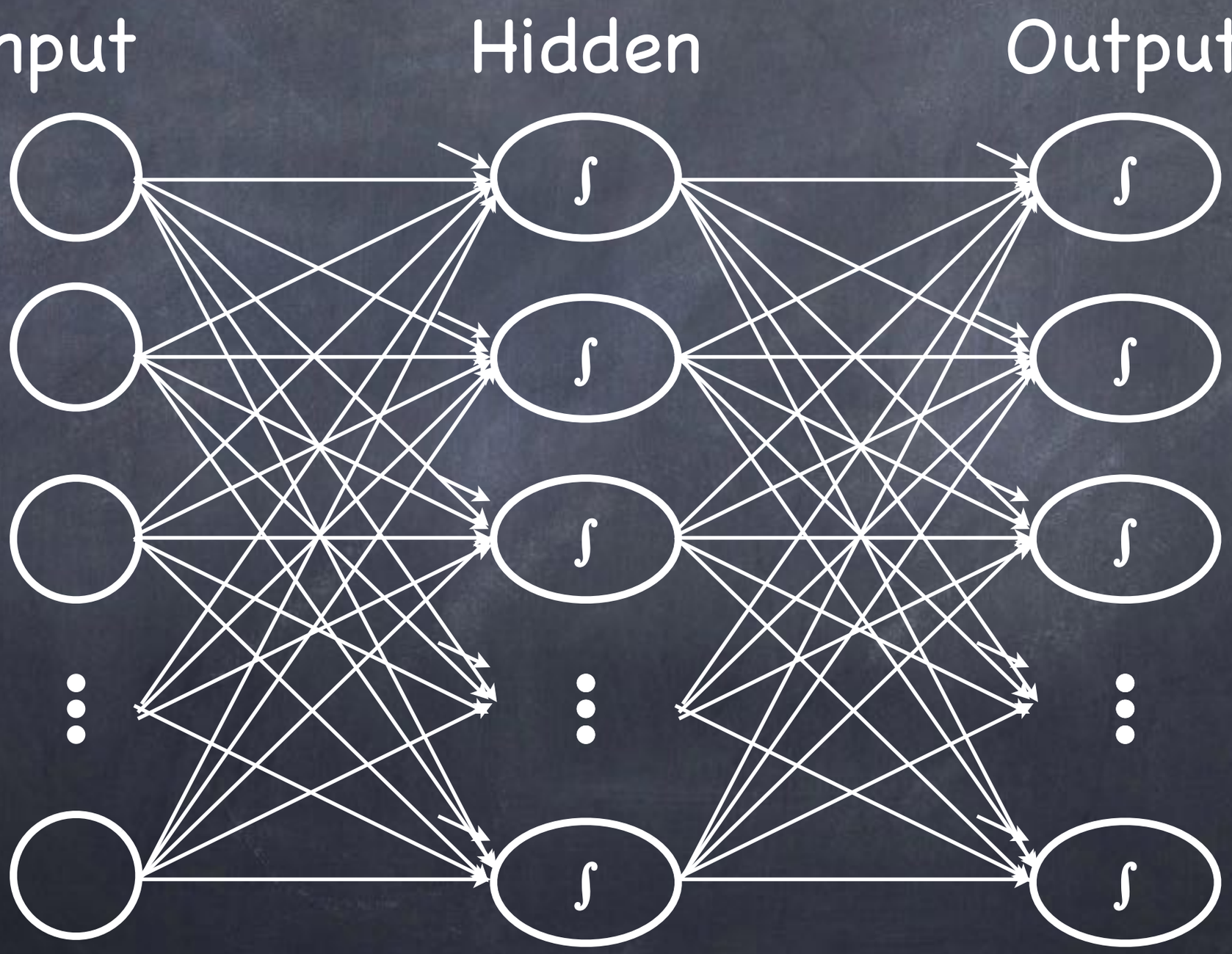


Multi-Layer Feed-Forward NNs

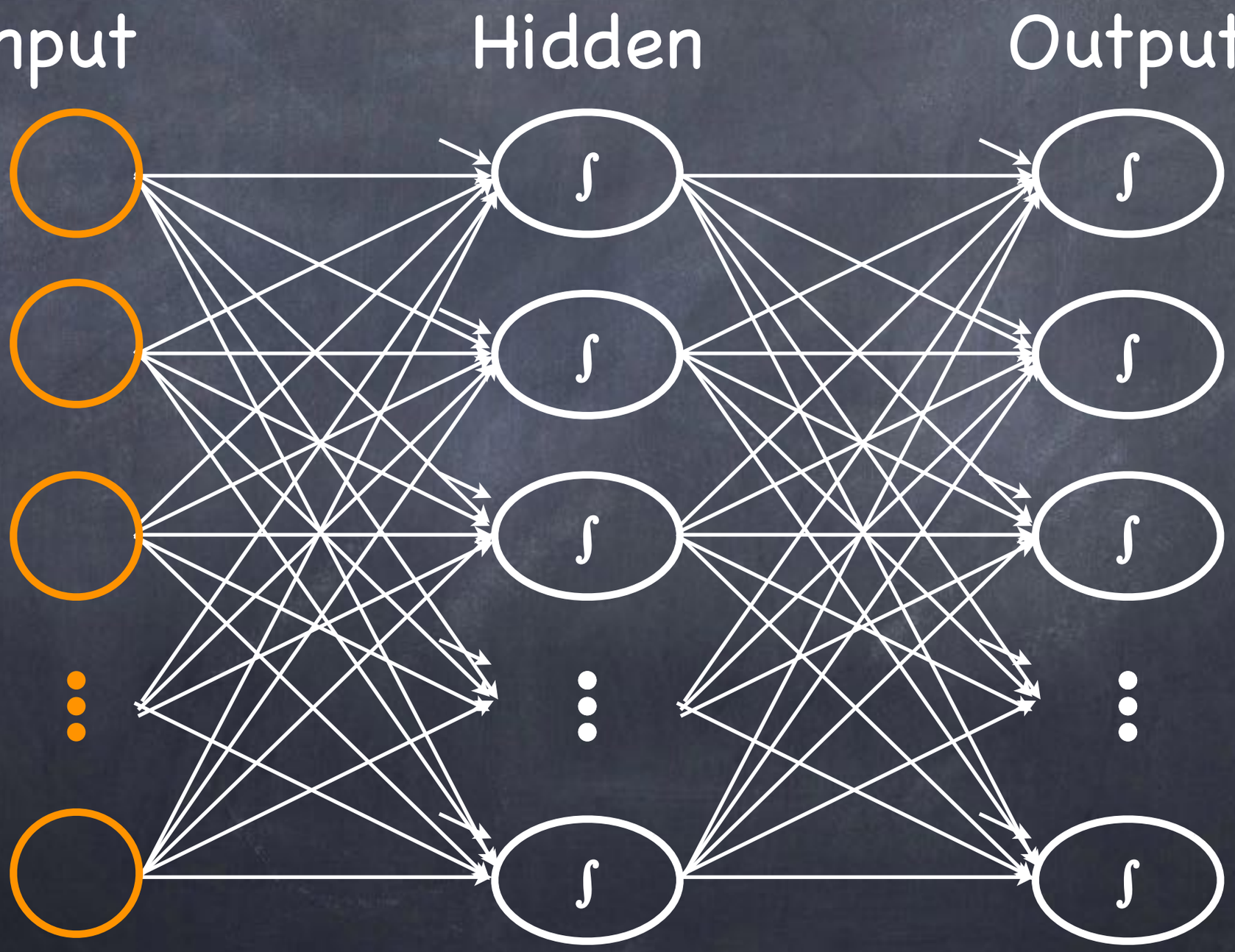


$$
\begin{aligned}
a_5 &= g(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4) \\
&= g(w_{0,5} + w_{3,5}g(w_{0,3} + w_{1,3}a_1 + w_{2,3}a_2) + \\
&\qquad w_{4,5}g(w_{0,4} + w1,4a_1 + w_{2,4}a_2)) \\
&= g(w_{0,5} + w_{3,5}g(w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) + \\
&\qquad w_{4,5}g(w_{0,4} + w1,4x_1 + w_{2,4}x_2))
\end{aligned}
$$

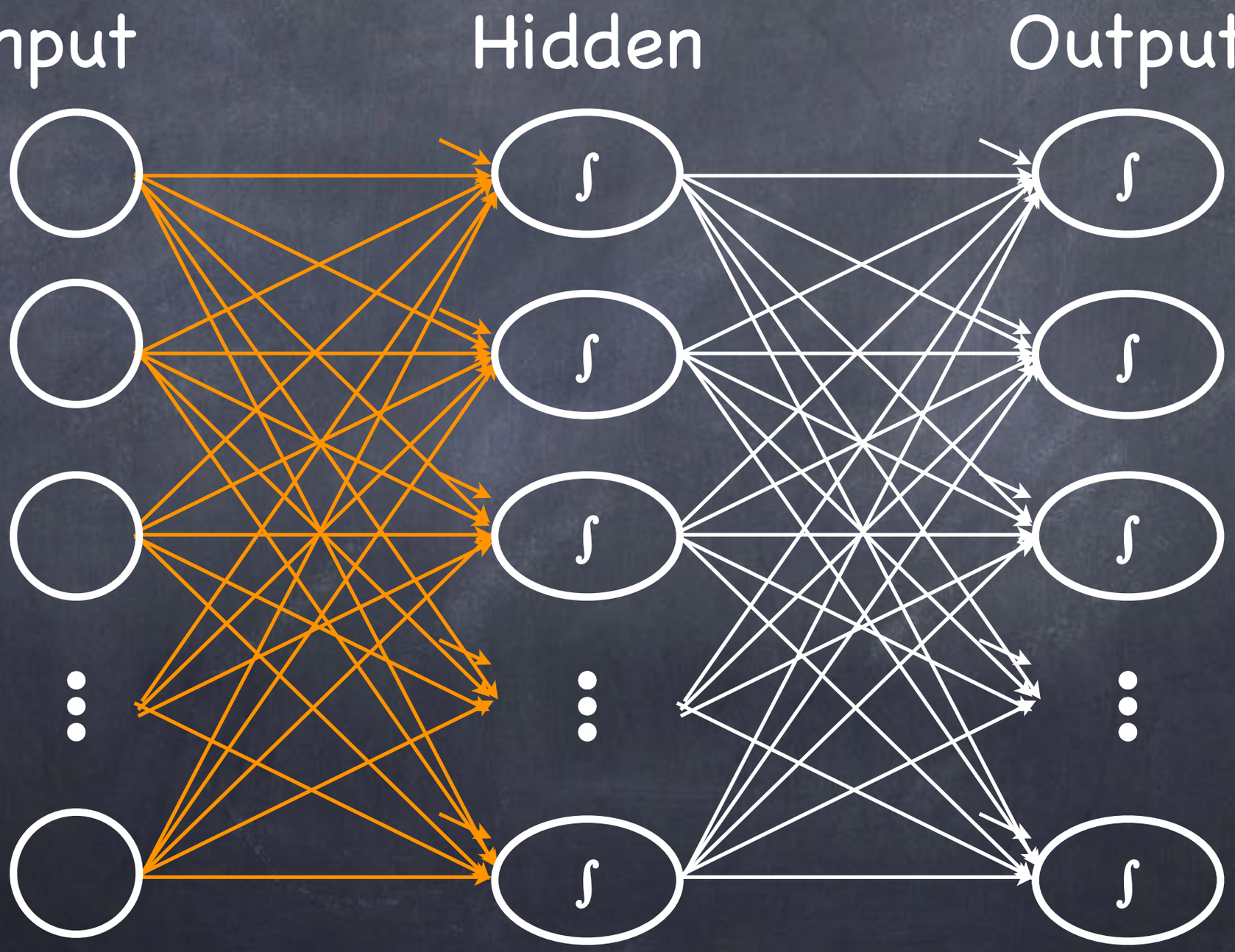
Learning in Multi-Layer Networks



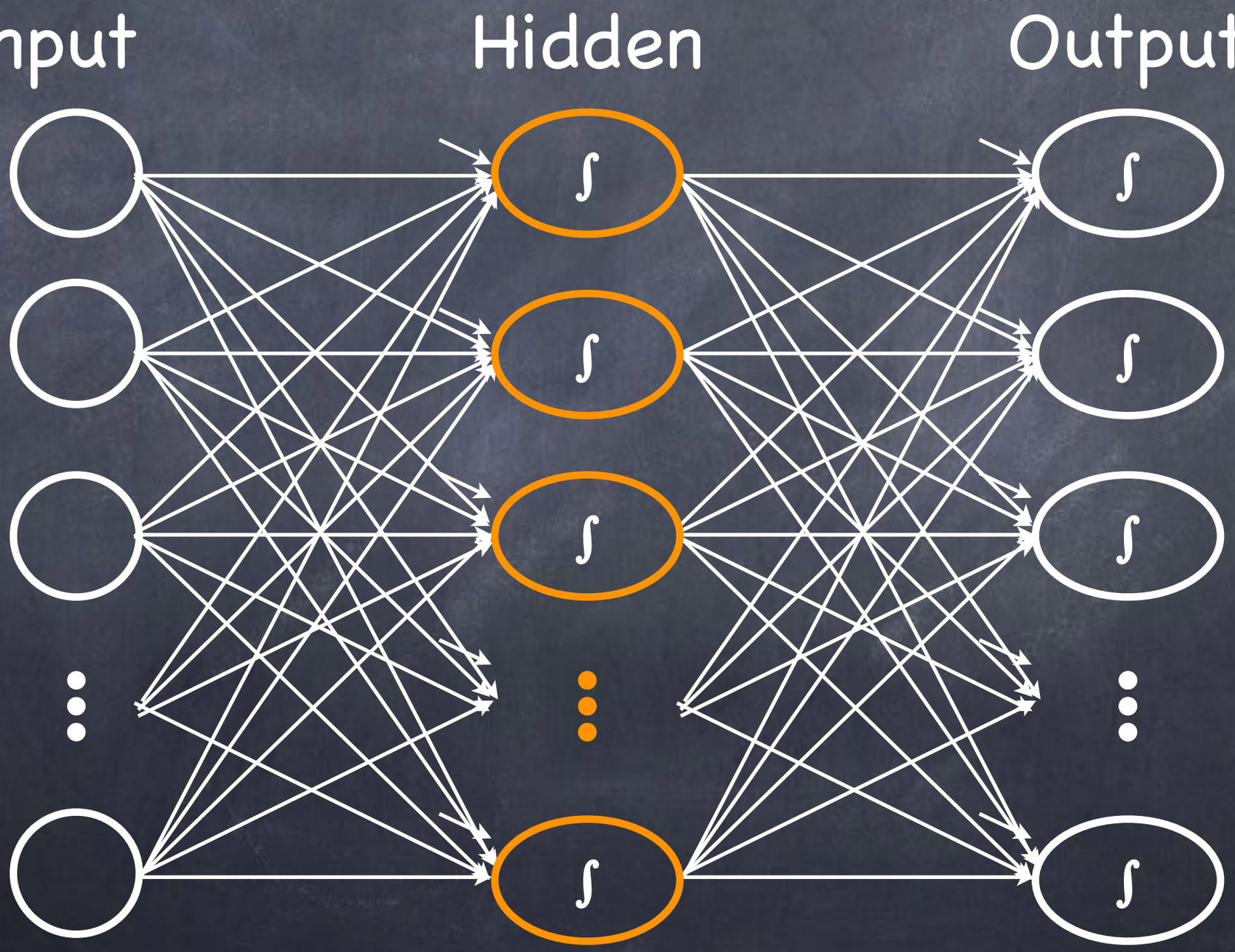
Learning in Multi-Layer Networks



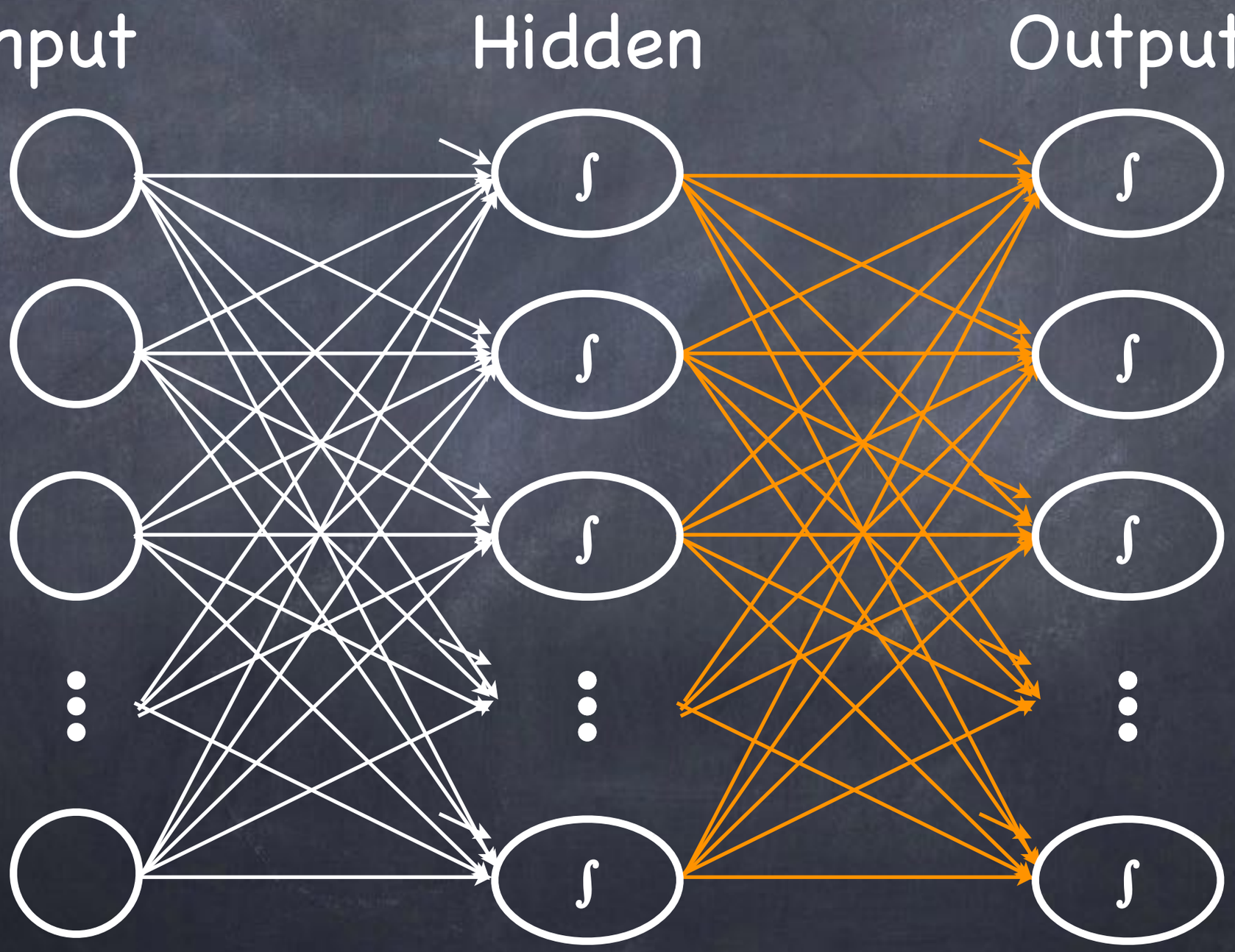
Learning in Multi-Layer Networks



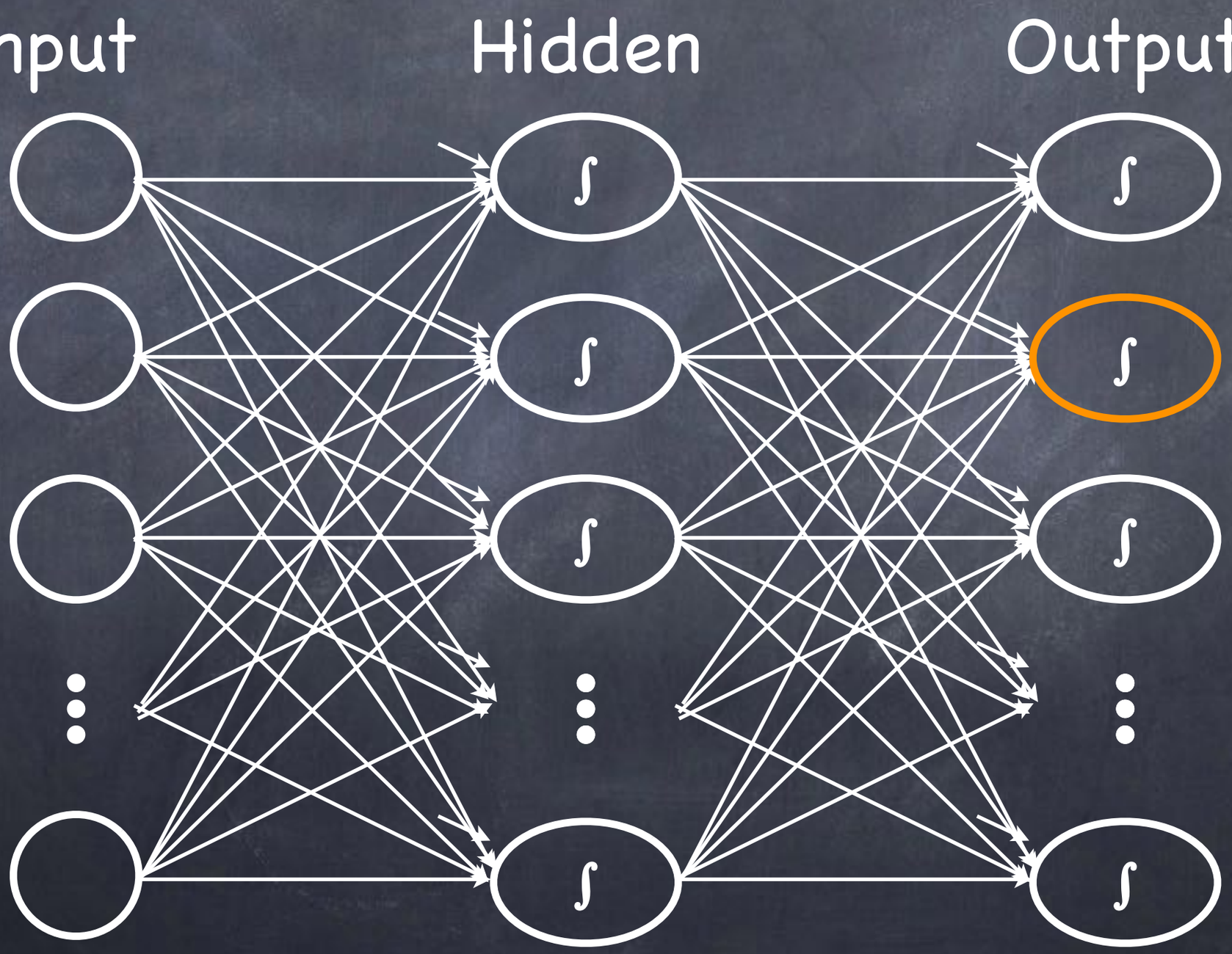
Learning in Multi-Layer Networks



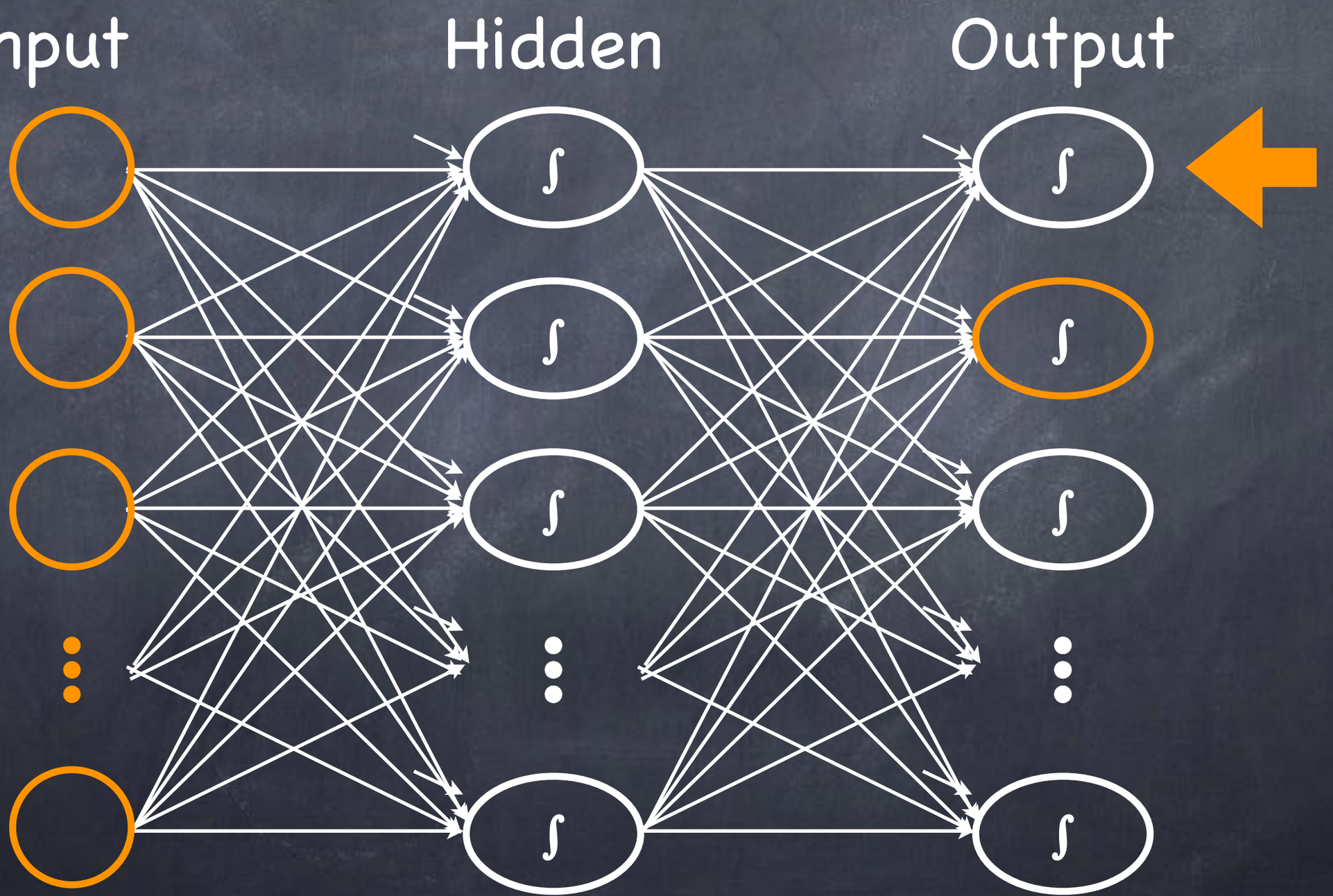
Learning in Multi-Layer Networks



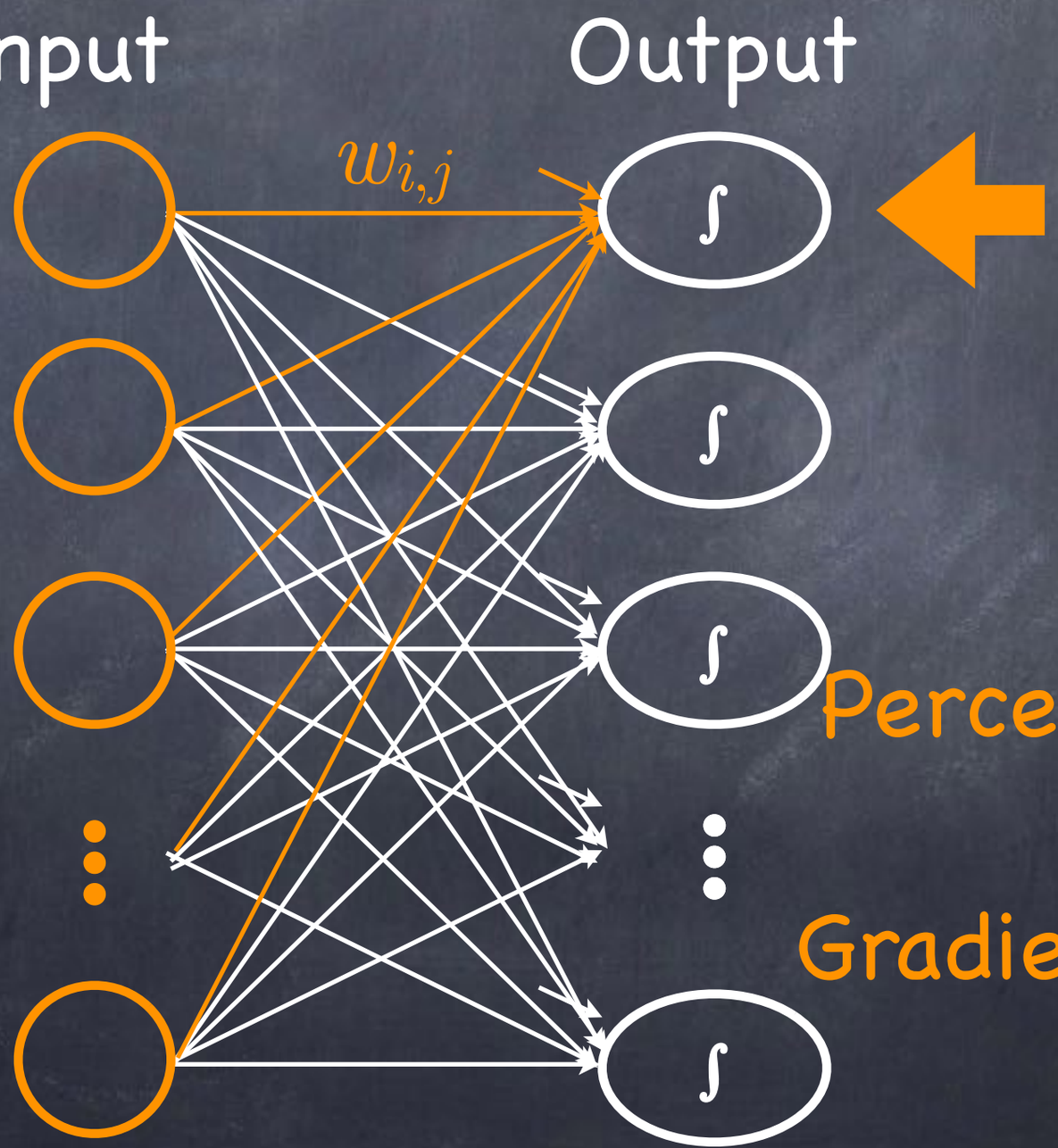
Learning in Multi-Layer Networks



Learning in Multi-Layer Networks

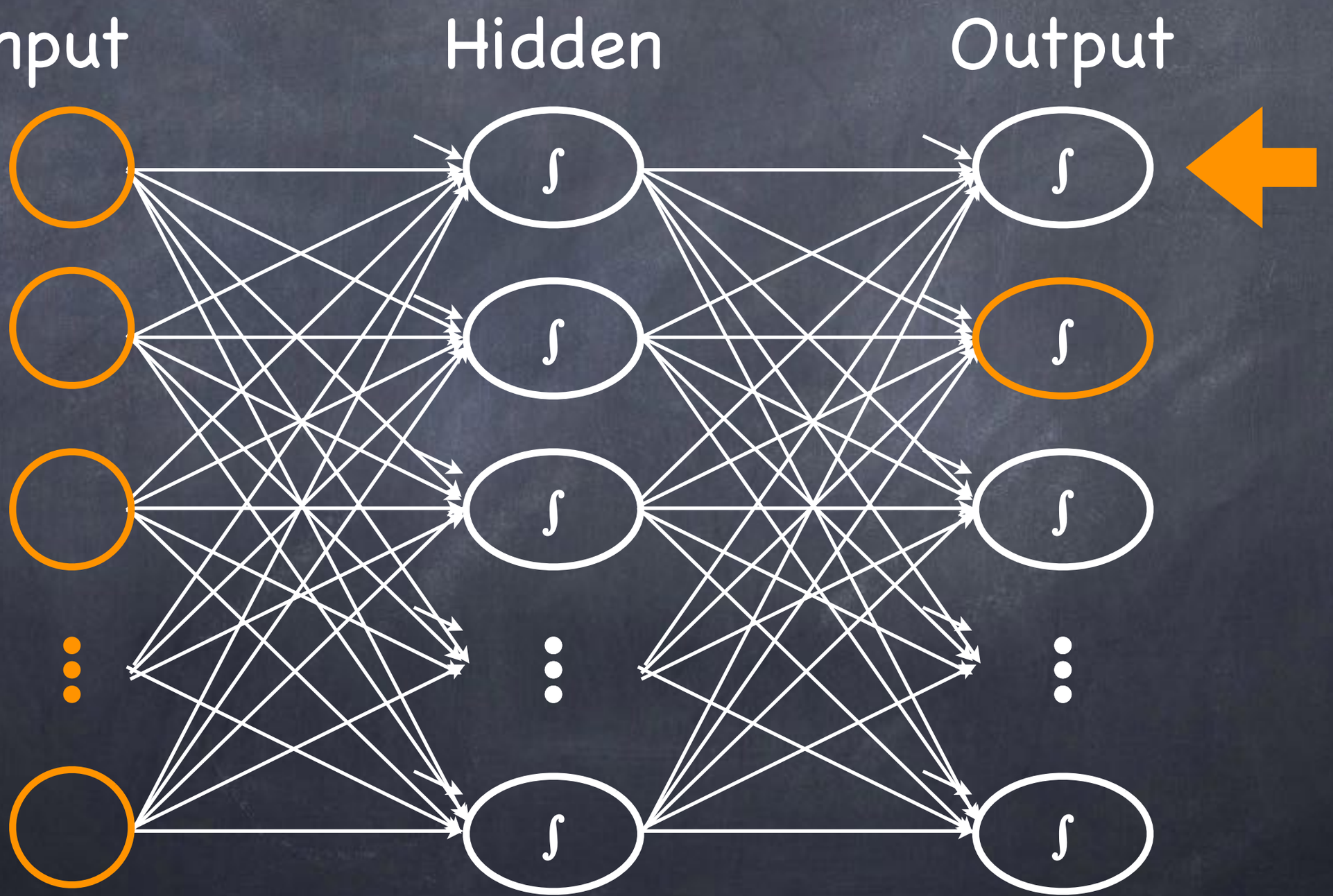


Learning in Single-Layer NNs

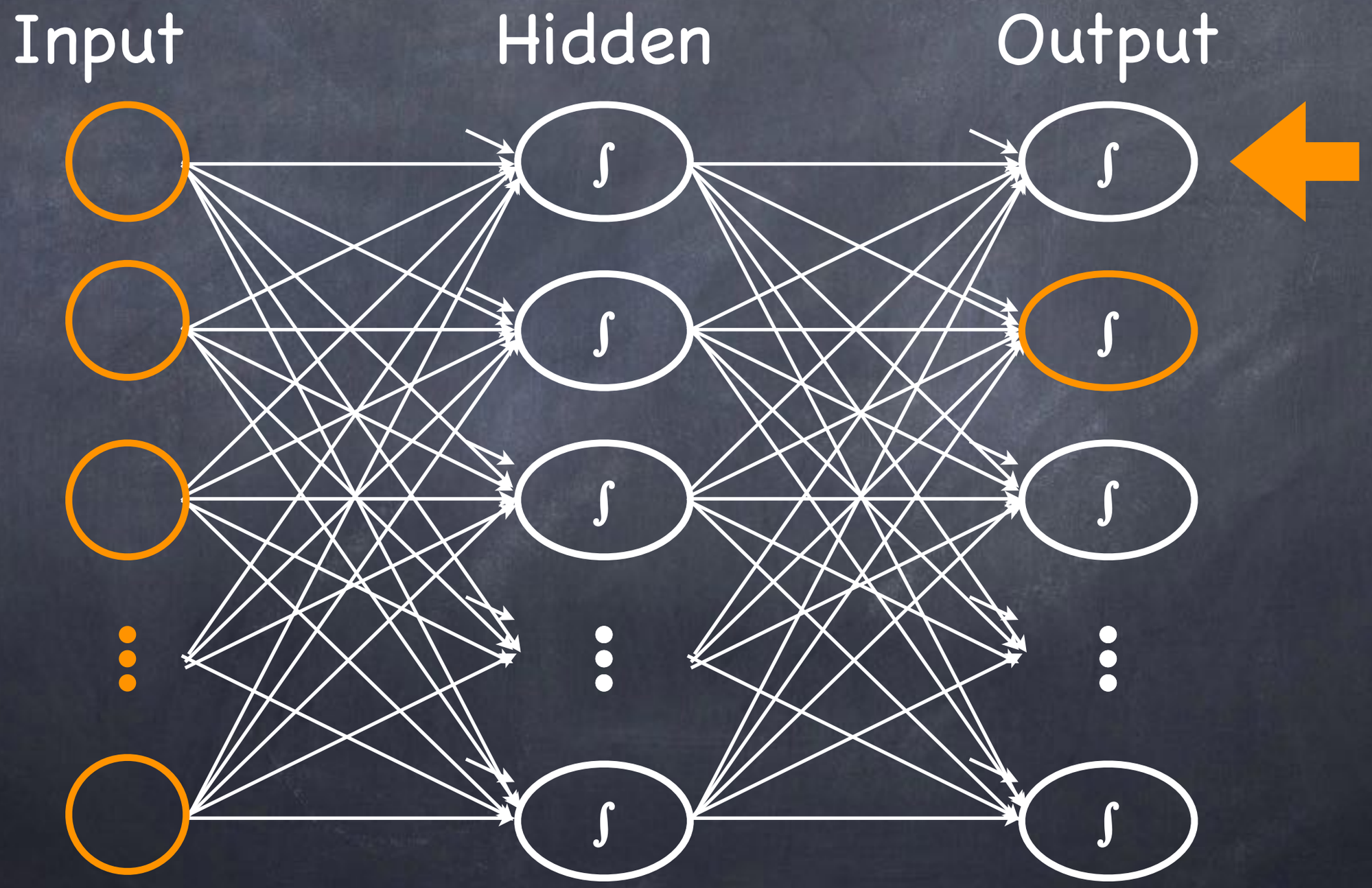


Perceptron Rule
or
Gradient Descent

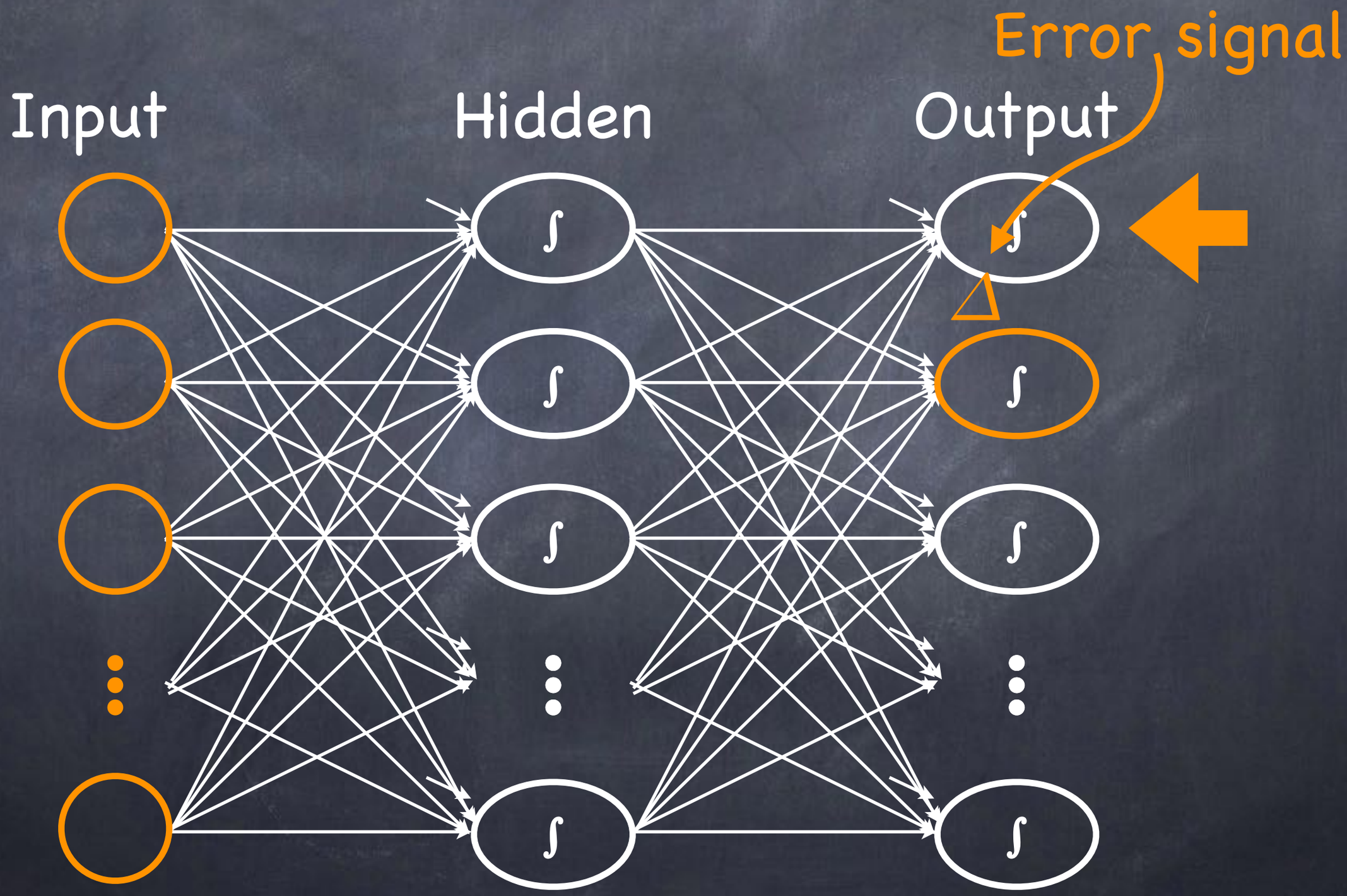
Learning in Multi-Layer Networks



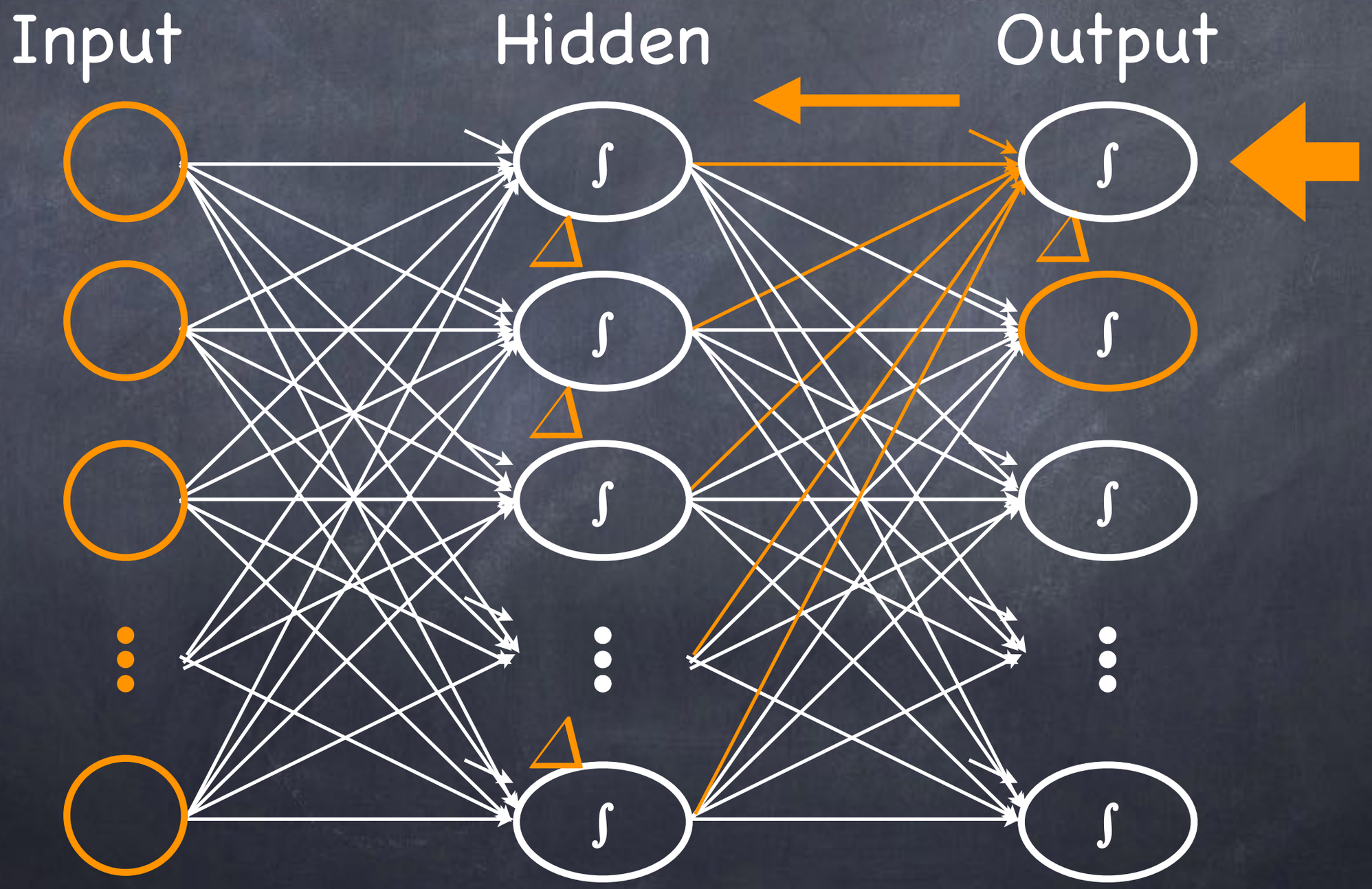
Back-Propagation



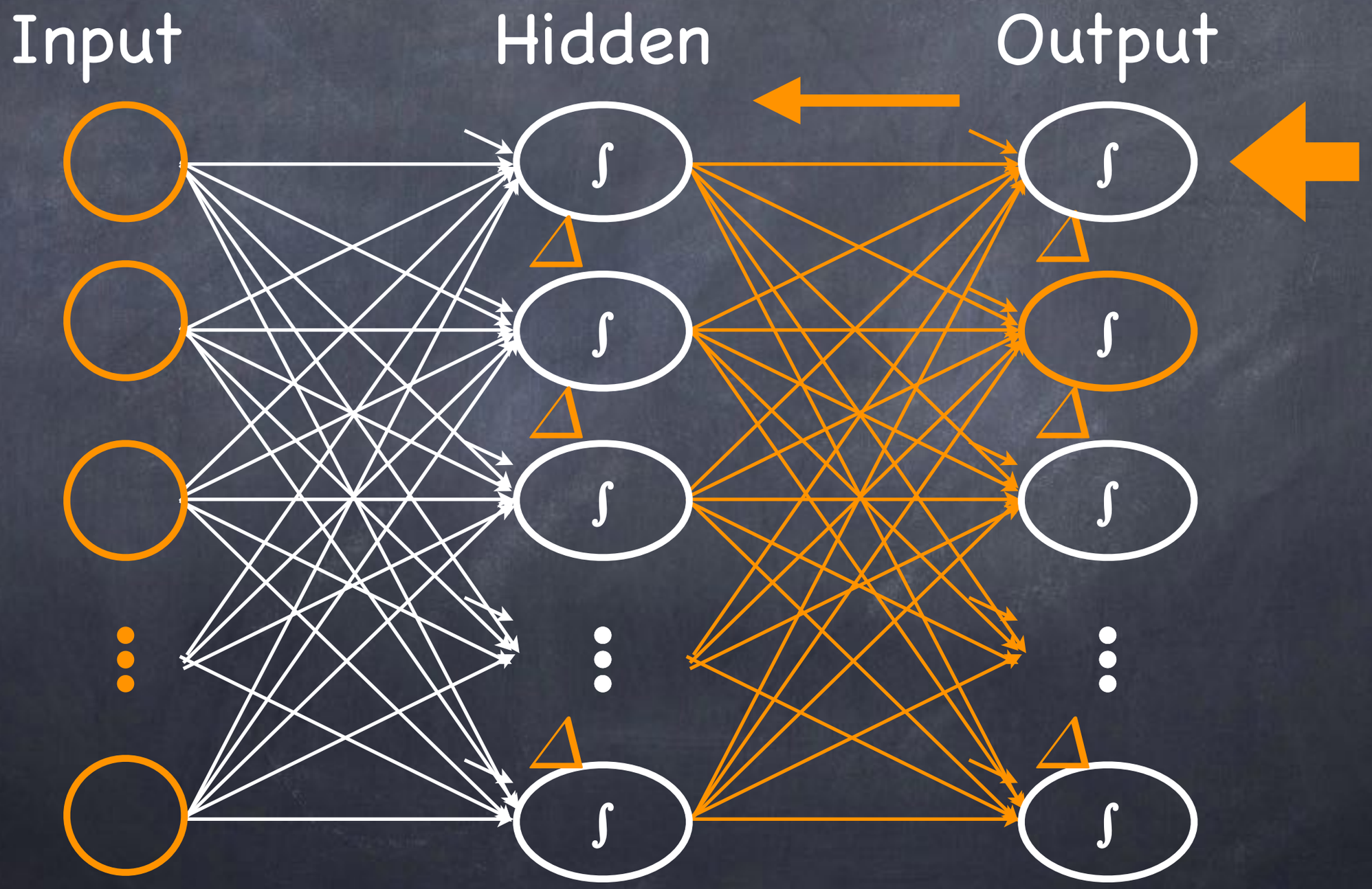
Back-Propagation



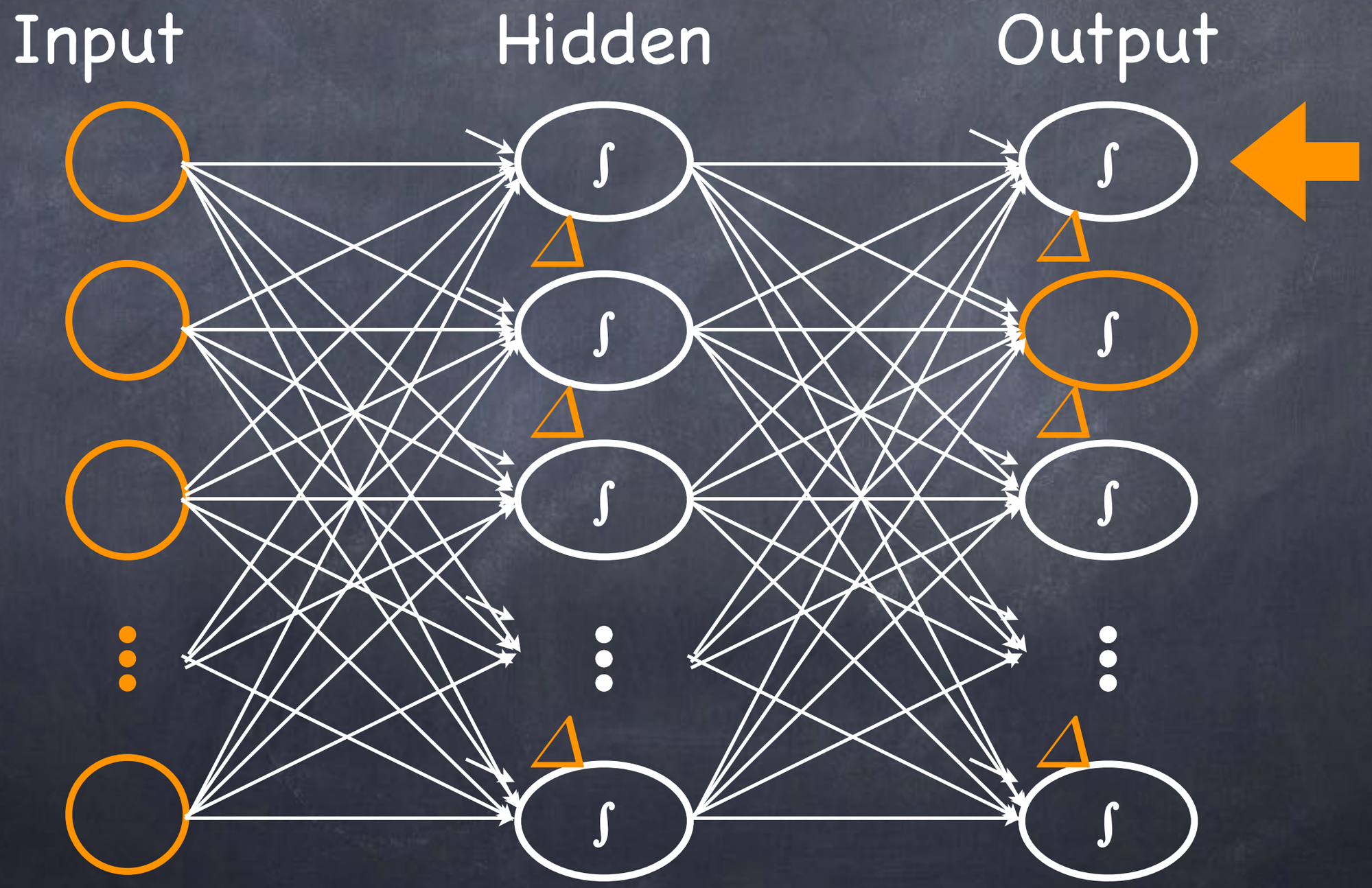
Back-Propagation



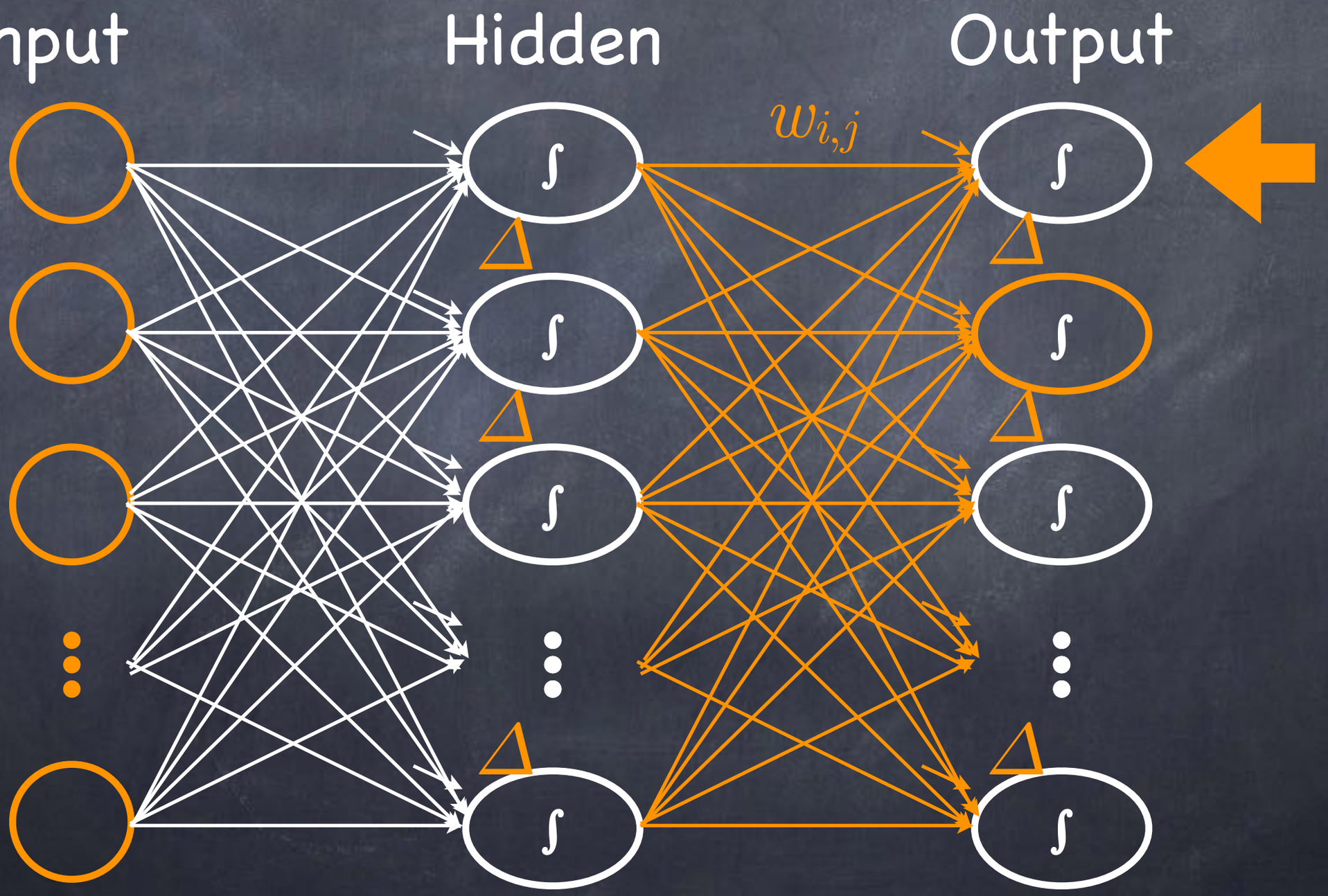
Back-Propagation



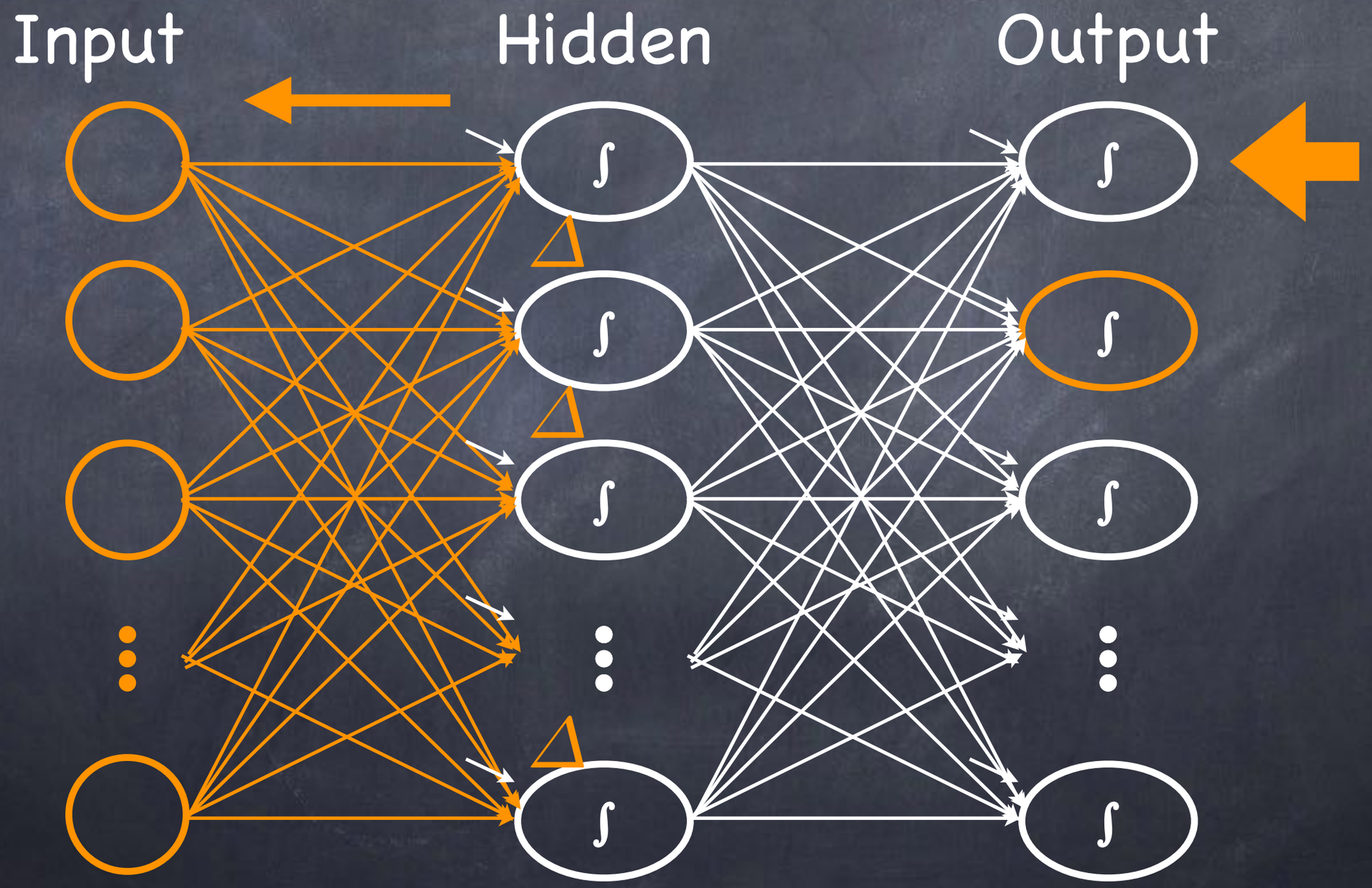
Back-Propagation



Back-Propagation



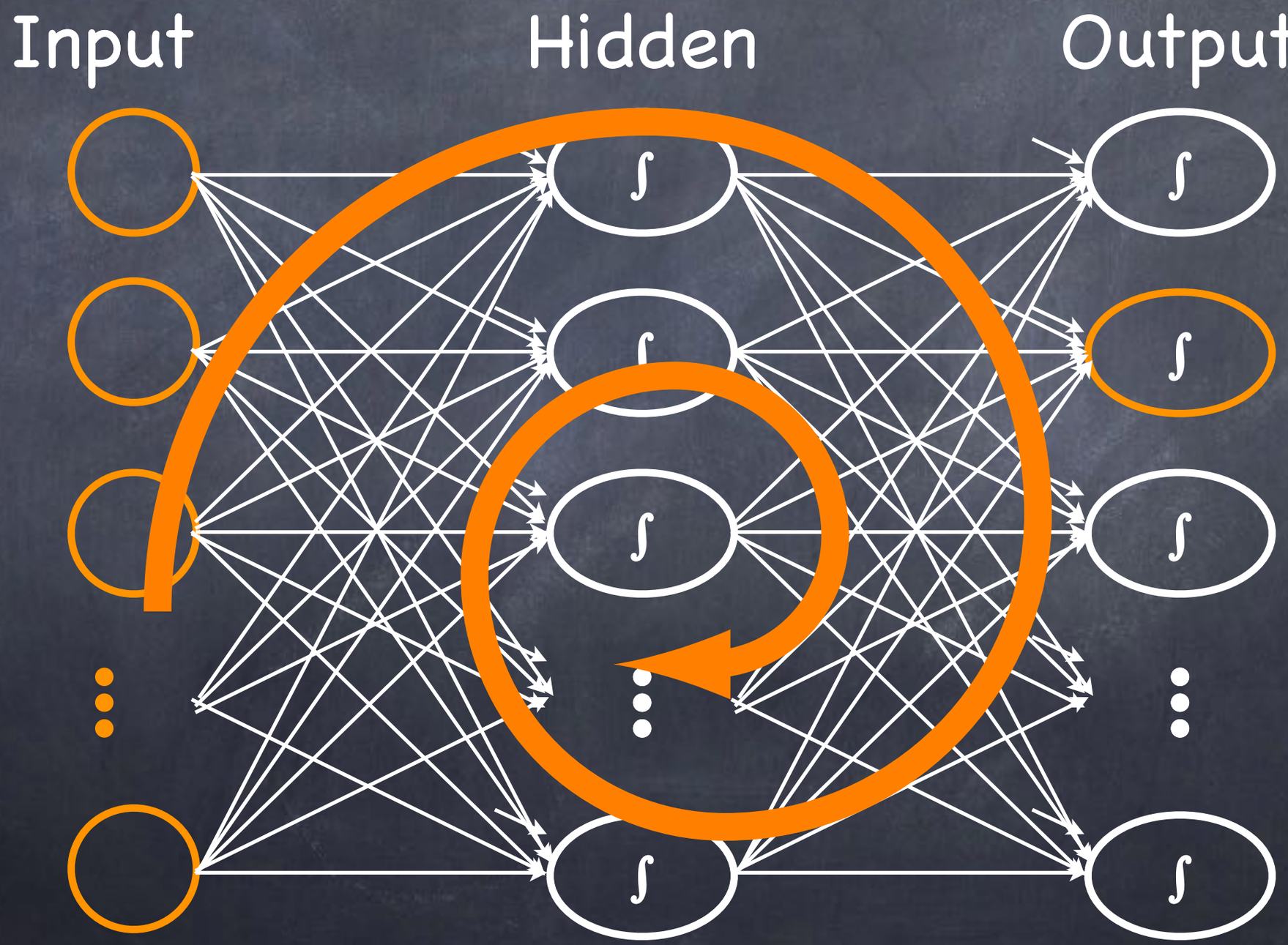
Back-Propagation



Back-Propagation

- Compute Δ values for output units using observed error
- Starting with output layer and working backwards:
- Propagate Δ values back to previous layer
- Update weights between the two layers

Back-Propagation



Recurrent NNs

- Allow connections from outputs to inputs
  - => Feedback loops!
- Mathematics is daunting...

Learning in Recurrent NNs

- Need to learn weights for each connection between nodes
  - Multiple logistic regression problems
  - Changes to weights of one node change its output => change inputs to other nodes
    - Convergence is complicated

Neural Nets

- Linear classifiers with logistic threshold functions, connected in a graph topology
- Learns classification function from inputs to outputs
- Trained from data:
  - Gradient descent for single-layer nets
  - Back-propagation for hidden units

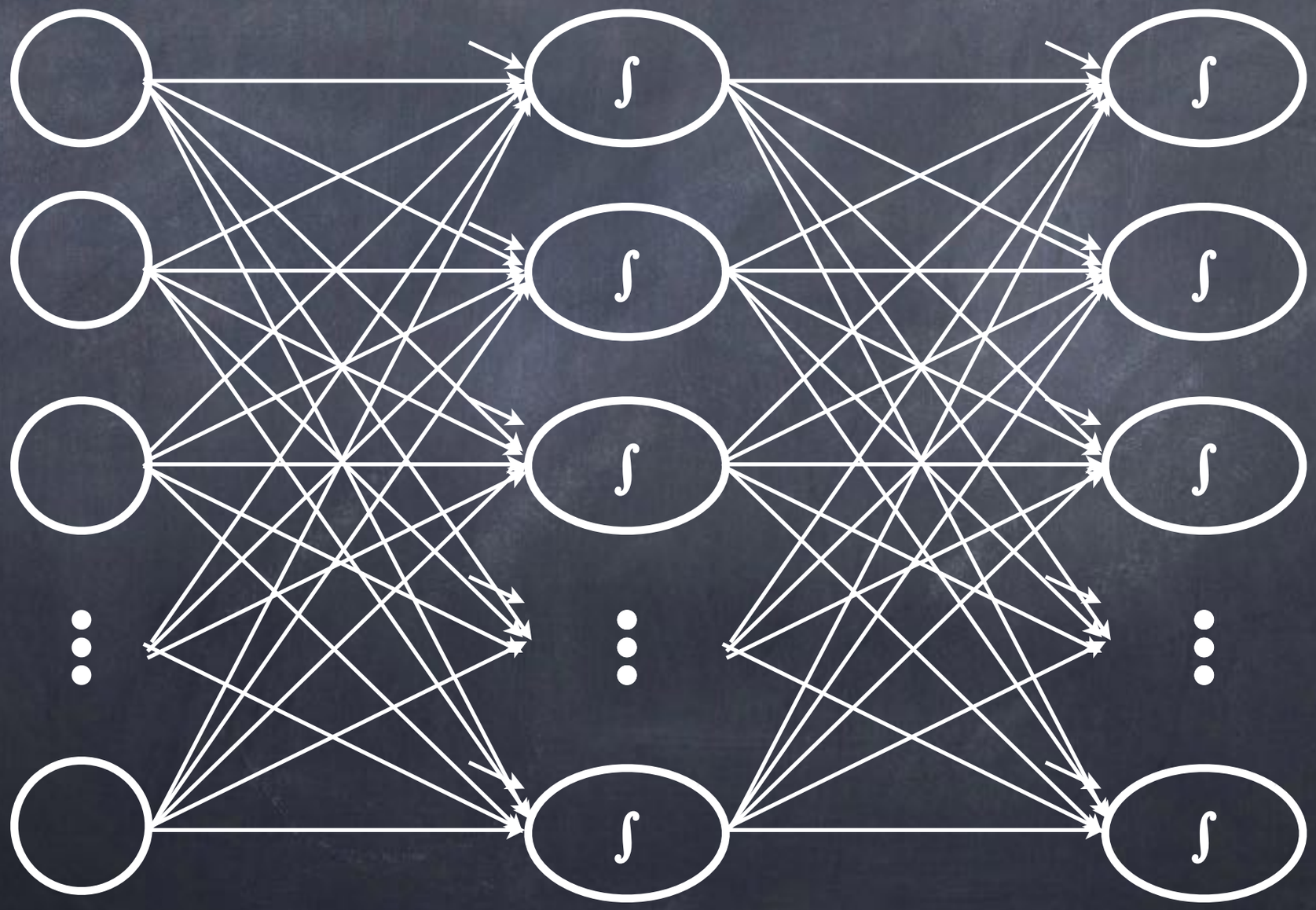
Deep Learning

Deep learning is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using multiple processing layers, with complex structures or otherwise, composed of multiple non-linear transformations.

- Wikipedia

Deep Learning

Concrete → Abstract



Deep Learning

Deep learning has been characterized as a buzzword, or a rebranding of neural networks.

- Wikipedia

Deep Learning

"Realistically, deep learning is only part of the larger challenge of building intelligent machines. Such techniques lack ways of representing causal relationships (...) have no obvious ways of performing logical inferences, and they are also still a long way from integrating abstract knowledge, such as information about what objects are, what they are for, and how they are typically used. The most powerful A.I. systems, like Watson (...) use techniques like deep learning as just one element in a very complicated ensemble of techniques, ranging from the statistical technique of Bayesian inference to deductive reasoning."

- Gary Marcus

Support Vector Machines

- Learn a maximum margin separator
  - Decision boundary with largest possible distance to example points
  - Learns linear separator (but possibly in higher-dimensional space)
  - Generalize well
  - Can represent complex functions (boundaries) without overfitting

Corinna Cortes
2008 ACM Paris Kanellakis
Theory and Practice Award
(with Vladimir Vapnik)

Summary

- Neural Networks: Directed graph of linear classifiers
  - Multi-layer networks can learn arbitrary (incl. nonlinear) functions
  - Train with gradient descent or backprop
- Deep Learning: NNs with many layers
- Support Vector Machines: State of the art for supervised learning of linear classifiers

For Next Time:

20.0-20.2.2; 20.2.5 fyi