CS57300
PURDUE UNIVERSITY

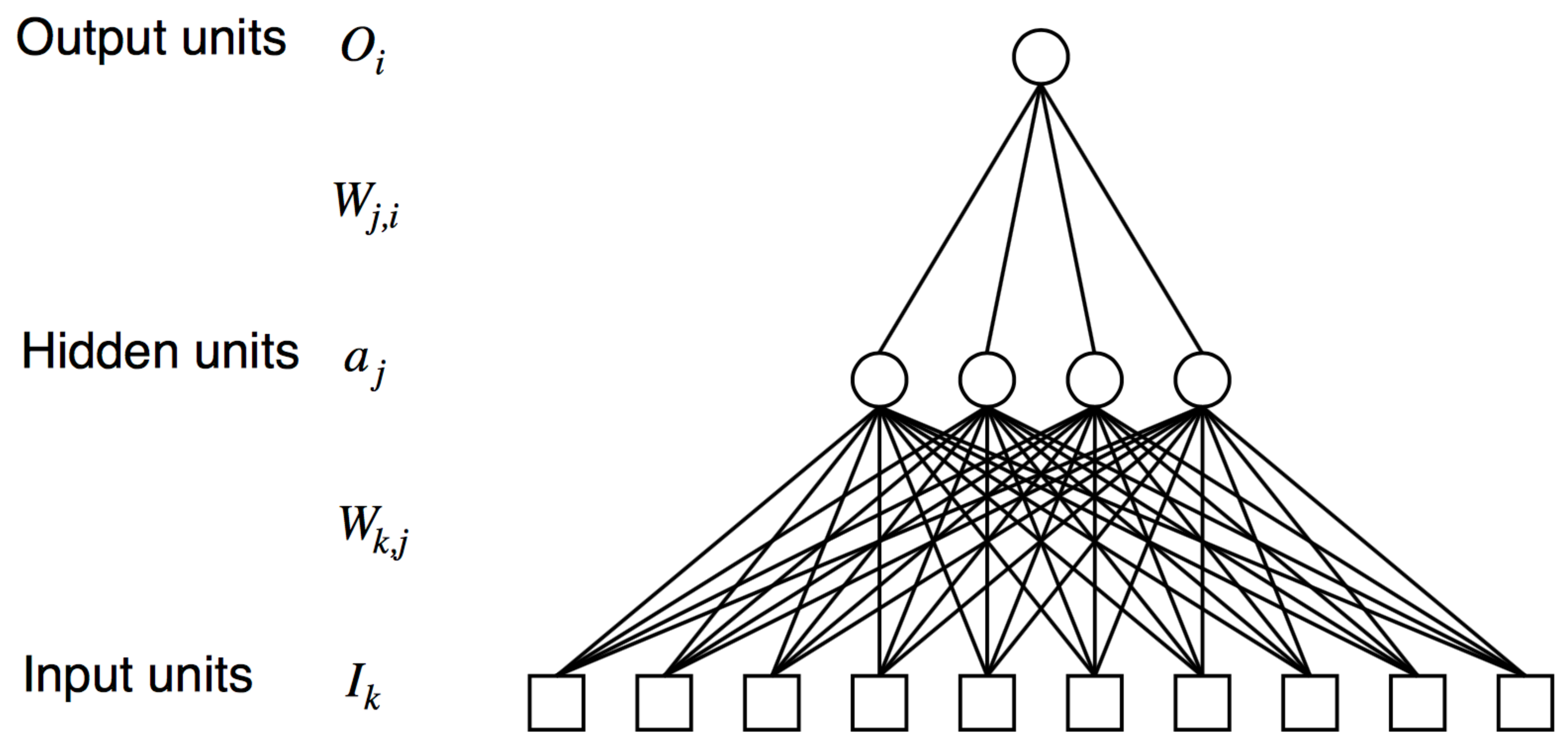
OCTOBER 20, 2021

DATA MINING

NEURAL NETWORK

MULTI-LAYER NEURAL NETWORK

- Increase expressive power by combining multiple perceptrons into ensemble
- Two-layer neural network: each perceptron output is a hidden unit, which are then aggregated into a final output



Output $$O_i = g(\sum_j W_{j,i} a_j)$$

Hidden units $$a_j = g(\sum_k W_{k,j} I_k)$$

Figure: M. Velosa

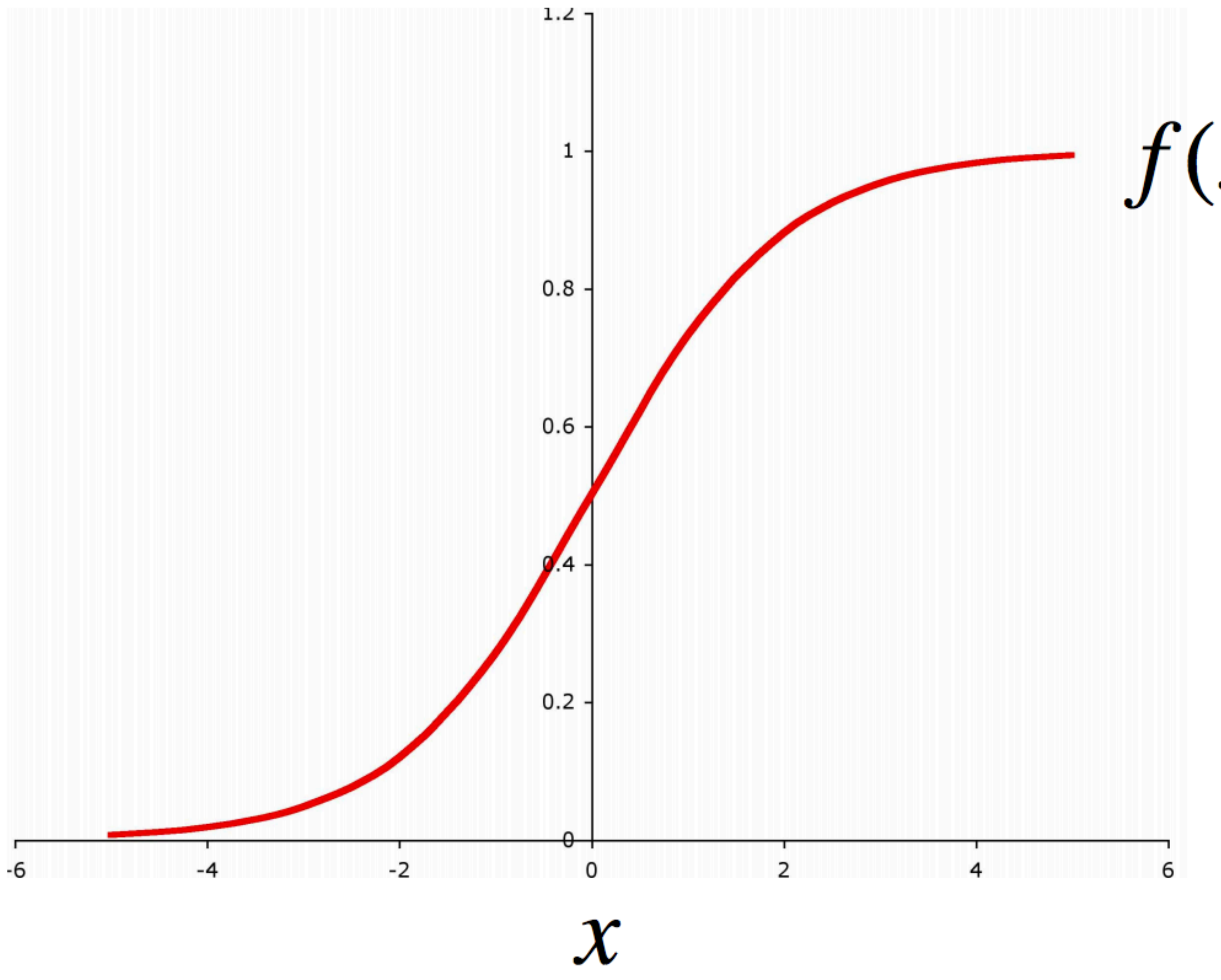
DIFFERENTIABLE SCORING FUNCTIONS AND ACTIVATION FUNCTIONS

- The scoring function *S* will take as inputs **x** (attributes), y (true label), $W_{k,j}$ (weights associated with hidden units), $W_{j,i}$ (weights associated with output units)
- If S is a differentiable function, we can use gradient-based optimization techniques to update weights!
- Differentiable scoring function: $E(\mathbf{w}) = \frac{1}{2}\sum_{d=1}^{N}(y^{(d)} - o^{(d)})^2$ instead of 0-1 loss
- Differentiable activation function: replacing step functions with something differentiable…

SIGMOID FUNCTION

- The output of a hidden unit (or a output unit) associated with weight ***w*** and input ***x*** will generate an output of:

$$f(x) = \frac{1}{1 + e^{-w^T x}}$$



$$f(x) = \frac{1}{1 + e^{-x}}$$

HIGH-LEVEL GRADIENT-BASED LEARNING FRAMEWORK

Given a training dataset with N data points: $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$

Initialize the weights: $\mathbf{w}=\mathbf{w}_0$

Repeat

for each $(\mathbf{x}^{(d)}, y^{(d)})$ in D:

$o^{(d)} = f(\mathbf{w}, \mathbf{x}^{(d)})$, f is given by the neural network's structure

$$E(\mathbf{w}) = \frac{1}{2}\sum_{d=1}^{N}(y^{(d)} - o^{(d)})^2$$

Compute the error gradient for the entire set of training data

Compute the gradient: $\nabla E(\mathbf{w})$

Update: $\mathbf{w} = \mathbf{w} - \eta \nabla E(\mathbf{w})$

Until stopping criteria is met

BATCH LEARNING

STOCHASTIC GRADIENT-BASED LEARNING FRAMEWORK

Given a training dataset with N data points: $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$

Initialize the weights: $\mathbf{w}=\mathbf{w}_0$

Repeat

for each $(\mathbf{x}^{(d)}, y^{(d)})$ in D:

$o^{(d)} = f(\mathbf{w}, \mathbf{x}^{(d)})$, f is given by the neural network's structure

$$E(\mathbf{w}) = \frac{1}{2}(y^{(d)} - o^{(d)})^2$$

Stochastic gradient descent

Compute the gradient: $\nabla E(\mathbf{w})$

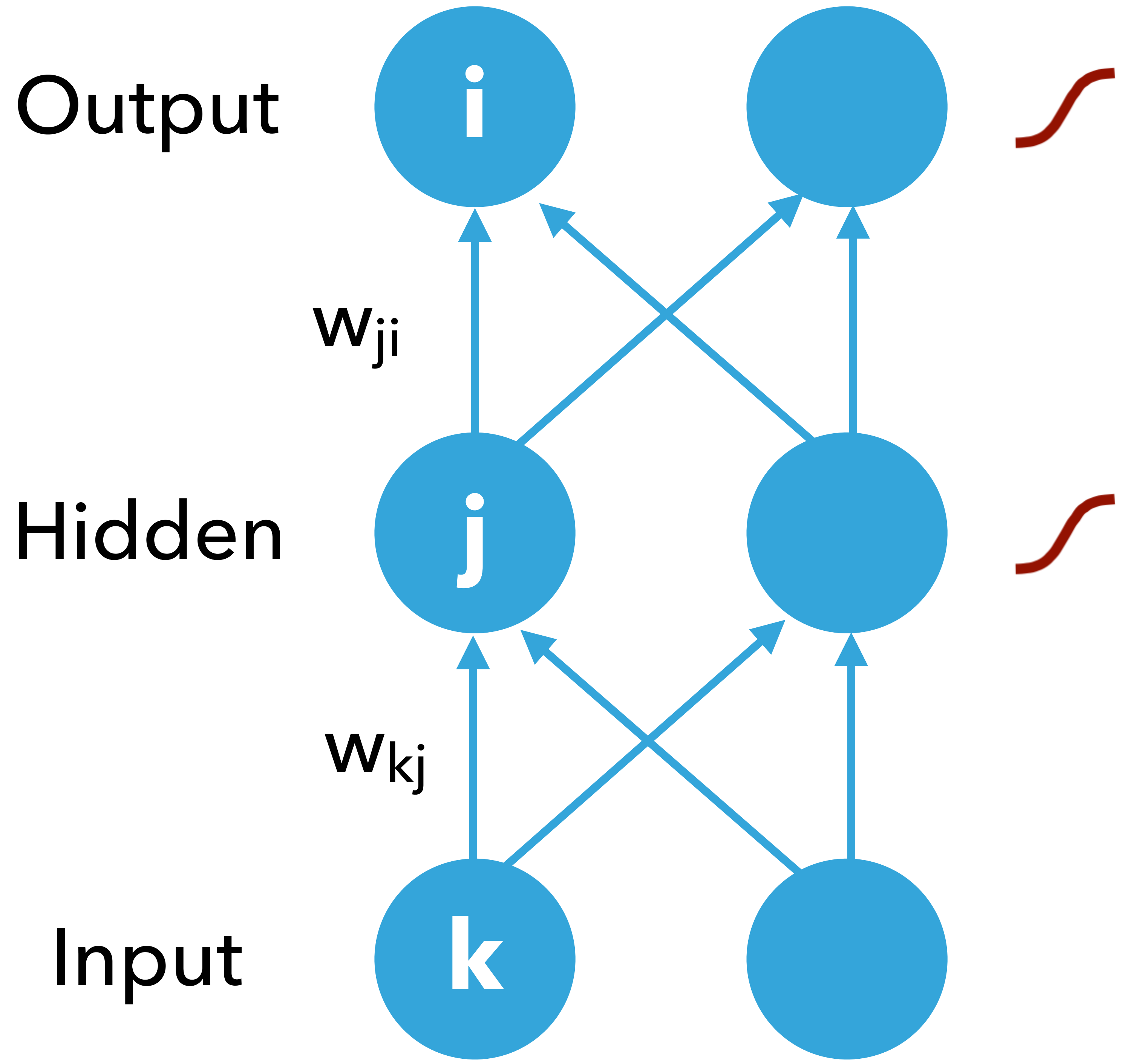
Update: $\mathbf{w} = \mathbf{w} - \eta \nabla E(\mathbf{w})$

Until stopping criteria is met

ONLINE LEARNING

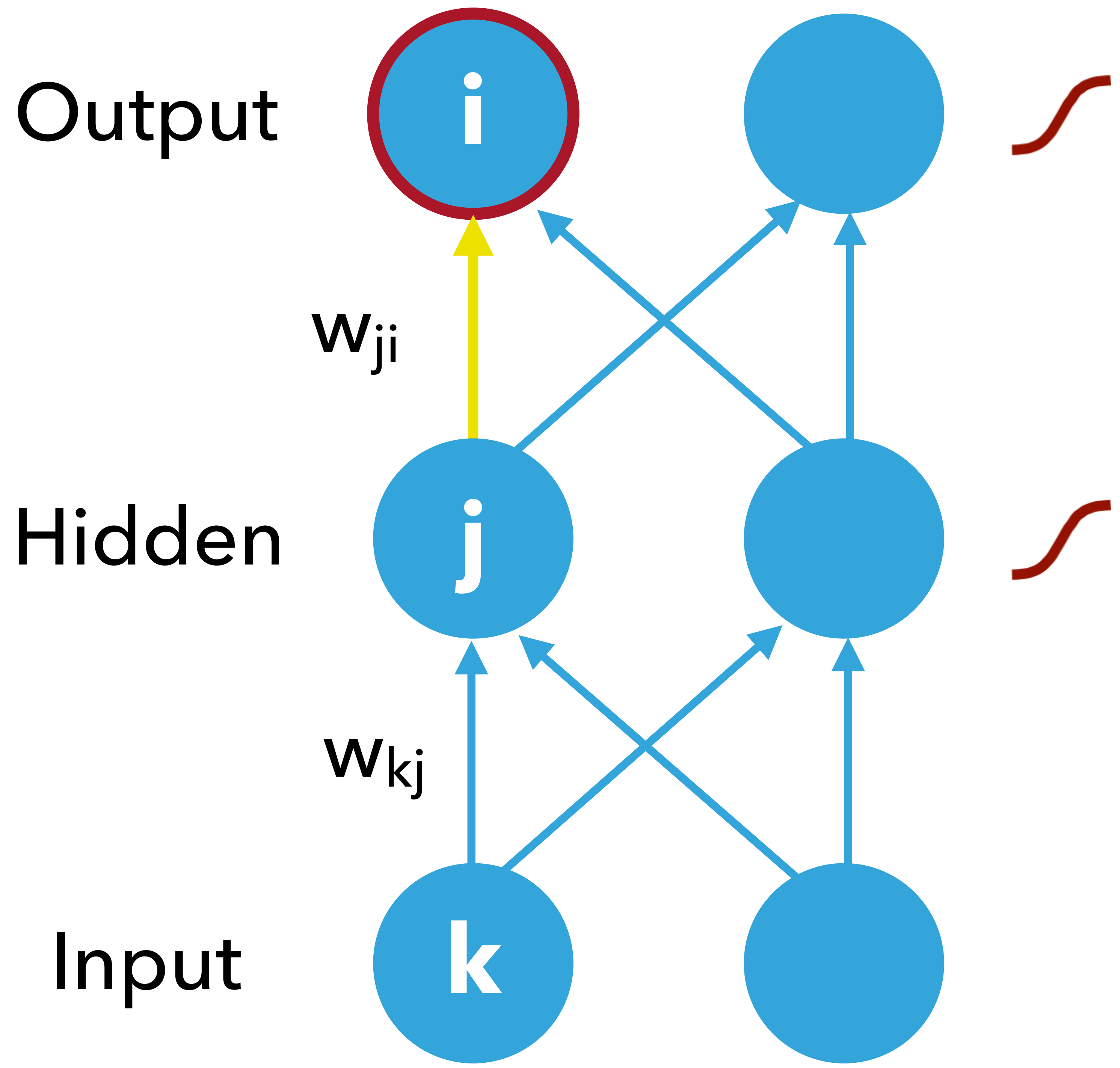
LEARNING NEURAL NETWORKS: SETUP

- Consider one training data (**x**, **y**), where the output **y** has M units
- Activation function for both hidden and output units are sigmoid functions
  - Suppose the output of node *z* is o_z, the input of node *z* is i_z (i_z=**x** if z is a hidden node, i_z are outputs of hidden nodes in the previous layer if z is an output node)
  - $o_z = \frac{1}{1+e^{-w^T i_z}}$, w is the weights associated with i_z
  - Denote $net_z = w^T i_z$, then $o_z = \frac{1}{1+e^{-net_z}}$



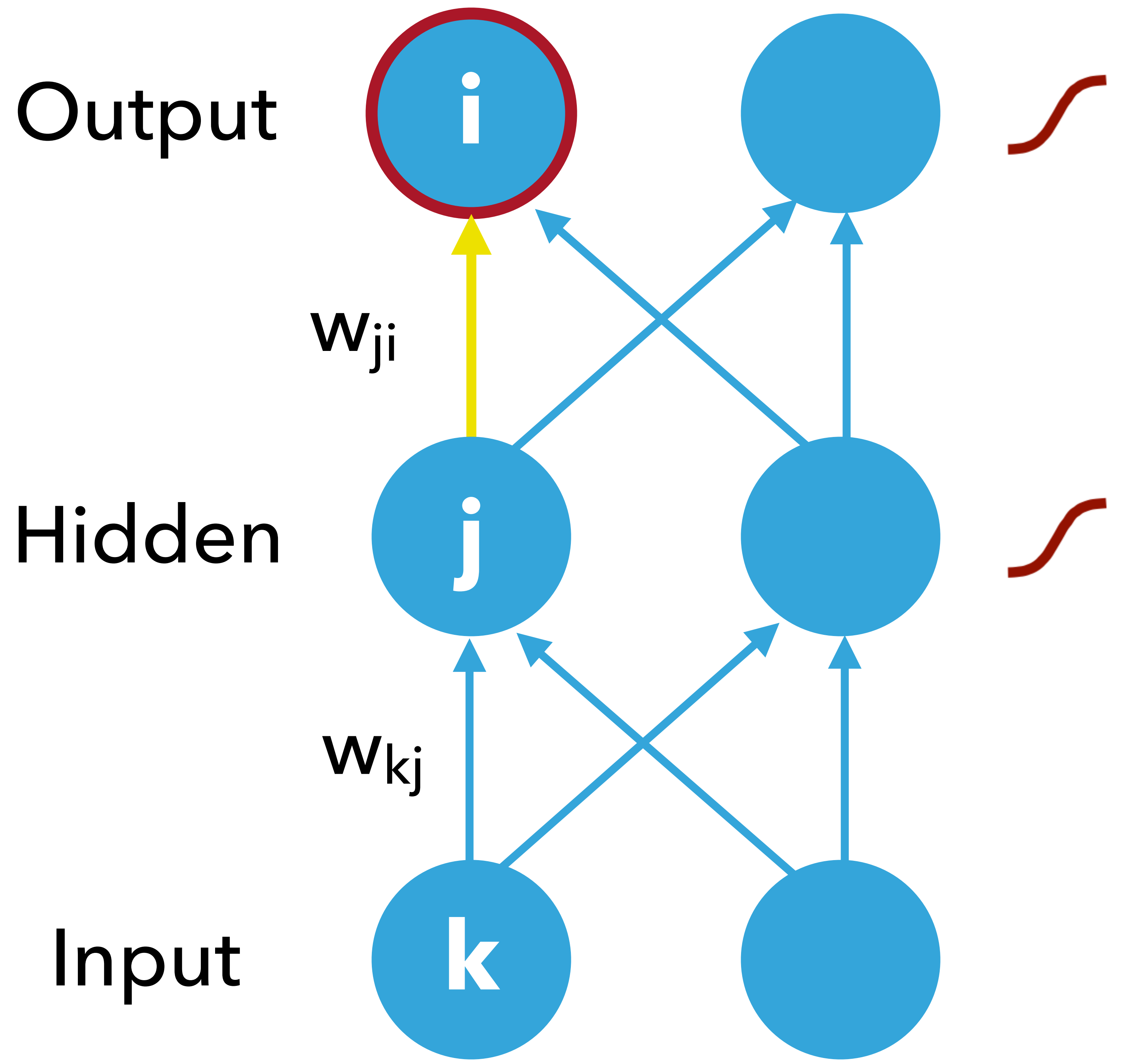
BACKPROPAGATION: LEARNING OUTPUT UNITS WEIGHTS

- Scoring function: $E(w) = \frac{1}{2} \sum_{m=1}^{M} (y_m - o_m)^2$



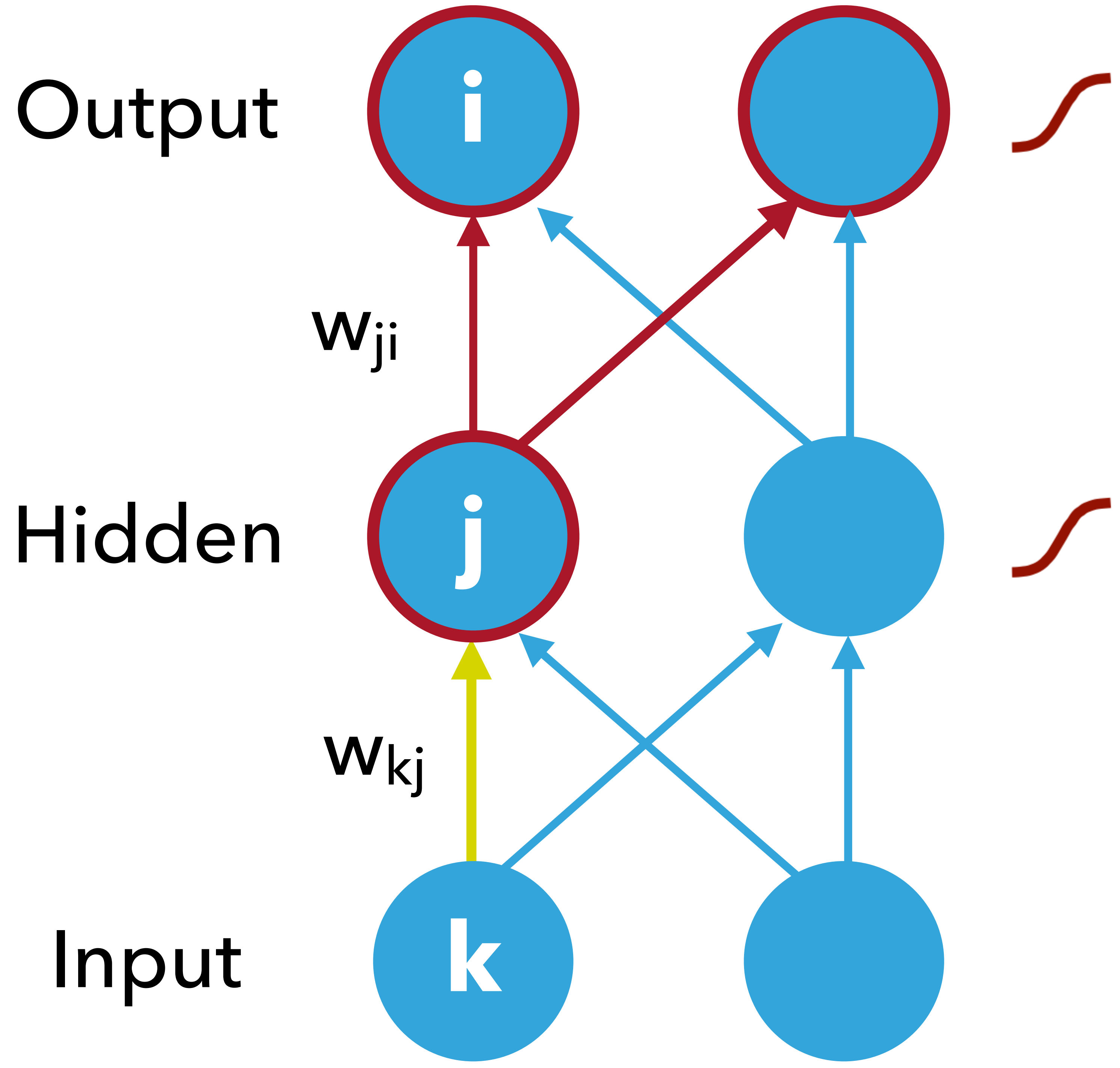
BACKPROPAGATION: LEARNING OUTPUT UNITS WEIGHTS

- Scoring function: $E(w) = \frac{1}{2}\sum_{m=1}^{M}(y_m - o_m)^2$
- Weights of output units w_{ji} will only affect E(w) through o_i
- $$\frac{\partial E(w)}{\partial w_{ji}} = \frac{\partial E(w)}{\partial o_i}\frac{\partial o_i}{\partial net_i}\frac{\partial net_i}{\partial w_{ji}}$$
$$= -(y_i - o_i)\frac{\partial o_i}{\partial net_i}\frac{\partial net_i}{\partial w_{ji}}$$
$$= -(y_i - o_i)o_i(1 - o_i)\frac{\partial net_i}{\partial w_{ji}}$$
$$= -(y_i - o_i)o_i(1 - o_i)o_j$$



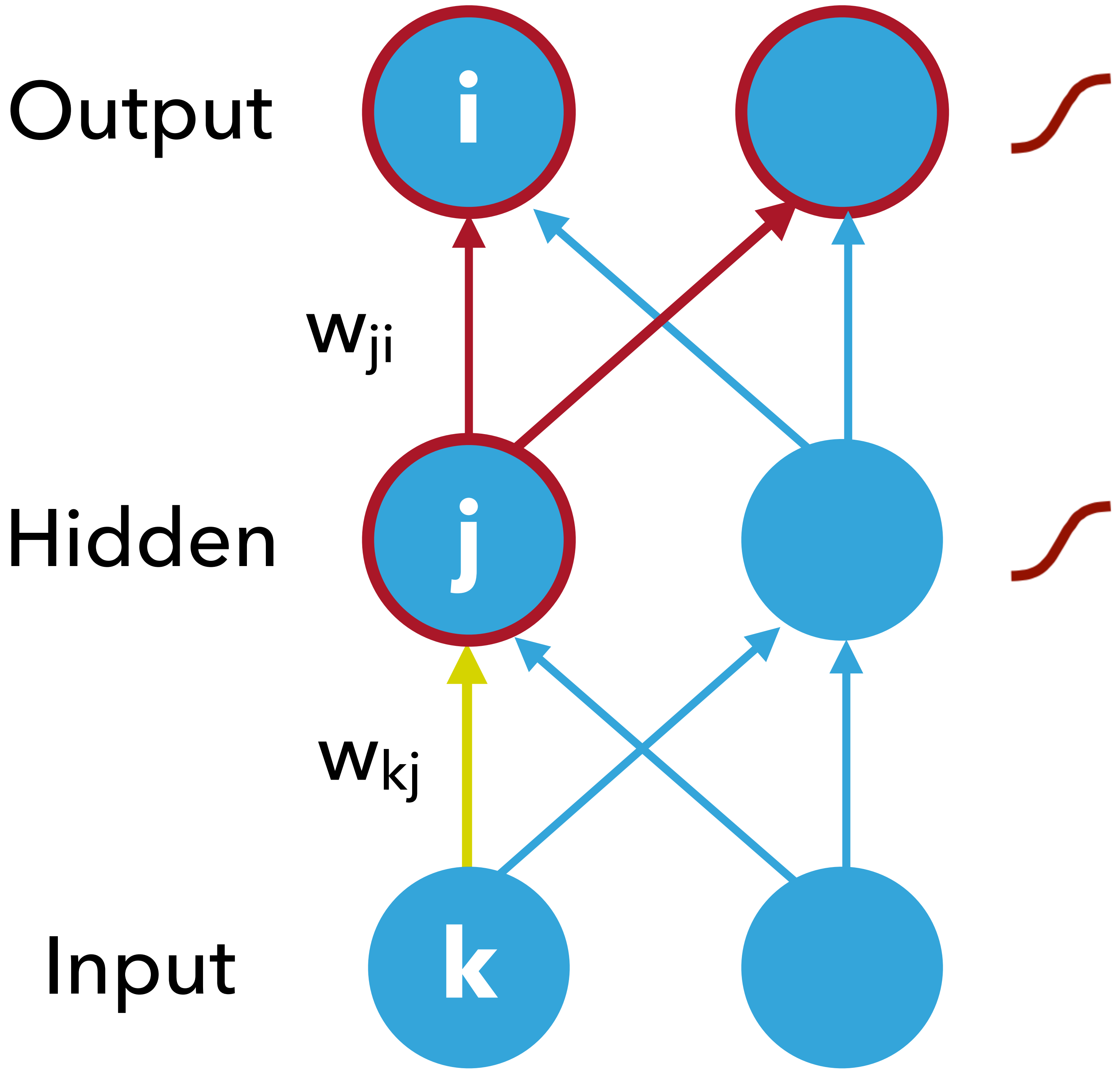
BACKPROPAGATION: LEARNING HIDDEN UNITS WEIGHTS

- Weights of hidden units w_{kj} will only affect E(w) through o_j
- Denote downstream(j) as the set of output units that take o_j as inputs



BACKPROPAGATION: LEARNING HIDDEN UNITS WEIGHTS

- Weights of hidden units w_{kj} will only affect $E(w)$ through o_j
- Denote downstream(j) as the set of output units that take o_j as inputs
- $$\frac{\partial E(w)}{\partial w_{kj}} = \sum_{i \in downstream(j)} \frac{\partial E(w)}{\partial o_i}\frac{\partial o_i}{\partial net_i}\frac{\partial net_i}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{kj}}$$
  $$= \sum_{i \in downstream(j)} -(y_i - o_i)o_i(1-o_i)\frac{\partial net_i}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{kj}}$$
  $$= \sum_{i \in downstream(j)} -(y_i - o_i)o_i(1-o_i)w_{ji}o_j(1-o_j)x_k$$



PUTTING TOGETHER: BACKPROPAGATION FOR LEARNING NEURAL NETWORK

Given a training data set with *N* data points: $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$

Initialize the weights: $\mathbf{w}=\mathbf{w}_0$

Repeat

for each $(\mathbf{x}^{(d)}, \mathbf{y}^{(d)})$ in D:

Compute the outputs $o_z^{(d)}$ for each hidden/output node *z* given the current weight **w** and data $\mathbf{x}^{(d)}$

For output units weights: $\nabla w_{ji} = -(y_i^{(d)} - o_i^{(d)})o_i^{(d)}(1 - o_i^{(d)})o_j^{(d)}$

For hidden units weights: $\nabla w_{kj} = -\sum_{i \in downstream(j)} (y_i^{(d)} - o_i^{(d)})o_i^{(d)}(1 - o_i^{(d)})w_{ji}o_j^{(d)}(1 - o_j^{(d)})x_k^{(d)}$

Update:

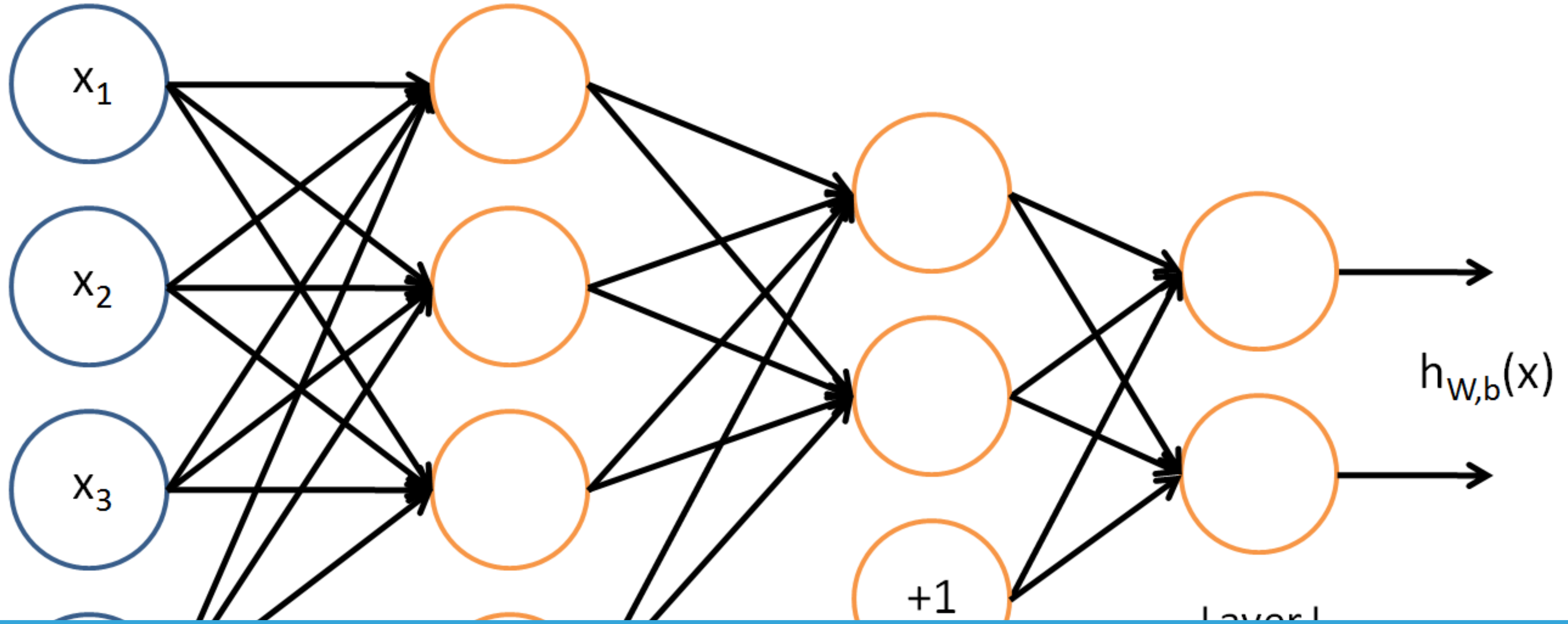
$$w_{ji} = w_{ji} - \eta \nabla w_{ji}; w_{kj} = w_{kj} - \eta \nabla w_{kj}$$

Until stopping criteria is met

NEURAL NETWORK COMPONENTS

- **Model space**
  - Set of weights **w** and b's (can combine them into a new weight vector)
- **Search algorithm**
  - Iterative refinement of weights, using backpropagation
- **Score function**
  - Minimize error (typically squared error)

FROM NEURAL NETWORKS TO DEEP LEARNING



ADDING LAYERS IN NEURAL NETWORKS GIVES THE MODEL MORE FLEXIBILITY —TRIED IN 1980S BUT DID NOT IMPROVE PERFORMANCE SUBSTANTIALLY BECAUSE BACK PROP ESTIMATION WOULD GET STUCK IN (SUBPAR) LOCAL MAXIMA

DEEP LEARNING

- Breakthrough in learning parameters for neural networks with a large number of hidden layers
- CS69000-DPL (Deep Learning)

PREDICTIVE MODELING: EVALUATION

WHAT WE'VE LEARNED SO FAR

- We've covered quite a bit of predictive models
  - Naive bayes
  - Decision trees
  - Nearest neighbors
  - Logistic regression
  - SVM
  - Neural networks

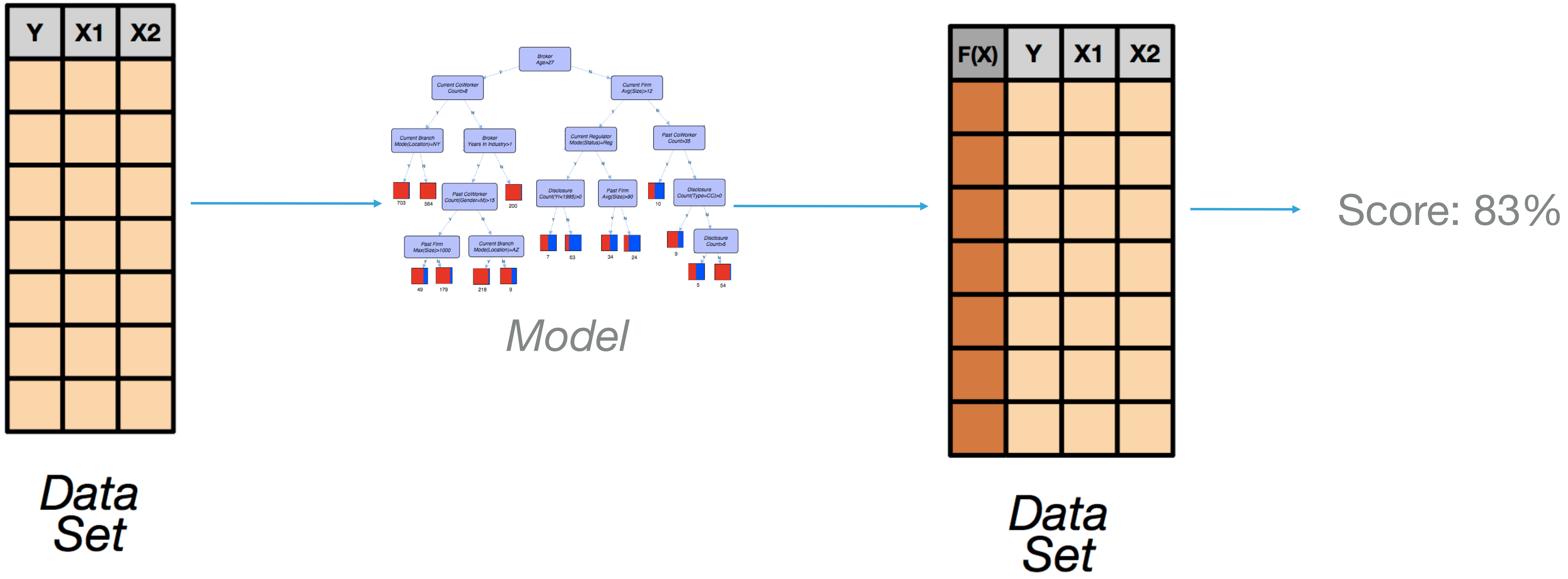
EMPIRICAL EVALUATION

- Given observed accuracy of a model on a limited amount of data, how well does this estimate generalize for additional examples?
- Given that one model outperforms another on some sample of data, how likely is it that this model is more accurate in general?
- When data are limited, what is the best way to use the data to both learn and evaluate a model?

EVALUATING CLASSIFIERS

- Goal: Estimate a classifier's performance on future (unseen) data
- **Approach 1**
  - Use the learned classifier to classify training data and estimate performance

APPROACH 1

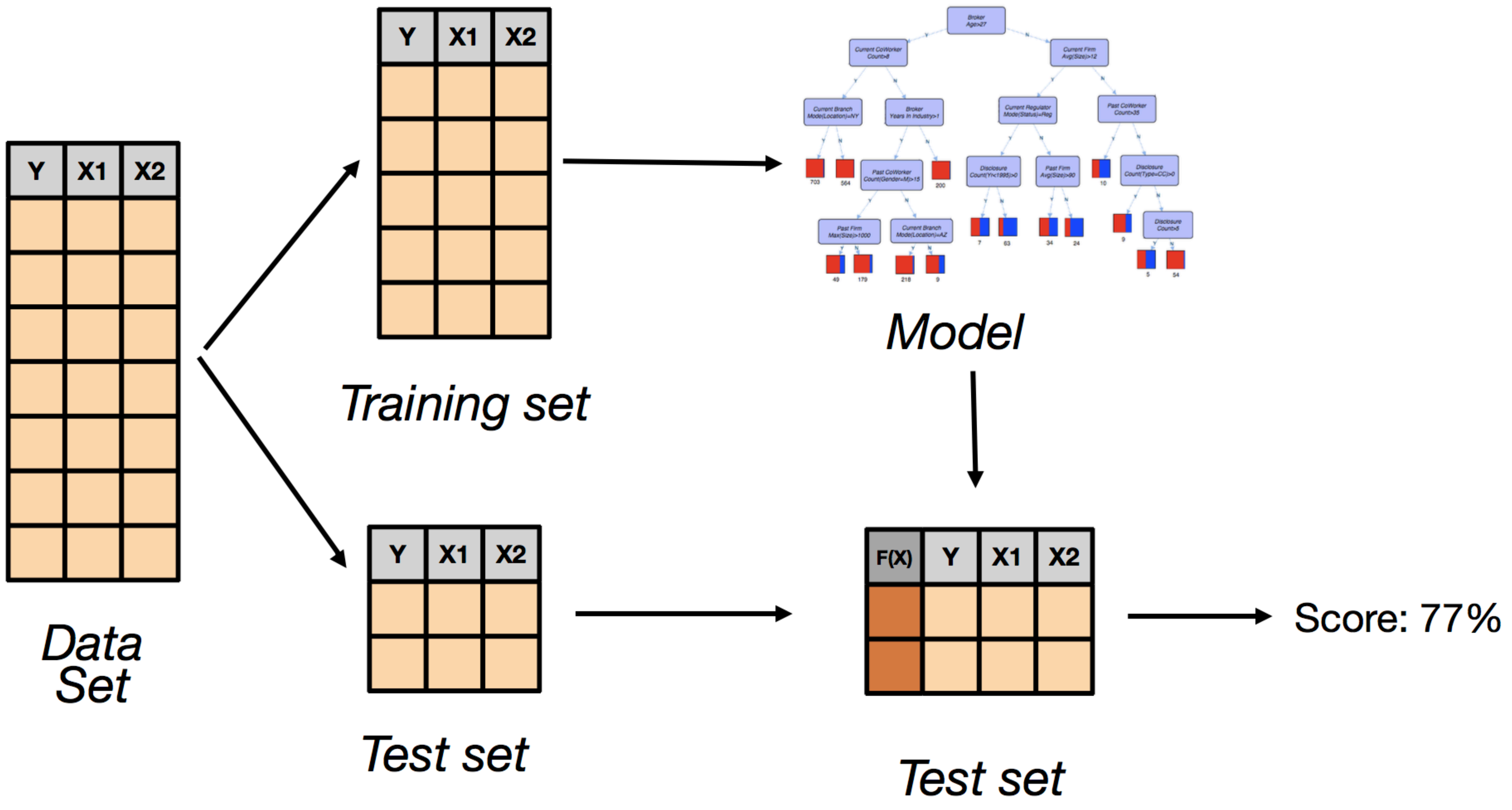


Typically produces a biased estimate of future performance

EVALUATING CLASSIFIERS

- **Approach 2**
  - Classify **disjoint** test set to estimate performance

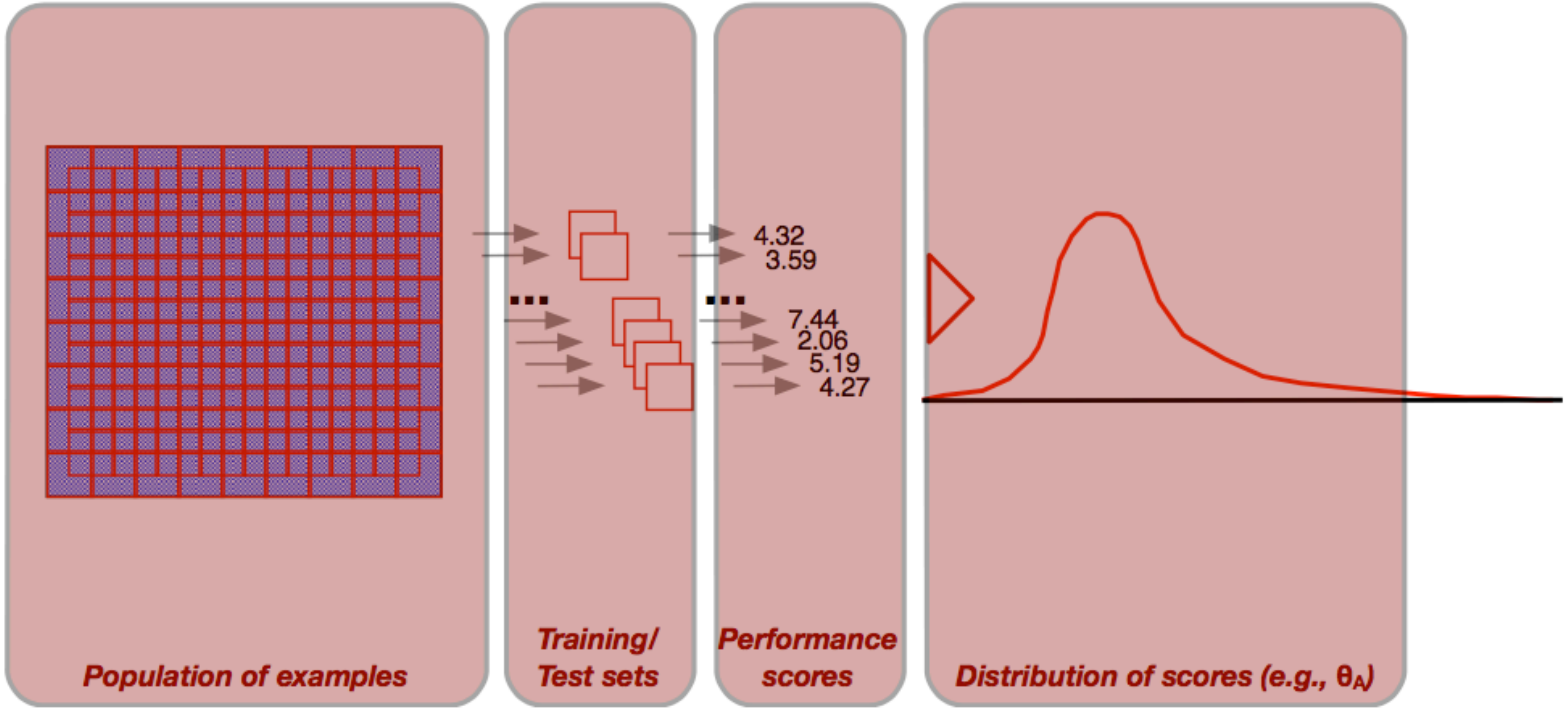
APPROACH 2



✓ An unbiased estimate of future performance

✗ But the estimate will vary due to size and makeup of test set

SAMPLING DISTRIBUTIONS



COMPARING CLASSIFIERS

- Given models A and B, how to decide which model has a better classification performance in general?
- Partition D_0 into two disjoint subsets, learn model on one subset, measure and compare performance on the other subset
- **Problem**: this is a point estimate of the model's performance, i.e., the estimate will vary due to size and makeup of test set

COMPARING CLASSIFIERS

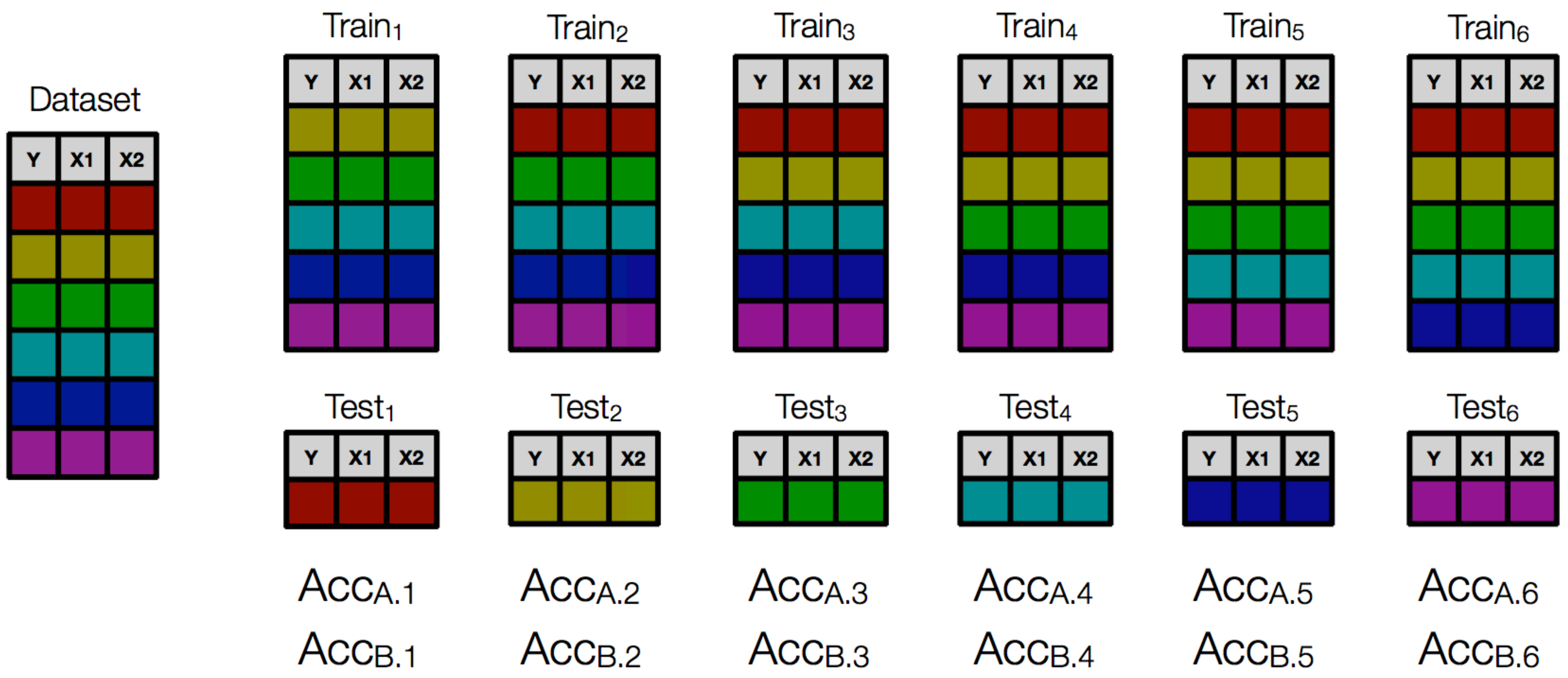
- Repeat Approach 2 for k times, i.e., randomly partition the entire dataset into disjoint training set and test set. Learn the model using the training set and evaluate on the test set.
- Compute the model's average performance over the k trials
- Plot average error and standard error bars
- Any problems?

OVERLAPPING TEST SETS

- Repeated sampling of test sets leads to overlap (i.e., dependence) among test sets… this will result in underestimation of variance
- Standard errors will be biased if performance is estimated from **overlapping** test sets *(Dietterich'98)*
- Recommendation: Use **cross-validation** to eliminate dependence between test sets

COMPARING CLASSIFIERS THROUGH CROSS VALIDATION

- Use k-fold cross-validation to get k estimates of performance for M_A and M_B

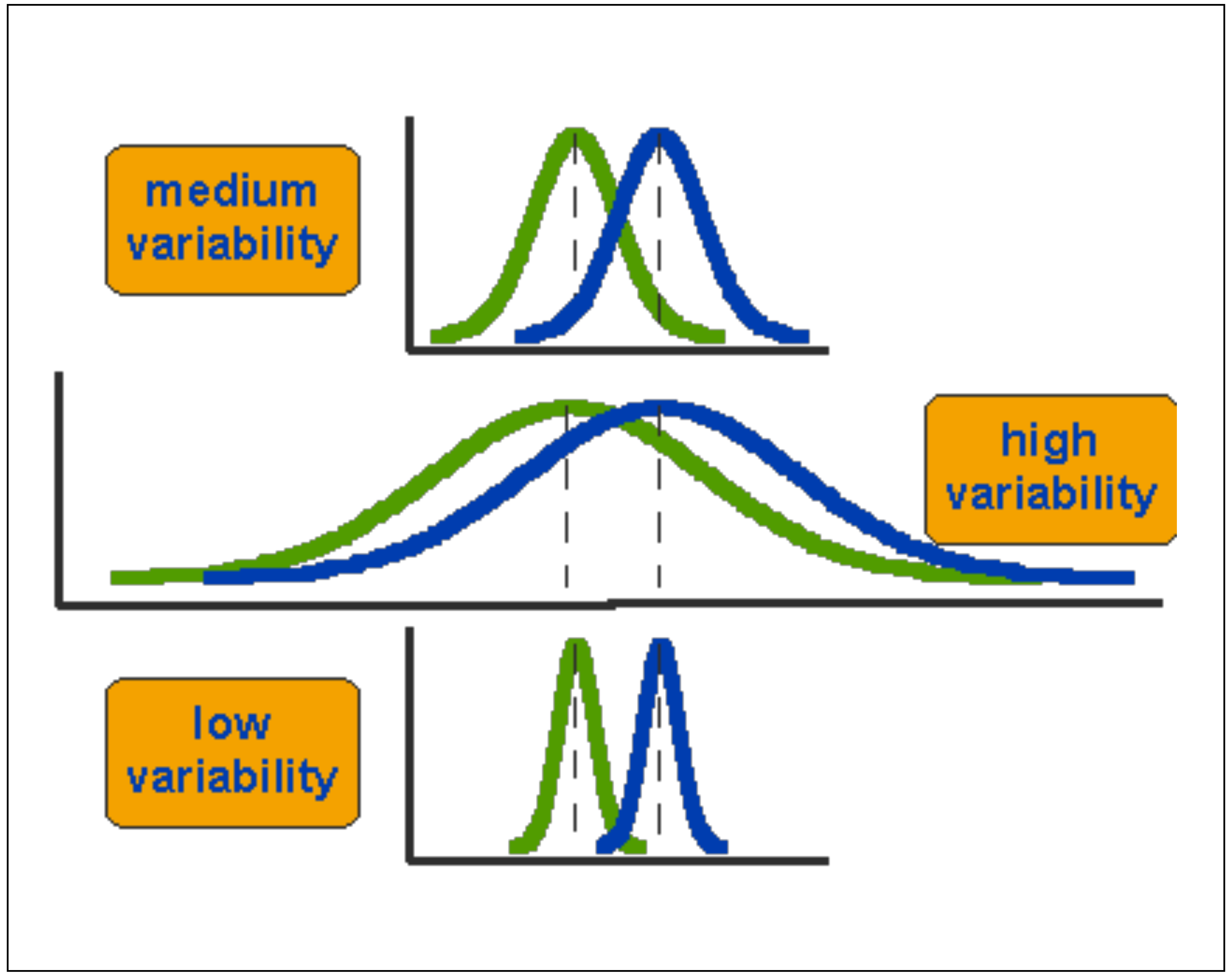


- Set of errors estimated over the test set folds provides empirical estimate of sampling distribution
- Mean is estimate of expected performance

ASSESSING SIGNIFICANCE

- Use **paired t-test** to assess whether the two distributions of errors are statistically different from each other

$Acc_{A.1}$	$Acc_{B.1}$
$Acc_{A.2}$	$Acc_{B.2}$
$Acc_{A.3}$	$Acc_{B.3}$
$Acc_{A.4}$	$Acc_{B.4}$
$Acc_{A.5}$	$Acc_{B.5}$
$Acc_{A.6}$	$Acc_{B.6}$



- Takes into account both the difference in means and the variability of the scores

USING CROSS-VALIDATION FOR MODEL SELECTION / TUNING

- Model evaluation
  - Estimate model performance across k-fold cross validation trials
  - Use performance measurement as empirical sampling distribution for model performance
  - Evaluate difference between algorithms with statistical test
- Parameter tuning
  - Decision tree example: Choose threshold for split function with cross validation
    - Repeatedly learn model with different thresholds
    - Pick threshold that shows best cross-validation performance