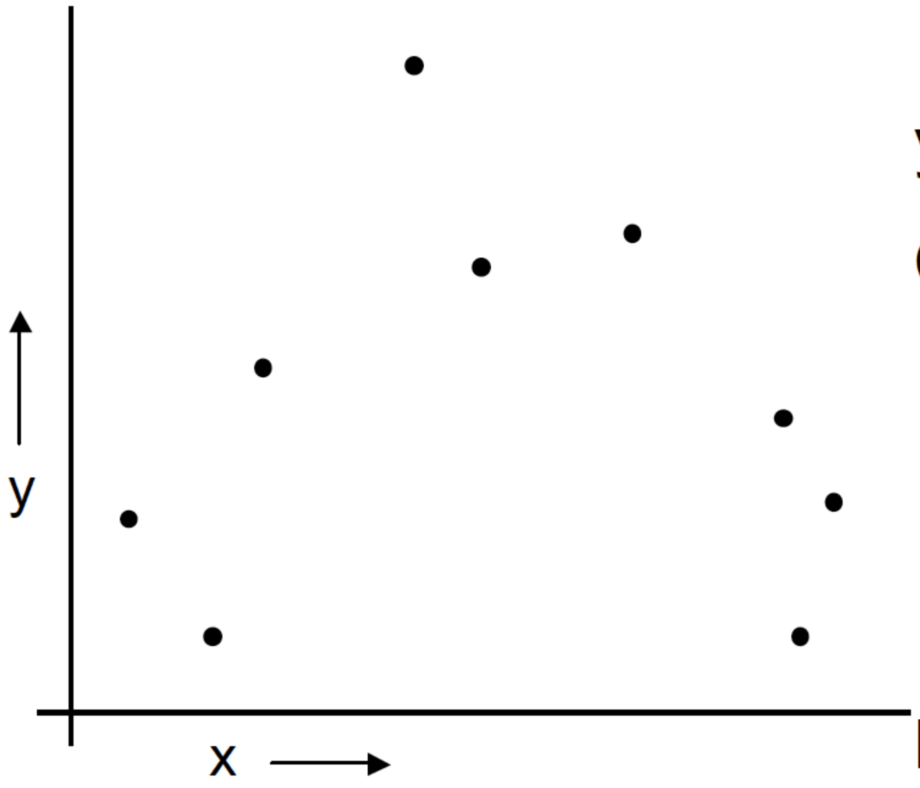
CS 440
Introduction to Artificial Intelligence

Lecture 24:

Cross-Validation – Linear Regression (continued)

May 14, 2020

A Regression Problem

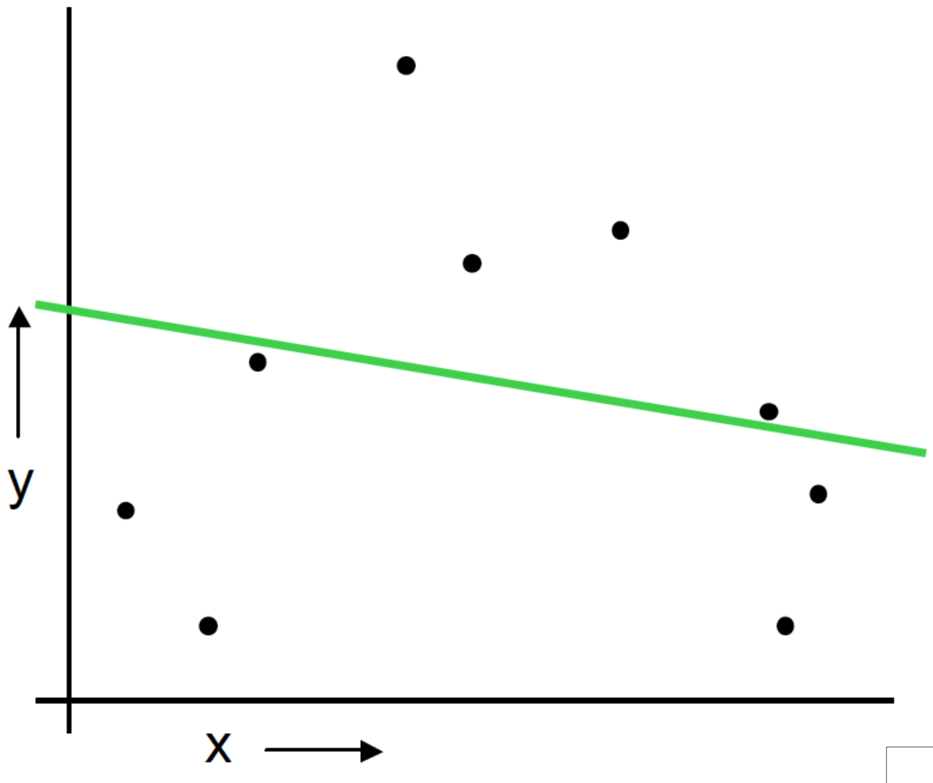


y = f(x) + noise

Can we learn f from this data?

Let’s consider three methods…

Linear Regression



Objective: Minimize the

Sum of Squared Errors

i.e. sum of squared differences

between y values

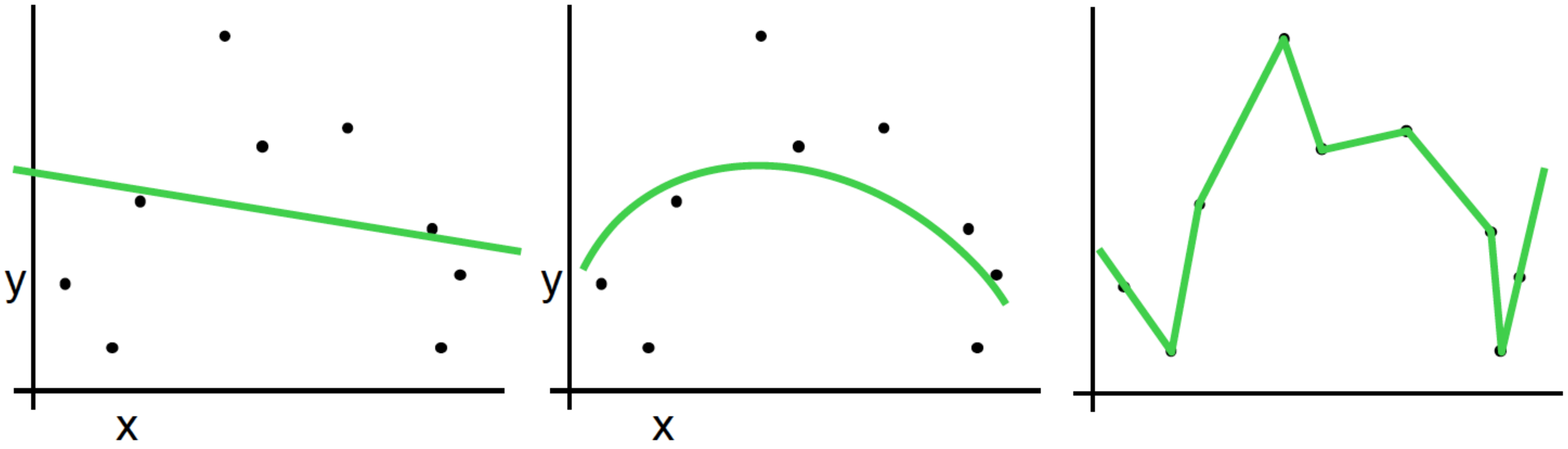
and the green line

$y = w_0 + w_1 \cdot x$

- ŷi is the prediction of the linear model
- yi the actual value for input xi
- Then minimize: Q = Σi (ŷi – yi)2

RUTGERS

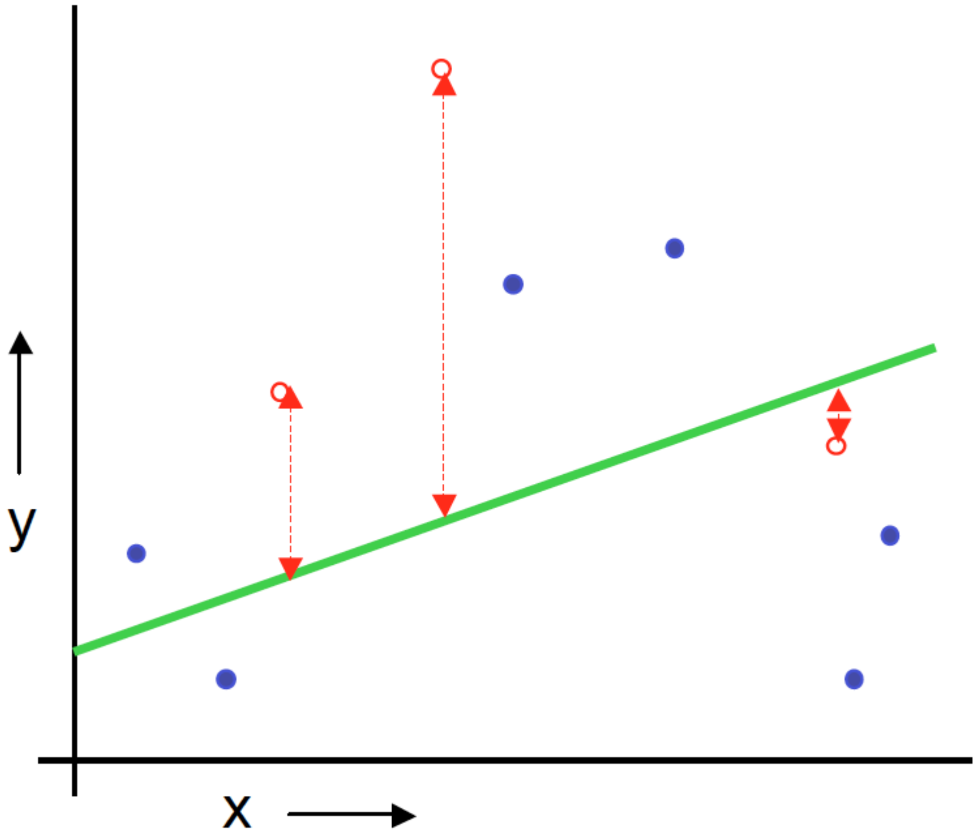
Which one is the best?



Why not choose the method with the best fit to the data?

"How well are you going to predict future data drawn from the same distribution?"

Test Set



(Linear regression example)

Mean Squared Error = 2.4

1. Randomly choose 30% of the data to be in a test set

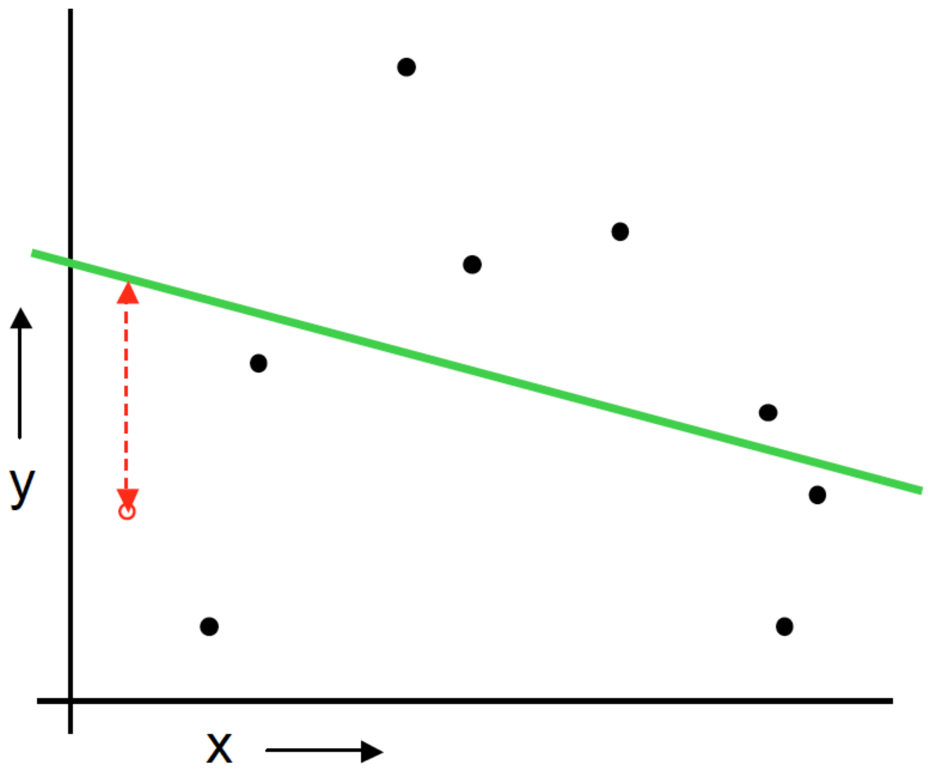
2. The remainder is a training set

3. Perform your regression on the training set

4. Estimate your future performance with the test set

RUTGERS

Leave One Out Cross Validation

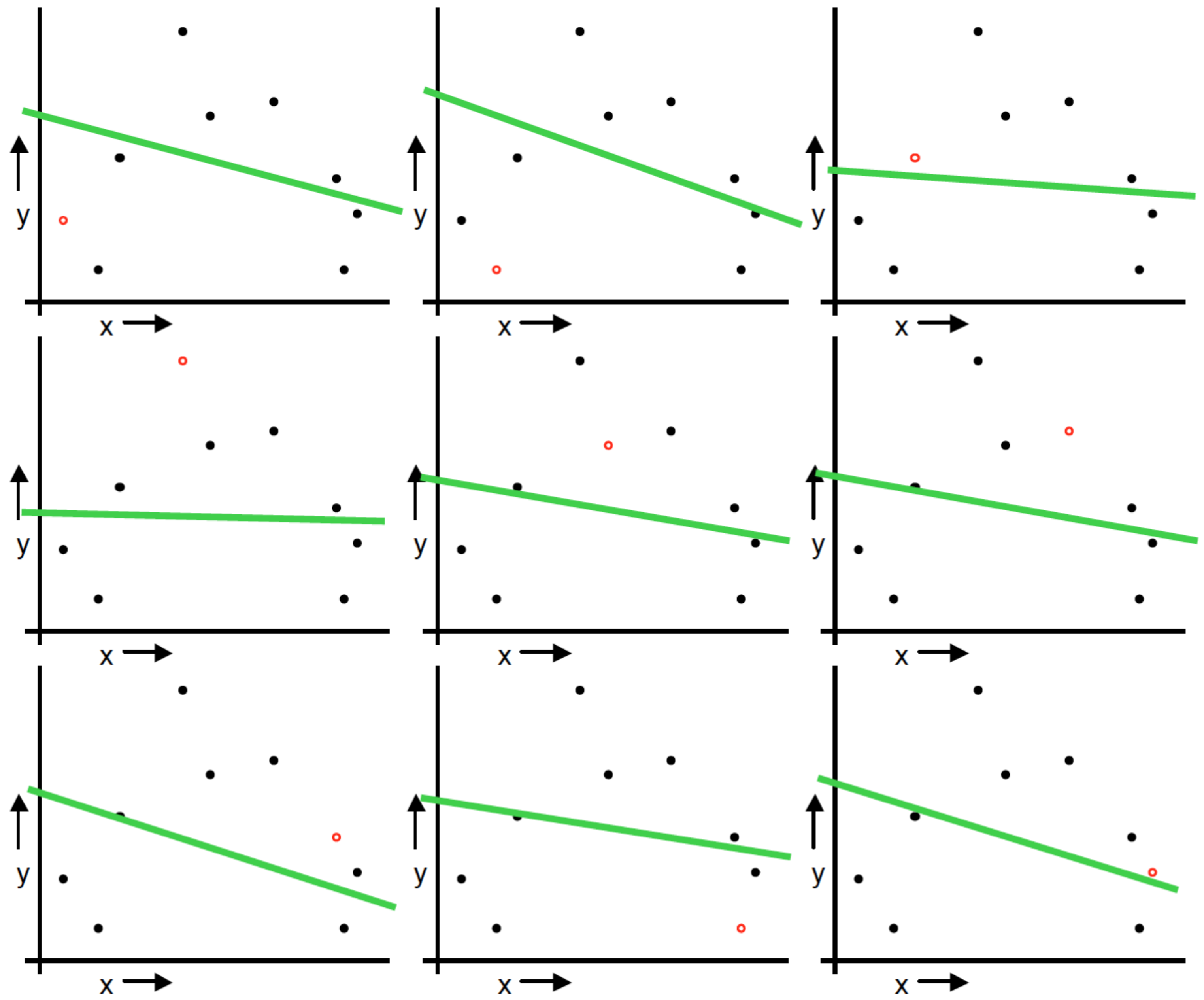


For k=1 to R

1. Let (x_k, y_k) be the kth record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

RUTGERS

Leave One Out Cross Validation



For k=1 to R

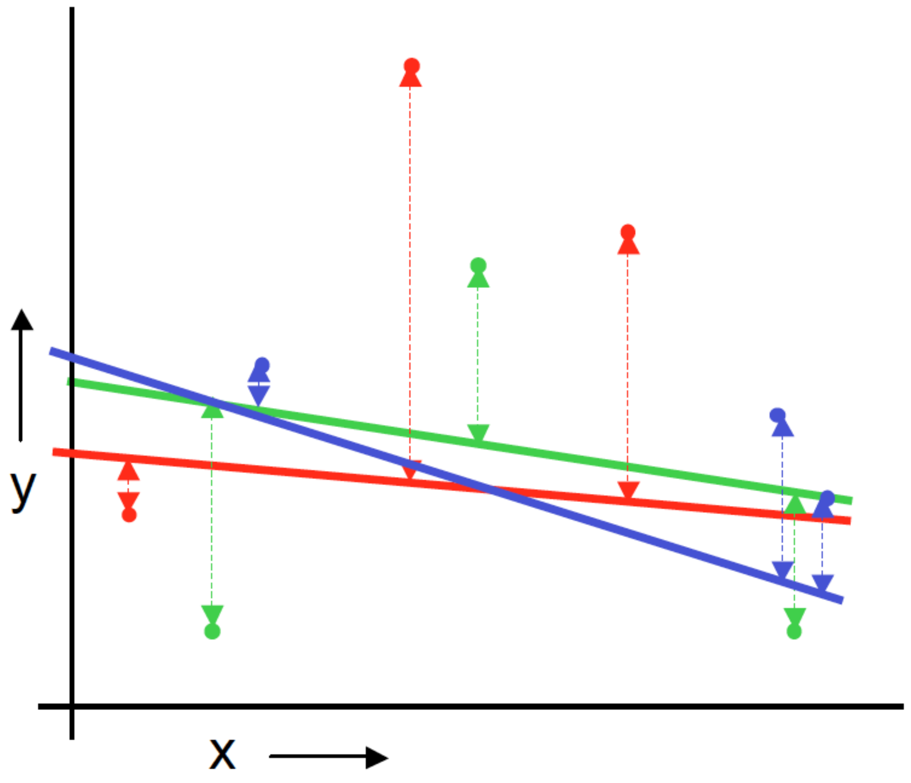
1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

MSE_{LOOCV} = 2.12

K-fold Cross Validation

k-fold Cross Validation



Linear Regression
MSE_{3FOLD}=2.05

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

Linear Regression

- It is traditional (going back to Gauss) to use the squared loss function L_2 summed over all the training examples

$$Loss(h_w) = \sum_{j=1}^{N} L_2(y_j, h_w(x_j)) = \sum_{j=1}^{N} (y_j - h_w(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

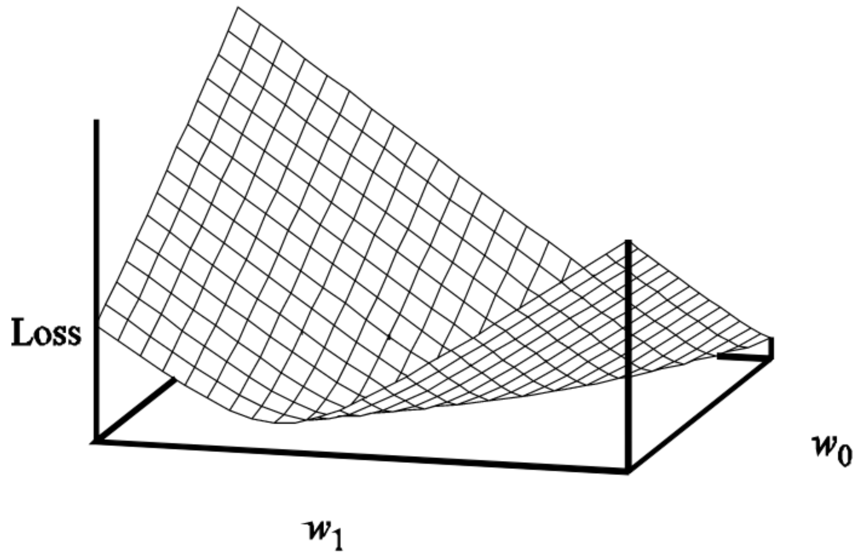
- We would like to find $\mathbf{w}^* = \arg\min_{\mathbf{w}} Loss(h_{\mathbf{w}})$
- The sum $\sum_j (y_j - (w_1 x + w_0))^2$ is minimized when its partial derivatives with respect to w_0 and w_1 are zero

$$\frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0$$

$$\frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0$$

RUTGERS

Linear Regression



- These equations have a unique solution

$$w_1 = \frac{N\left(\sum x_j y_j\right) - \left(\sum x_j\right)\left(\sum y_j\right)}{N\left(\sum x_j^2\right) - \left(\sum x_j\right)^2}$$

$$w_0 = \left(\sum y_j - w_1\left(\sum x_j\right)\right) \Big/ N$$

- The weight space defined by w_0 and w_1 is convex
- This is true for every linear regression problem with an L_2 loss function, and it implies that there are no local minima

RUTGERS

Discussion

How would you minimize lost for other functions?

- Parabolas
- Third order polynomial
- ect.

RUTGERS

Beyond Linear Regression

- To go beyond linear models, we will need to face the fact that the equation defining minimum loss will often have no closed-form solution
- Instead, we will face a general optimization search in continuous weight space
- As we already know, such problems can be addressed by a hill-climbing algorithm that follows the ***gradient*** of the function to be optimized
- Because we are trying to minimize the loss, we will use ***gradient descent***
- We choose any starting point in weight space and then move to a neighboring point that is downhill, repeating until we converge on the minimum possible loss

RUTGERS

Learning Hypothesis Parameters

```
$\mathbf{w}$ ← any point in the parameter space
loop until convergence do
for each $w_i$ in $\mathbf{w}$ do
        $w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$
```

- The step size parameter α is usually called the ***learning rate*** when we are trying to minimize loss in a learning problem
- It can be a fixed constant, or it can decay over time as the learning problem proceeds
- For univariate regression, the loss function is a quadratic function, so the partial derivative will be a linear function

RUTGERS

One training example

- Let's consider the case of only one training example, (x, y):

$$\begin{aligned}\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))^2 \\ &= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x)) \\ &= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_1 x + w_0))\end{aligned}$$

- Applying this to both w_0 and w_1 we get:

$$\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x))$$

$$\frac{\partial}{\partial w_1} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x$$

Many Training Examples

- Plugging these values back to the gradient descent update rule (folding the constant 2 into the learning rate α), we get the following learning rules for the weights

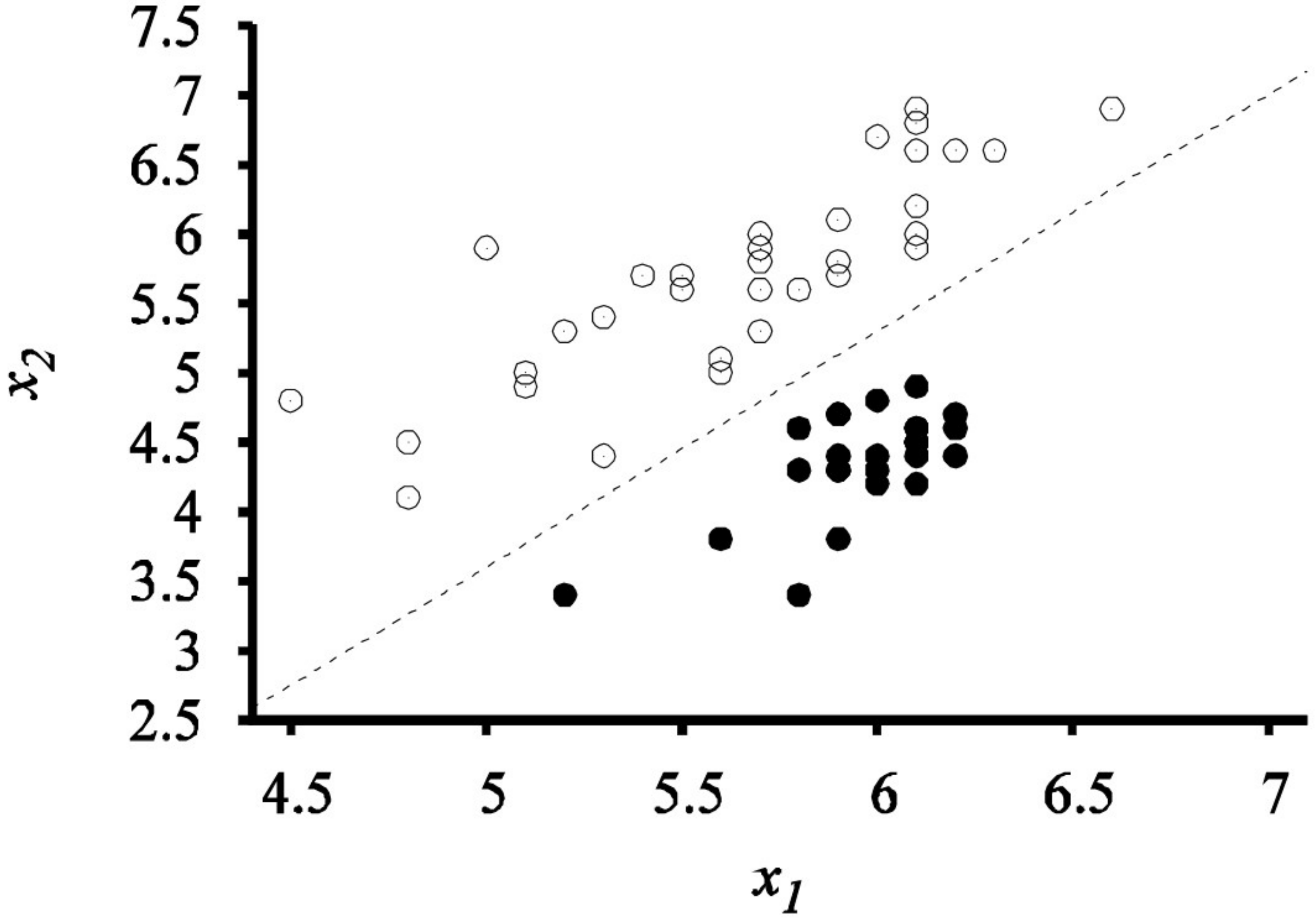
$$w_0 \leftarrow w_0 + \alpha(y - h_w(x))$$
$$w_1 \leftarrow w_1 + \alpha(y - h_w(x)) \cdot x$$

- Intuitively:
  - if $h_w(x) > y$ – the output of the hypothesis is too large – reduce w_0 a bit
  - Reduce w_1 if x was a positive input but increase w_1 if x was a negative input
- For N training examples the derivative of a sum is the sum of the derivatives, and we have

$$w_0 \leftarrow w_0 + \alpha \textstyle\sum_j (y_j - h_w(x_j))$$
$$w_1 \leftarrow w_1 + \alpha \textstyle\sum_j (y_j - h_w(x_j)) \cdot x_j$$

RUTGERS

Linear Classification Example



RUTGERS

Linear Classifiers with a Threshold

- Linear functions can be used to do classification by finding a ***decision boundary*** (a linear separator) – a line or a surface in higher dimensions – that separates the two classes (if the data is ***linearly separable***)
- $h_{\mathbf{w}}(\mathbf{x}) = 1$ if $\mathbf{w} \cdot \mathbf{x} \geq 0$ and 0 otherwise
- We can think of h as passing the linear function $\mathbf{w} \cdot \mathbf{x}$ through a threshold function

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

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

- For regression minimizing the loss could be considered through closed form solution and by gradient descent in weight space
- Here we cannot do either of those things because the gradient is zero almost everywhere in weight space except at those points where $\mathbf{w} \cdot \mathbf{x} = 0$, and at those points the gradient is undefined

Perceptron Learning Rule

- There is a simple weight update rule that converges to a solution
- It provides a linear separator that classifies the data perfectly provided the data are linearly separable

- For a single example $(\mathbf{x}, y)$, we have the ***perceptron learning rule***

$$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x})) \cdot x_i$$

  which is essentially identical to the update rule for linear regression

- Because we are considering 0/1 classification problem, the behavior is somewhat different

Perceptron Rule

$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x})) \cdot x_i$

- Both the true value y and the hypothesis output $h_{\mathbf{w}}(\mathbf{x})$ are either 0 or 1:
  - If $y = h_{\mathbf{w}}(x)$ the output is correct, and the weights are not changed
  - If $y = 1$ but $h_{\mathbf{w}}(\mathbf{x}) = 0$, then w_i is increased when the corresponding input x_i is positive and decreased when x_i is negative.
    - This makes sense, because we want to make $\mathbf{w} \cdot \mathbf{x}$ bigger so that $h_{\mathbf{w}}(\mathbf{x})$ outputs a 1
  - If $y = 0$ but $h_{\mathbf{w}}(\mathbf{x}) = 1$, then w_i is decreased when the corresponding input x_i is positive and increased when x_i is negative.
    - This makes sense, because we want to make $\mathbf{w} \cdot \mathbf{x}$ smaller so that $h_{\mathbf{w}}(\mathbf{x})$ outputs a 0

RUTGERS

Convergence

- Typically the learning rule is applied one example at a time, choosing examples at random (stochastic gradient descent)

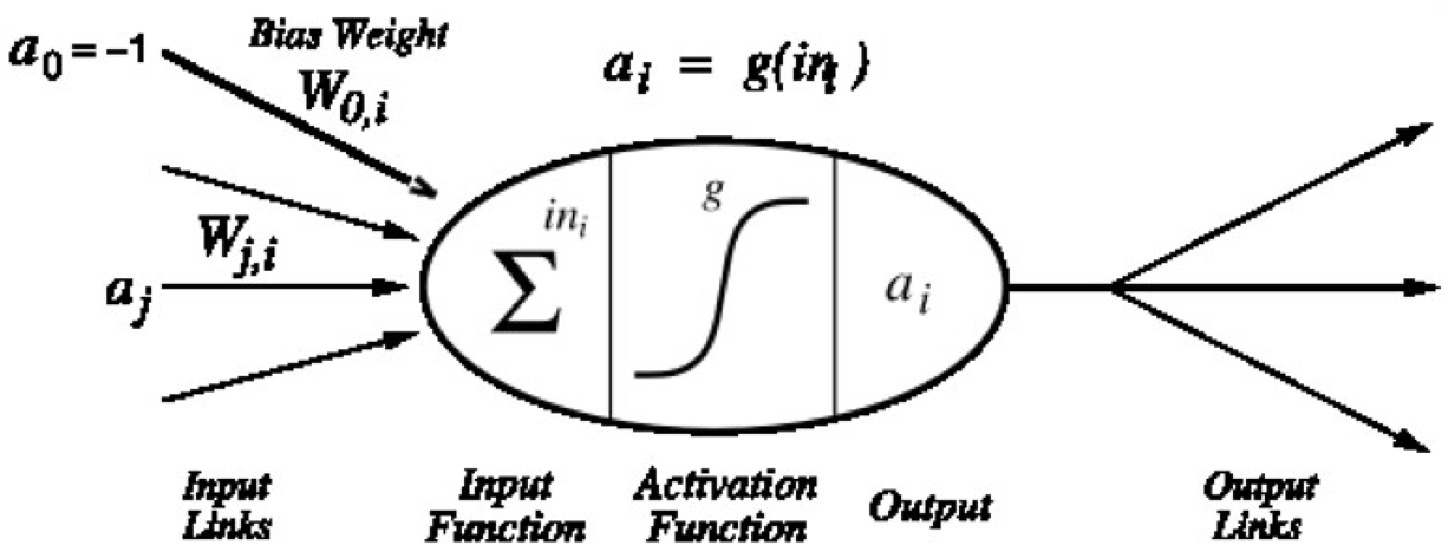
- The perceptron rule may not converge to a stable solution for fixed learning rate α
- However, if α decays as $O(1/t)$ where t is the iteration number, then the perceptron learning rule converges to a minimum error solution when examples are presented in a random sequence

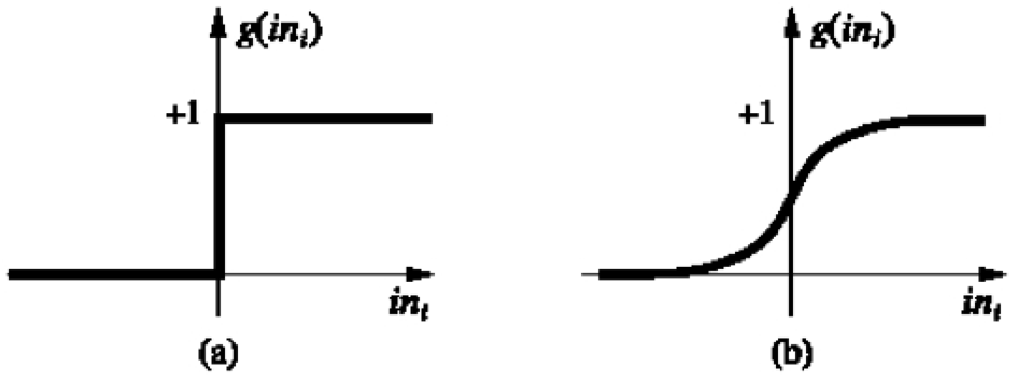
- If data points are not linearly separable, the learning rule may fail to converge

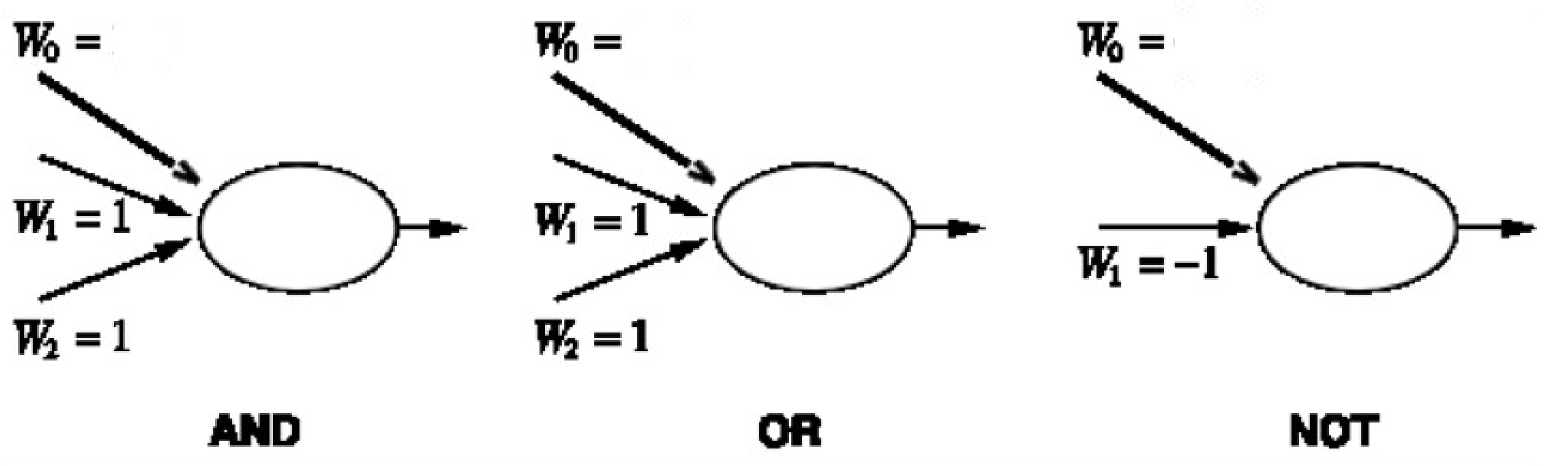
- Finding the minimum-error solution is NP-hard

RUTGERS

Artificial Model of a Neuron



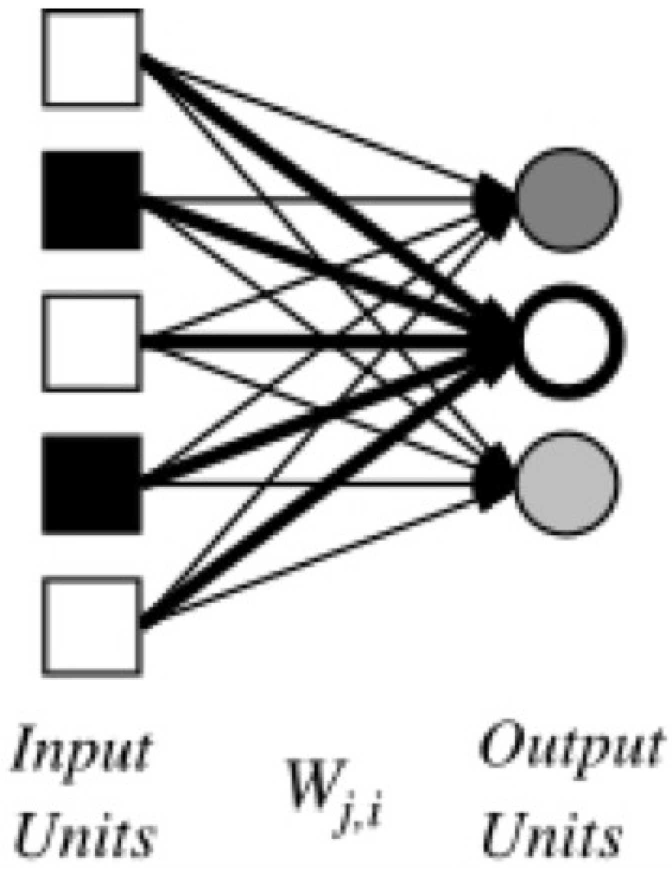




$W_0 =$
$W_1 = 1$
$W_2 = 1$
AND
$W_0 =$
$W_1 = 1$
$W_2 = 1$
OR
$W_0 =$
$W_1 = -1$
NOT

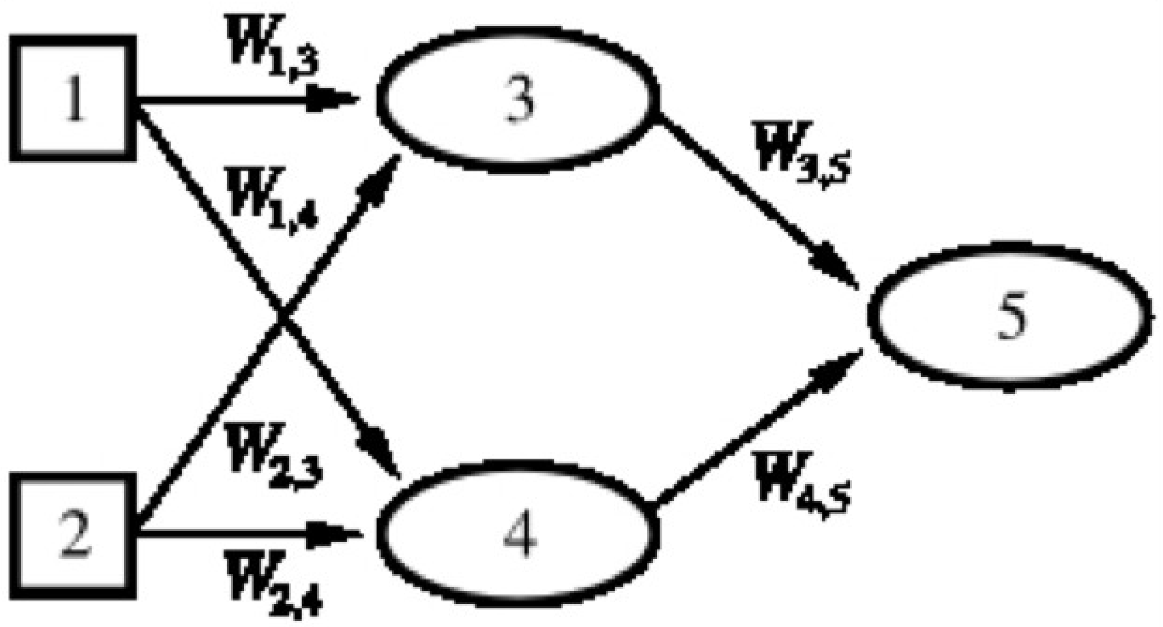
RUTGERS

Perceptron

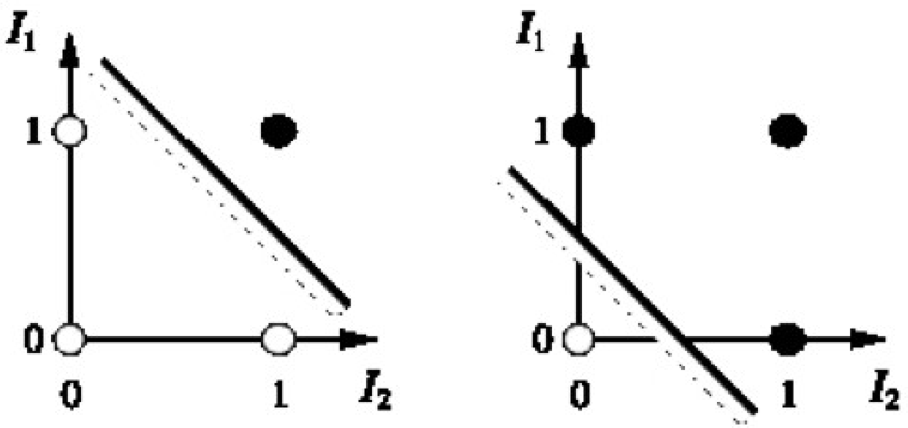


RUTGERS

Multi-Layer Network Structure

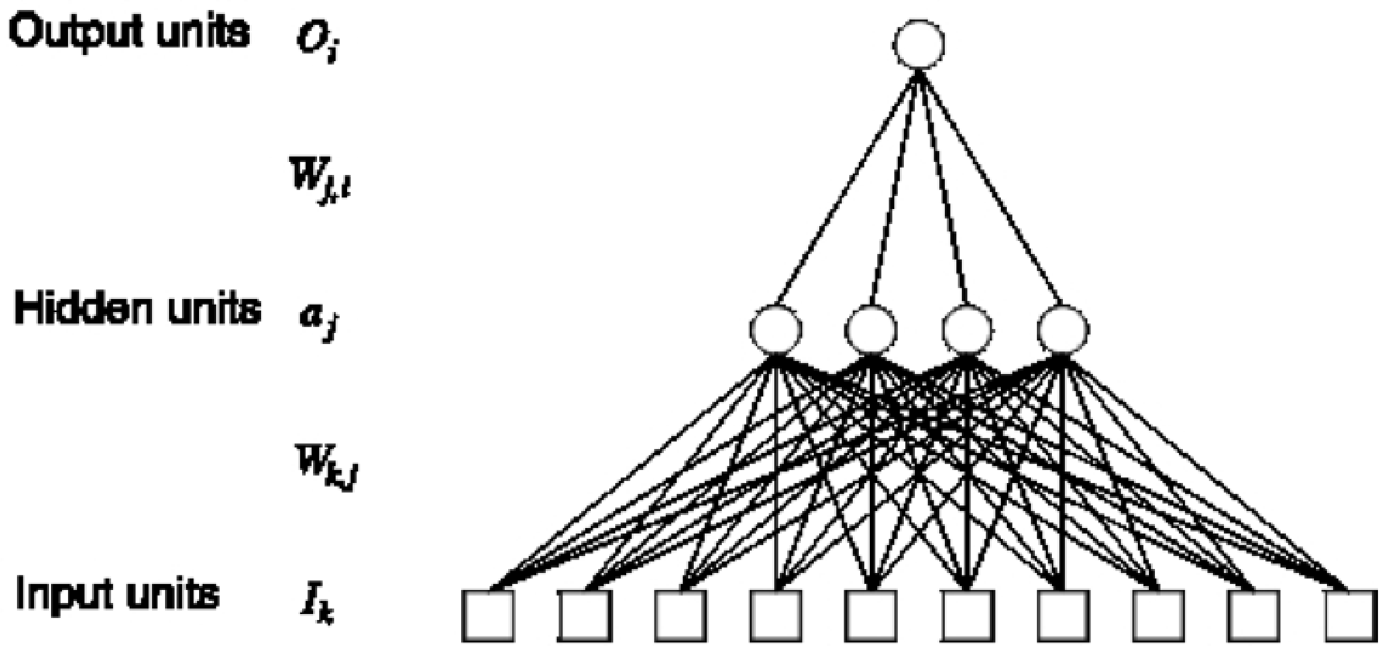


Linear Separation



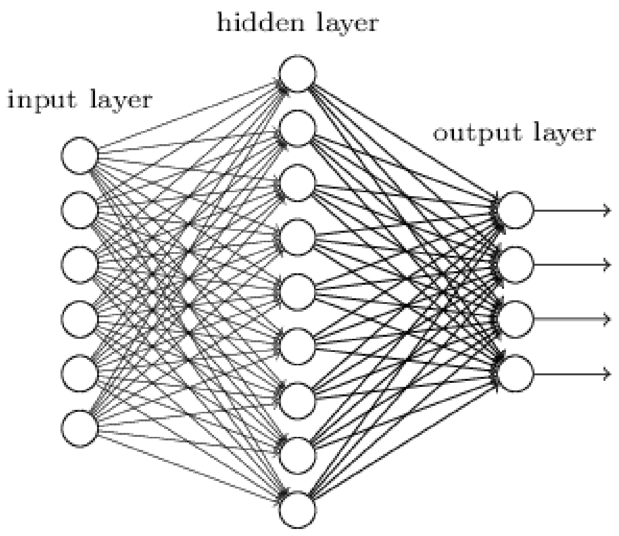
(a) I_1 and I_2 (b) I_1 or I_2

Multi-Layer Network

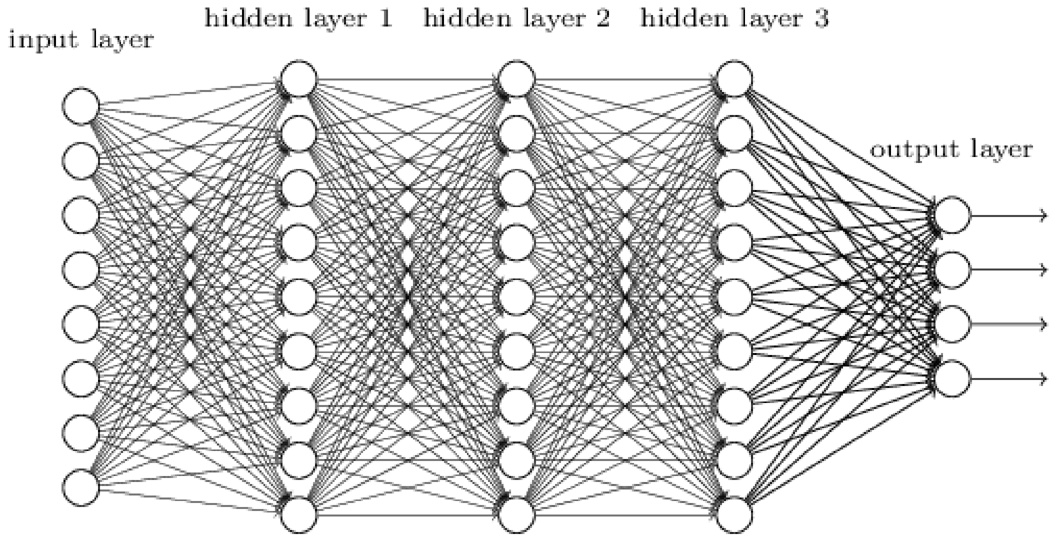


Deep Network Structures

"Non-deep" feedforward neural network



Deep neural network


input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer