Machine Learning

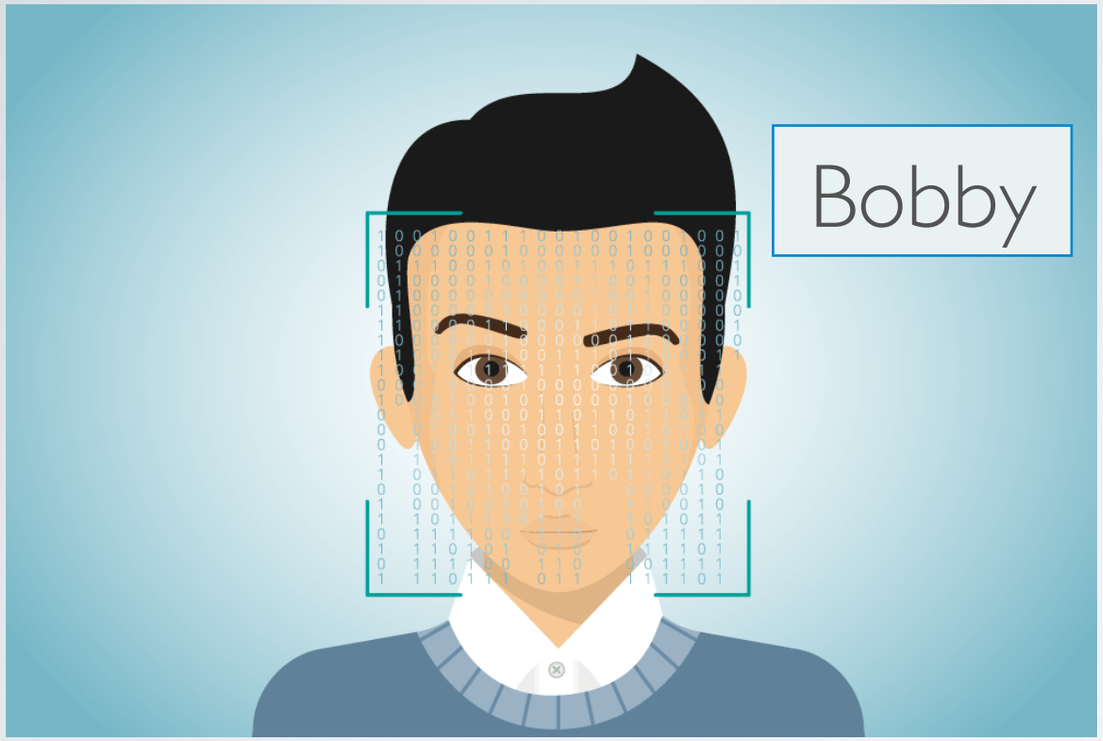
CS16: Introduction to Data Structures & Algorithms

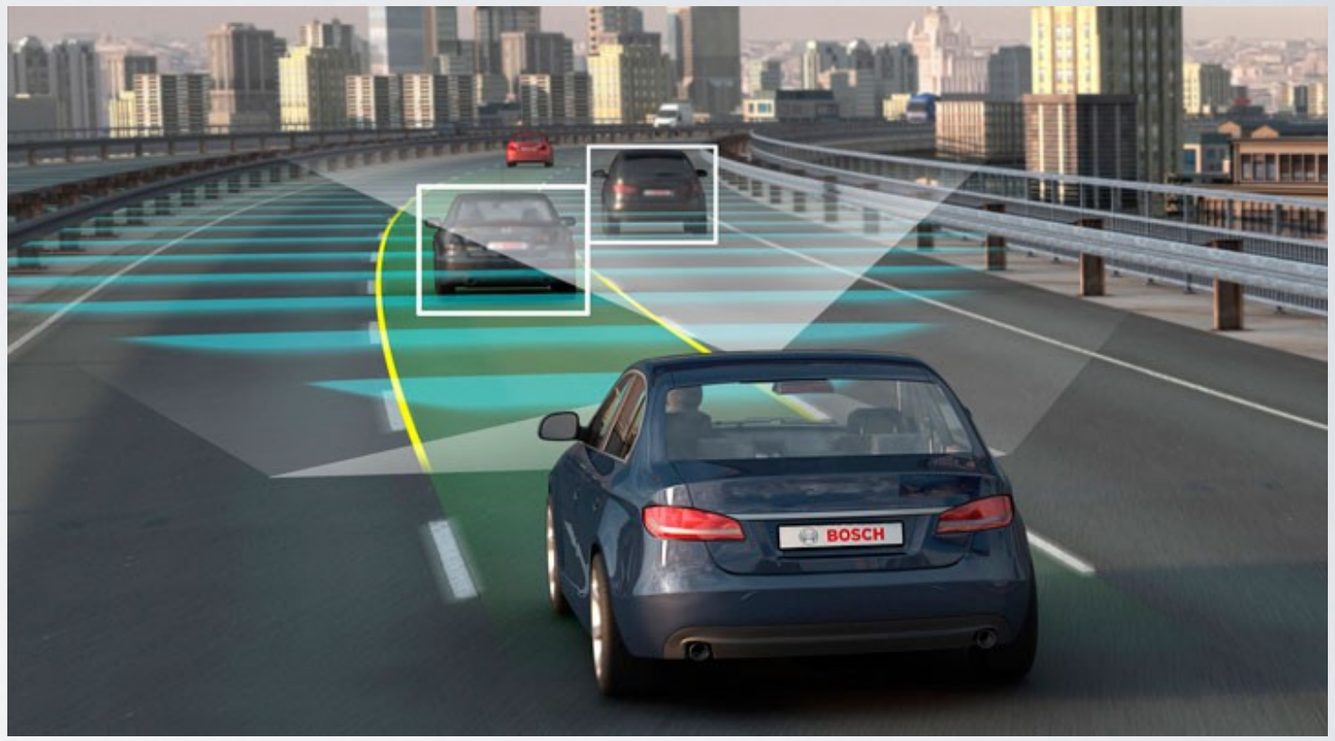
Spring 2019

Outline

- Overview
- Artificial Neurons
- Single-Layer Perceptrons
- Multi-Layer Perceptrons
- Overfitting and Generalization
- Applications

What do think of when you hear "Machine Learning"?





"Alexa, play Despacito."

What does it mean for machines to learn?

- Can machines think?
- Difficult question to answer because vague definition of "think":
  - Ability to process information/perform calculations
  - Ability to arrive at 'intelligent' *results*
  - Replication of the 'intelligent' *process*

Let's Think About This Differently

- Alan Turing, in "Computing Machinery and Intelligence" (1950)
  - Turing's test: the Imitation Game
  - Proposed that we instead consider the question, "Can machines do what we (as thinking entities) do?"
- A machine learns when its *performance* at a particular *task* improves with *experience*

Machine Learning Algorithm Structure

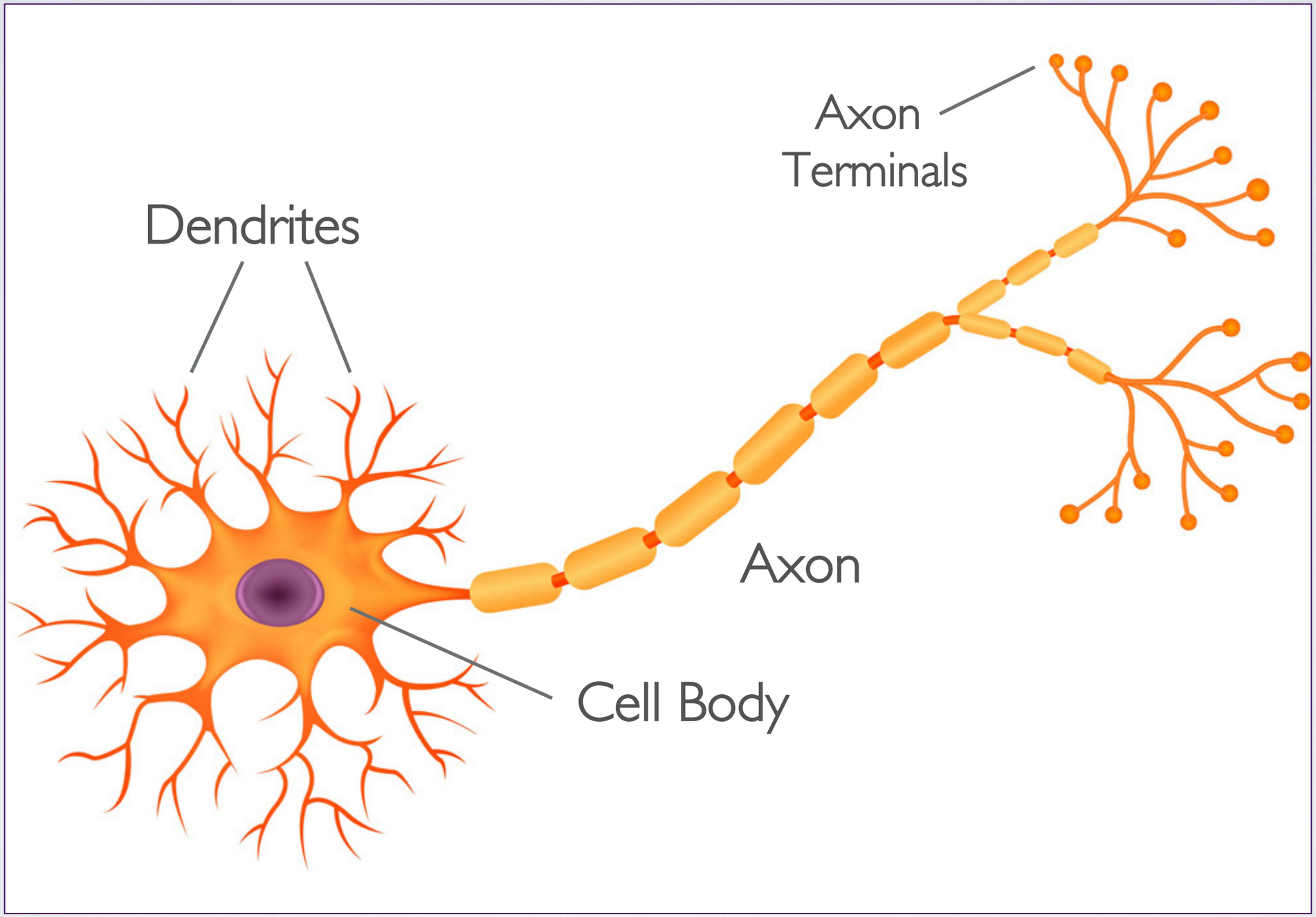
- Three key components:
  - **Representation**: define a space of possible programs
  - **Loss function**: decide how to score a program's performance
  - **Optimizer**: how to search the space for the program with the highest score

- Let's revisit decision trees:
  - **Representation**: space of possible trees that can be built using attributes of the dataset as internal nodes and outcomes as leaf nodes
  - **Loss function**: percent of testing examples misclassified
  - **Optimizer**: choose attribute that maximizes information gain

Neurons

- The brain has **100** billion neurons
- Neurons are connected to **1000**'s of other neurons by synapses
- If the neuron's electrical potential is high enough, neuron is activated and fires
- Each neuron is very simple
  - it either fires or not depending on its potential
  - but together they form a very complex "machine"

Neuron Anatomy (…very simplified)



Artificial Neuron

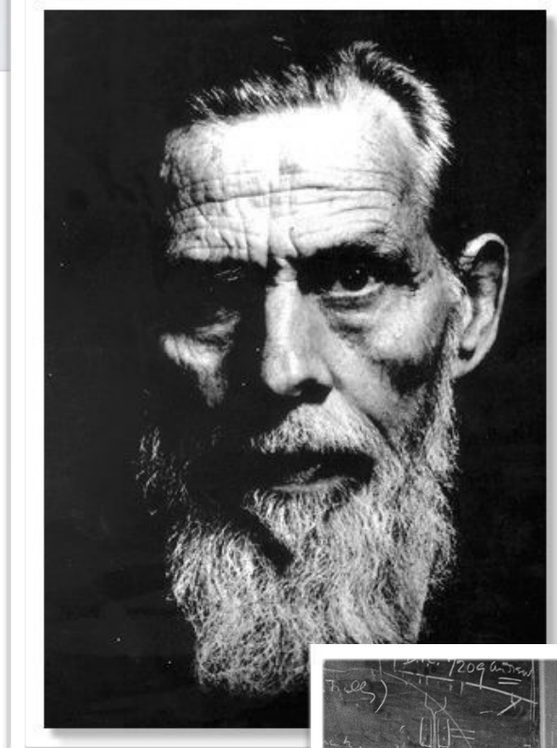
Bulletin of Mathematical Biology Vol. 52, No. 1/2, pp. 99–115, 1990.
Printed in Great Britain.

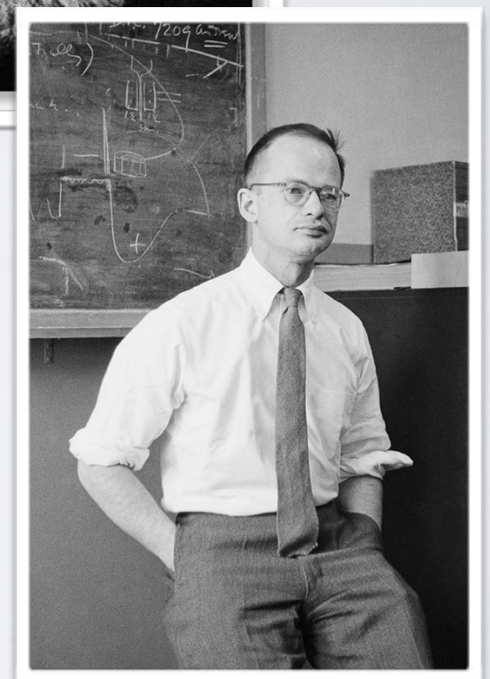
0092–8240/90$3.00 + 0.00
Pergamon Press plc
Society for Mathematical Biology

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*

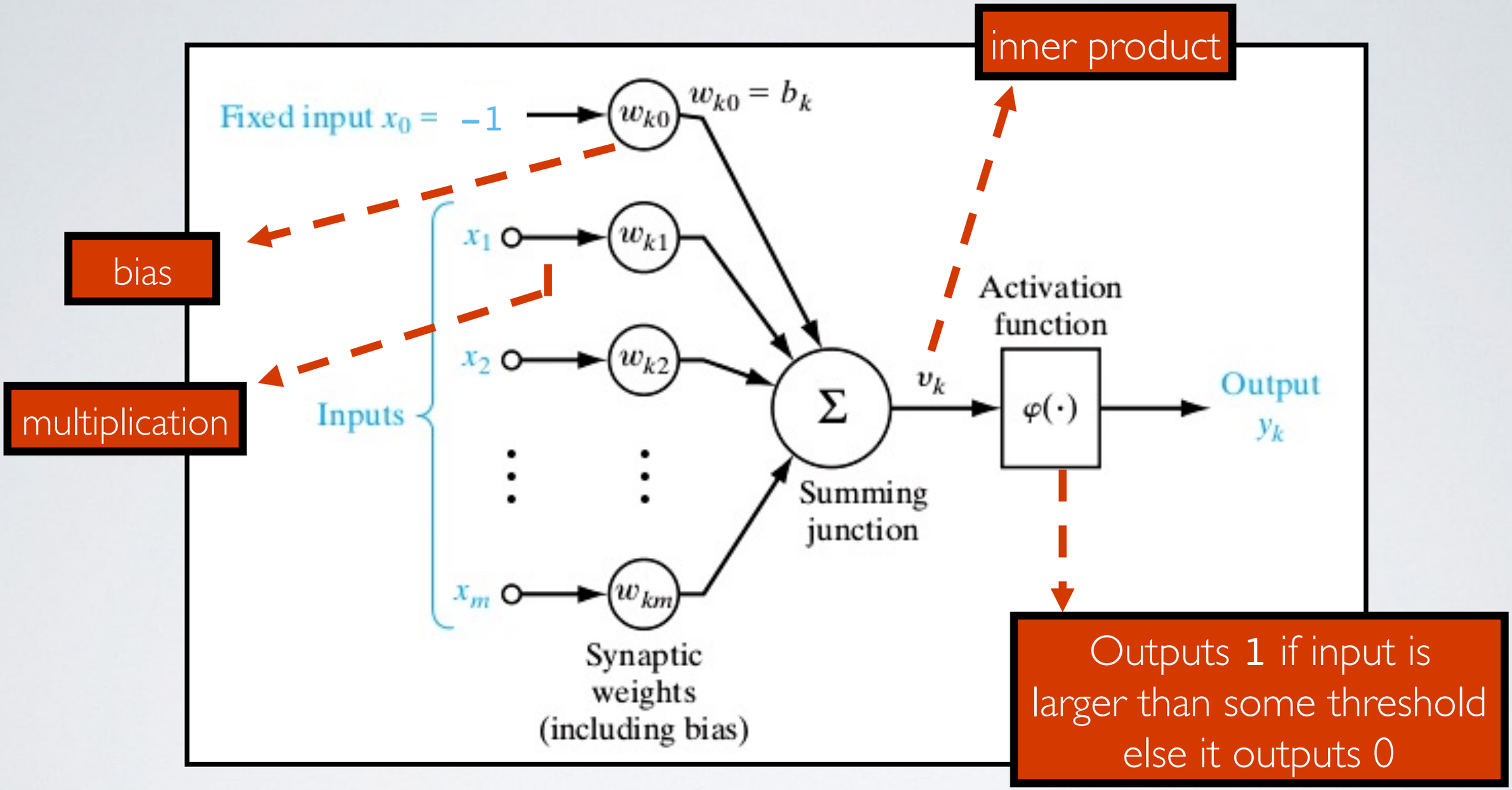
■ WARREN S. MCCULLOCH AND WALTER PITTS
University of Illinois, College of Medicine,
Department of Psychiatry at the Illinois Neuropsychiatric Institute,
University of Chicago, Chicago, U.S.A.

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

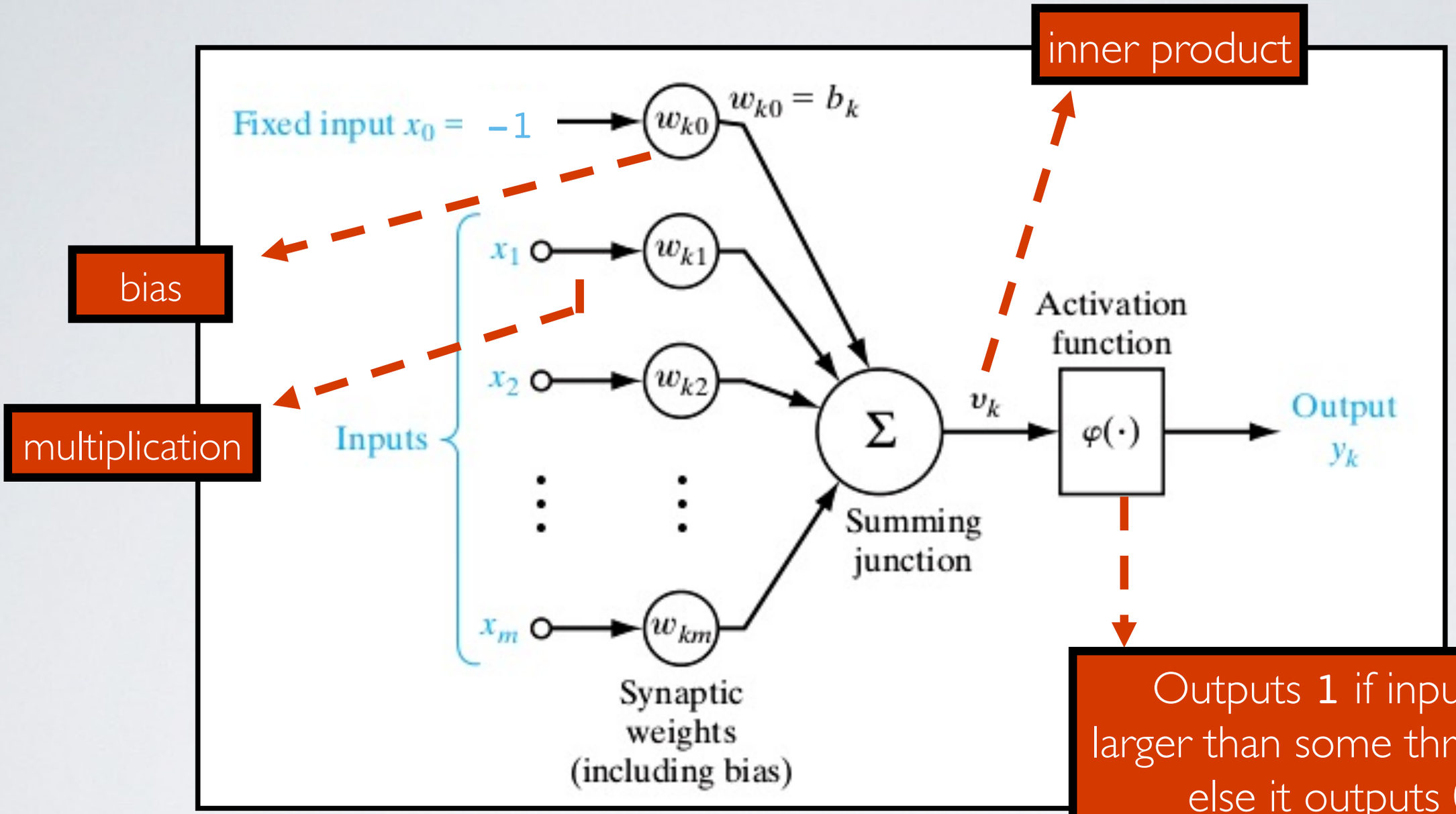




Artificial Neuron



Artificial Neuron



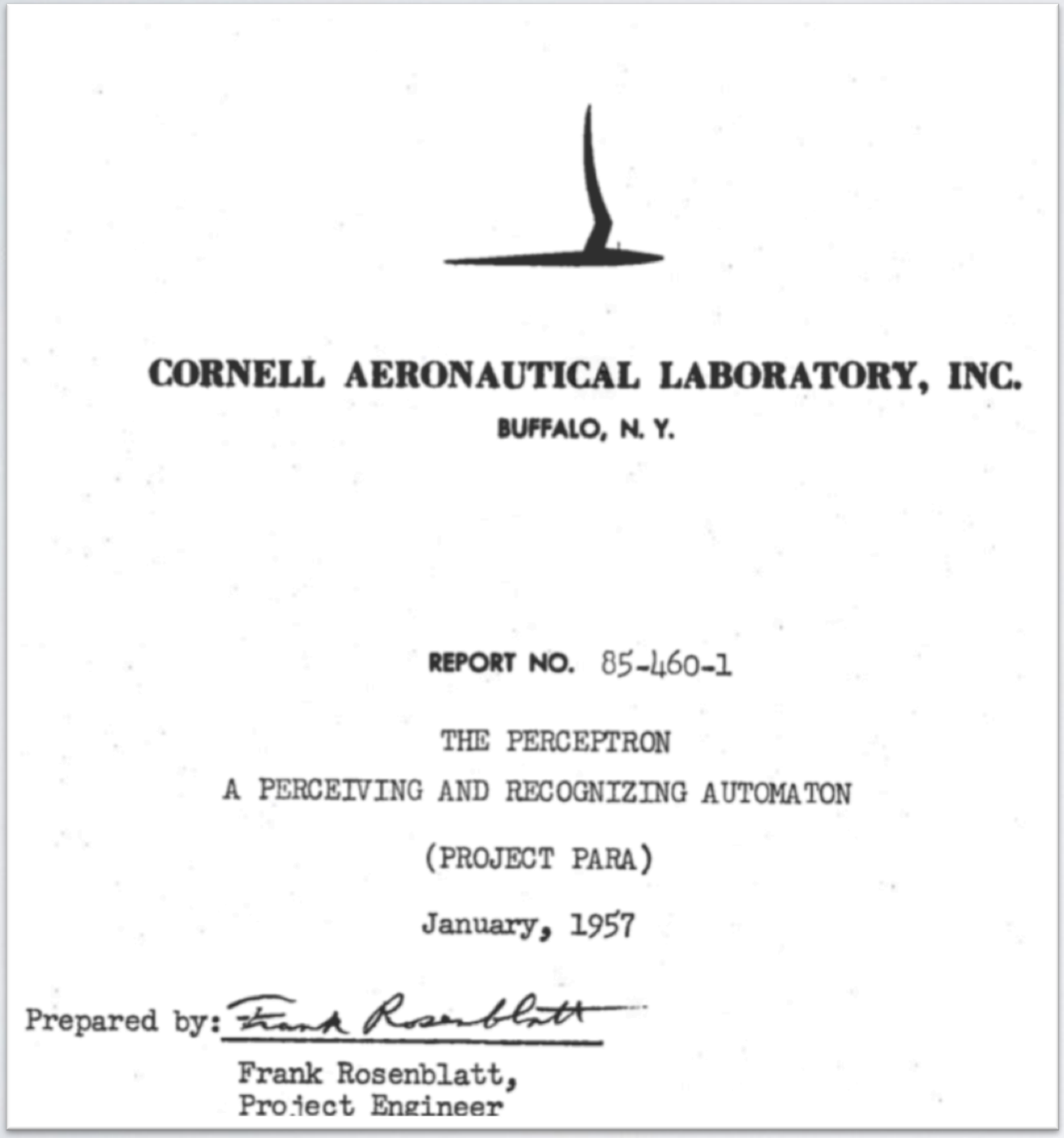
$$y = \varphi\left(\sum_{i=1}^{m} x_i \cdot w_i - b\right) = \varphi(\mathbf{x}^\intercal \cdot \mathbf{w} - b) = \begin{cases} 0, & \text{if } \mathbf{x}^\intercal \cdot \mathbf{w} - b \leq 0 \\ 1, & \text{if } \mathbf{x}^\intercal \cdot \mathbf{w} - b > 0 \end{cases}$$

Artificial Neuron

$$
\begin{aligned}
y &= \varphi\left(\sum_{i=1}^{m} x_i \cdot w_i - b\right) \\
&= \varphi(\mathbf{x}^\intercal \cdot \mathbf{w} - b) \\
&= \begin{cases} 0, & \text{if } \mathbf{x}^\intercal \cdot \mathbf{w} - b \leq 0 \\ 1, & \text{if } \mathbf{x}^\intercal \cdot \mathbf{w} - b > 0 \end{cases} \\
&= \begin{cases} 0, & \text{if } \mathbf{x}^\intercal \cdot \mathbf{w} \leq b \\ 1, & \text{if } \mathbf{x}^\intercal \cdot \mathbf{w} > b \end{cases}
\end{aligned}
$$

- The bias **b** allows us to control the threshold of **φ**
  - we can change the threshold by changing the weight/bias b
  - this will simplify how we describe the learning process

The Perceptron (Rosenblatt, 1957)



CORNELL AERONAUTICAL LABORATORY, INC.

BUFFALO, N. Y.

REPORT NO. 85-460-1

THE PERCEPTRON

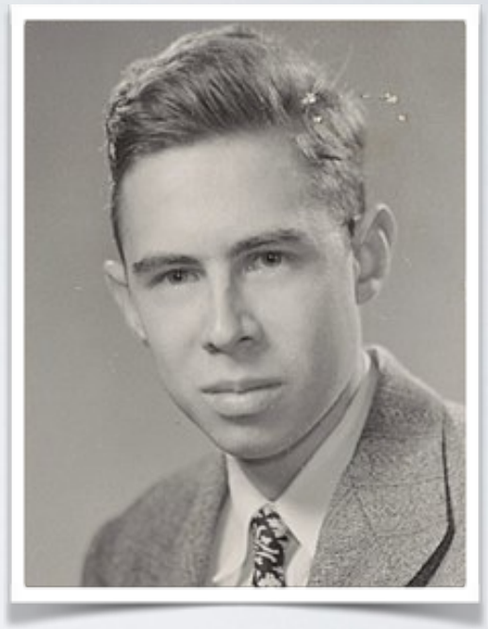
A PERCEIVING AND RECOGNIZING AUTOMATON

(PROJECT PARA)

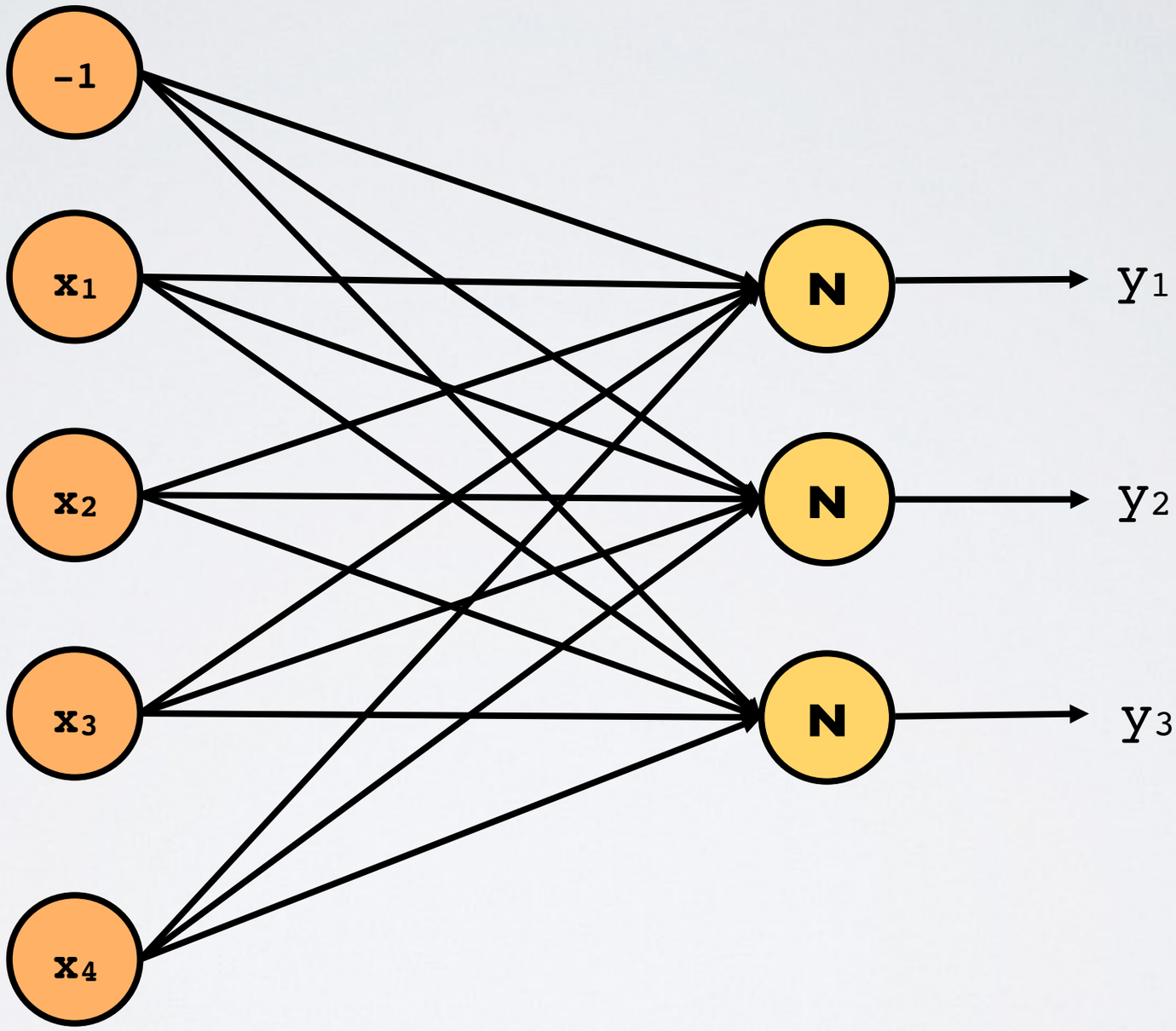
January, 1957

Prepared by: Frank Rosenblatt

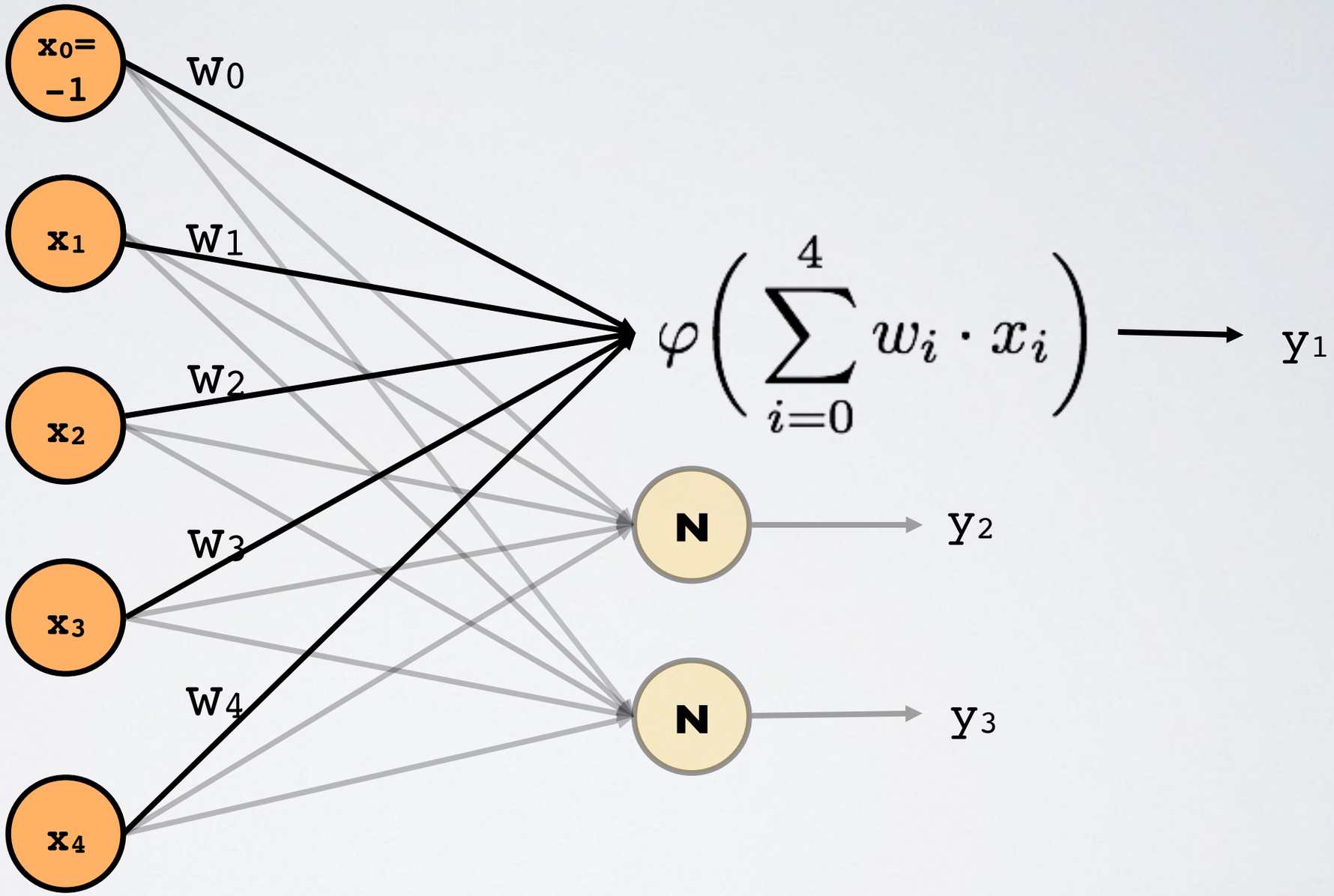
Frank Rosenblatt,
Project Engineer



Perceptron Network



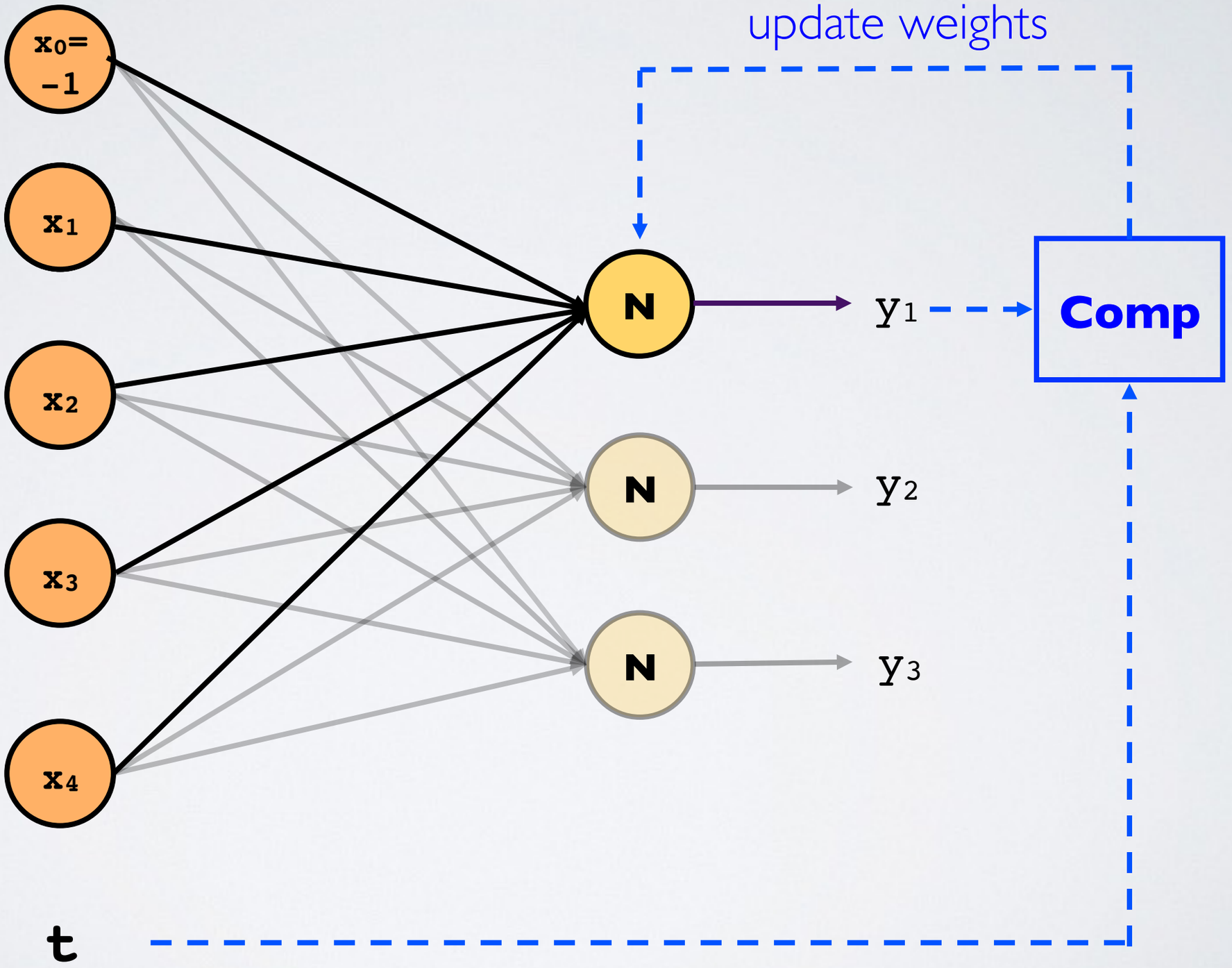
Perceptron Network



Training a Perceptron

- What does it mean for a perceptron to learn?
  - as we feed it more examples (i.e., input + classification pairs)
  - it should get better at classifying inputs
- Examples have the form $(x_1,\ldots,x_n,t)$
  - where t is the "target" classification (the right classification)
- How can we use examples to ***improve*** a (artificial) neuron?
  - which aspects of a neuron can we change/improve?
  - how can we get the neuron to output something closer to the target value?

Perceptron Network

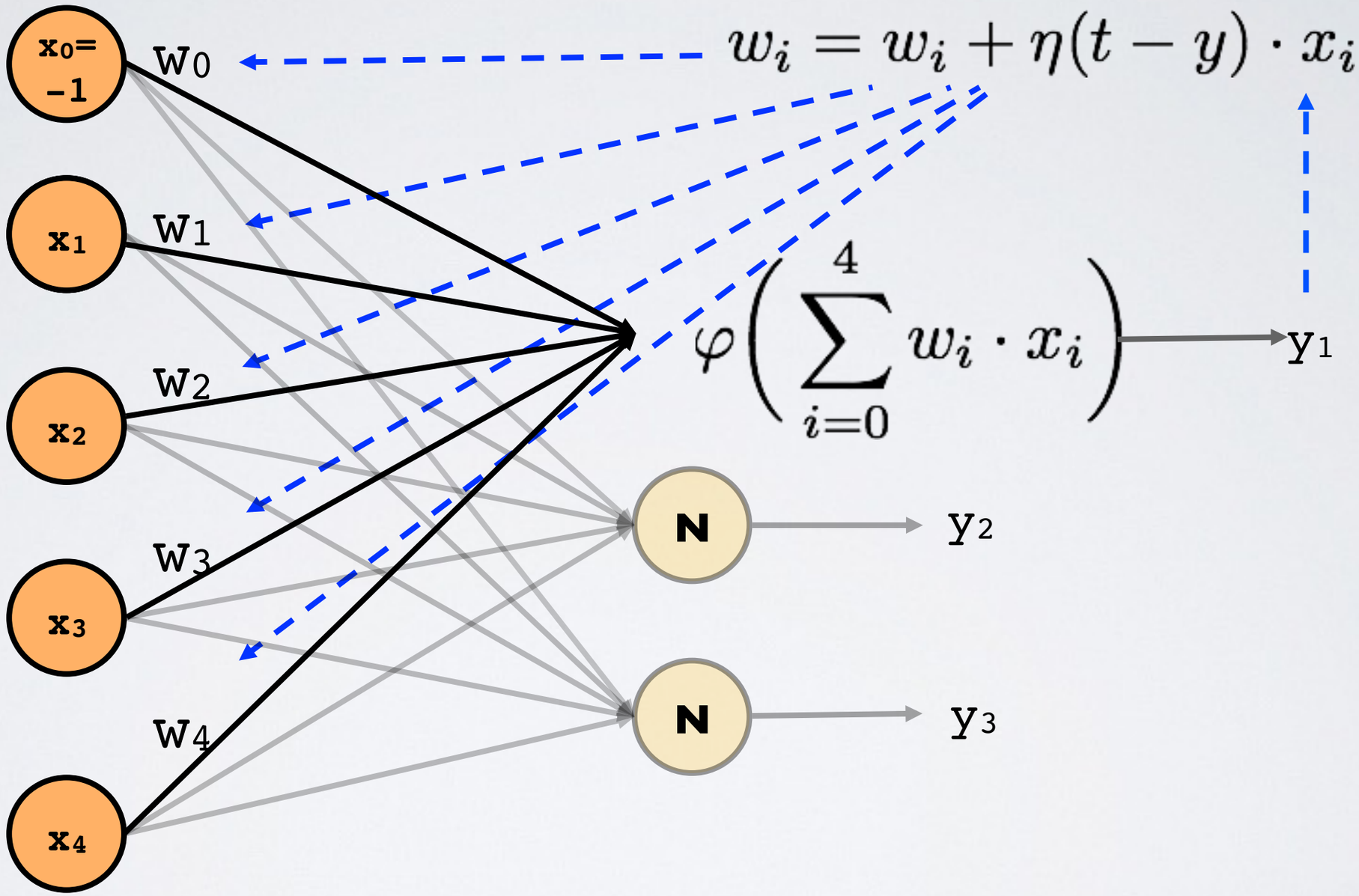


Perceptron Training

- Set all weights to small random values (positive and negative)
- For each training example $(x_1,\ldots,x_n,t)$
  - feed $(x_1,\ldots,x_n)$ to a neuron and get a result y
    - if $y=t$ then we don't need to do anything!
    - if $y<t$ then we need to ***increase*** the neuron's weights
    - if $y>t$ then we need to ***decrease*** the neuron's weights
- We do this with the following update rule

$$w_i = w_i + \eta(t - y) \cdot x_i$$

Perceptron Network



Artificial Neuron Update Rule

$$w_i = w_i + \eta(t - y) \cdot x_i$$

$$\Delta_i$$

- If $y=t$ then $\Delta_i=0$ and $w_i=w_i$
- if $y<t$ and $x_i>0$ then $\Delta_i>0$ and w_i increases by Δ_i
- if $y>t$ and $x_i>0$ then $\Delta_i<0$ and w_i decreases by Δ_i
- What happens when $x_i<0$?
  - last two cases are inverted! why?
  - recall that w_i gets multiplied by x_i so when $x_i<0$, so if we want y to increase then w_i needs to be decreased!

Artificial Neuron Update Rule

$$w_i = w_i + \eta(t - y) \cdot x_i$$

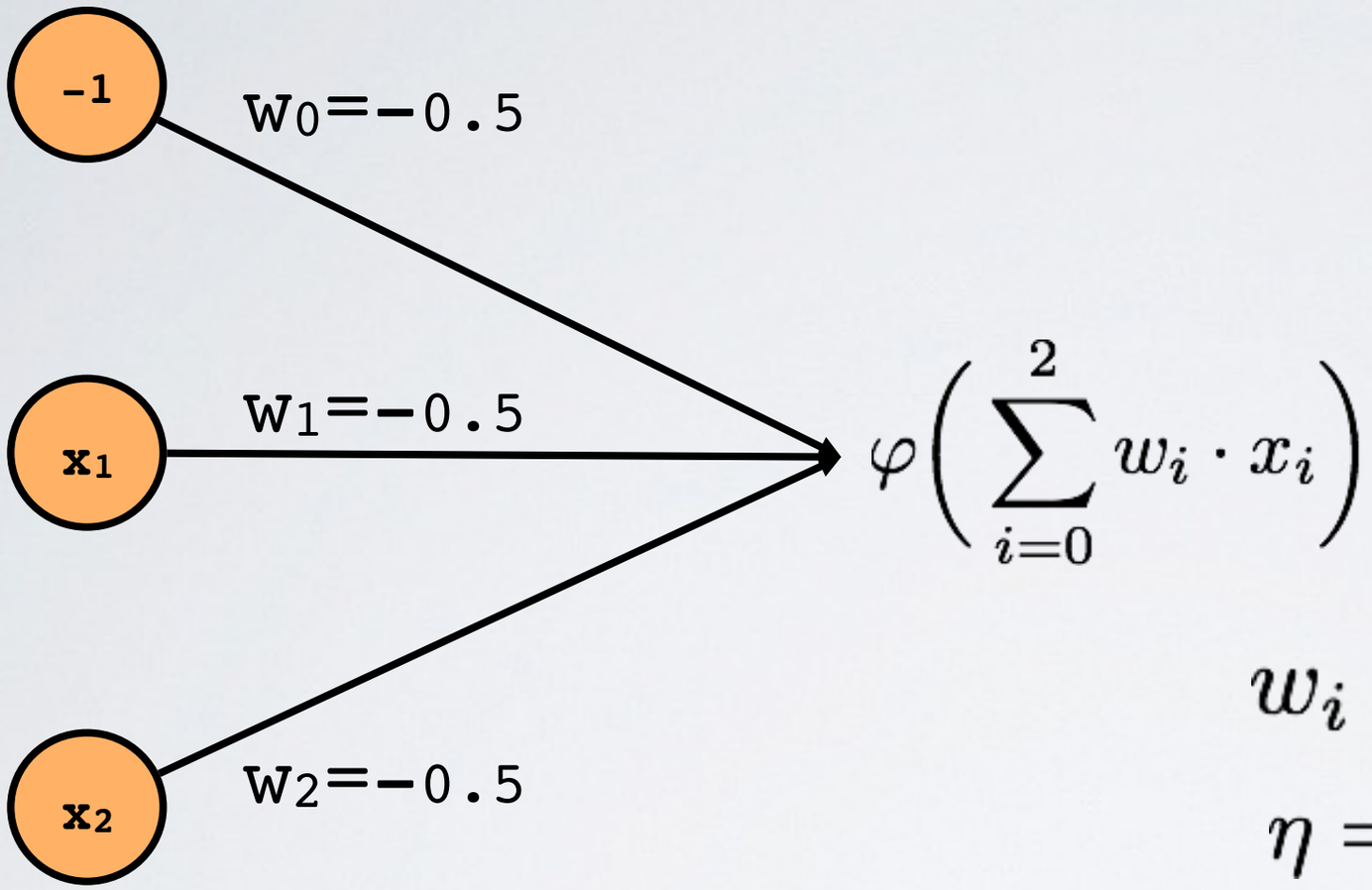
$$\Delta_i$$

- What is η for?
  - to control by how much w_i should increase or decrease
  - if η is large then errors will cause weights to be changed a lot
  - if η is small then errors will cause weights to be change a little
- large η increases speed at which a neuron learns but increases sensitivity to errors in data

Perceptron Training Pseudocode

```
Perceptron(data, neurons, k):
   for round from 1 to k:
      for each training example in data:
         for each neuron in neurons:
            y = output of feeding example to neuron
            for each weight of neuron:
               update weight
```

Perceptron Training



x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	1

$$w_i = w_i + \eta(t - y_i) \cdot x_i$$

$$\eta = 0.5$$

Activity #1

3 min

Perceptron Training

bias

target

- Example (-1,0,0,0)
  - $y=\varphi(-1\times-0.5+0\times-0.5+0\times-0.5)=\varphi(0.5)=\mathbf{1}$
  - $w_0=-0.5+0.5(0-1)\times-1=\mathbf{0}$
  - $w_1=-0.5+0.5(0-1)\times0=\mathbf{-0.5}$
  - $w_2=-0.5+0.5(0-1)\times0=\mathbf{-0.5}$
- Example (-1,0,1,1)
  - $y=\varphi(-1\times0+0\times-0.5+1\times-0.5)=\varphi(-0.5)=\mathbf{0}$
  - $w_0=0+0.5(1-0)\times-1=\mathbf{-0.5}$
  - $w_1=-0.5+0.5(1-0)\times0=\mathbf{-0.5}$
  - $w_2=-0.5+0.5(1-0)\times1=\mathbf{0}$

Perceptron Training

bias

target

- Example (-1,1,0,1)
  - $y=\varphi(-1\times-0.5+1\times-0.5+0\times0)=\varphi(0)=\mathbf{0}$
  - $w_0=-0.5+0.5(1-0)\times-1=\mathbf{-1}$
  - $w_1=-0.5+0.5(1-0)\times1=\mathbf{0}$
  - $w_2=0+0.5(1-0)\times0=\mathbf{0}$
- Example (-1,1,1,1)
  - $y=\varphi(-1\times-1+1\times0+1\times0)=\varphi(1)=\mathbf{1}$
  - $w_0=\mathbf{-1}$
  - $w_1=\mathbf{0}$
  - $w_2=\mathbf{0}$

Perceptron Training

- Are we done?
- No!
  - perceptron was wrong on examples: **(0,0,0), (0,1,1),&(1,0,1)**
    - so we keep going until weights stop changing, or change only by very small amounts (*convergence*)
- For sanity, check if our final weights correctly classify **(0,0,0)**
  - w_0**=-1,** w_1**=0,** w_2**=0**
  - **y=φ(**-1×-1+0×0+0×0**)=φ(**1**)=1**

Perceptron Training Pseudocode

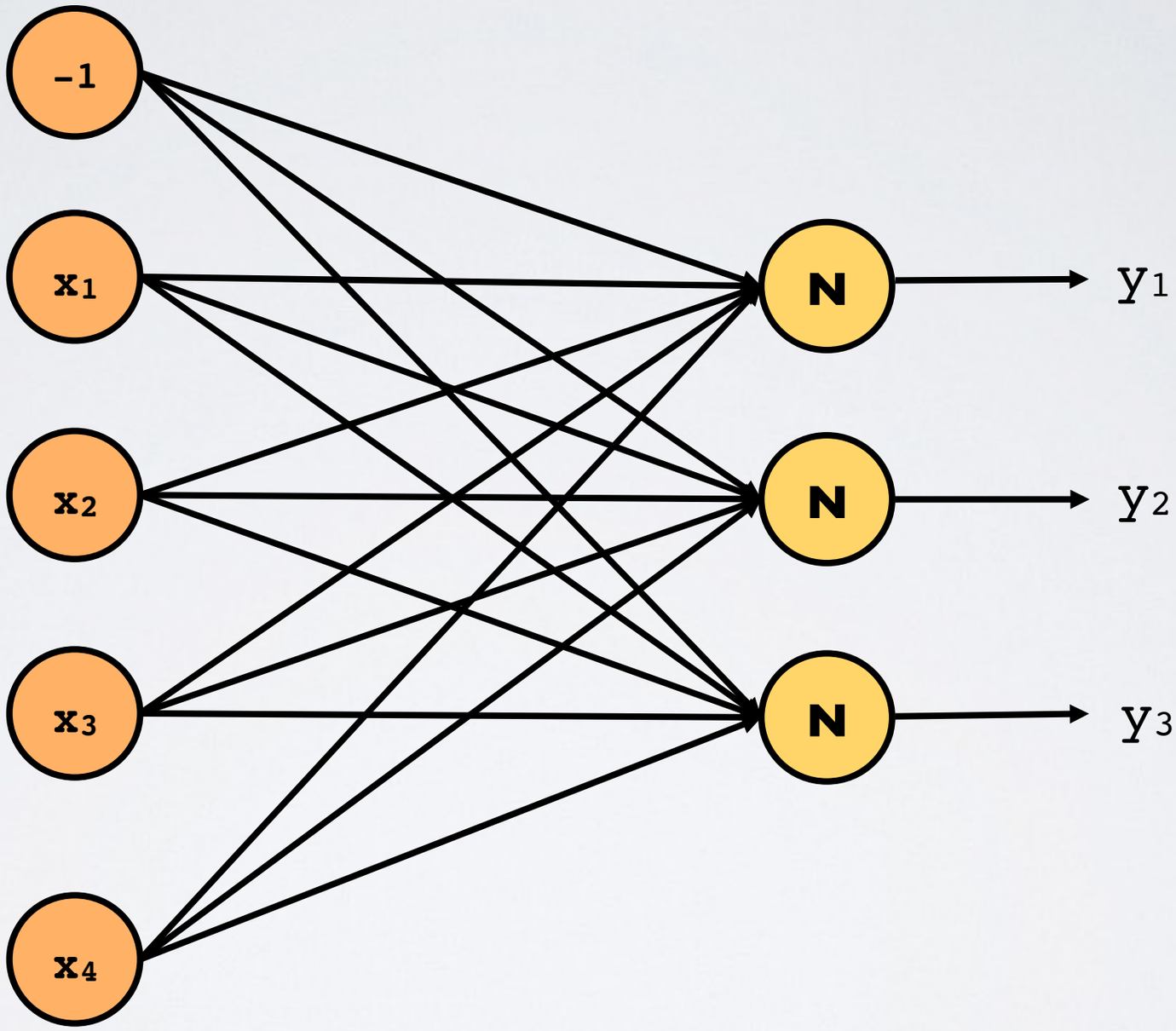
- **`data[n][m]`**: **`n`** examples, each of size **`m+1`** (**`m`** attributes & **`1`** classification)
- **`neurons[q][m]`**: **`q`** neurons, each with **`m`** weights

```
Perceptron(data, neurons, k):
   for round from 1 to r:
         for i from 0 to n-1: // loop over training examples
         for j from 0 to q-1: // loop over neurons
         inner_prod = 0
         for k from 0 to m-1: // loop over inputs & neuron weights
           inner_prod += data[i][k] x neuron[j][k]
           y = activation(inner_prod)
            for k from 0 to m-1: //loop over weights to update them
           neuron[j][k] += eta x (data[i][m] - y) x data[i][k]
```

- runtime of **`O(rnqm)`**

Perceptron Animation

Single-Layer Perceptron

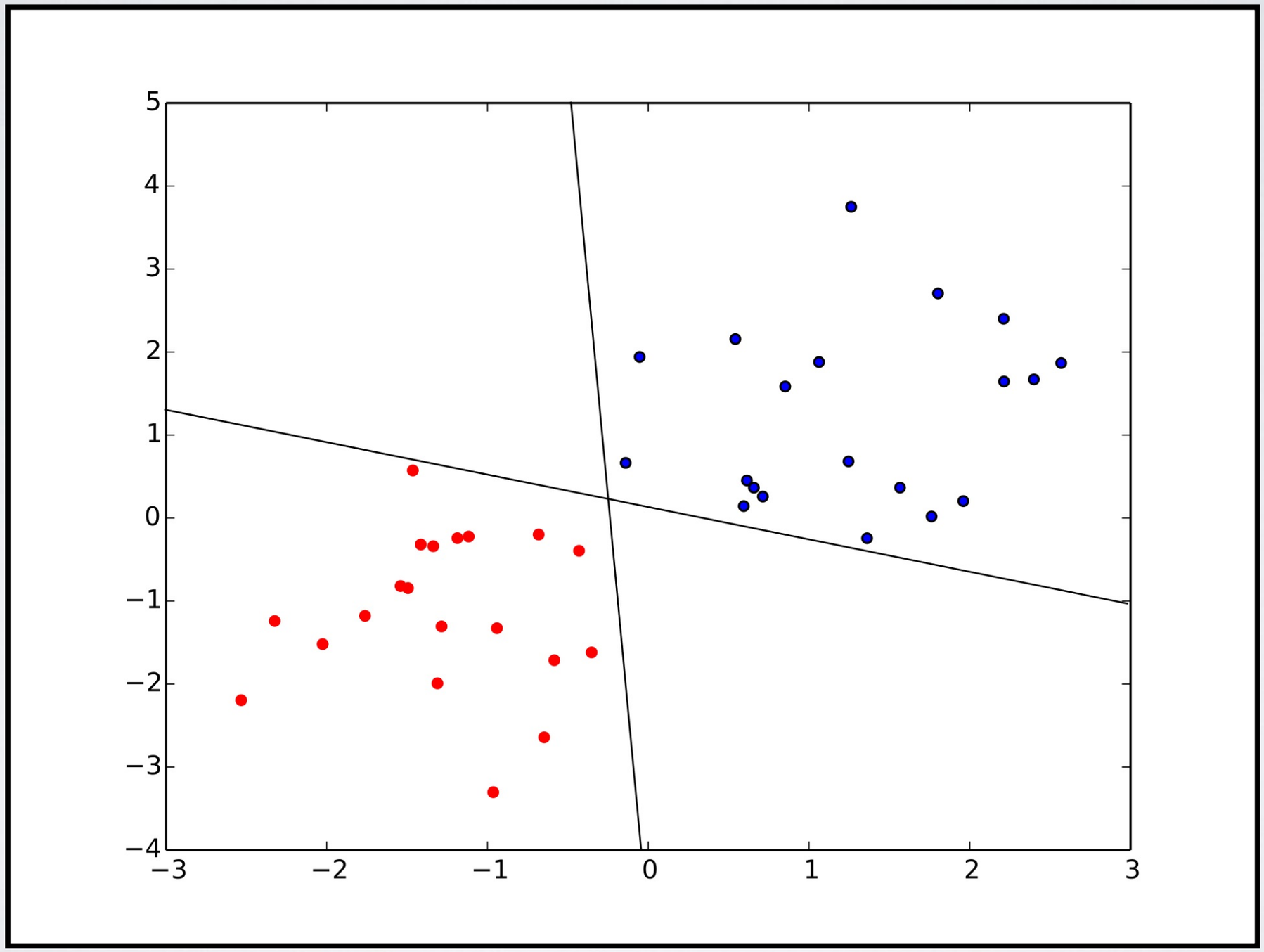


Limits of Single-Layer Perceptrons

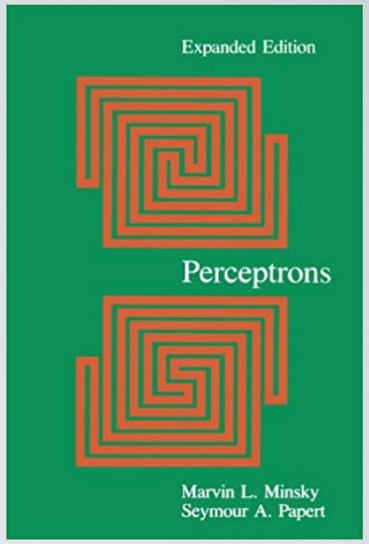
- Perceptrons are limited
  - there are many functions they cannot learn
- To better understand their power and limitations, it's helpful to take a geometric view

- If we plot classifications of all possible inputs in the plane (or hyperplane if high-dimensional)
  - perceptrons can learn the function if classifications can be separated by a line (or hyperplane)
  - data is **linearly separable**

Linearly-Separable Classifications

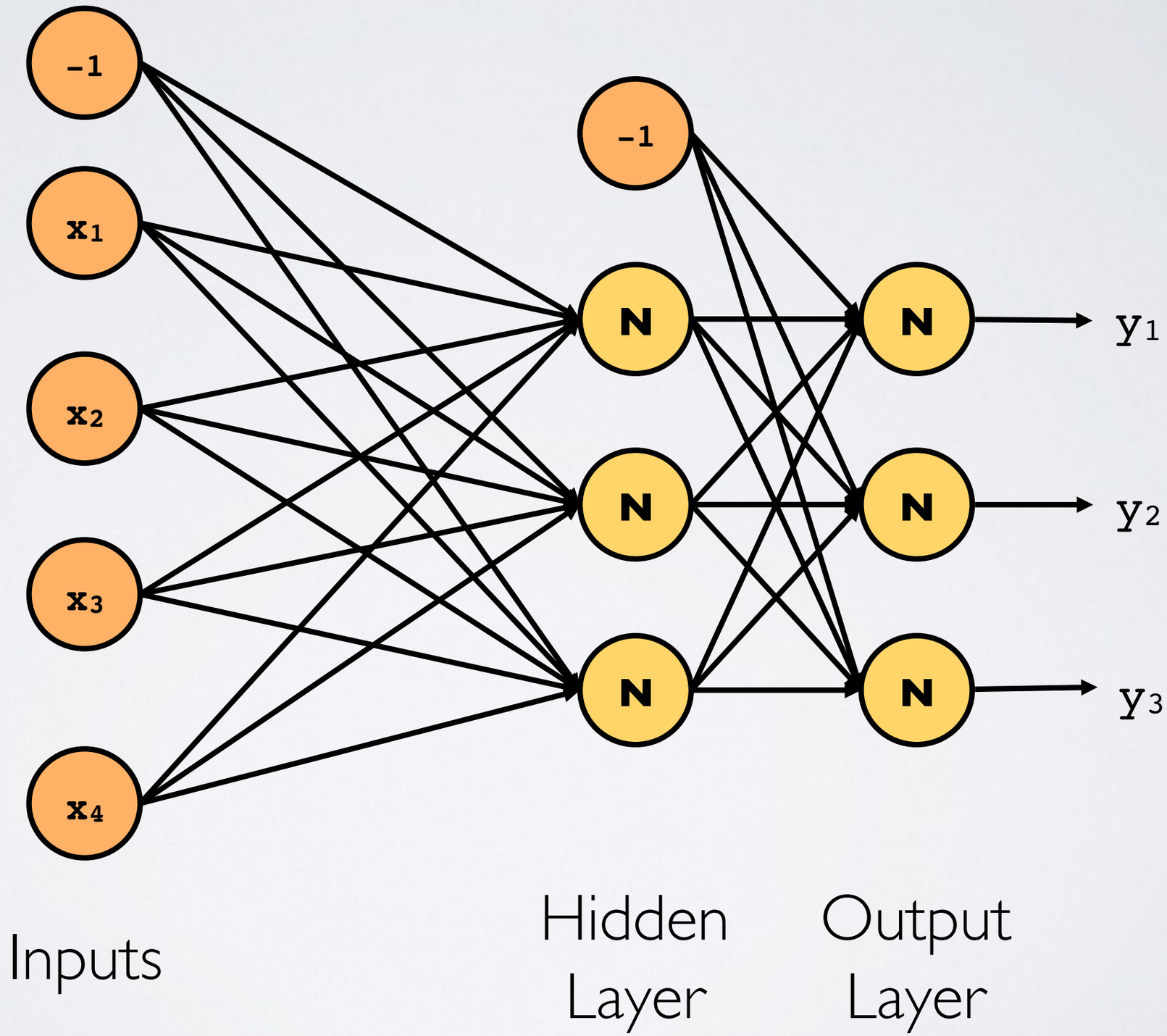


Single-Layer Perceptrons



- In 1969, Minksy and Papert published
  - *Perceptrons: An Introduction to Computational Geometry*
- In it they proved that single-layer perceptrons
  - could not learn some simple functions
- This really hurt research in neural networks…
  - …many became pessimistic about their potential

Multi-Layer Perceptron



Training Multi-Layer Perceptrons

- Harder to train than a single-layer perceptron
  - if output is wrong, do we update weights of hidden neuron or of output neuron? or both?
  - update rule for neuron requires knowledge of target but there is no target for hidden neurons
- MLPs are trained with stochastic gradient descent (SGD) using ***backpropagation***
  - invented in 1986 by Rumelhart, Hinton and Williams
  - technique was known before but Rumelhart et al. showed precisely how it could be used to train MLPs

Training Multi-Layer Perceptrons

CHAPTER 8

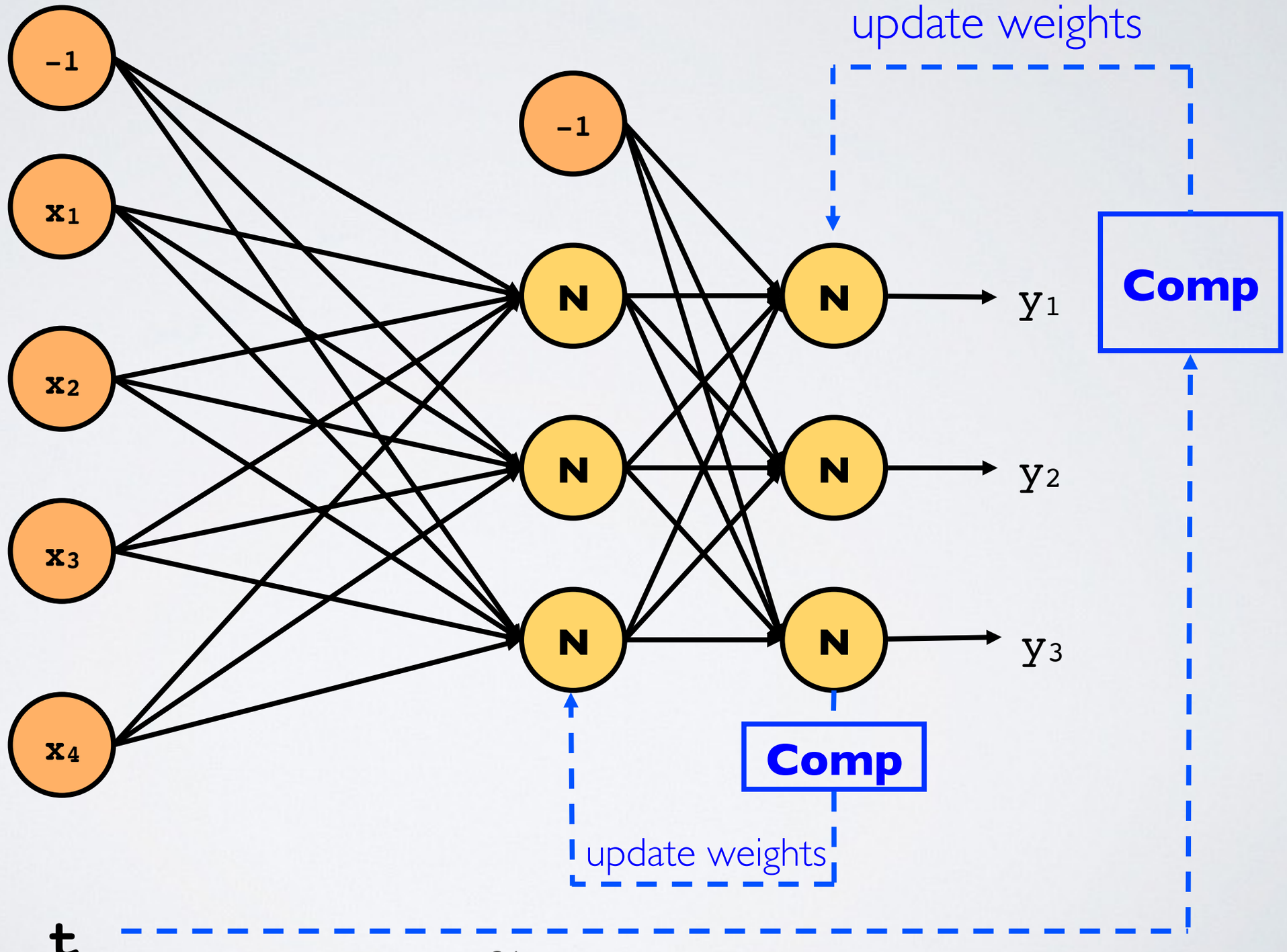
Learning Internal Representations by Error Propagation

D. E. RUMELHART, G. E. HINTON, and R. J. WILLIAMS

THE PROBLEM

We now have a rather good understanding of simple two-layer associative networks in which a set of input patterns arriving at an input layer are mapped directly to a set of output patterns at an output layer. Such networks have no *hidden* units. They involve only *input* and *output* units. In these cases there is no *internal representation.* The coding provided by the external world must suffice. These networks have proved useful in a wide variety of applications (cf. Chapters 2, 17, and 18). Perhaps the essential character of such networks is that they map similar input patterns to similar output patterns. This is what allows these networks to make reasonable generalizations and perform reasonably on patterns that have never before been presented. The similarity of patterns in a PDP system is determined by their overlap. The overlap in such networks is determined outside the learning system itself—by whatever produces the patterns

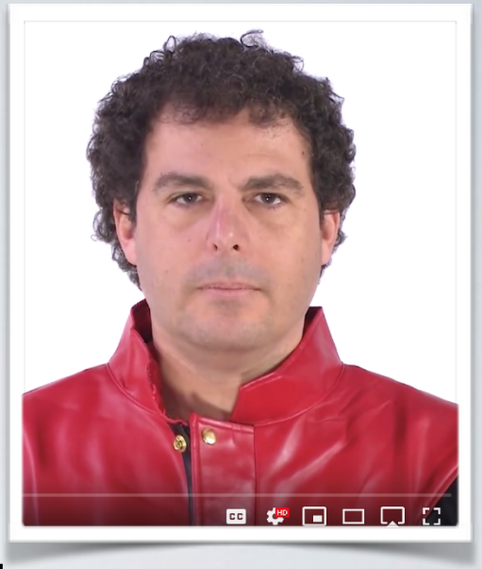
Training by Backpropagation



Training Multi-Layer Perceptrons

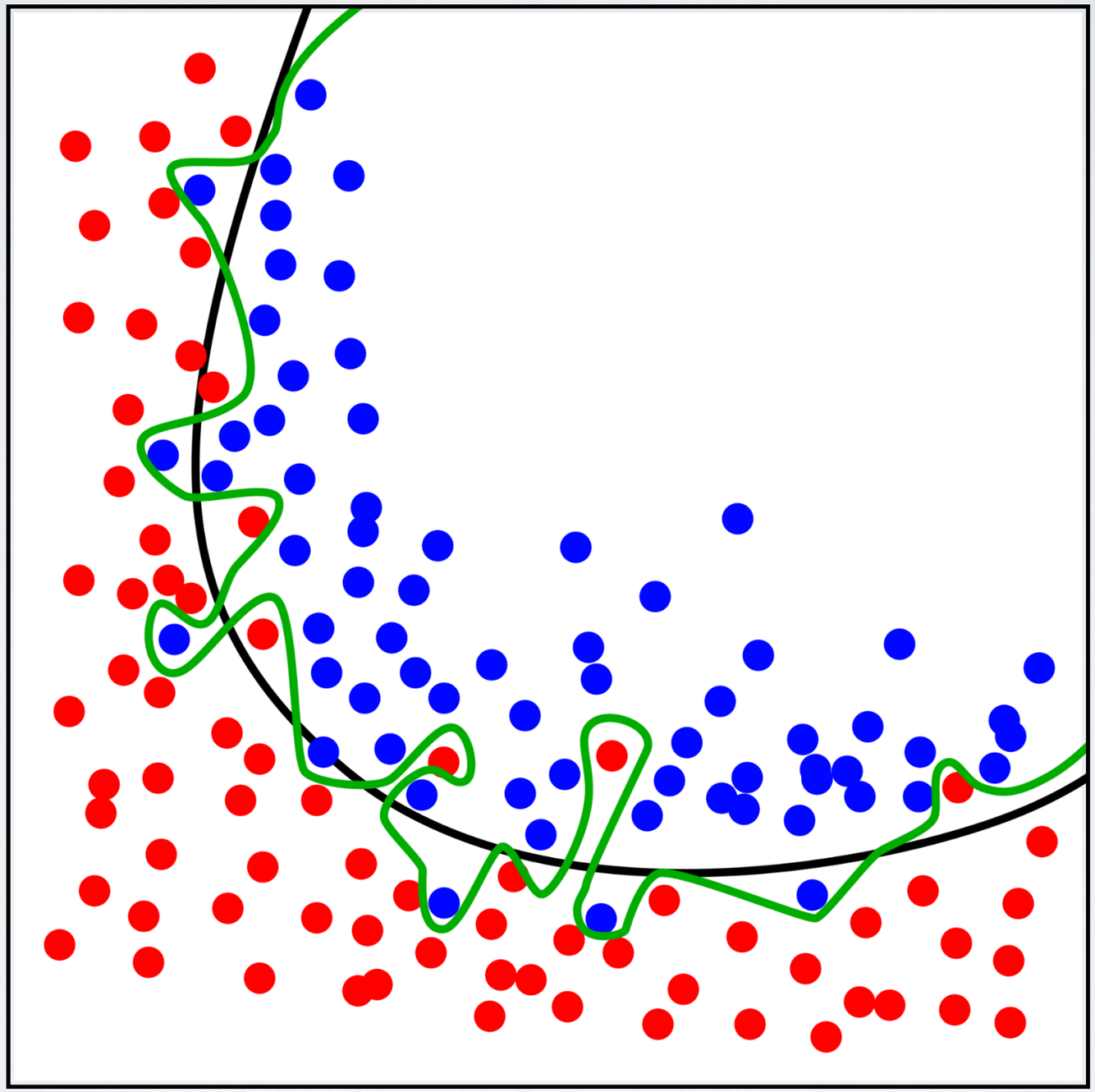
- Specifics of the algorithm are beyond CS16
  - covered in CS142 and CS147
- We often use **2** hidden layers & **1** output layer
  - more layers don't seem to add much more power
    - tradeoff between complexity and number of parameters needed to tune
- Other kinds of neural nets
  - convolutional neural nets (image & video recognition)
  - recurrent neural nets (speech recognition)
  - many many more

Overfitting



- A challenge in ML is deciding ***how much*** to train a model
  - if a model is overtrained then it can ***overfit*** the training data
  - which can lead it to make mistakes on new/unseen inputs
- Why does this happen?
  - training data can contain errors and noise
  - if model overfits training data then it "learns" those errors and noise
  - and won't do as well on new unseen inputs
  - for more on overfitting see
    - https://www.youtube.com/watch?v=DQWI1kvmwRg

Overfitting & Generalization

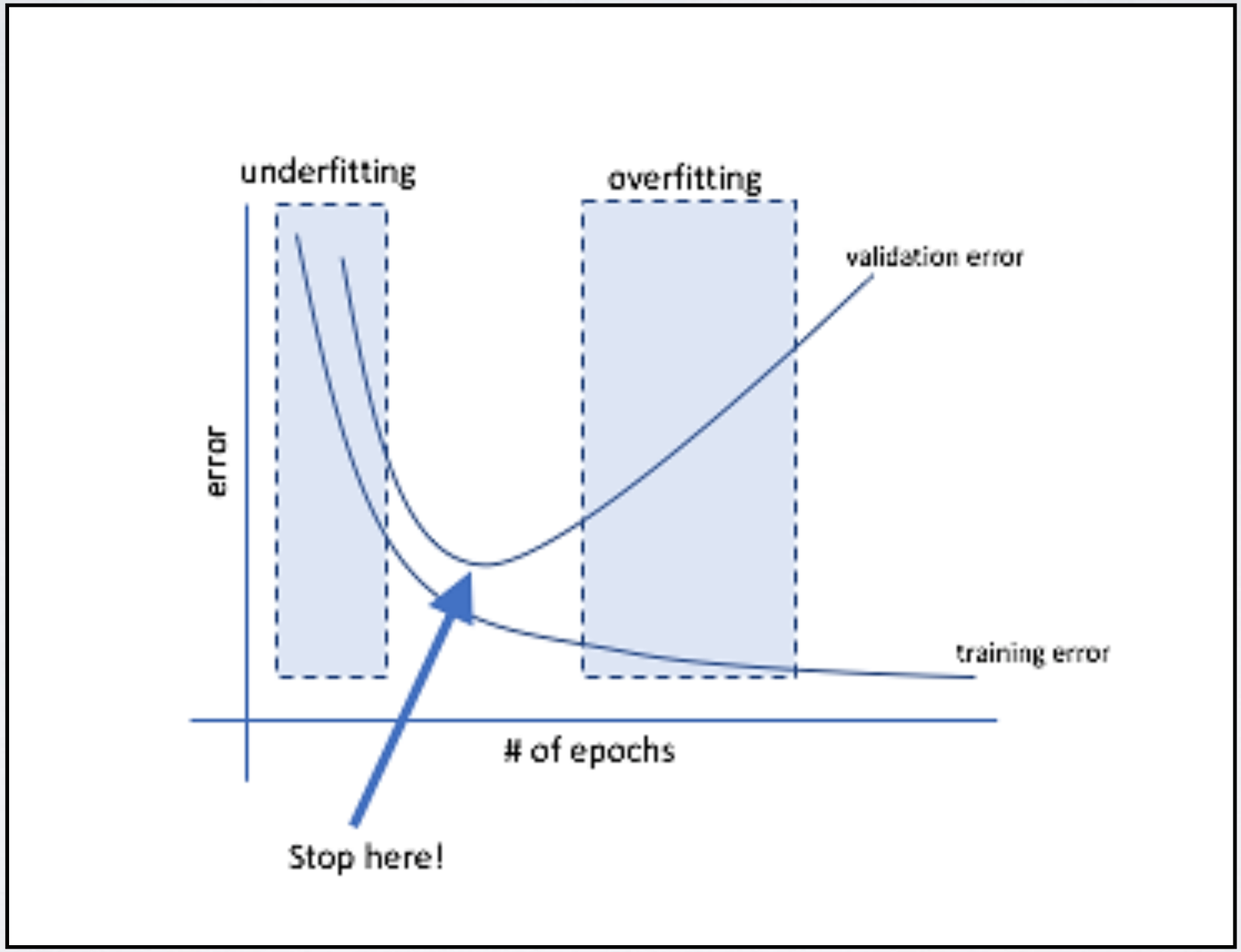


Overfitting & Generalization

- So how do we know when to stop training?
  - one approach is to use the ***early stopping*** technique
- Split the training examples into 3 sets
  - a training set (50%), a validation set (25%), a testing set (25%)

- Train on the training set but
  - every 5 rounds, run NN on validation set
  - compute the NN' s error over entire validation set
  - compare current error to previous error
  - if error is increasing, stop and use previous version of NN

Early Stopping



Applications

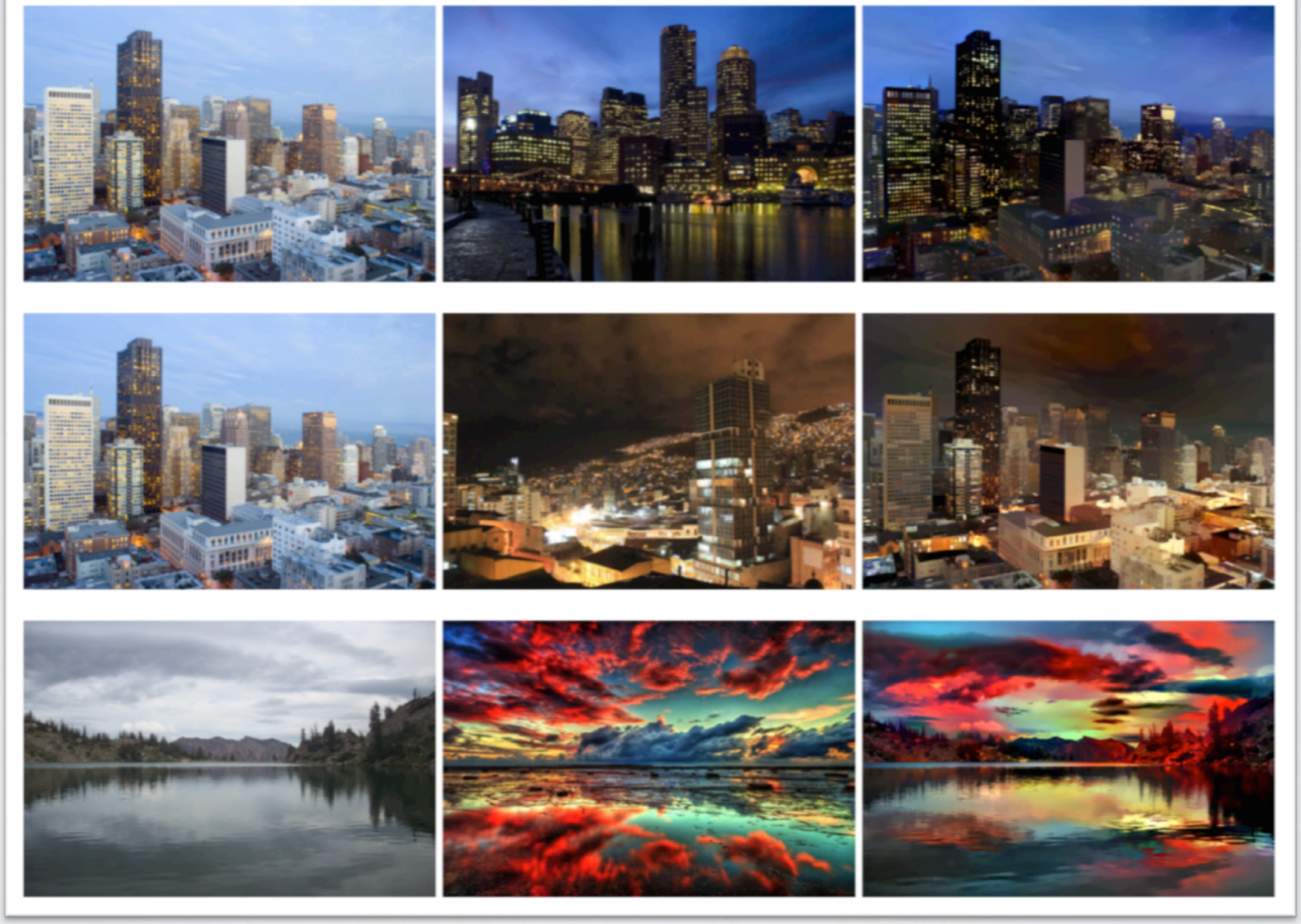
- Musical composition
- Francesco Marchesani – imitating the music of Chopin using a recurrent neural network (RNN)

Applications (continued)

- Style Transfer

Applications (continued)

- Style Transfer



Applications

- Advertising
- Credit card fraud detection
- Skin-cancer diagnosis
- Predicting earthquakes
- Lip-reading from video
- Even…neural networks to help you write neural networks! (Neural Complete)

Questions?