Feedforward Neural Networks (FNNs)

Math of Machine Learning

Department of Mathematics
University *of* Nebraska-Lincoln

Agenda

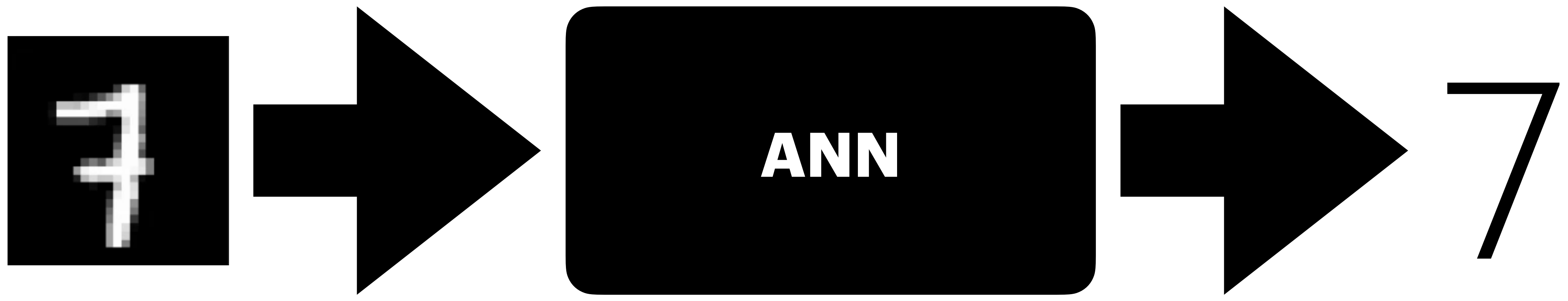
Learning Objectives

1. ***Feedforward Neural Networks (FNNs)***: a classical machine learning model.
   1.1. Introduce neural networks via an example.
   1.2. (Re)call linear algebra and calculus.
   1.3. Training ANNs with gradient descent.

Artificial Neural Networks (ANNs)

A Classification Problem

- Suppose we want to classify handwritten digits.
- Our goal is to construct a function that achieves the following:

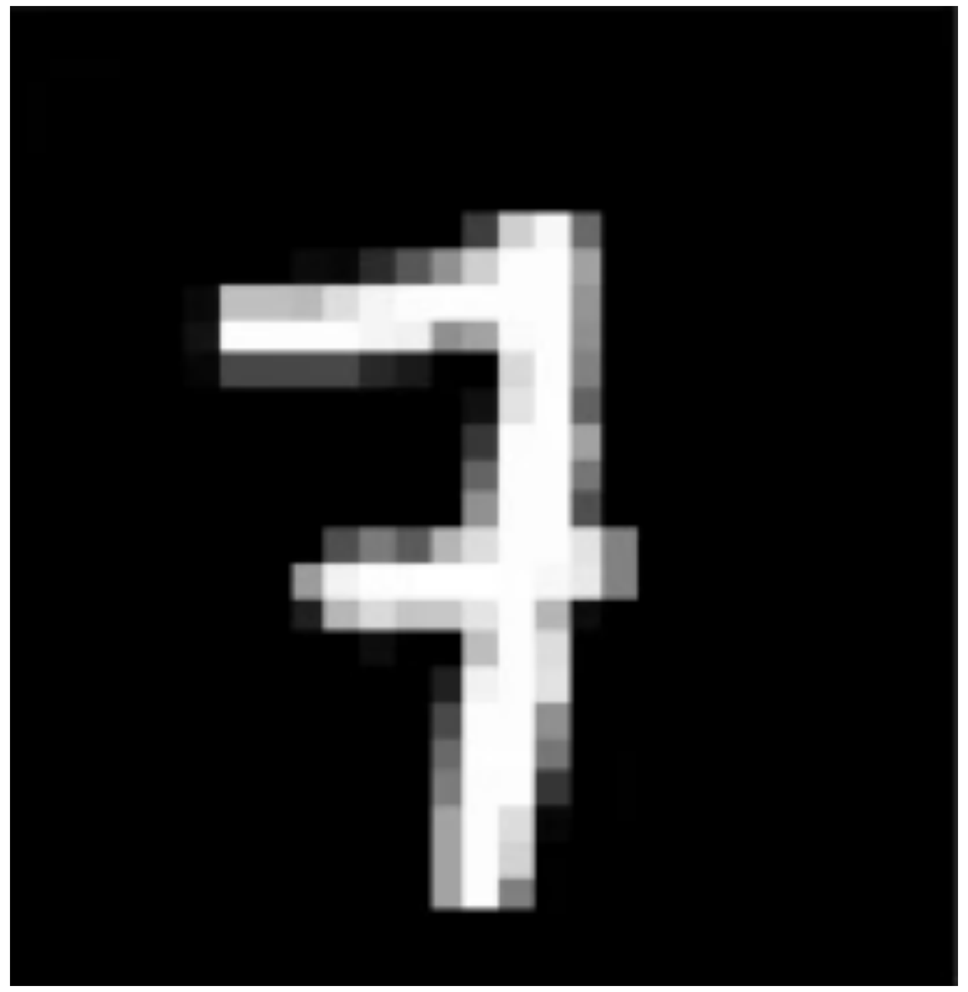


- How do computers store images?

Artificial Neural Networks

A Classification Problem

Matrices!



$28 \times 28 = 784$ entries

$$
\begin{array}{cc}
28 & \\
\left[\begin{matrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 26 & 111 & 195 & 230 & 30 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 28 & 107 & 195 & 254 & 254 & 254 & 244 & 20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 46 & 167 & 248 & 254 & 222 & 146 & 150 & 254 & 174 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 65 & 223 & 246 & 254 & 153 & 61 & 10 & 0 & 48 & 254 & 129 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 85 & 175 & 164 & 80 & 2 & 0 & 0 & 0 & 48 & 254 & 120 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 182 & 254 & 16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 207 & 254 & 16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 207 & 202 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 28 & 248 & 170 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 107 & 254 & 61 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 166 & 252 & 30 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 191 & 206 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 191 & 206 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 14 & 246 & 186 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 91 & 254 & 77 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 175 & 254 & 48 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 175 & 240 & 27 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 215 & 222 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 115 & 255 & 152 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 134 & 255 & 68 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{matrix}\right] & 28
\end{array}
$$

Artificial Neural Networks

A Classification Problem

- Let's reorganize this information.
  - Label each of the 28 columns of the image $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{28}$.
  - Stack the columns to get a ***vector*** (or point).

$$\begin{bmatrix} | & | & & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_{28} \\ | & | & & | \end{bmatrix} \longrightarrow \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \vdots \\ \mathbf{c}_{28} \end{bmatrix} \in \mathbb{R}^{784}$$

E.g.

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \to \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}$$

- So, we can reframe our problem as building a function that maps vectors in $\mathbb{R}^{784}$ to the set $\{0,1,2,\ldots,9\}$.

Artificial Neural Networks

What kinds of functions can we apply to vectors?

- Matrix Multiplication:

$$\begin{bmatrix} 1 & 2 & -1 \\ 0 & 3 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 7 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \cdot 5 & + 2 \cdot 7 & + -1 \cdot 4 \\ 0 \cdot 5 & + 3 \cdot 7 & + \quad 1 \cdot 4 \end{bmatrix} = \begin{bmatrix} 15 \\ 25 \end{bmatrix}$$

  - This 2×3 matrix sends vectors in $\mathbb{R}^3$ to vectors in $\mathbb{R}^2$.

- We can also make this map more interesting. Consider

$$W = \begin{bmatrix} 1 & 2 & -1 \\ 0 & 3 & 1 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

- Define the function $T(\mathbf{x}) = W\mathbf{x} + \mathbf{b}$ for any vector $\mathbf{x}$ in $\mathbb{R}^3$. For example,

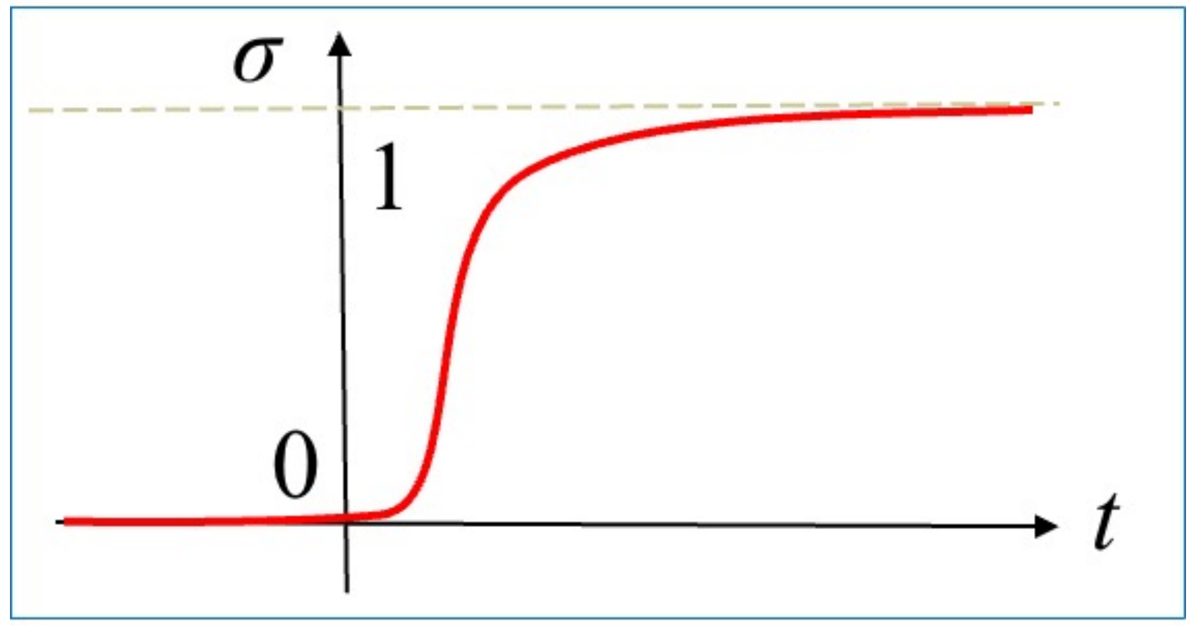
$$T\left(\begin{bmatrix} 5 \\ 7 \\ 4 \end{bmatrix}\right) = \begin{bmatrix} 17 \\ 24 \end{bmatrix}$$

Artificial Neural Networks

What kinds of functions can we apply to vectors?

- Consider the ***sigmoid function***:

$$\sigma(t) = \frac{e^t}{1 + e^t}$$



- We apply the sigmoid function to vectors component-wise like so

$$\sigma\left(\begin{bmatrix} -1 \\ 0.3 \end{bmatrix}\right) \approx \begin{bmatrix} 0.27 \\ 0.57 \end{bmatrix}$$

- We can also have more interesting functions. For example,

$$\text{softmax}\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}\right) = \begin{bmatrix} \frac{e^{x_1}}{e^{x_1} + e^{x_2} + \cdots + e^{x_n}} \\ \frac{e^{x_2}}{e^{x_1} + e^{x_2} + \cdots + e^{x_n}} \\ \vdots \\ \frac{e^{x_n}}{e^{x_1} + e^{x_2} + \cdots + e^{x_n}} \end{bmatrix}$$

E.g.

$$\text{softmax}\left(\begin{bmatrix} 1 \\ 2 \\ -1 \\ 0 \end{bmatrix}\right) \approx \begin{bmatrix} 0.237 \\ 0.644 \\ 0.032 \\ 0.087 \end{bmatrix}$$

- The ***softmax*** function outputs a ***probability vector***, which means the sum of the components equal 1.

Artificial Neural Networks

Constructing a ANN

- Recall our goal:

E.g.

$$ANN(\text{image}) = \begin{bmatrix} 0.02 \\ 0.73 \\ 0.005 \\ \vdots \\ 0.149 \end{bmatrix} \xrightarrow{\text{predict}} 1$$

$$\longrightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

- Let's edit this g
  - Notice that s ... s 1 and 7 are actually furth

  - We need to treat the labels $\{0,1,2,\ldots,9\}$ as categorical data.

  - Relabel each digit as a probability vector.

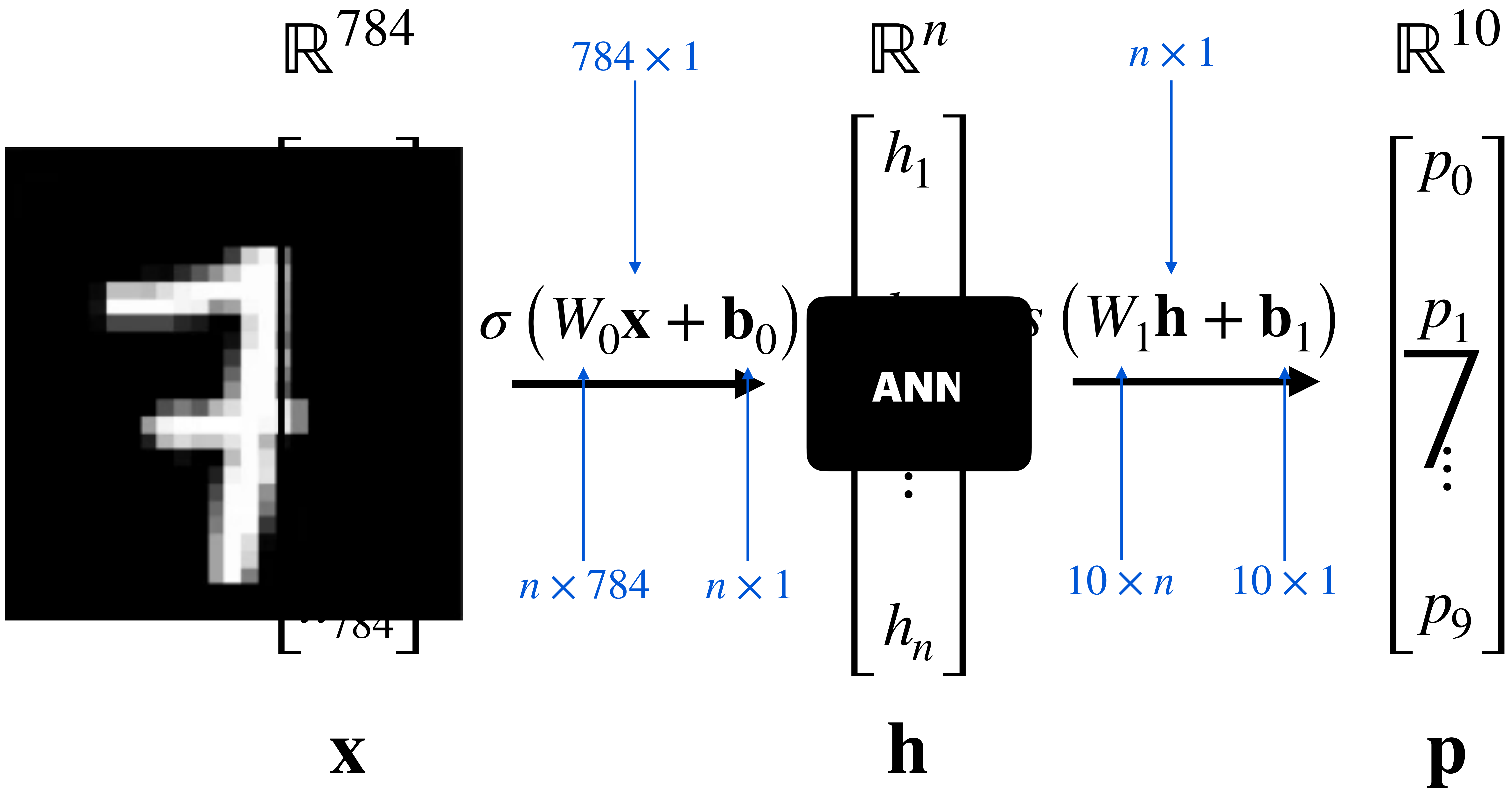
$$i \longrightarrow e_i = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

Feedforward Neural Network

Putting it all together

$r =$ sigmoid

$s =$ softmax

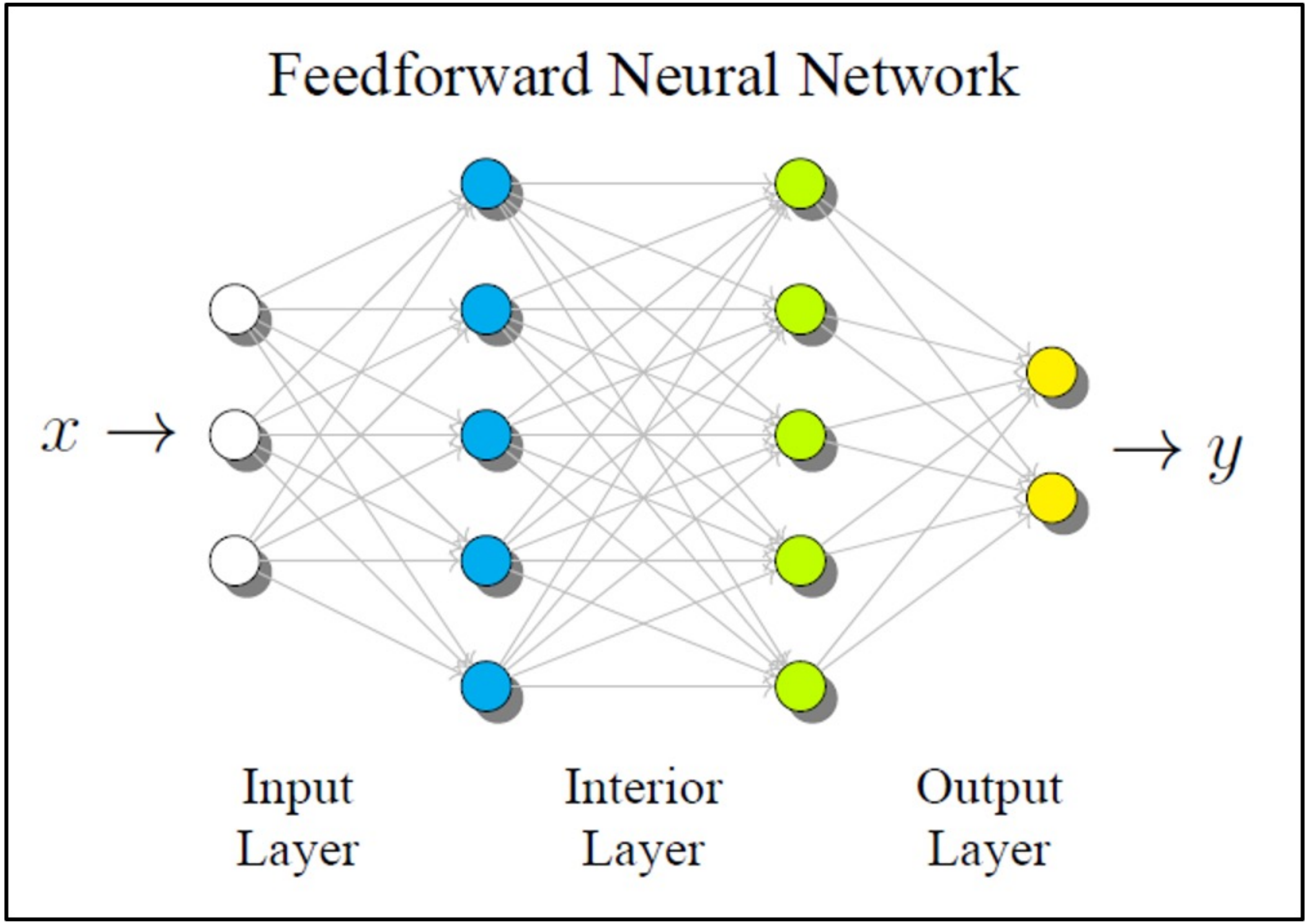


We call the numbers in $W_0, \mathbf{b}_0, W_1, \mathbf{b}_1$ the model ***weights*** or ***parameters***.

This ANN has $n \cdot 784 + n \cdot 1 + 10 \cdot n + 10 \cdot 1 = \boxed{795n + 10 \text{ parameters.}}$

Feedforward Neural Network

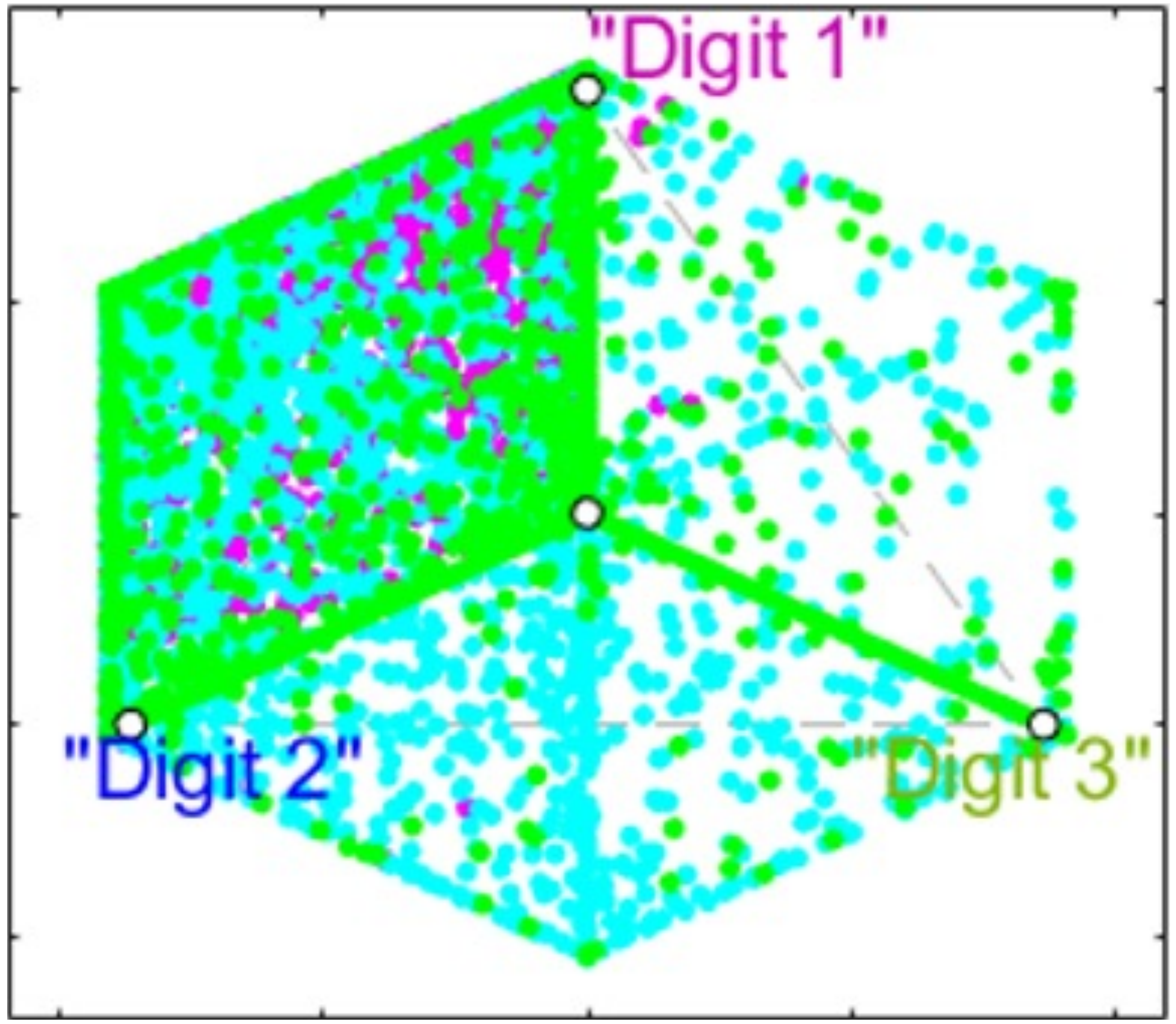
Biological Inspiration



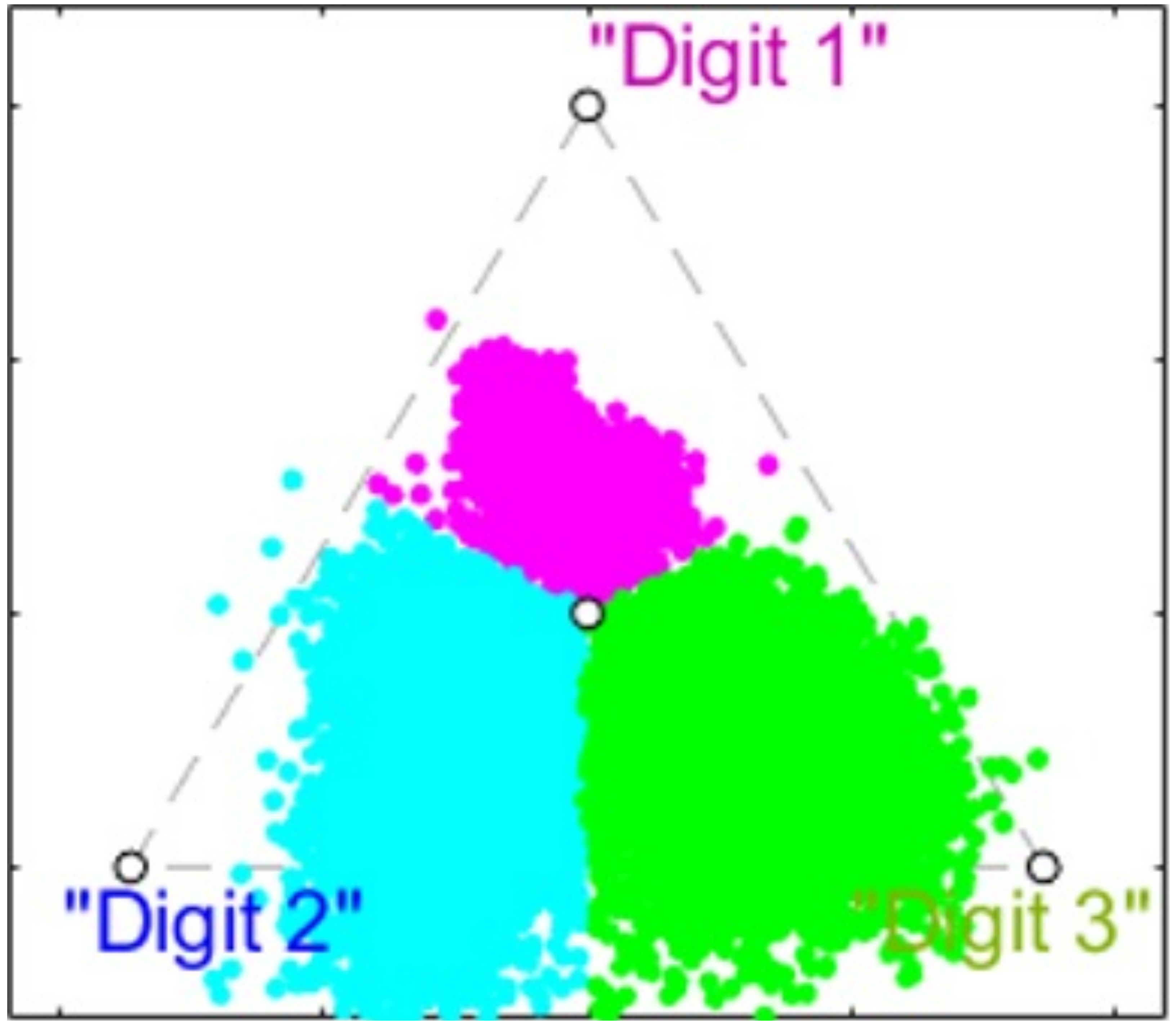
Artificial Neural Networks

Visualization

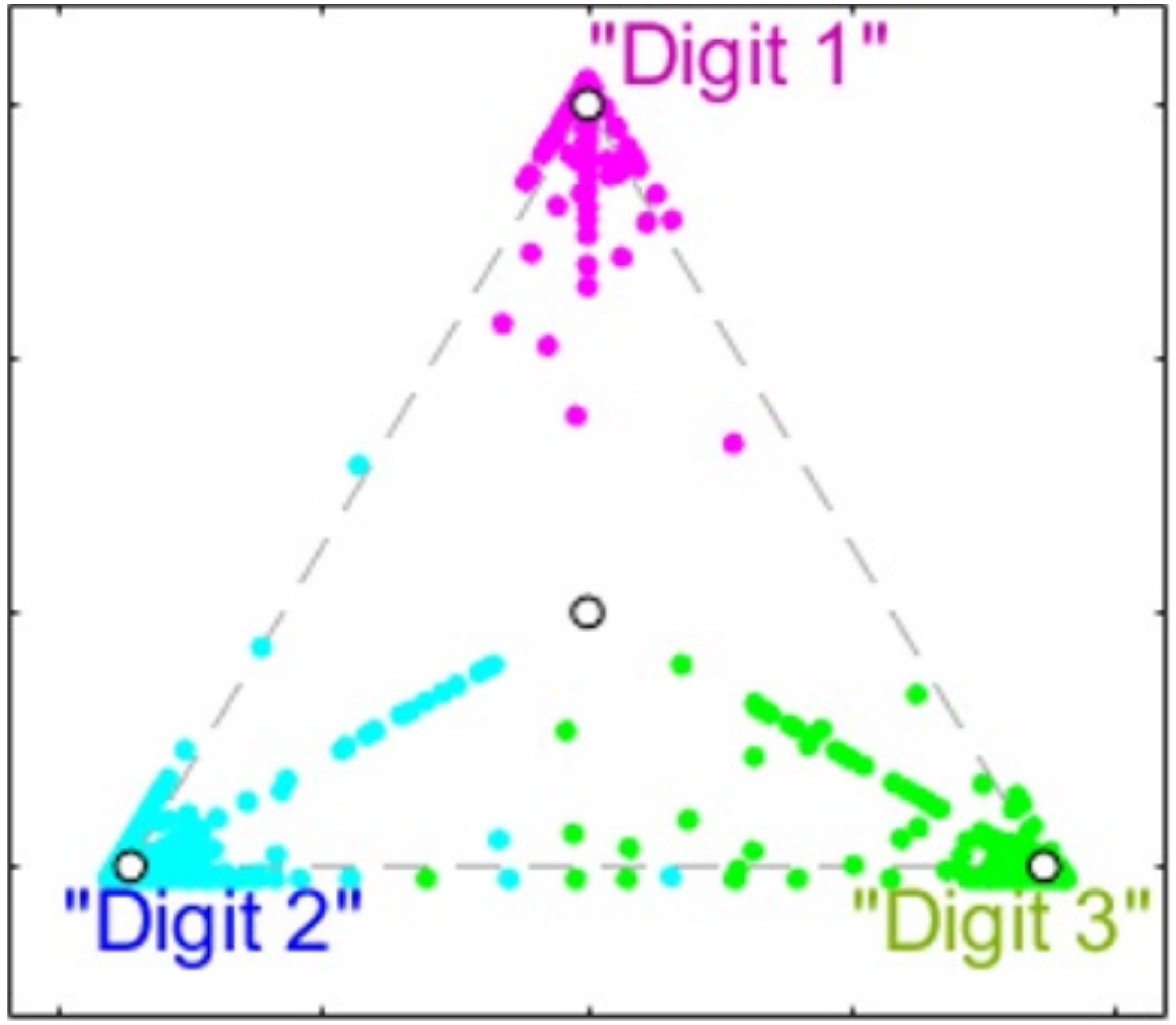
Initial Clustering



$W_1\sigma\left(W_0\mathbf{x} + \mathbf{b}_0\right) + \mathbf{b}_1$

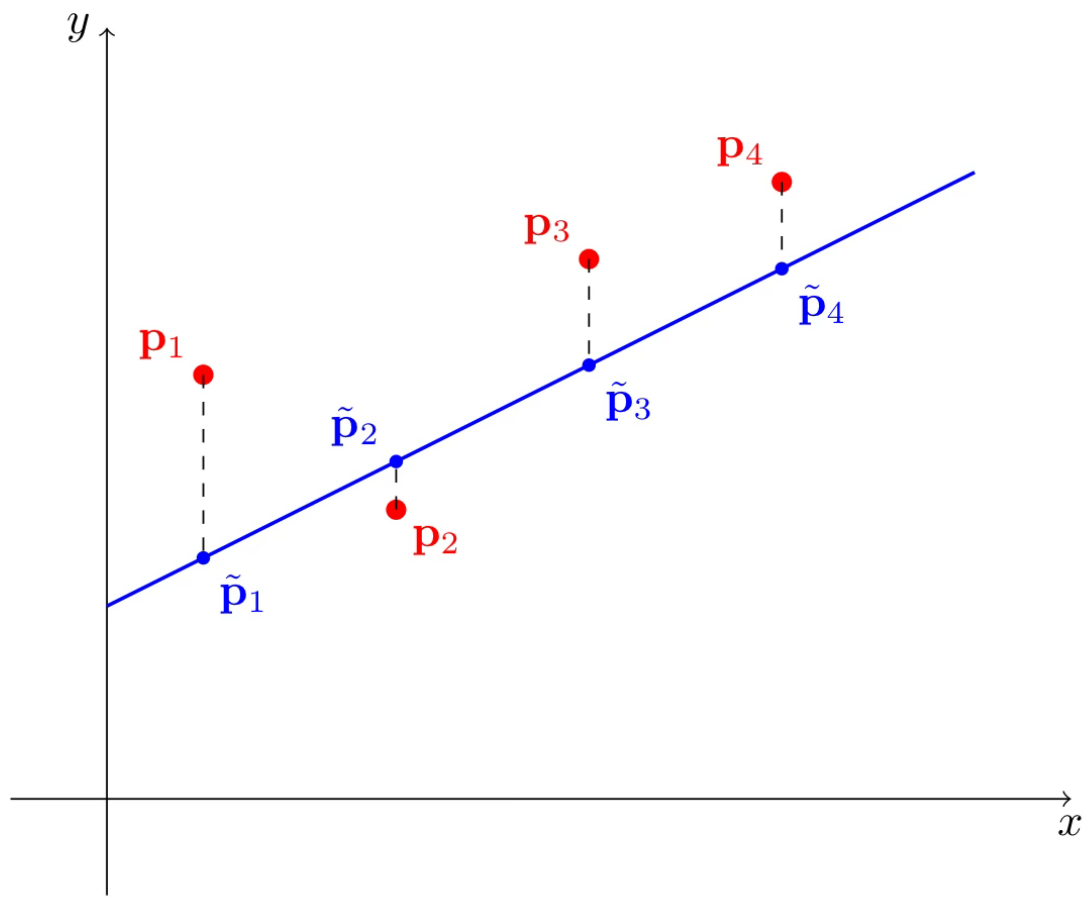


Softmax



Artificial Neural Networks

Evaluating Performance



- **Loss Function:** Squared Error

$$L(\mathbf{p}, \tilde{\mathbf{p}}) = \| \mathbf{p} - \tilde{\mathbf{p}} \|^2$$

The vector $\mathbf{p}$ is the true label $(e_0, \ldots, e_9)$ and the vector $\tilde{\mathbf{p}}$ is the ANN's prediction.

  - If we train on more than one image, we might compute the average of the sum over all images:

$$L(\mathbf{p}, \tilde{\mathbf{p}}) = \frac{1}{m} \sum_{i=1}^{m} \| \mathbf{p}_i - \tilde{\mathbf{p}}_i \|^2$$

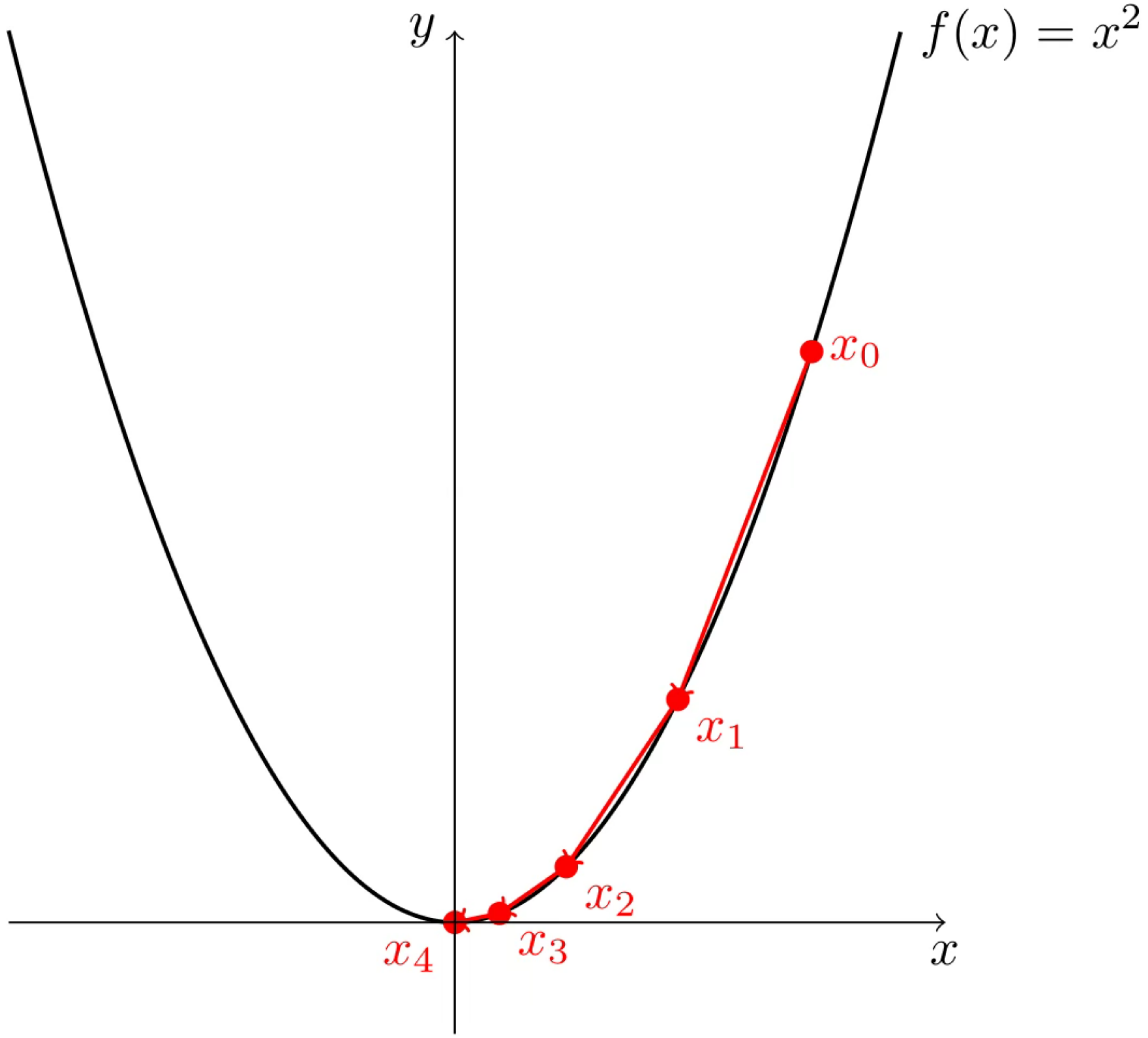
You may have heard of this loss function before. It's called ***Mean Squared Error (MSE)***.

- **Positive Rate:**

$$PR = \frac{\textbf{number of correctly predicted images}}{\textbf{number of images}}$$

Training

Finding Minimums with Derivatives



Goal: Minimize the function

$$f(x) = x^2$$

by moving along f starting at a “random” point

$$\left(x_0, f(x_0)\right).$$

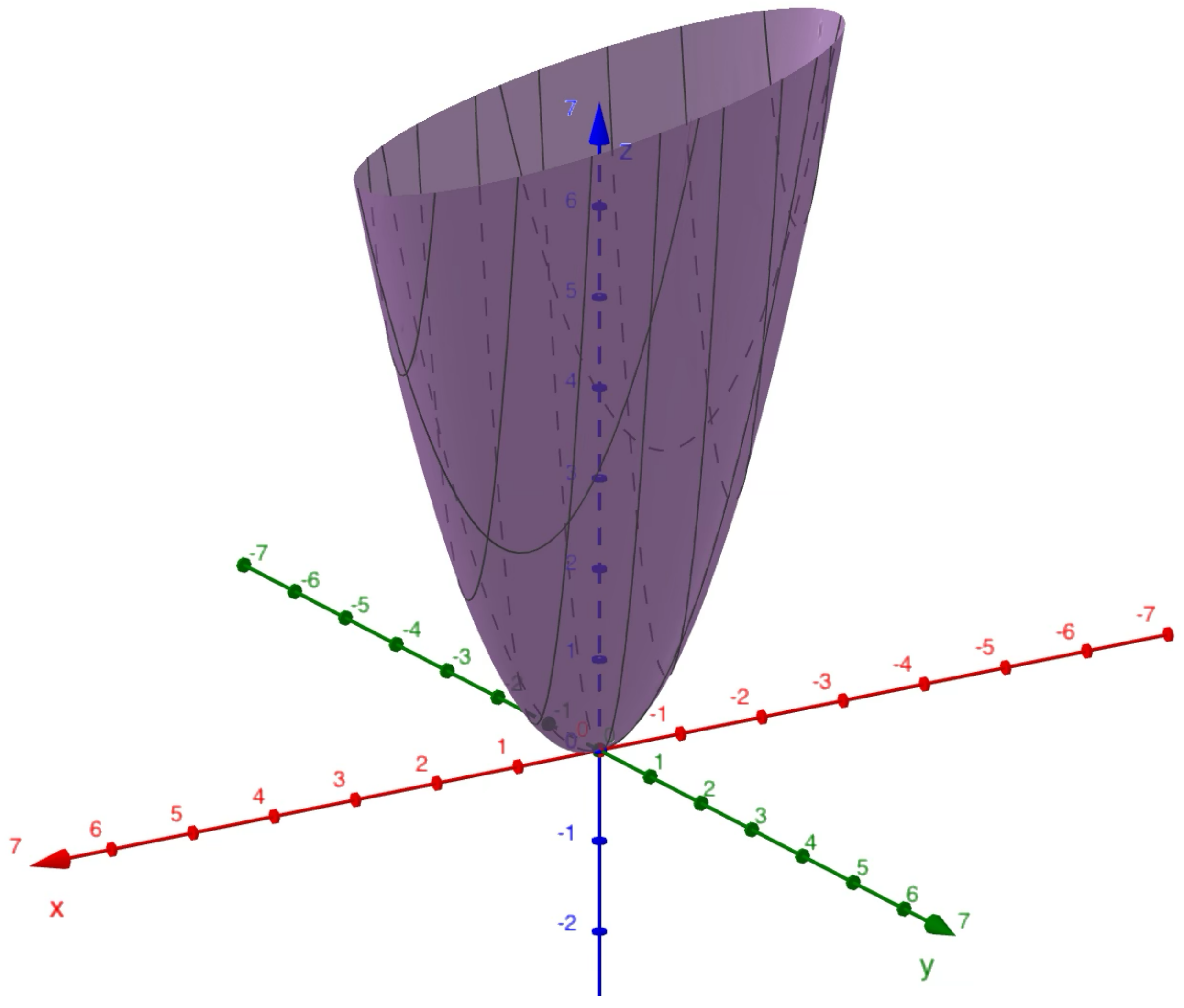
Strategy: “Update” x_0 moving in the opposite direction direction of the slope:

$$x_1 = x_0 - l \cdot f'(x_0)$$

The constant l is called the ***learning rate***.

Training

Multivariable Functions and Partial Derivatives



- We can define functions with more than one input/variable:

$$f(x, y) = x^2 - 3xy + 5y^2$$

- Partial Derivatives:

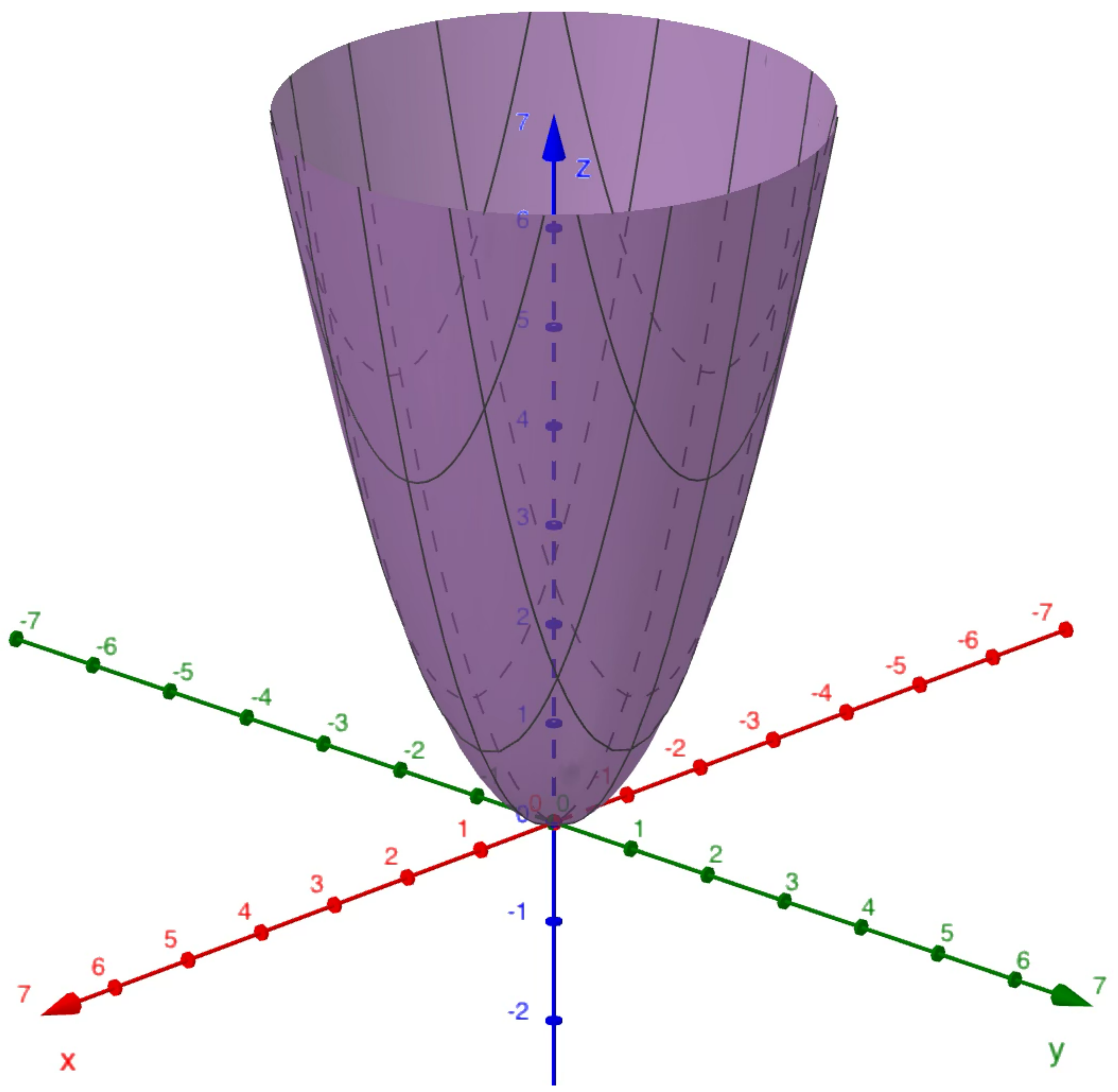
$$\frac{\partial f}{\partial x} f(x, y) = 2x - 3y \qquad \frac{\partial f}{\partial y} f(x, y) = -3x + 10y$$

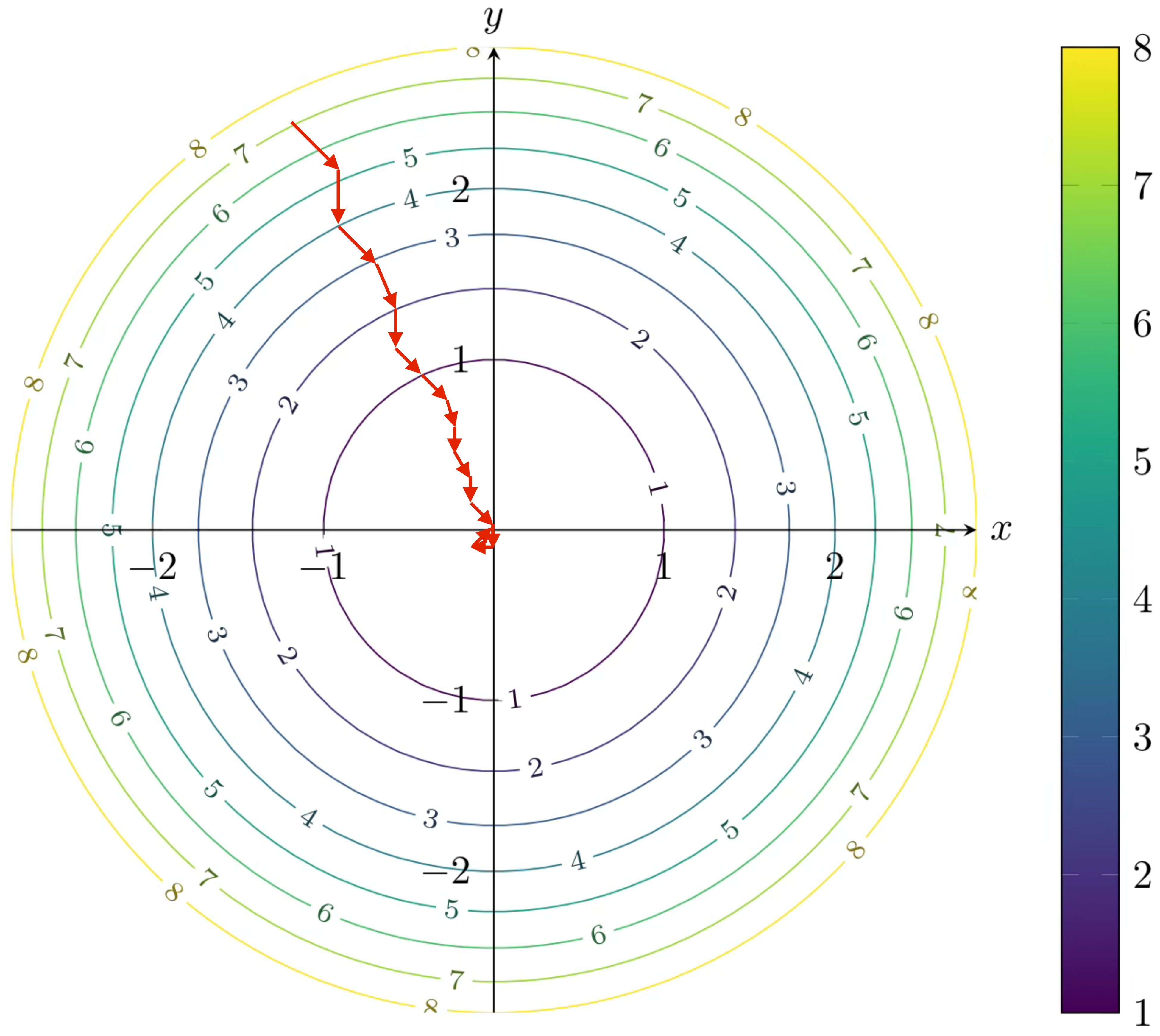
- Gradient Vector:

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x - 3y \\ -3x + 10y \end{bmatrix}$$

Training

Gradient Descent



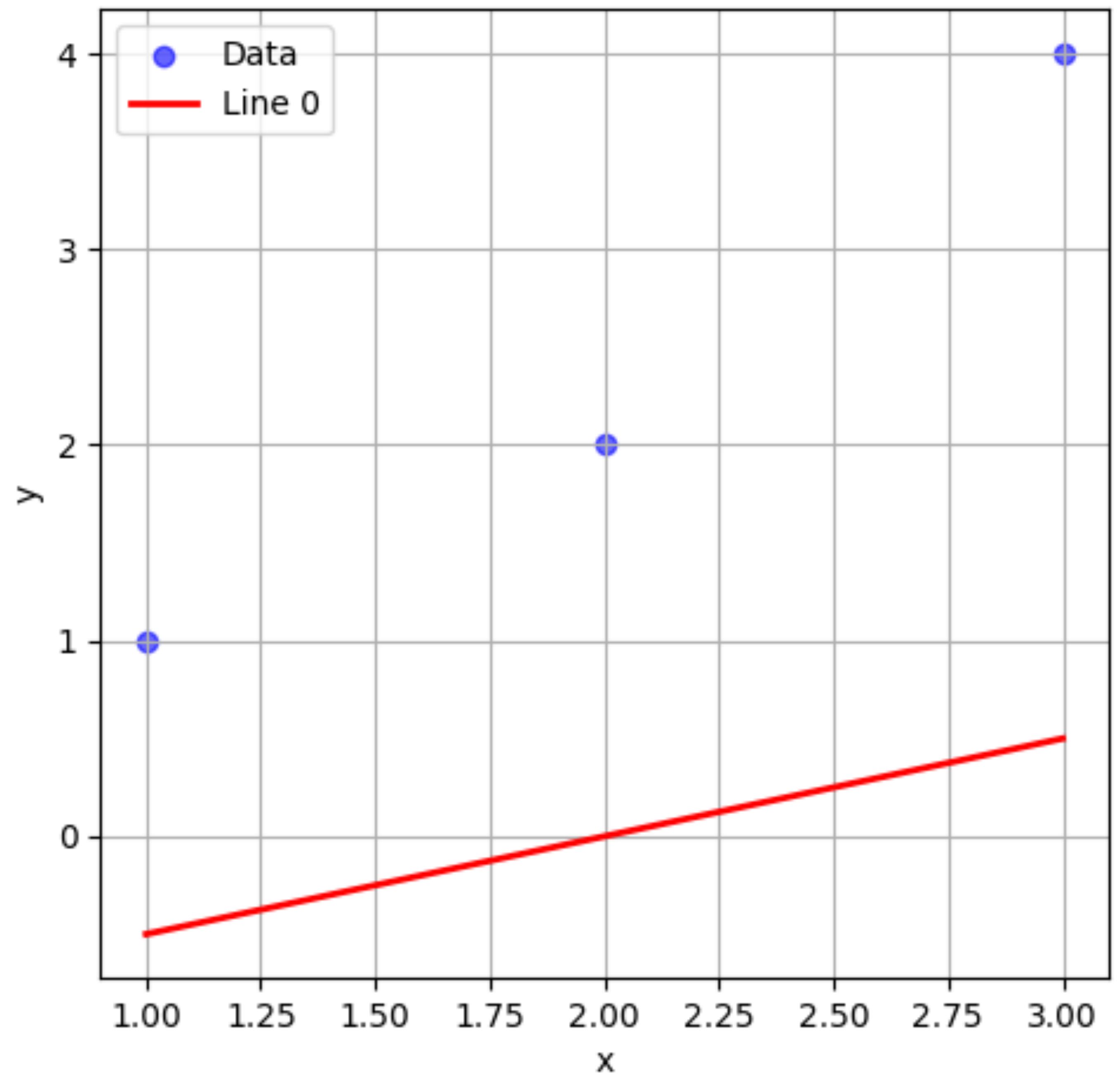


Training

Example

- Consider the dataset:

x	y
1	1
2	2
3	4



- Let's build a familiar ANN:

$$x \xrightarrow{mx+b} \tilde{y}$$

$$L\left(y, \tilde{y}\right) = \frac{1}{3}\sum_{i=1}^{3} \|y - \tilde{y}\|^2 = \frac{1}{3}\sum_{i=1}^{3} \left(y - (mx + b)\right)^2$$

$$= \frac{1}{3}\left((1 - m - b)^2 + (2 - 2m - b)^2 + (4 - 3m - b)^2\right)$$

$$L(m, b) = \frac{1}{3}\left(21 - 34m - 14b + 14m^2 + 12mb + 3b^2\right)$$

Training

Example

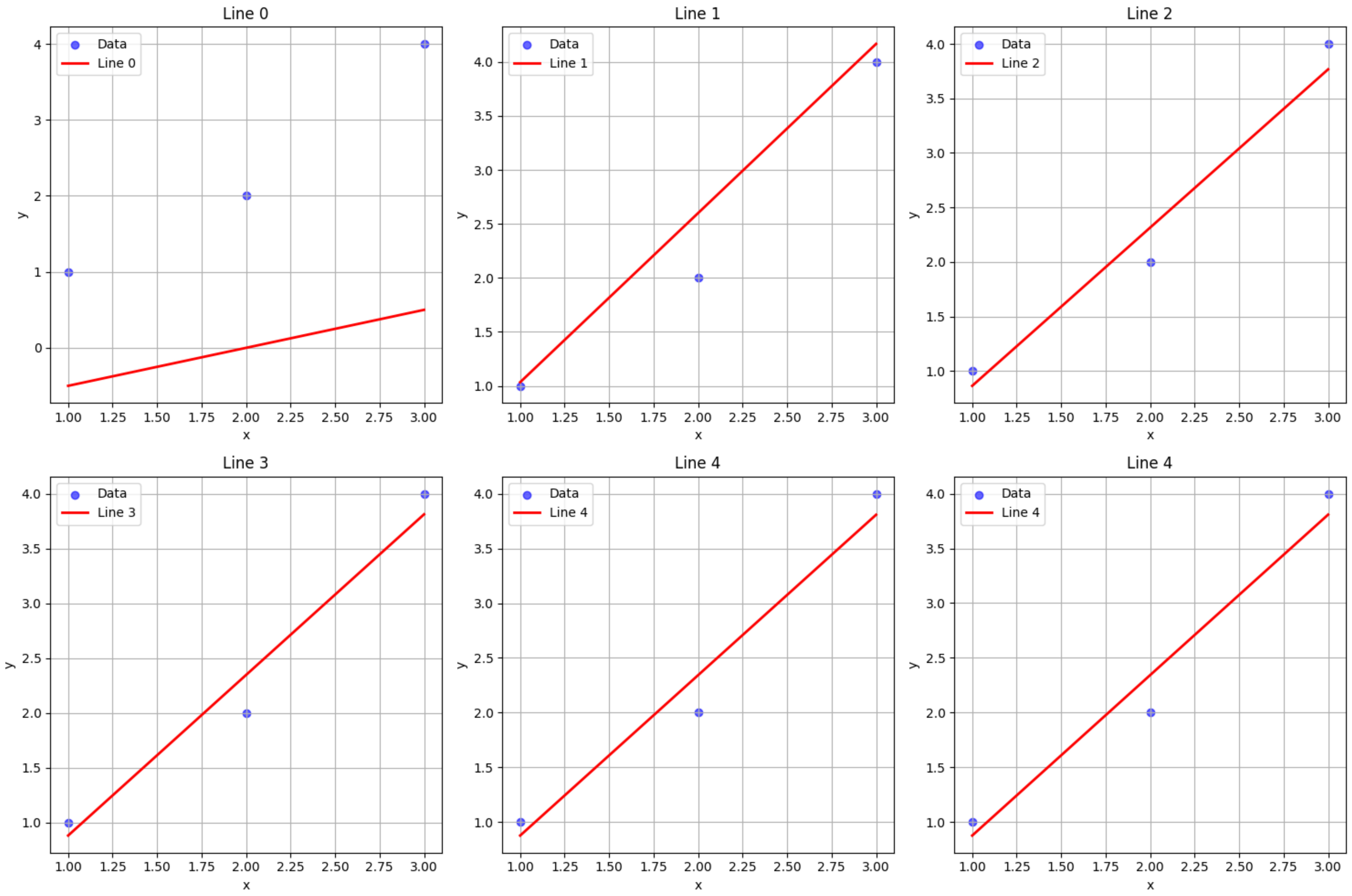
$$L(m,b) = \frac{1}{3}\left(21 - 34m - 14b + 14m^2 + 12mb + 3b^2\right)$$

x	y
1	1
2	2
3	4

$$\nabla L\left(m,b\right) = \frac{1}{3}\begin{bmatrix} -34 + 28m + 12b \\ -14 + 12m + 6b \end{bmatrix}$$

- Suppose we pick $l = 0.1$ to be our learning rate. Then:

i	m_i	b_i	$L(m,b)$
0	0.5	-1	6.16667
1	1.567	-0.533	0.12963
2	1.451	-0.587	0.05747
3	1.465	-0.583	0.05655
4	1.464	-0.586	0.05650
5	1.465	-0.588	0.05645


Line 0
Data
Line 0
Line 1
Data
Line 1
Line 2
Data
Line 2
Line 3
Data
Line 3
Line 4
Data
Line 4
Line 4
Data
Line 4
x
y

Feedforward Neural Network

Loss Function

$\sigma =$ sigmoid

$s =$ softmax

$$\underset{\mathbf{x}}{\overset{\mathbb{R}^{784}}{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{784} \end{bmatrix}}} \xrightarrow{\sigma\left(\underbrace{W_0}_{n \times 784}\overbrace{\mathbf{x}}^{784 \times 1} + \underbrace{\mathbf{b}_0}_{n \times 1}\right)} \underset{\mathbf{h}}{\overset{\mathbb{R}^{n}}{\begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix}}} \xrightarrow{s\left(\underbrace{W_1}_{10 \times n}\overbrace{\mathbf{h}}^{n \times 1} + \underbrace{\mathbf{b}_1}_{10 \times 1}\right)} \underset{\mathbf{p}}{\overset{\mathbb{R}^{10}}{\begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_9 \end{bmatrix}}}$$

Our loss function

$$L\left(W_0, \mathbf{b}_0, W_1, \mathbf{b}_1\right)$$

is a map of points

$$\mathbb{R}^{795n+10} \to \mathbb{R}$$

This ANN has $n \cdot 784 + n \cdot 1 + 10 \cdot n + 10 \cdot 1 = \boxed{795n + 10 \text{ parameters.}}$

Feedforward Neural Network

Loss Function

σ = sigmoid

s = softmax

$$\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{784} \end{bmatrix}}_{\mathbf{x}} \in \mathbb{R}^{784} \xrightarrow{\sigma\left(W_0\mathbf{x} + \mathbf{b}_0\right)} \underbrace{\begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix}}_{\mathbf{h}} \in \mathbb{R}^{n} \xrightarrow{s\left(W_1\mathbf{h} + \mathbf{b}_1\right)} \underbrace{\begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_9 \end{bmatrix}}_{\mathbf{p}} \in \mathbb{R}^{10}$$

784×1 $n \times 784$ $n \times 1$ $n \times 1$ $10 \times n$ 10×1

Our loss function

$$L\left(W_0, \mathbf{b}_0, W_1, \mathbf{b}_1\right)$$

is a map of points

$$\mathbb{R}^{795n+10} \to \mathbb{R}$$

This ANN has $n \cdot 784 + n \cdot 1 + 10 \cdot n + 10 \cdot 1 = \boxed{795n + 10 \text{ parameters.}}$

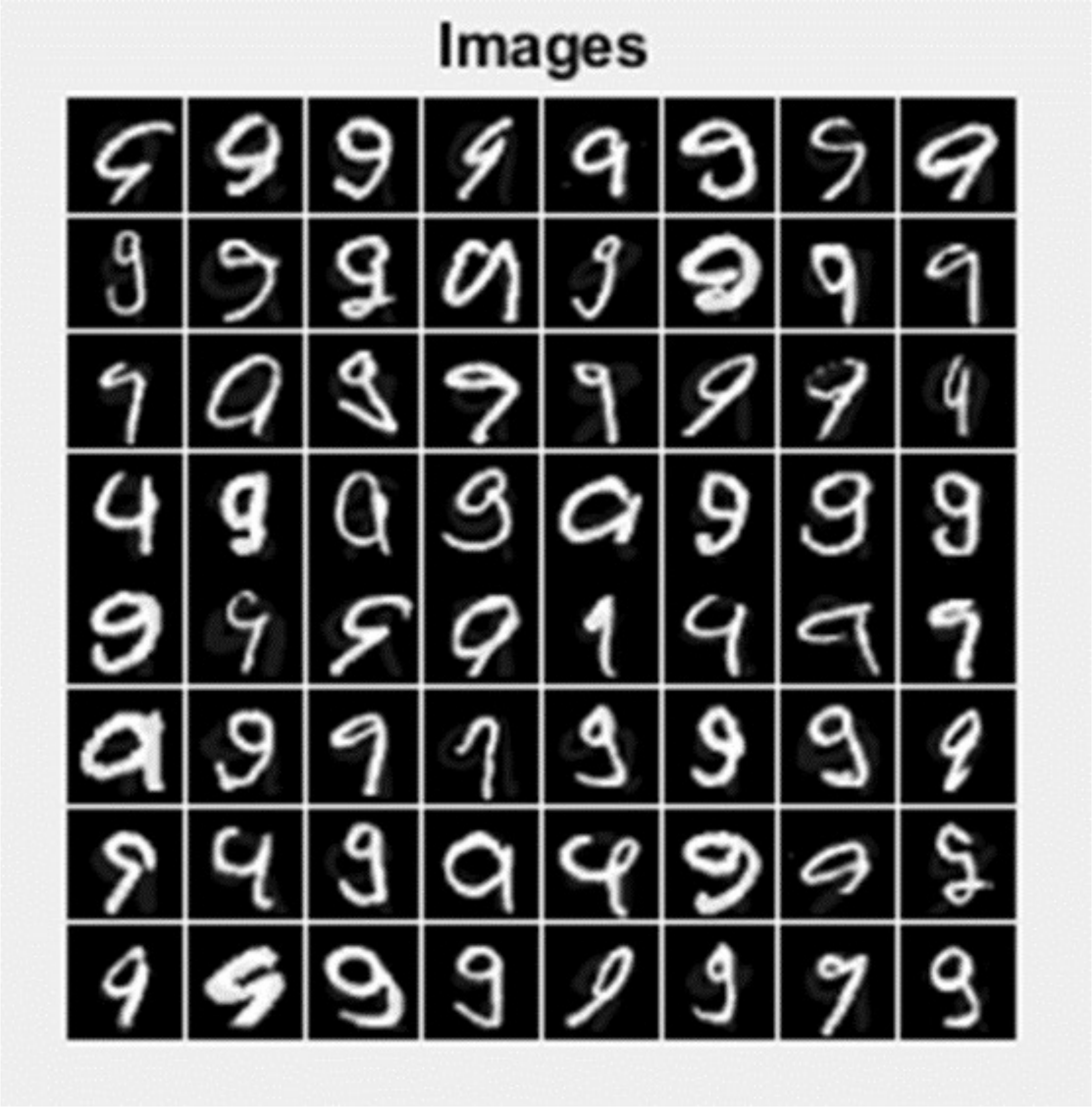
Training on A LOT of Data

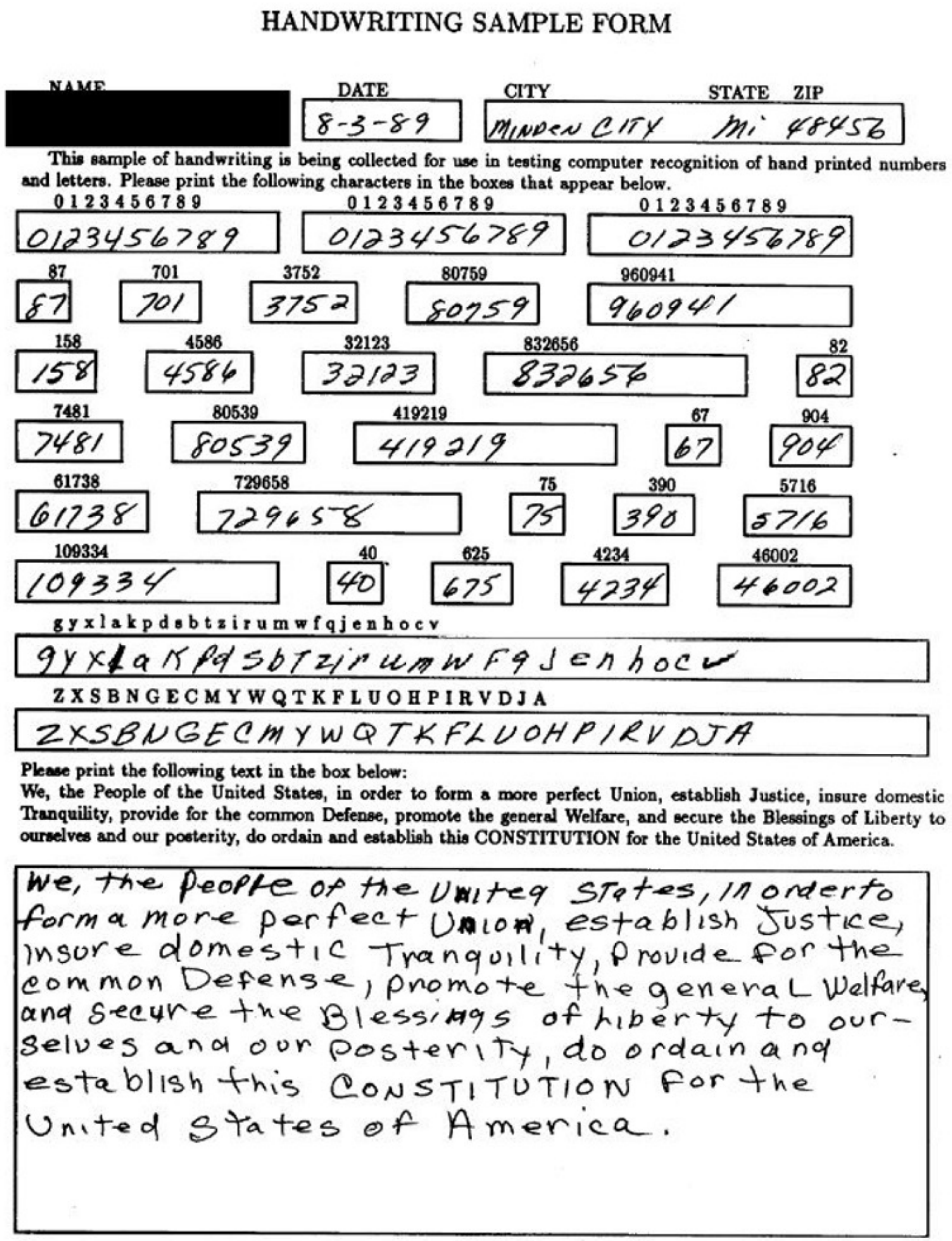
MNIST Dataset

MNIST Handwritten Dataset
- 60,000 training samples from federal employees
- 10,000 test samples from high school students

Modified for Digit Classification in 1994
→ MNIST Data Set

MNIST = Modified National Institute
Standards and Technology





HANDWRITING SAMPLE FORM

NAME [redacted] DATE CITY STATE ZIP

This sample of handwriting is being collected for use in testing computer recognition of hand printed numbers and letters. Please print the following characters in the boxes that appear below.

0 1 2 3 4 5 6 7 8 9 | 0 1 2 3 4 5 6 7 8 9 | 0 1 2 3 4 5 6 7 8 9

87 | 701 | 3752 | 80759 | 960941

158 | 4586 | 32123 | 832656 | 82

7481 | 80539 | 419219 | 67 | 904

61738 | 729658 | 75 | 390 | 5716

109334 | 40 | 625 | 4234 | 46002

g y x l a k p d s b t z i r u m w f q j e n h o c v

Z X S B N G E C M Y W Q T K F L U O H P I R V D J A

Please print the following text in the box below:
We, the People of the United States, in order to form a more perfect Union, establish Justice, insure domestic Tranquility, provide for the common Defense, promote the general Welfare, and secure the Blessings of Liberty to ourselves and our posterity, do ordain and establish this CONSTITUTION for the United States of America.

Training on A LOT of Data

Stochastic Gradient Descent

- Recall our loss function:

$$L\left(W_0, \mathbf{b}_0, W_1, \mathbf{b}_1\right) = \frac{1}{60{,}000} \sum_{i=1}^{60{,}000} \left\| y_i - \tilde{y}_i \right\|$$
$$= \frac{1}{60{,}000} \sum_{i=1}^{60{,}000} \left\| \mathbf{y}_i - s\left(W_1 \sigma\left(W_0 \mathbf{x} + \mathbf{b}_0\right) + \mathbf{b}_1\right) \right\|$$

- We need to perform gradient descent on this function to train our model—yikes!

- For context, computing the loss function for given weights $W_0, \mathbf{b}_0, W_1, \mathbf{b}_1$ requires more than $60{,}000(1{,}593n + 10)$ operations. Performing gradient descent requires much more!
    - Even a single iteration of gradient descent for this example would be too many computations for most computers to handle because they run out of random access memory (RAM).

Training on A LOT of Data

Stochastic Gradient Descent

- **New Strategy:** Perform gradient descent in batches.
- ***Stochastic gradient descent*** is the process of performing gradient descent in random batches of your training set.
- For example, suppose we take batches of 100, then at each gradient descent step our loss function simplifies to

$$\begin{aligned} L\left(W_0, \mathbf{b}_0, W_1, \mathbf{b}_1\right) &= \frac{1}{100} \sum_{i=1}^{100} \left\| y_i - \tilde{y}_i \right\| \\ &= \frac{1}{100} \sum_{i=1}^{100} \left\| \mathbf{y}_i - s\left(W_1 \sigma\left(W_0 \mathbf{x} + \mathbf{b}_0\right) + \mathbf{b}_1\right) \right\| \end{aligned}$$

Training on A LOT of Data

Stochastic Gradient Descent

- Let's continue with the example of splitting the training data into batches of 100.
  - **Step 1:** Randomly partition the 60,000 samples into 6,000 batches of 100.
  - **Step 2:** Perform gradient descent on each batch, updating our model weights $W_0, \mathbf{b}_0, W_1, \mathbf{b}_1$ at after each gradient descent step. That is, 6,000 iterations of gradient descent to get through each batch!
- One iteration of steps 1 and 2 is called an ***epoch***.
- When training a model, we usually perform hundreds or thousands of epochs.

Feedforward Neural Networks

Recap and Next Steps

- You now have all the intuition for training a neural network!
  - Structure or *architecture* of a FNN.
  - Evaluate performance with a loss function.
  - Gradient descent on a loss function (with few parameters).
- Now we'll train our first models utilizing software called TensorFlow that automates training the model for us.
- What's next?
  - Chain Rule for multivariable functions.
  - Training a model from scratch (without the help of TensorFlow).