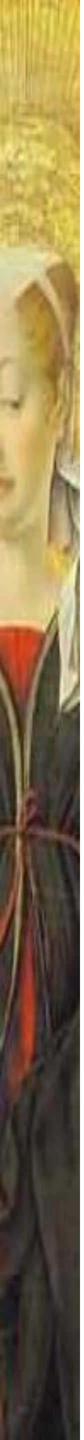


Topics in Machine Learning

CS212 Topics in Computing 2
Second Half
Lecture 3

Term 2 2019

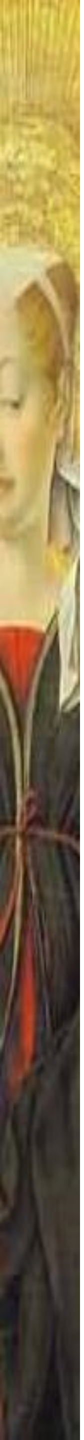
Dr. Dmitri Roussinov
dmitri.roussinov@strath.ac.uk

Admin

- We will use 11^{th} floor labs only (for providing help and recording attendance)
- Let me know if you don’t find a spot to work

Recap

- Modern trainable systems consist of millions of artificial neurons that are very similar to a Perceptron
- Multiple layers allow to learn interaction between several variables
- So the learned function does not need to be monotone
- Real data has some elements of AND and OR
- If we teach perceptrons to learn combinations of AND and OR in the data, we can teach them to learn anything!



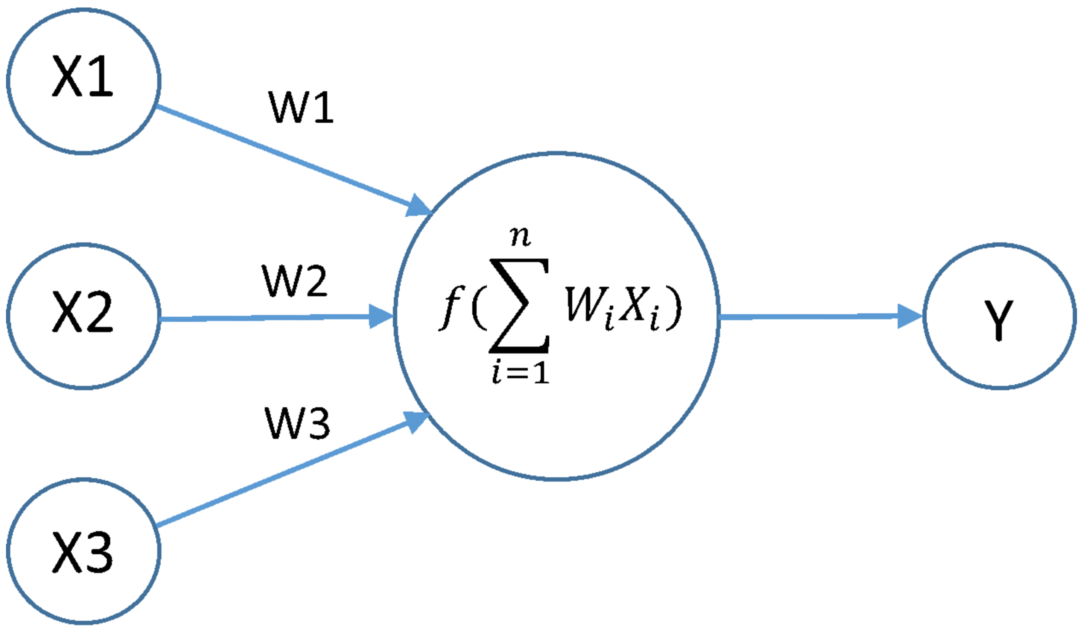
Where we Are

- We know how perceptrons operate
- We know what they can learn
- We know how to find solutions analytically
  - in some “easy” situations (e.g. symmetric monotone boolean functions)
- Next: general perceptron training algorithm
- After Next: so we can train them to recognize hand-written digits (MNIST)

Next: Getting Rid of Bias

- So our algorithm is more simple/uniform:
  - We don’t need to handle bias in a different way as we handle W-s
- As we will see …

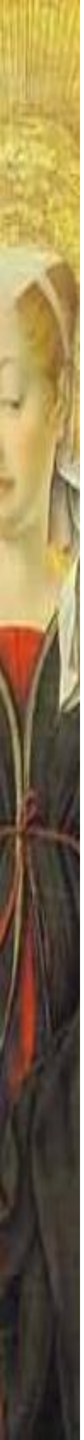
Recall A Single Neuron



- Activations: $a = \sum_{i=1}^{n} W_i X_i$
- Inputs: *{X1,X2,…}*
- Prediction: $Y = f(a)$

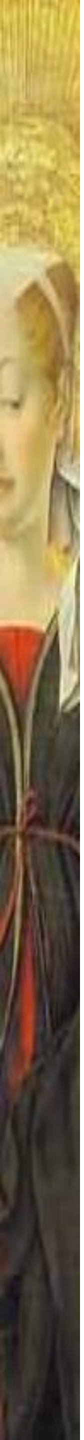
What is a Bias

- Sometime the nonlinearity function f() can also have a bias b added to the activation
- As it is in the perceptron:
- $Y = 1 \text{ for } W1*X1 + W2*X2 + b >= 0, 0$ otherwise
- It helps to shift the typical activation values up or down as needed
- But it also makes the formula more complex



Instead: Introducing another Weight

- a = $W1 * X1 + W2 * X2 + W3 * X3$
- We will always set values $X3 = 1$
- This way W3 plays the same role as b
- So we don’t need b
- Instead we have one more weight to train
- Updated output formula:
- Y = 1 if $W1 * X1 + W2 * X2 + W3 * X3 >= 0$, and 0 otherwise



Perceptron Training Algorithm

Inputs:

training examples: label, features X[]

perceptron with any initial weights W[]

Outputs:

perceptron with trained weights W[]

Algorithm:

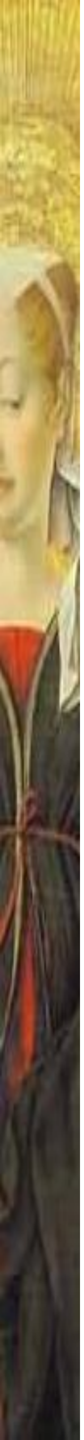
While error exists:

For all training examples do:

error = label - prediction

For i from 1 to length(W) do:

W[i] += learning_rate * error * X[i]



Terminology: and Epoch

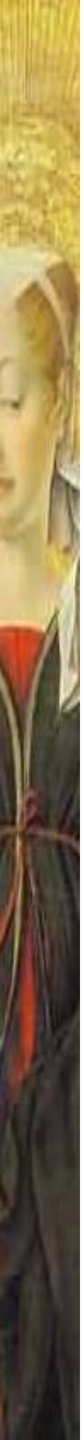
- An **epoch** is one complete presentation of the training data set
- Recall: in real applications we also use **testing set** which is different from training set
- This is to simulate **real-life conditions** when your trained system will encounter new data

Exercise: Apply it to Boolean OR

X1	X2	Yt
0	0	0
1	0	1
0	1	1
1	1	1

- We will start with initial values
  W1 = W2 = W3 = 0
- We will use learning_rate of 1.
- Yt is “target label” or simply “label”

- Correct the small typo in the exercise printed, should be W[i] **+=** error * X[i]



Very First Training Iteration

update rule: W[i] += error * X[i]

initial W1,W2,W3 = (0,0,0)

- X1 = ?

  0
- X2 = ?

  0
- X3 = ?

  1
- Yt = ?

  0

- a = ?

  0*0 + 0*0 + 0*1

  = 0
- Y = ?

  1
- **error = Yt – Y =**

  -1

- W1 = ?

  0
- W2 = ?

  0
- W3 = ?

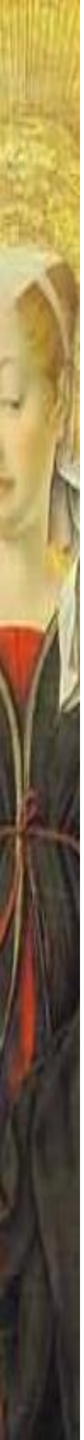
  -1

Observations

- X1 and X2: we can only have 0 or 1
- X3: we will always have 1
  - Recall this is how we implement a bias!
- error: can be?

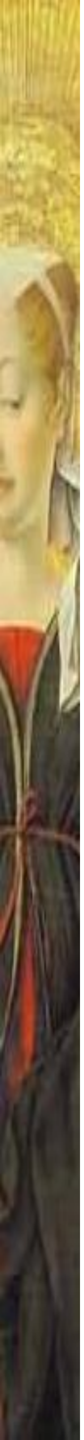
  {-1,0,1}

- if error = -1, that means our perceptron "overshoots"
- So W-s are decreased
- But only those that connect X-s that are 1
- For X-s that are 0s, W-s are not changed



To Check

- What will be the error if we had submitted the same training example again?



Second Training Iteration

update rule: W[i] += error * X[i]

prior W1,W2,W3 = (0,0,-1)

- X1 = ?

  1
- X2 = ?

  0
- X3 = ?

  1
- Yt = ?

  1

- a = ?

  0*1 + 0*0 + (-1)*1

  = -1
- Y = ?

  0
- **error = Yt – Y =**

  1

- W1 = ?

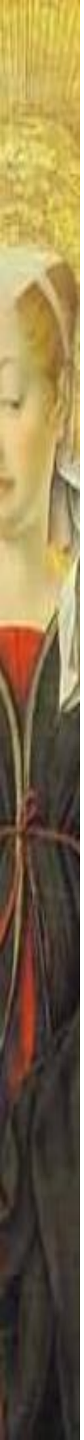
  1
- W2 = ?

  0
- W3 = ?

  -1+1 = 0

Observations

- if error = 1, that means our perceptron "undershoots"
- So W-s are increased
- But only those that connect X-s that are 1
- For X-s that are 0s, W-s are not changed
- The weights became (1, 0, 0)
- Since positive X1 was positive, so its weight went up



Third Training Iteration

update rule: W[i] += error * X[i]

prior W1,W2,W3 = (1,0,0)

- X1 = ?

  0
- X2 = ?

  1
- X3 = ?

  1
- Yt = ?

  1

- a = ?

  1*0 + 0*1 + 0*1

  = 0
- Y = ?

  1
- **error = Yt – Y =**

  0

- W1 = ?

  1
- W2 = ?

  0
- W3 = ?

  0

Observations

- if error = 0
- We don’t need to change anything!
- Does it mean we are done?
- No!
- Since some other training examples still may result in errors

Fourth Training Iteration

update rule: W[i] += error * X[i]

prior W1,W2,W3 = (1,0,0)

- X1 = ?
  1
- X2 = ?
  1
- X3 = ?
  1
- Yt = ?
  1

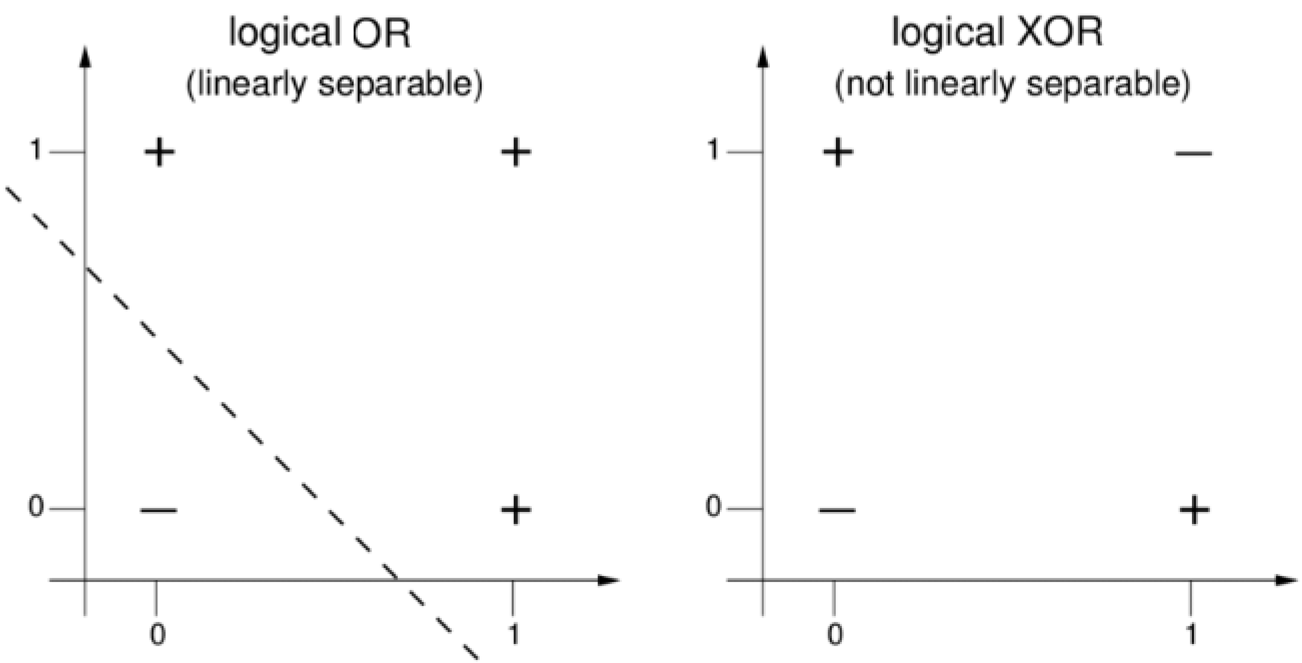
- a = ?
  1*1 + 0*1 + 0*1
  = 1
- Y = ?
  1
- **error = Yt – Y =**
  0

- W1 = ?
  1
- W2 = ?
  0
- W3 = ?
  0

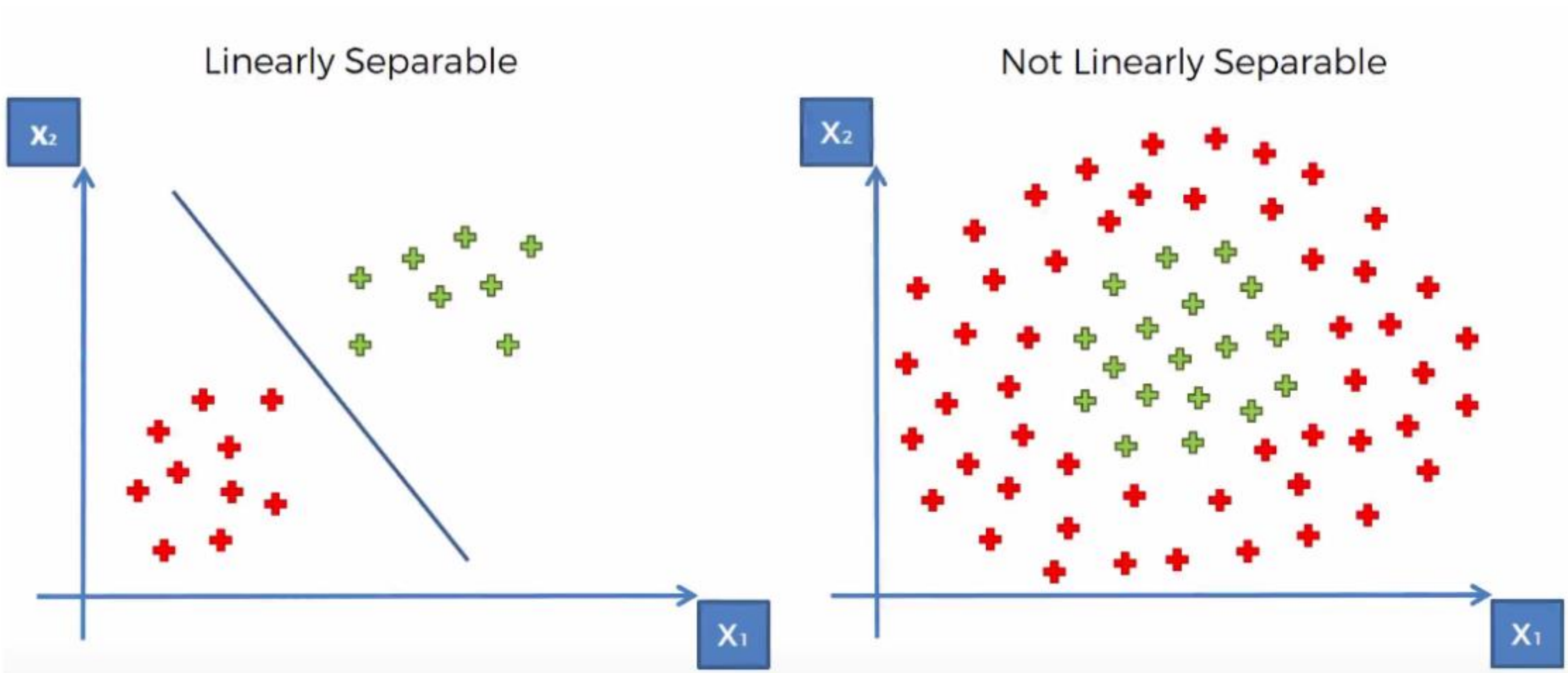
Math Properties

- The algorithm converges to 0 error solution if the dataset is linearly separable (examples follow)
- Otherwise, it does not converge
- Most real life datasets are not linearly separable
- So, a compromise needs to be found
- Typically, convergence is forced by gradually decreasing learning rate

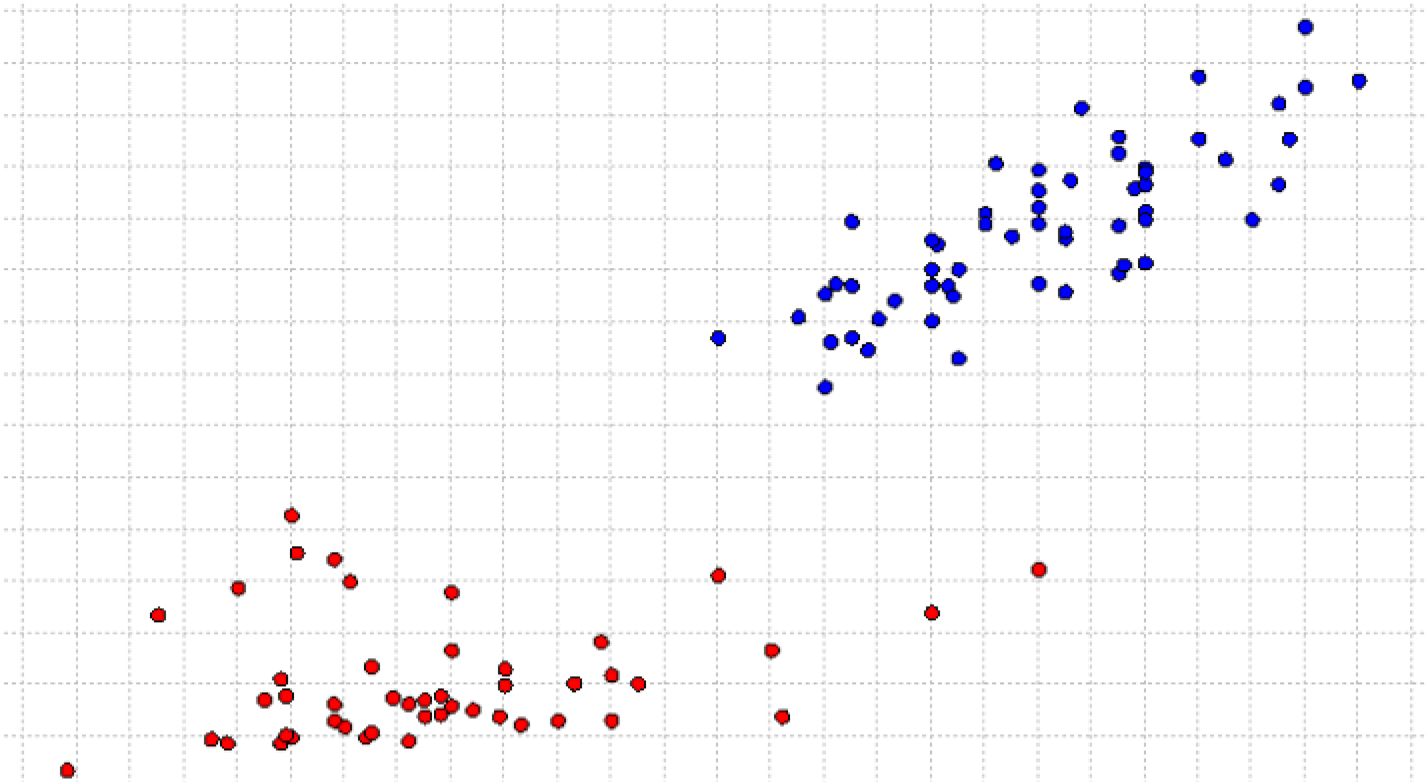
Examples: OR vs XOR



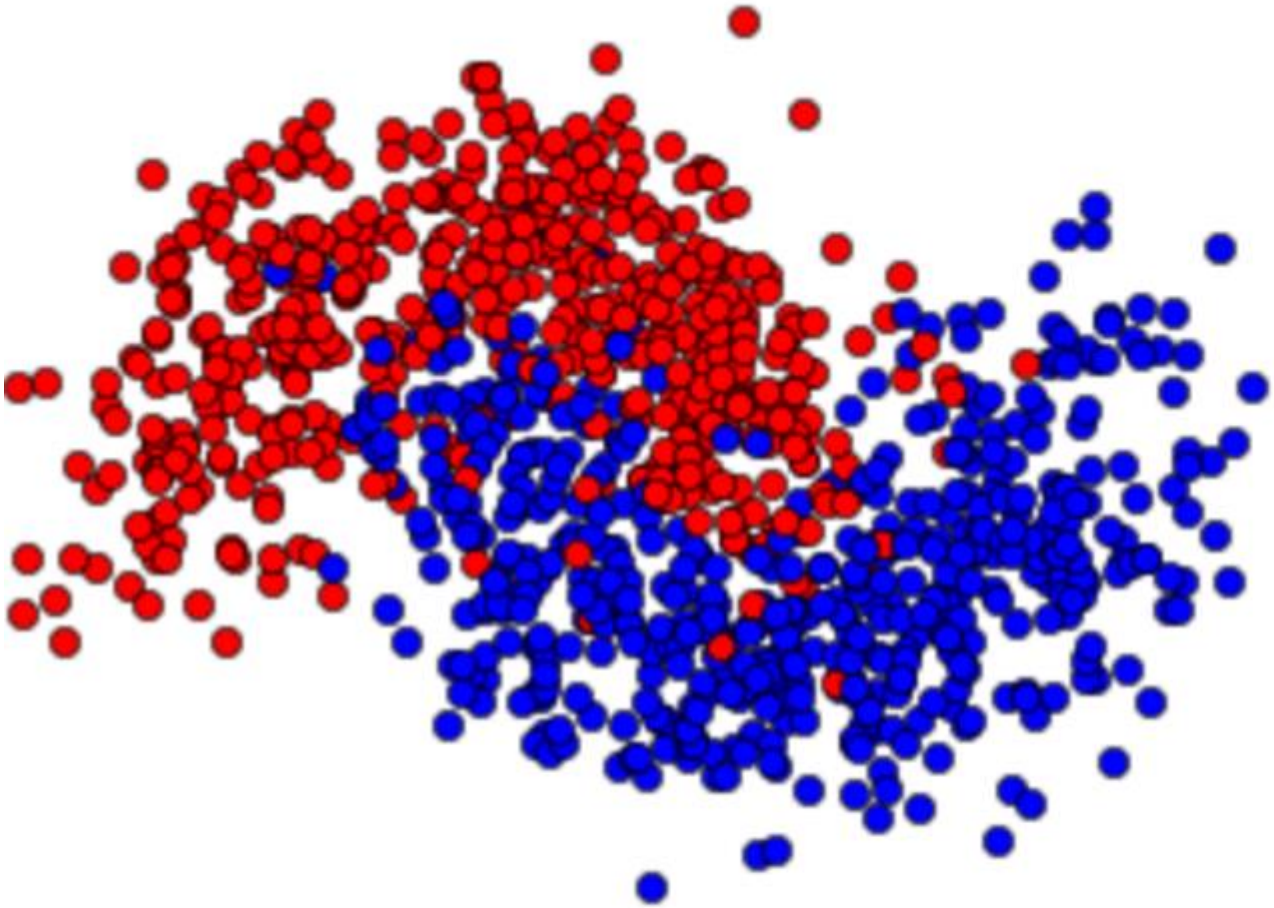
More realistic examples



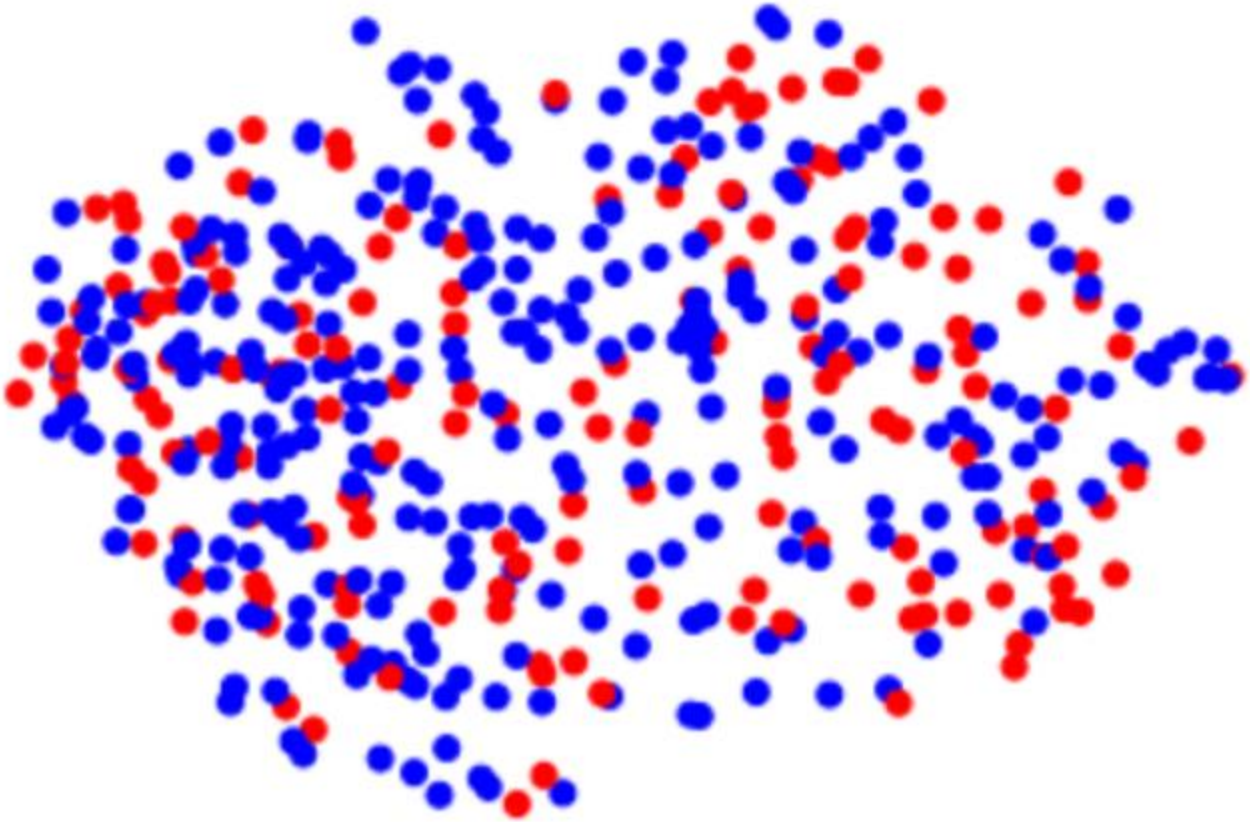
How about this?



And this?



And finally this?



MNIST task

Data Format

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	84	185	159	151	60	36	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	222	254	254	254	254	241	198	198	198	198	198	198	198	198	170	52	0
7	0	0	0	0	0	0	67	114	72	114	163	227	254	225	254	254	254	250	229	254	254	140	0
7	0	0	0	0	0	0	0	0	0	0	0	17	66	14	67	67	67	59	21	236	254	106	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	83	253	209	18	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22	233	255	83	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	129	254	238	44	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59	249	254	62	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	133	254	187	5	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	205	248	58	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	126	254	182	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	75	251	240	57	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	19	221	254	166	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	3	203	254	219	35	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	38	254	254	77	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	31	224	254	115	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	133	254	254	52	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	61	242	254	254	52	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	121	254	254	219	40	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	121	254	207	18	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	116	125	171	255	255	150	93	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	169	253	253	253	253	253	253	218	30	0	0	0	0	0
2	0	0	0	0	0	0	0	0	169	253	253	253	213	142	176	253	253	122	0	0	0	0	0
2	0	0	0	0	0	0	0	52	250	253	210	32	12	0	6	206	253	140	0	0	0	0	0
2	0	0	0	0	0	0	0	77	251	210	25	0	0	0	122	248	253	65	0	0	0	0	0
2	0	0	0	0	0	0	0	0	31	18	0	0	0	0	209	253	253	65	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	117	247	253	198	10	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	76	247	253	231	63	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	128	253	253	144	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	176	246	253	159	12	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	25	234	253	233	35	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	198	253	253	141	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	78	248	253	189	12	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	19	200	253	253	141	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	134	253	253	173	12	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	248	253	253	25	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	248	253	253	43	20	20	20	20	5	0	5	20	20	37	150
2	0	0	0	0	0	0	0	0	248	253	253	253	253	253	253	253	168	143	166	253	253	253	253
2	0	0	0	0	0	0	0	0	174	253	253	253	253	253	253	253	253	253	253	253	249	247	247
2	0	0	0	0	0	0	0	0	0	118	123	123	123	166	253	253	253	155	123	123	41	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Things To Note:

- 28 lines per image
- Label (digit) is the very first number on each line
- It is separated by \t from the pixel values
- Pixel values are separated by 1+ empty spaces
- 9000 images to train
- 1000 images to test

What We Want

- To read a file with any number of images
  - So we can use train and test sets separately
- To store images in memory
  - So we can use them to train or to test
- We want to represent each image as a flat array of pixels
  - Why?
  - So we can feed it to our perceptron

Possible Implementations

- Next few slides show some hints on how it can be implemented
- But you can do it in your own way

Useful Java Functions

- split("[\t]+")
- Integer.*parseInt()*

Possible Java Class To Store Images

```java
static int  image_size = 28 * 28;

static class Image {
    double[] pixels; // 28x28 = 784 numbers in flat array
    int label;

    Image() {
        pixels = new double[image_size];
    }
    void set(int pixel, int value) {
        pixels[pixel] = value;
    }
    void set_label(int value) {
        label = value;
    }
}
```

Getting the Number of Images in the File

```
static Image[] Read(String fileName) throws Exception {

  File file = new File("E:\\212-labs\\test-10.txt");

  BufferedReader br = new BufferedReader(new FileReader(file));
  String st;
  int number_of_lines = 0;
  while ((st = br.readLine()) != null)
    number_of_lines++;
  int number_of_images = number_of_lines / 28;
  Image[] t = new Image[number_of_images];

  ...
```

Getting All Pixel Values

```
br = new BufferedReader(new FileReader(file));
for(int i = 0; i < number_of_images; i++) {
  t[i] =  new Image();
  for(int row = 0; row < 28; row ++) {

      st = br.readLine();
      String[] line_parts = st.split("[\t ]+");
      int label = Integer.parseInt(line_parts[0]);
      for(int pixel = 0; pixel < 28; pixel++) {
          int value = Integer.parseInt((line_parts[1 + pixel]));
          t[i].set(row * 28 + pixel, value);
      }
      t[i].set_label(label);
  }
}
```