Neural networks 1 Oct 12, 2022

- Ch 8 of Ertel's book and Ch 25 of [RN]
  - neuron model
  - perceptron abstraction of neuron
  - perceptron training algorithm, convergence, linear separability
  - multilayered neural network
  - backpropagation algorithm for training

- T. Sejnowski : Deep learning revolution
  - Deep networks and applications

Machine Learning

- Supervised learning
  - Bayesian classifier, Bayes networks
  - regression models
  - decision trees, decision lists
  - perceptrons, neural networks
- Unsupervised learning
  - clustering techniques
- reinforcement learning
- PCA, dimension reduction
- genetic algorithms, fuzzy logic

Connectionist models

Consider humans:

- Neuron switching time ~ .001 second
- Number of neurons ~ 10^{10}
- Connections per neuron ~ 10^{4-5}
- Scene recognition time ~ .1 second
- 100 inference steps doesn't seem like enough

$\rightarrow$ much parallel computation

Properties of artificial neural nets (ANN's):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically

When to Consider Neural Networks

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant

Examples:

- Speech phoneme recognition [Waibel]
- Image classification [Kanade, Baluja, Rowley]
- Financial prediction

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH AND WALTER PITTS

FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,
AND THE UNIVERSITY OF CHICAGO

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.

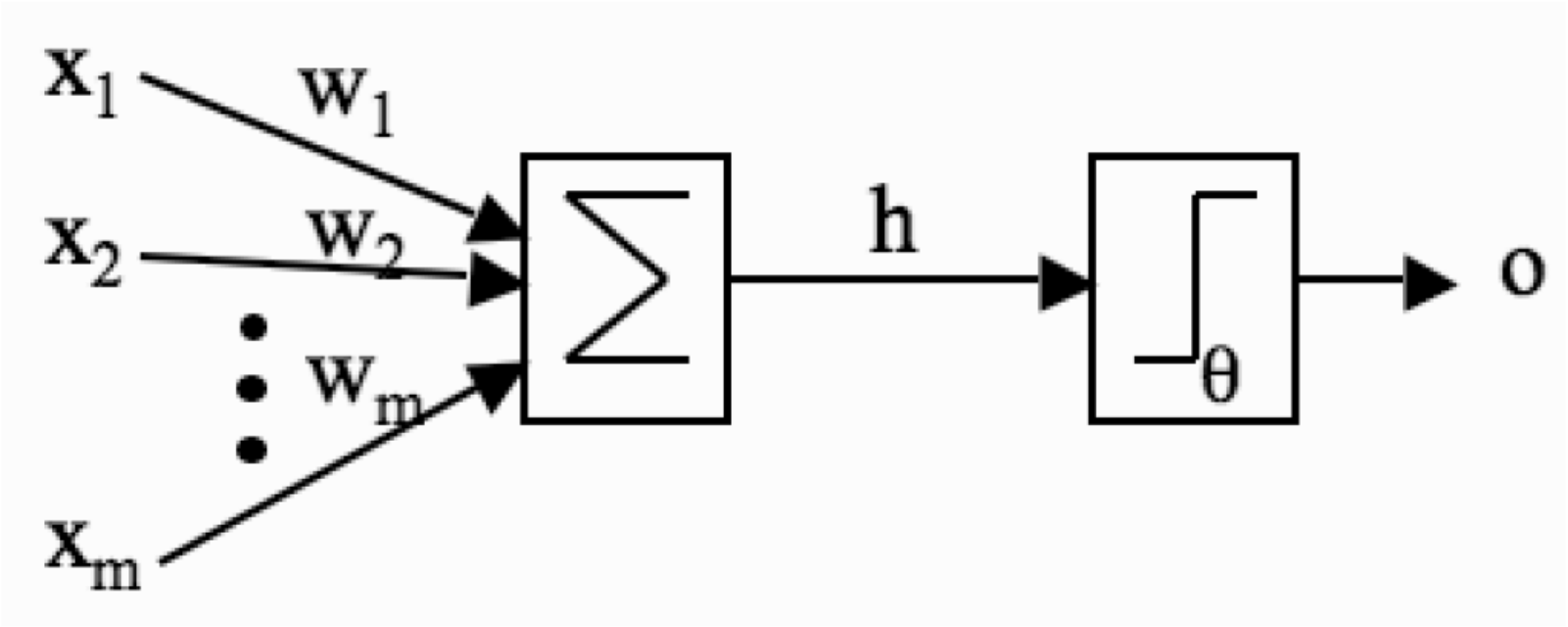


FIGURE 3.1 A picture of McCulloch and Pitts' mathematical model of a neuron. The inputs x_i are multiplied by the weights w_i, and the neurons sum their values. If this sum is greater than the threshold θ then the neuron fires; otherwise it does not.

McCulloch and Pitts Neurons

$$h = \sum_{i=1}^{m} x_i w_i$$

$$o = \begin{cases} 1 & h \geq \theta \\ 0 & h < \theta \end{cases}$$

for some threshold q

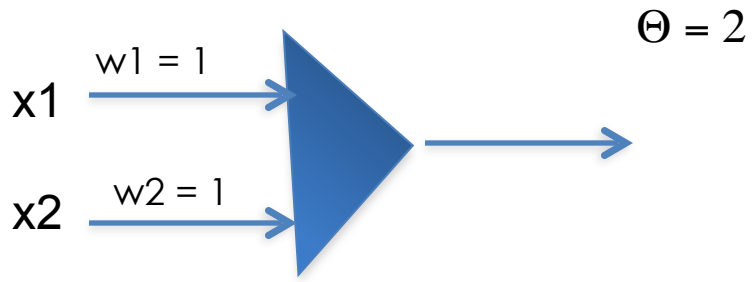
- The weight w_j can be positive or negative
  - Inhibitory or exitatory
- Use only a linear sum of inputs

Neural Networks

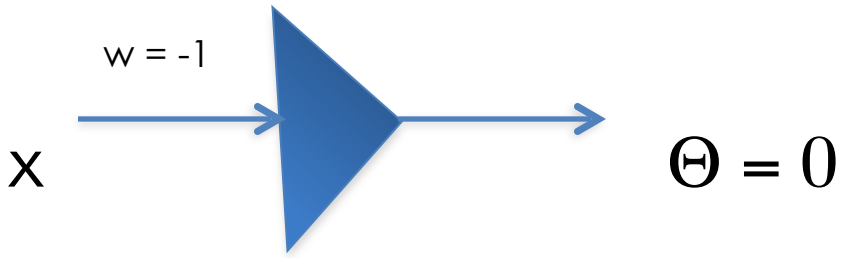
- Can put many neurons together.
- Connect them up like a combinational circuit.
- In fact, assemblies of the neurons are capable of universal computation
  - Can perform any computation that a normal computer can. (To show this, it is enough to show that a neuron can simulate OR, AND and NOT gate.) This is a straight-forward exercise.

AND OR and NOT gates

- AND gate:



- OR gate: same weights as AND, $\Theta = 1$
- NOT gate:



ALVINN (Carnegie Mellon Univ)

Automated driving at 70 mph on a public highway

Camera
image

30 outputs
for steering

4 hidden
units

30x32 pixels
as inputs

30x32 weights
into one out of
four hidden
unit

Another Example

NETtalk – Program that learns to pronounce English text. (Sejnowski and Rosenberg 1987).

- A difficult task using conventional programming models.
- Rule-based approaches are too complex since pronunciations are very irregular.
- NETtalk takes as input a sentence and produces a sequence of phonemes and an associated stress for each letter.

-https://www.youtube.com/watch?v=gakJlr3GecE

Image classification

Jeff Hinton's neural network

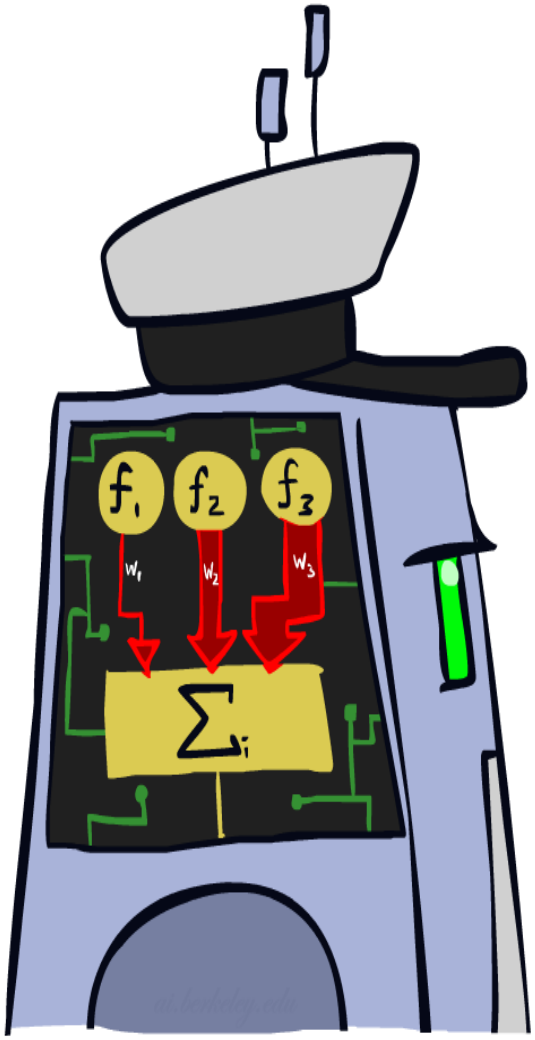
Hand-writing recognition

- MNIST dataset (handwriting samples of digits 0 to 9)
- Yann Le Cunn's neural network performs the classification of digits in real-time.

https://www.youtube.com/watch?v=yxuRnBEczUU

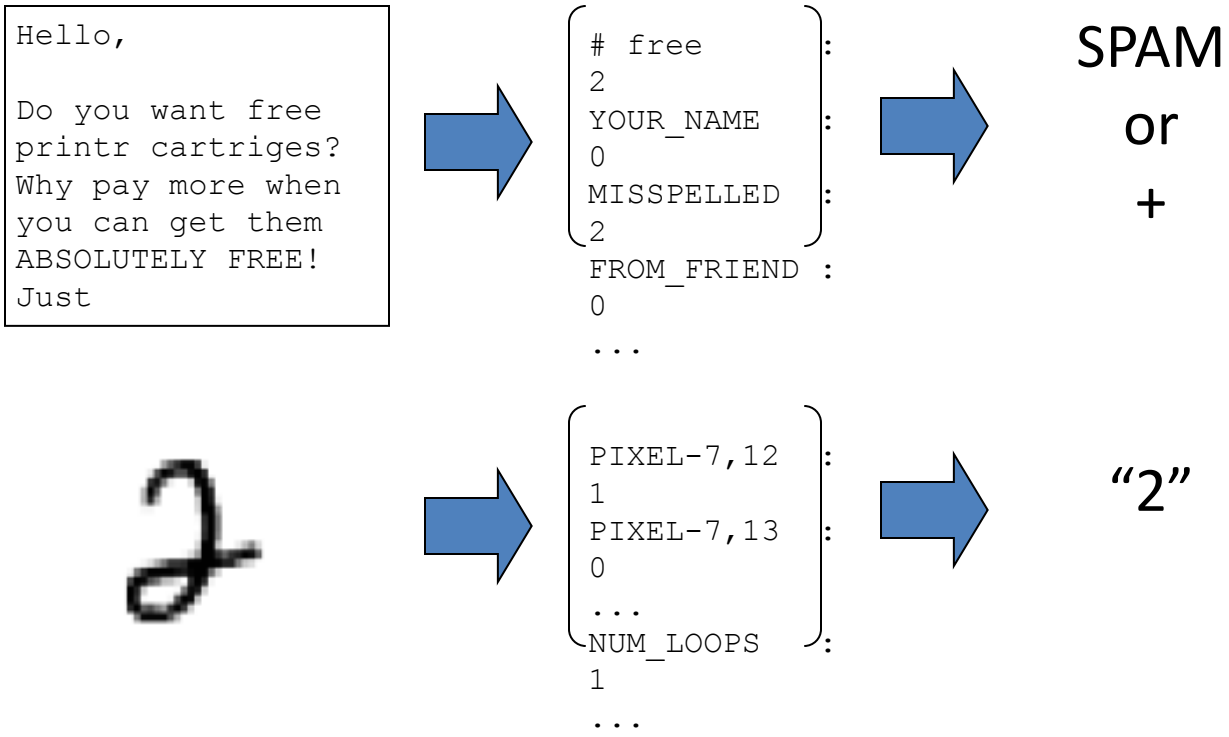
His web site at NYU has (had?) an animation of a neural network performing digit classification.

Linear Classifier – a single perceptron



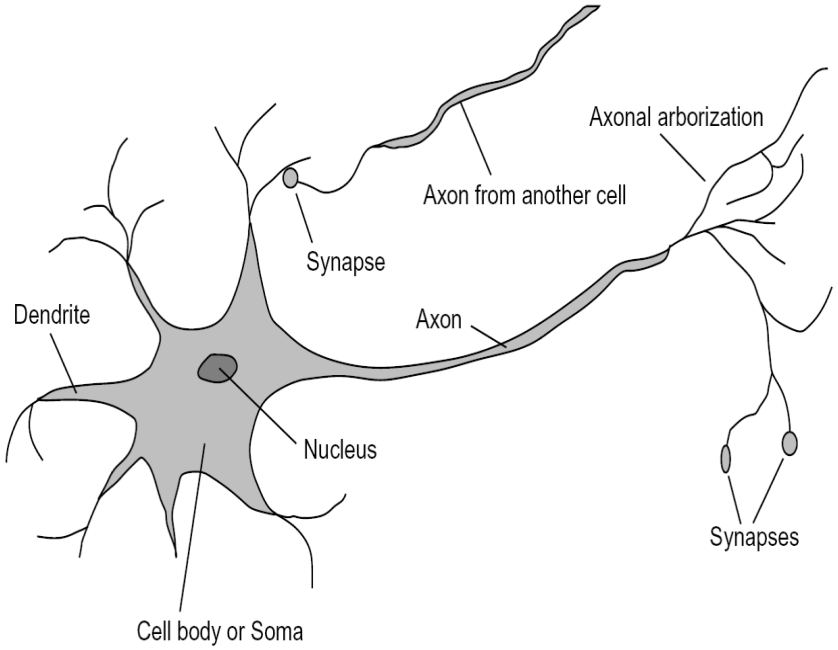
Feature Vectors

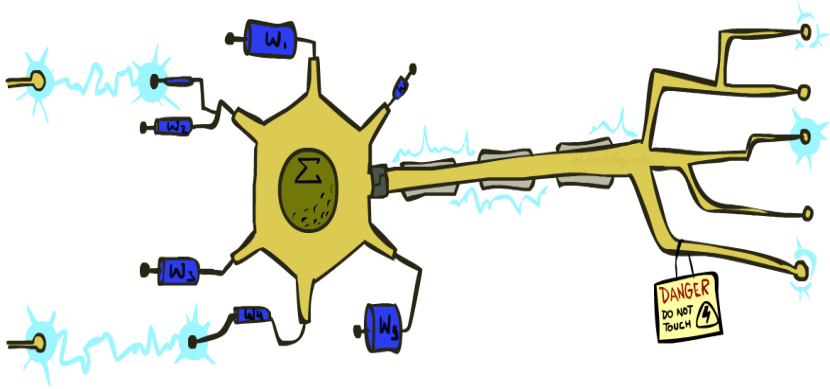
x $f(x)$ y



Some (Simplified) Biology

- Very loose inspiration: human neurons

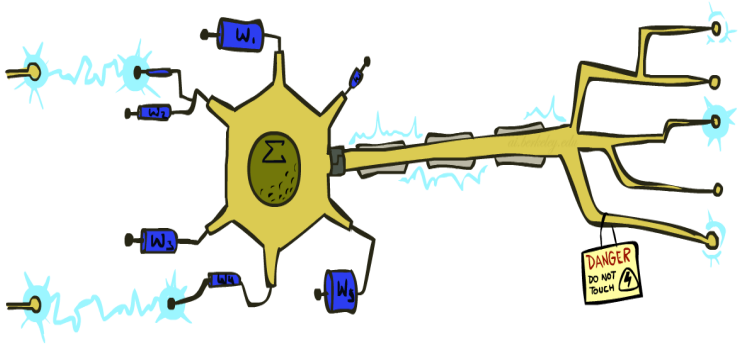




- A video clip from Khan academy: https://www.youtube.com/watch?v=1h4kW8RX-6k

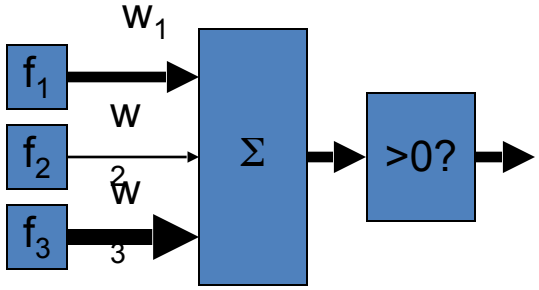
Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

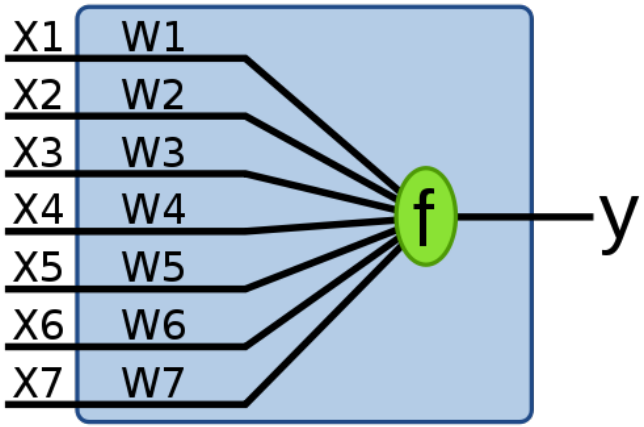
- If the activation is:
  - Positive, output +1
  - Negative, output -1



The Perceptron

Definition:

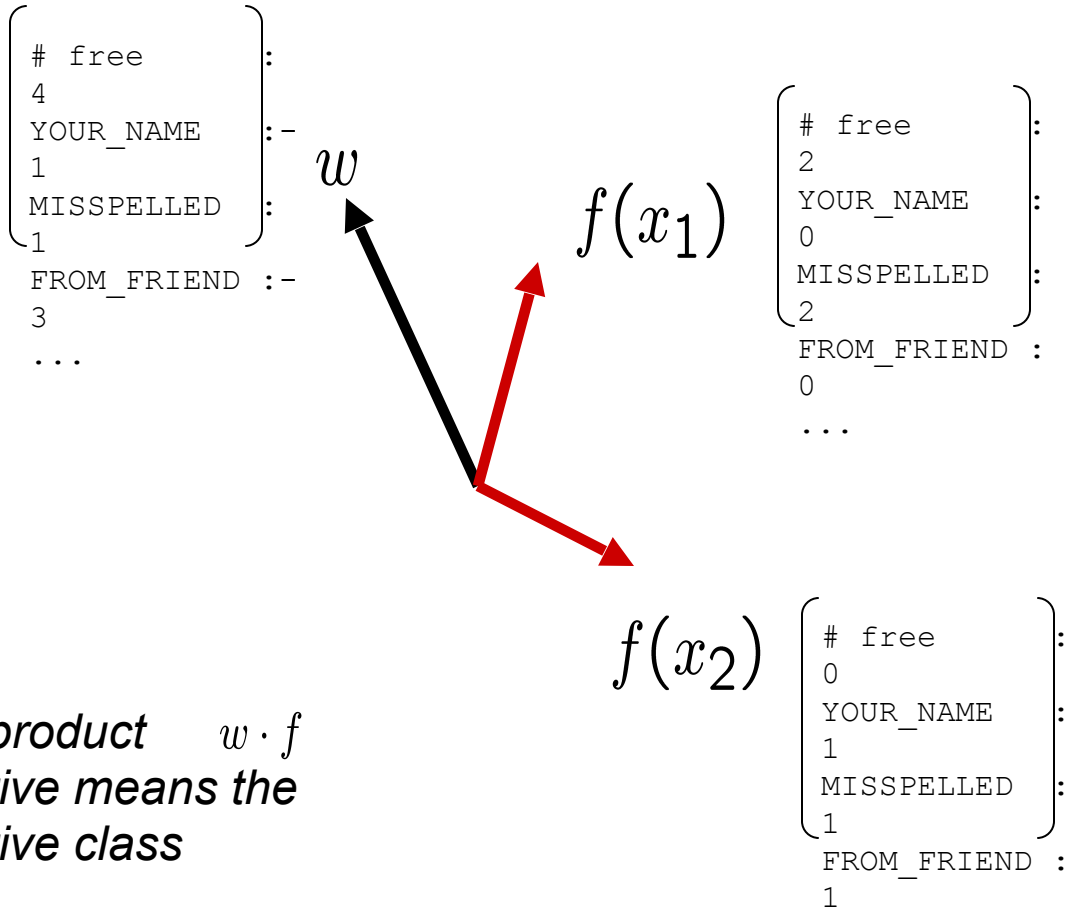
The **perceptron** is a binary classifier which maps its input x (a real-valued vector) to an output value f(x) (a single binary value) across the matrix:



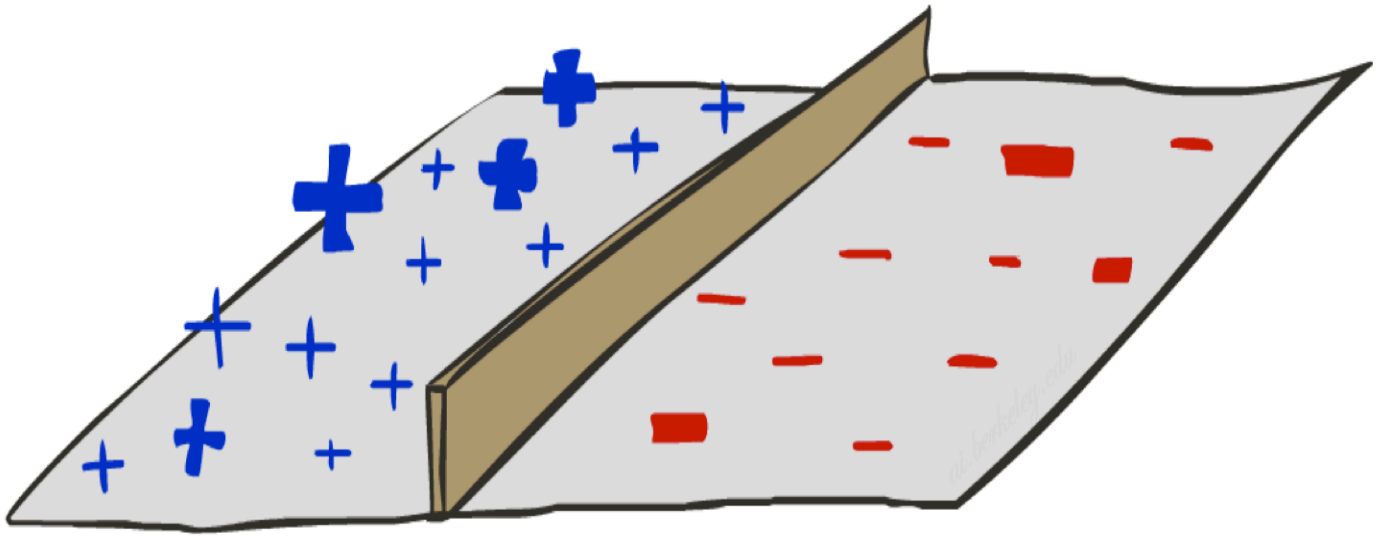
In order to not explicitly write b, we extend the input vector x by one more dimension that is always set to -1, e.g., $\mathbf{x} = (x_0, x_1, \ldots, x_7)$ with $x_0 = -1$, and extend the weight vector to $\mathbf{w} = (w_0, w_1, \ldots, w_7)$. Then adjusting w_0 corresponds to adjusting b.

Weights

- Binary case: compare features to a weight vector
- Learning: figure out the weight vector from examples

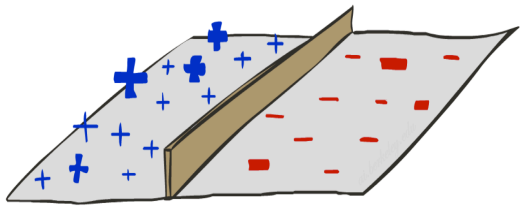


Decision Rules



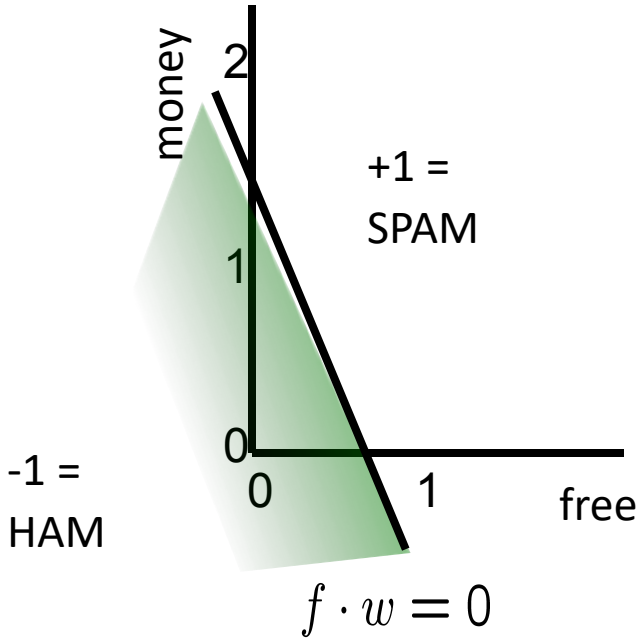
Binary Decision Rule

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to Y=+1
  - Other corresponds to Y= -1



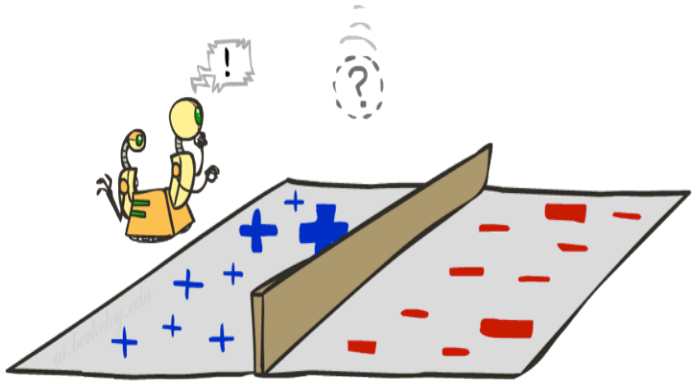
w

```
BIAS   :
-3
free   :
4
money  :
2
...
```

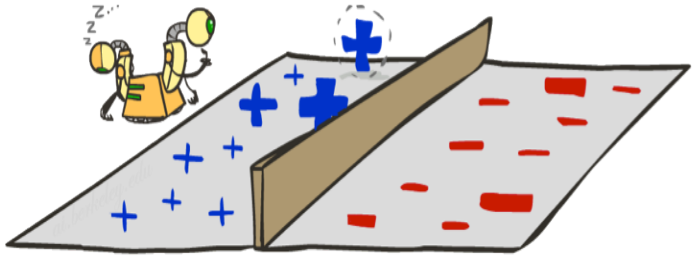


Learning: Binary Perceptron

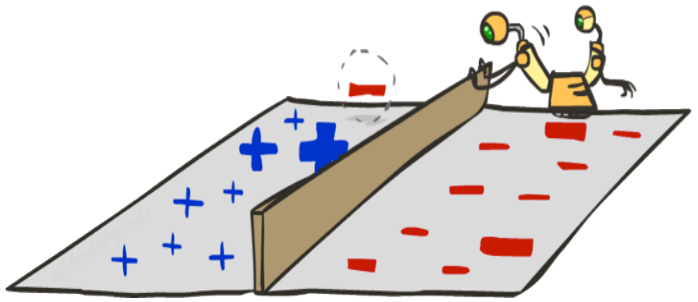
- Start with weights = 0
- For each training instance:
  - Classify with current weights



  - If correct (i.e., y=y*), no change!



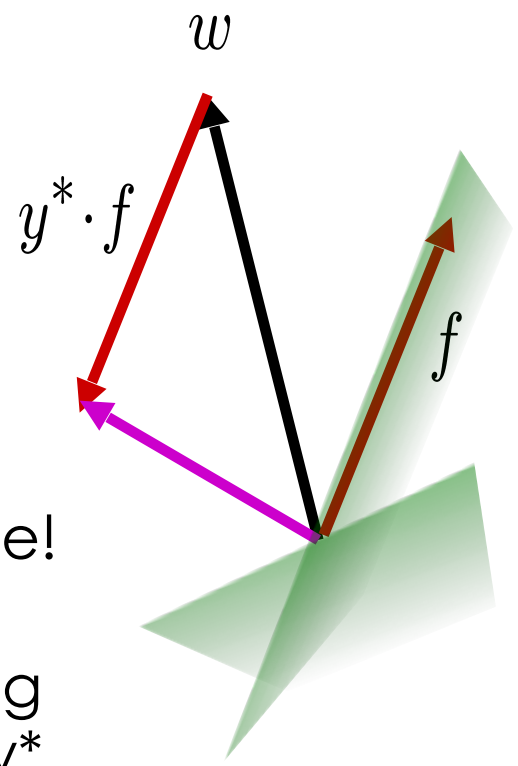
  - If wrong: adjust the weight vector



Learning: Binary Perceptron

- Start with weights = 0
- For each training instance:
  - Classify with current weights

$$y = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$



  - If correct (i.e., y=y*), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

$$w = w + y^* \cdot f$$

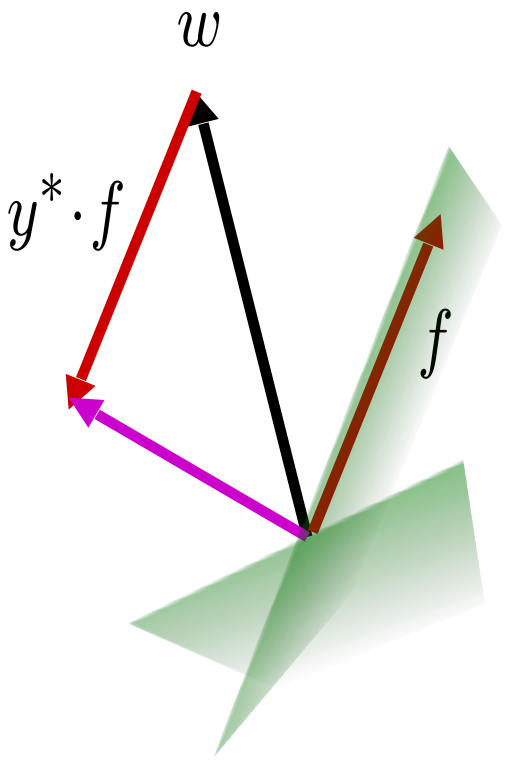
Learning: Binary Perceptron

- To avoid drastic change in w, use a quantity τ called learning rate:

$$y = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$

- Weight correction:

  $w = w + \tau (y^* - y) f$



Here y* is the actual label of the instance x. (Thus, the above expression will not make any update when y = y*

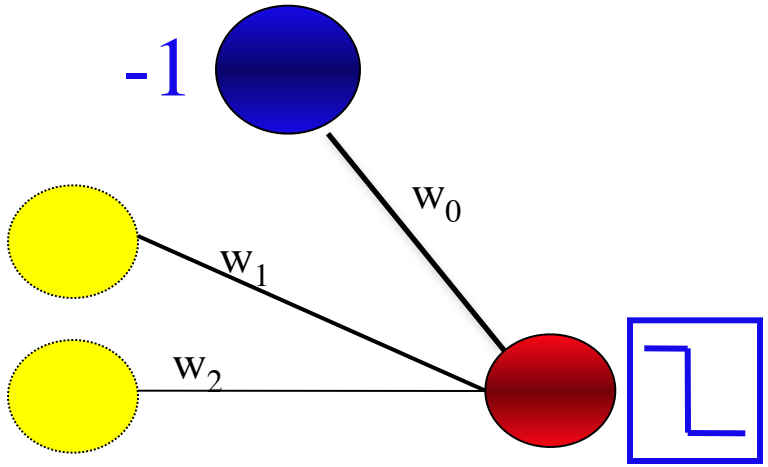
Pseudo-code taken from Ertel's book

PERCEPTRONLEARNING[M_+, M_-]
$\boldsymbol{w}$ = arbitrary vector of real numbers
Repeat
  For all $\boldsymbol{x} \in M_+$
    If $\boldsymbol{w}\,\boldsymbol{x} \leq 0$ **Then** $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{x}$
  For all $\boldsymbol{x} \in M_-$
    If $\boldsymbol{w}\,\boldsymbol{x} > 0$ **Then** $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{x}$
Until all $\boldsymbol{x} \in M_+ \cup M_-$ are correctly classified

1. This version goes through each data point in training set and performs updates if needed. This would work for small or medium size data.
2. A single iteration will not make the algorithm work correctly on all training data. So algorithm makes many passes improving the weights.
3. This version may potentially loop forever.

Example 1: The Logical OR

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



Initial values: $w_0(0)=-0.05$, $w_1(0)=-0.02$, $w_2(0)=0.02$, and $\eta=0.25$

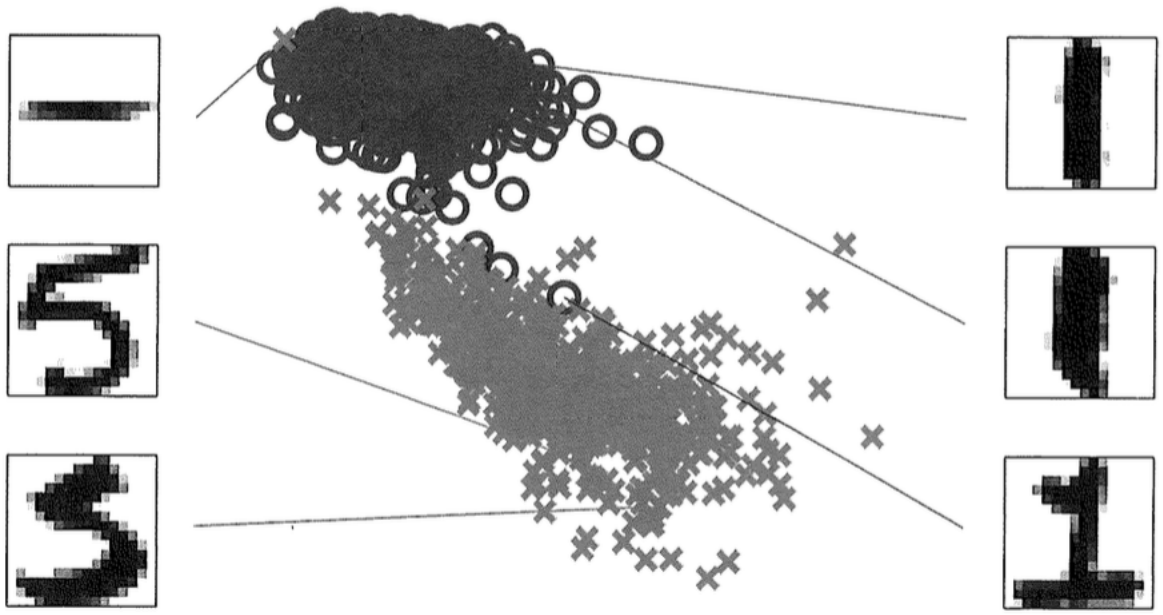
Take first row of our training table:

$$y_1 = \text{sign}(-0.05\times-1 + -0.02\times0 + 0.02\times0) = 1$$

$$w_0(1) = -0.05 + 0.25\times(0-1)\times-1=0.2$$
$$w_1(1) = -0.02 + 0.25\times(0-1)\times0=-0.02$$
$$w_2(1) = 0.02 + 0.25\times(0-1)\times0=0.02$$

We continue with the new weights and the second row, and so on.
We make several passes over the training data until no errors or some

Example 2: Hand-written characters 1 and 5

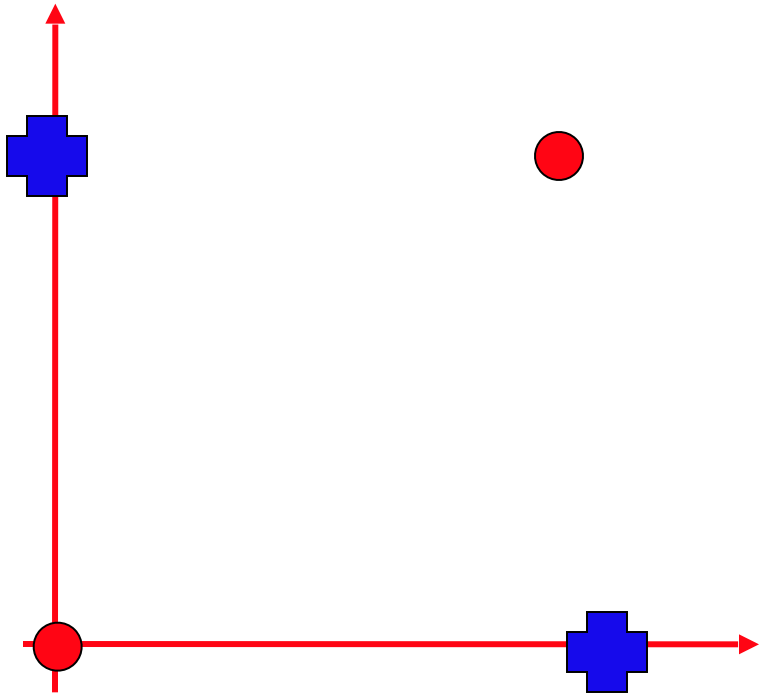


Two features used: symmetry and intensity. Just these two features almost linearly separate the two classes.

Limitations of the Perceptron

No perceptron exists for XOR function.

The Exclusive Or (XOR) function.

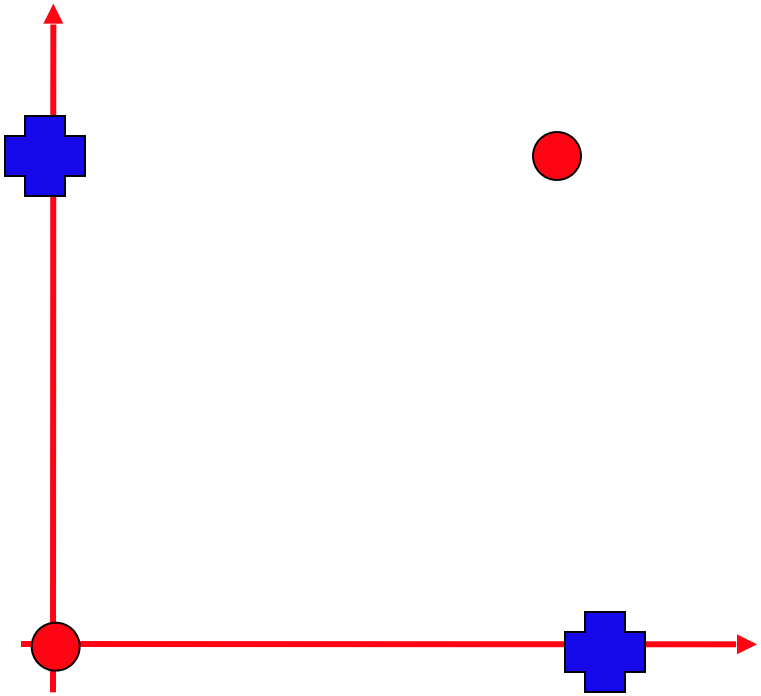


A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Limitations of the Perceptron

No perceptron exists for XOR function.

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0



Assume w1, w2 and Θ exist. Then, we get:

$w1*0 + w2*0 < \Theta \Rightarrow 0 < \Theta$

$w1*0 + w2*1 \geq \Theta \Rightarrow \Theta \leq w1$

$w1*1 + w2*0 \geq \Theta \Rightarrow \Theta \leq w2$

$w1*1 + w2*1 < \Theta \Rightarrow w1 + w2 < \Theta$

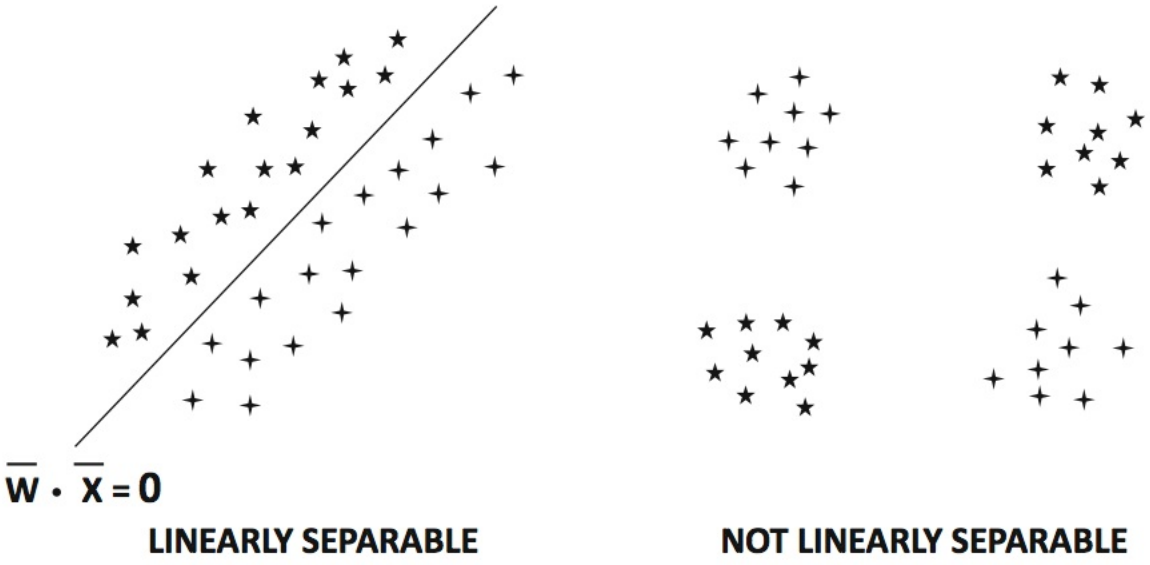
(2), (3) and (1) imply $w1 + w2 \geq 2\Theta > \Theta$ which contradicts (4)

Perceptron

- Separable Case

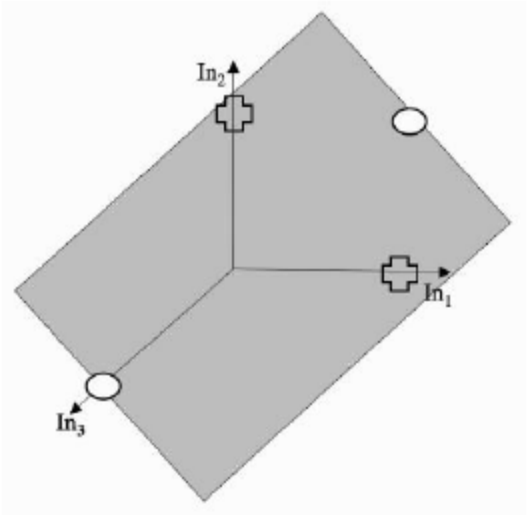
Linearly separable data

All training data x with label 1 lie on one side of the plane and those x with label 0 lie on the other side.



Adding a new dimension can make XOR linearly separable

In_1	In_2	In_3	Output
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	1




```
>>> inputs = np.array([[0,0,1],[0,1,0],[1,0,0],[1,1,0]])
>>> pcn.pcntrain(inputs,targets,0.25,15)
Iteration:  14
[[-0.27757663]
 [-0.21083089]
 [-0.23124407]
 [-0.53808657]]
Final outputs are:
[[0]
 [1]
 [1]
 [0]]
```

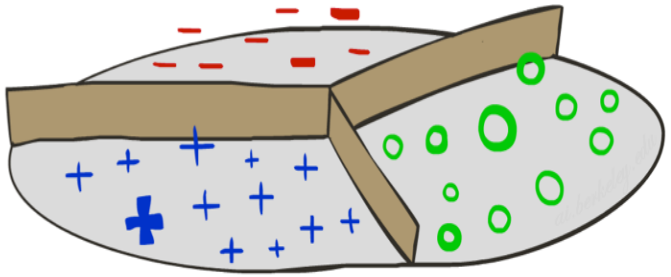
Multiclass Decision Rule

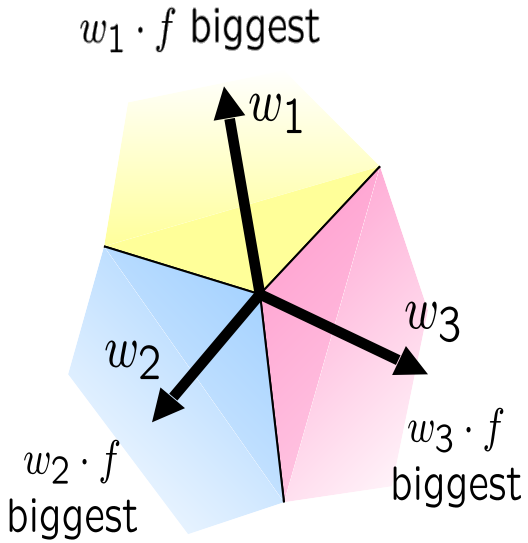
- If we have multiple classes:
  - A weight vector for each class: w_y
  - Score (activation) of a class y:

$$w_y \cdot f(x)$$

  - Prediction: highest score wins

$$y = \arg\max_y \; w_y \cdot f(x)$$





Binary = multiclass where the negative class has weight zero

Learning: Multiclass Perceptron

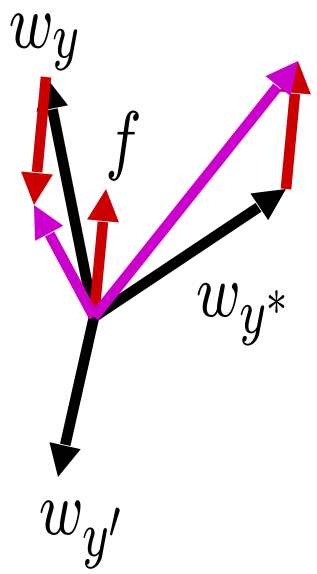
- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

$$y = \arg\max_y \; w_y \cdot f(x)$$

- If correct, no change!
- If wrong: lower score of wrong answer:

- raise s$w_y = w_y - f(x)$nswer

$$w_{y^*} = w_{y^*} + f(x)$$



Multiclass linear separable data

Iris setosa

Iris versicolor

Iris virginica

Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:

-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

Multiclass perceptron algorithm

Input: training data $D = \{(x_i, y_i)\}$, $i = 0, 1, 2, \ldots, n - 1$ where x_i is in R^d, and y_i is in $L = \{1, 2, \ldots, k\}$, the class labels. ($x_0 = -1$ is the bias.)

Output: Weight vectors, $\{w^{(1)}, \ldots, w^{(k)}\}$ for each class j.
Initialize each $\{ w^{(j)} \}$ with small real values.

- predict the class label of x_i as

  argmax $\{w^{(k)} \cdot x_k\}$ over all k

- Update:
  - for correct label y_i: $w^{(i)} = w^{(i)} + \eta x$
  - for predicted label y_j: $w^{(j)} = w^{(j)} - \eta x$

Example – MNIST data set

Hand-written character recognition:

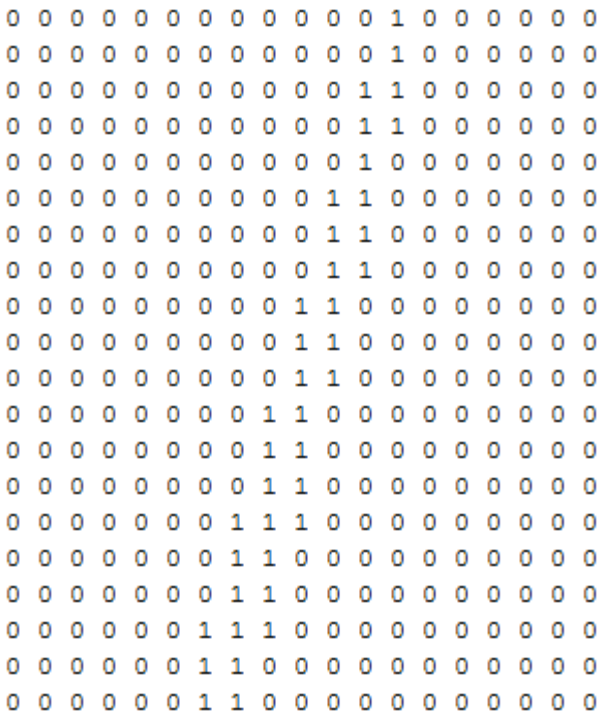
There are 10 classes – one for each letter.

Some of the features could be:

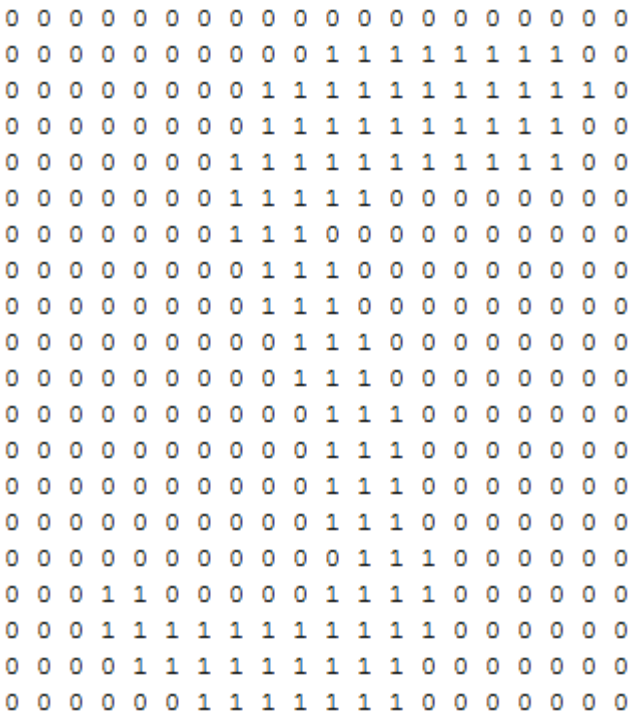
1) Density (average gray scale value)
2) Degree of symmetry (8 and 1 are more symmetrical then 4 or 5.)
3) Number of loops (8 has two, 7 has 0 etc.)
4) Average curvature (7 and 4 have low curvature, 8 has a high curvature etc.)

Example – MNIST data set

Black-white image of digit 1



Black-white image of digit 5



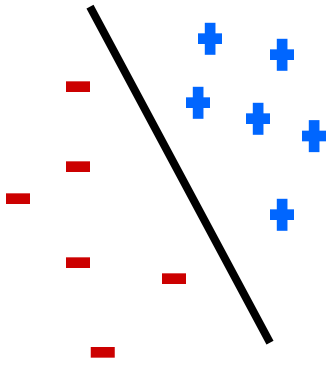
Sample images – MNIST dataset

Properties of Perceptrons

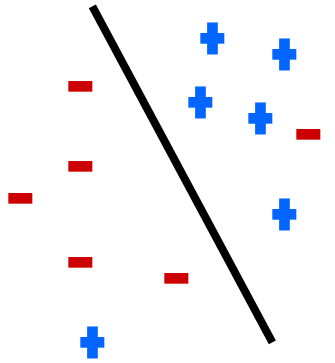
- Linear Separability: true if some parameters get the training set perfectly correct
- Convergence: if the training data is linearly separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

$$\text{mistakes} < \frac{k}{\delta^2}$$

Separable



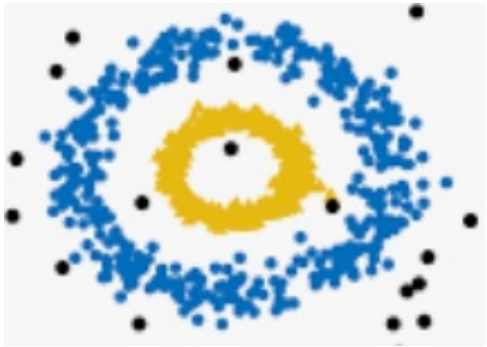
Non-Separable



Examples: Perceptron

- Non-Separable Case

Transformation can convert nonseparable to separable case



Ignore black dots, consider the other two colors – blue and yellow as labels with (x, y) values as features (Assume origin is at the center of the two circles).

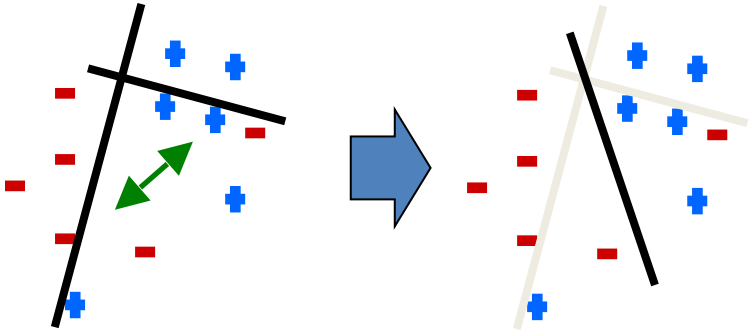
Clearly the data is not linearly separable, but if we add a new feature $x^2 + y^2$, then it becomes linearly separable.

This idea is used in support vector

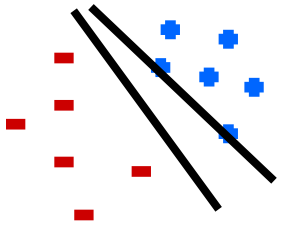
Perceptron algorithm – variations and improvements

Problems with the Perceptron

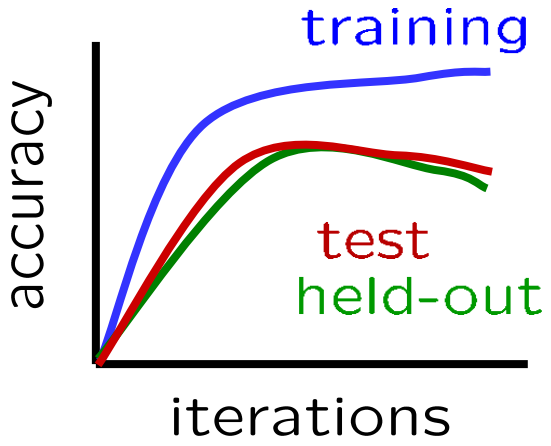
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



- Mediocre generalization: finds a "barely" separating solution

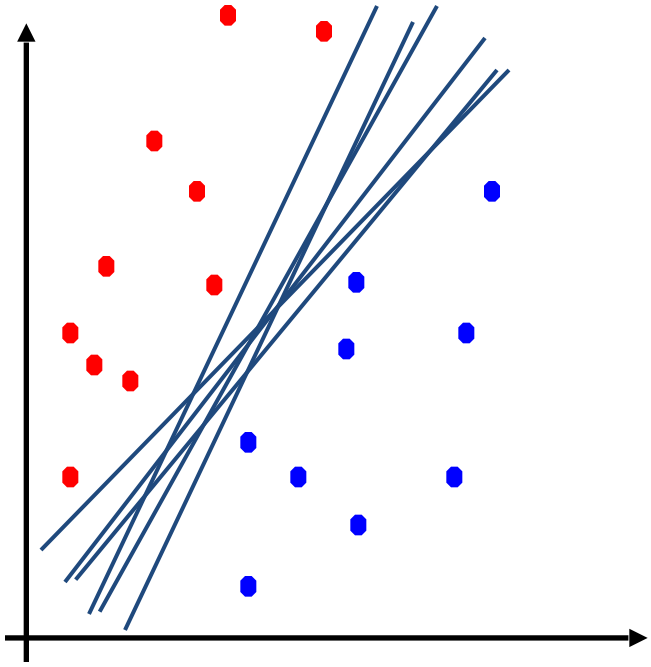


- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



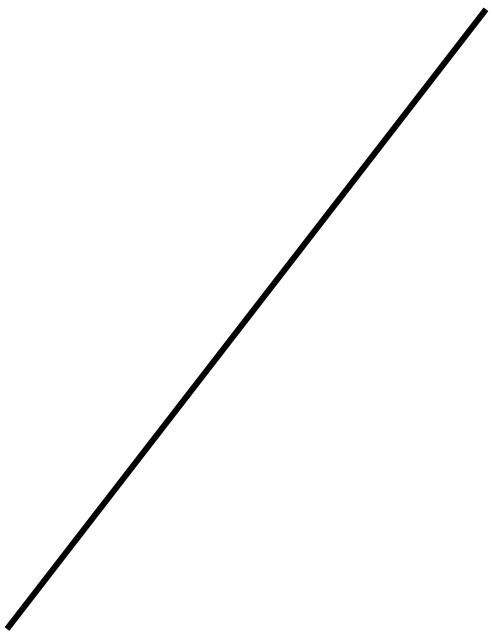
Linear Separators

- Which of these linear separators is optimal?

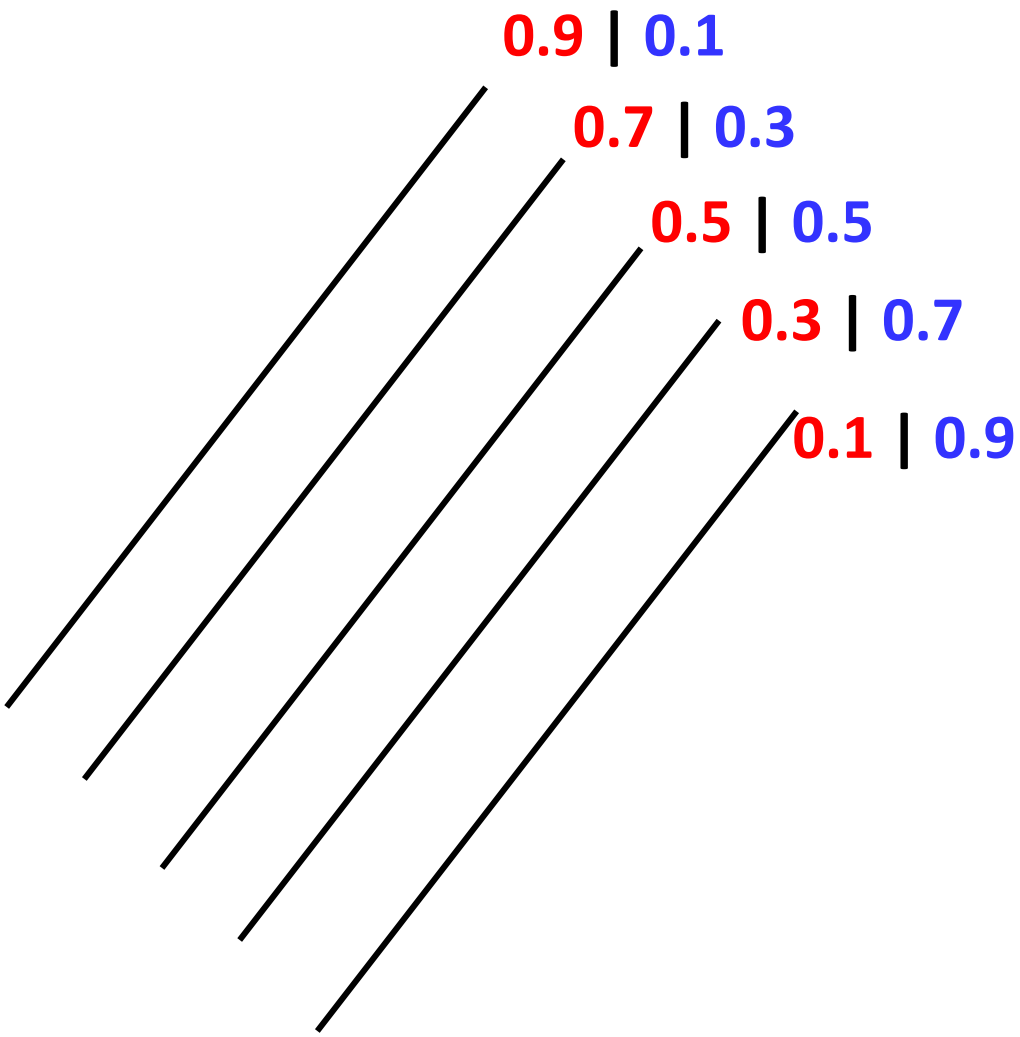


Non-Separable Case: Deterministic Decision

Even the best linear boundary makes at least one mistake



Non-Separable Case: Probabilistic Decision

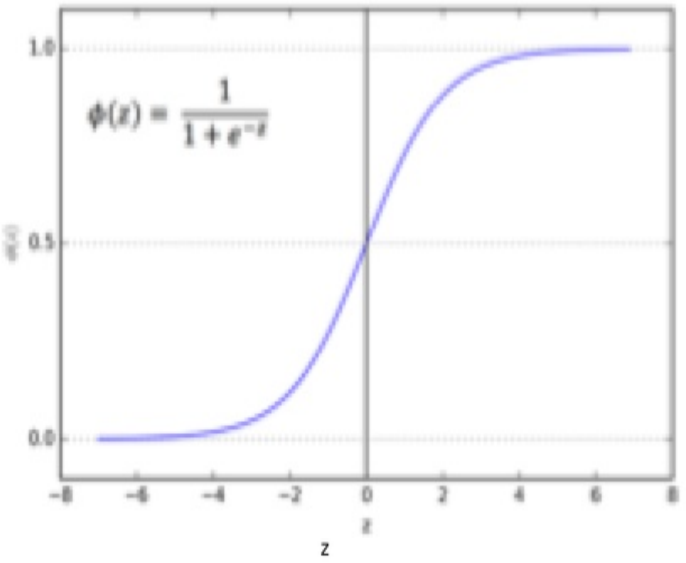


How to get probabilistic decisions?

- Perceptron scoring: $z = w \cdot f(x)$
- If $z = w \cdot f(x)$ very positive → want probability going to 1
- If $z = w \cdot f(x)$ very negative → want probability going to 0

- Sigmoid function

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



Learning the weights as an optimization problem

For a given weight vector w, we can associate the 'gain' on a specific training data point (x(i), y(i)) as:

$$P(y^{(i)} = +1 | x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

$$P(y^{(i)} = -1 | x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

The 'gain' is high when the classification is correct and low when it is wrong. (ideally this value is 1 for each data point – when a + point has prob = 1, and negative point has prob = 0).

Best w?

- Maximum likelihood estimation:

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

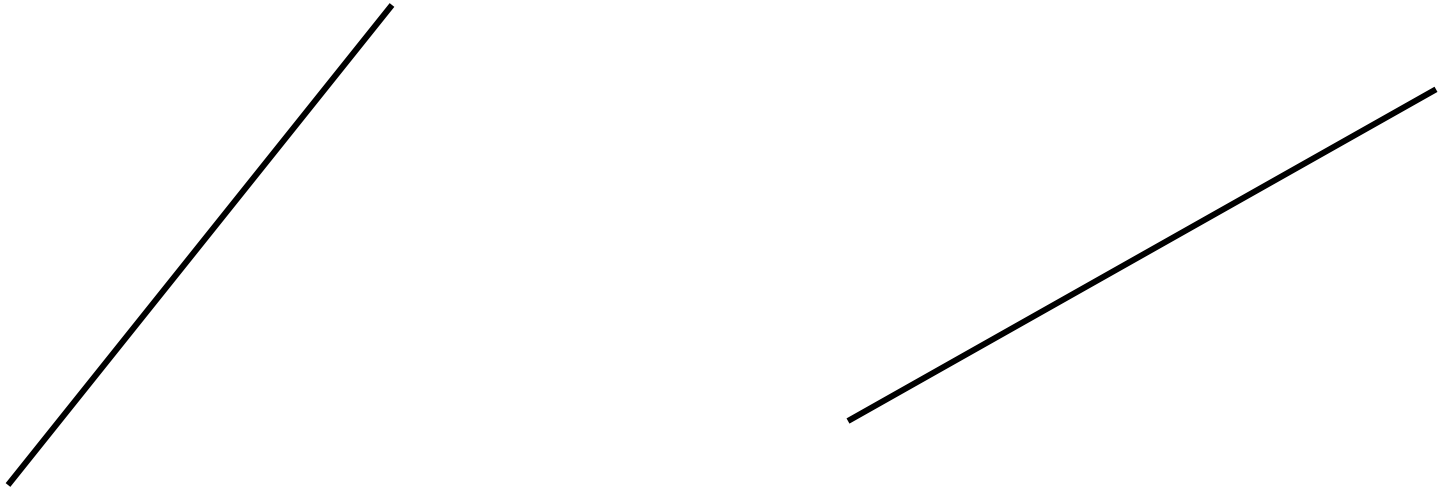
with:

$$P(y^{(i)} = +1|x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

$$P(y^{(i)} = -1|x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

known as logistic regression

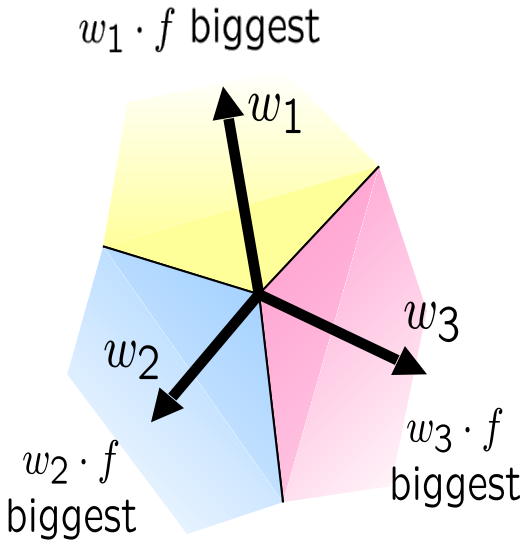
Separable Case: Deterministic Decision – Many Options



Multiclass Logistic Regression

- Recall Perceptron:
  - A weight vector for each class: w_y
  - Score (activation) of a class y: $w_y \cdot f(x)$
  - Prediction highest score wins

$$y = \arg\max_y \; w_y \cdot f(x)$$



- How to make the scores into probabilities?

$$z_1, z_2, z_3 \rightarrow \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

original activations

softmax activations

Best w?

- Maximum likelihood estimation:

$$\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)};w)$$

with:

$$P(y^{(i)}|x^{(i)};w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

= multi-Class Logistic Regression

Summary

- Perceptron and regression optimize the same target function
- In both cases we compute the gradient (vector of partial derivatives)
- In the case of regression, we set the gradient to zero and solve for vector w. As the solution we have a closed formula for w such that the target function obtains the global minimum.
- In the case of perceptron, we iteratively go in the direction of the minimum by going in the direction of minus the gradient. We do this incrementally making small steps for each data point.

Summary

- Perceptron is a single neuron that attempts to build a weight vector w such that $w.x \geq 0$ ($w.x < 0$) for class with output $y = 1$ (class with output $y = 0$).
- The weight vector is updated for each input x for which the w.x gives a wrong conclusion of its class label. The process is iterated many times over training data until some stopping condition is met.
- If the data is linearly separable, then the above algorithm will terminate with all training data correctly classified.
- The algorithm has a natural extension for multiclass data.
- Many variations of algorithm: t the learning rate, stopping conditions, initialization of weight vector etc.