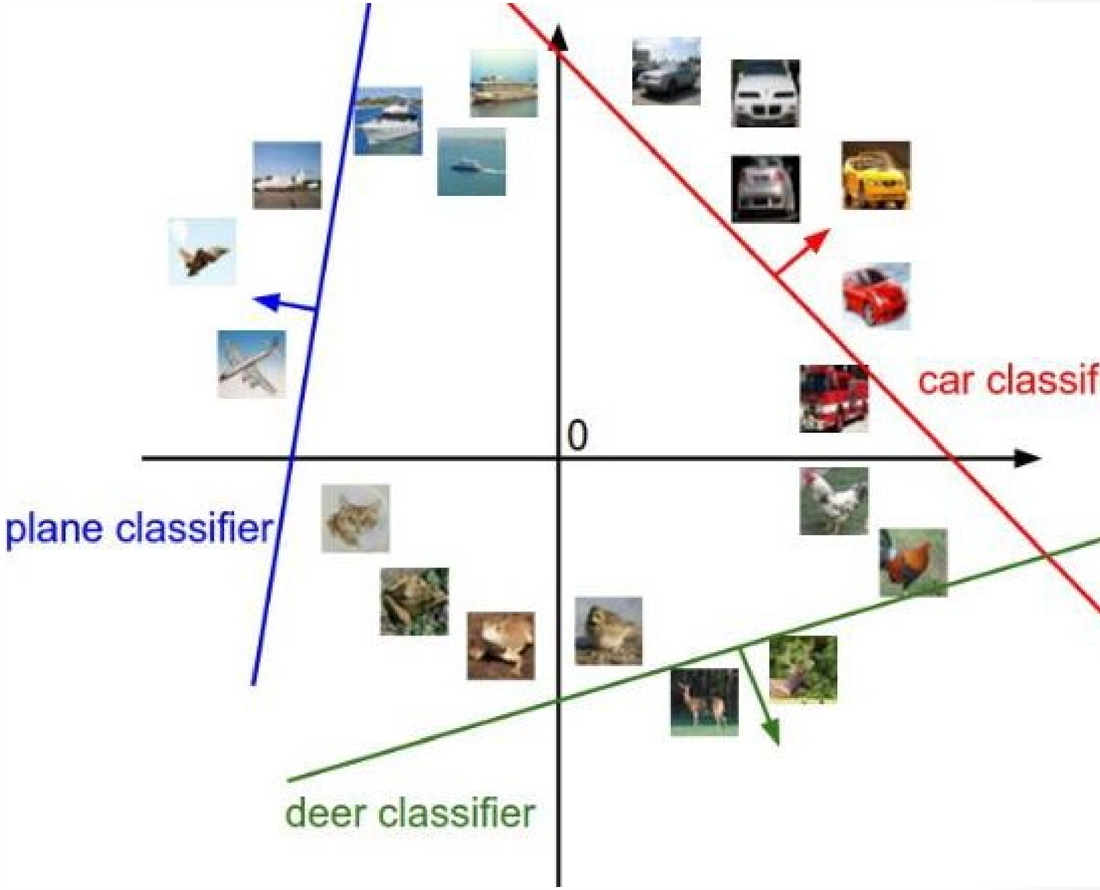
LINEAR CLASSIFIER
LEARNING
PERCEPTRON

Chih-Chung Hsu (許志仲)

Assistant Professor
ACVLab, Institute of Data Science
National Cheng Kung University



Outline

- Introduction to Machine Learning
- Supervised learning
  - Nearest neighbor classifier
  - Decision tree
  - Linear classifier
  - Support vector machine
  - Bayer classifier
    - Two-dim data
    - Multivariant data
  - Ensemble and boosting
- Feature selection
  - PCA and LDA
- Unsupervised learning
  - K-means
  - EM-algorithm
  - Affinity-propagation

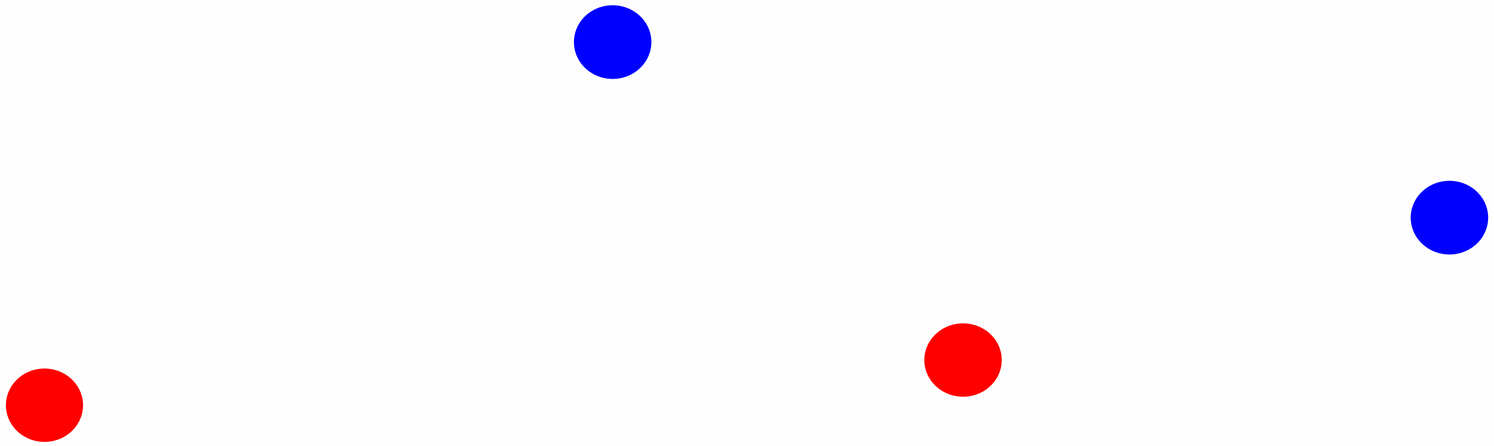
Outline

Machine learning models

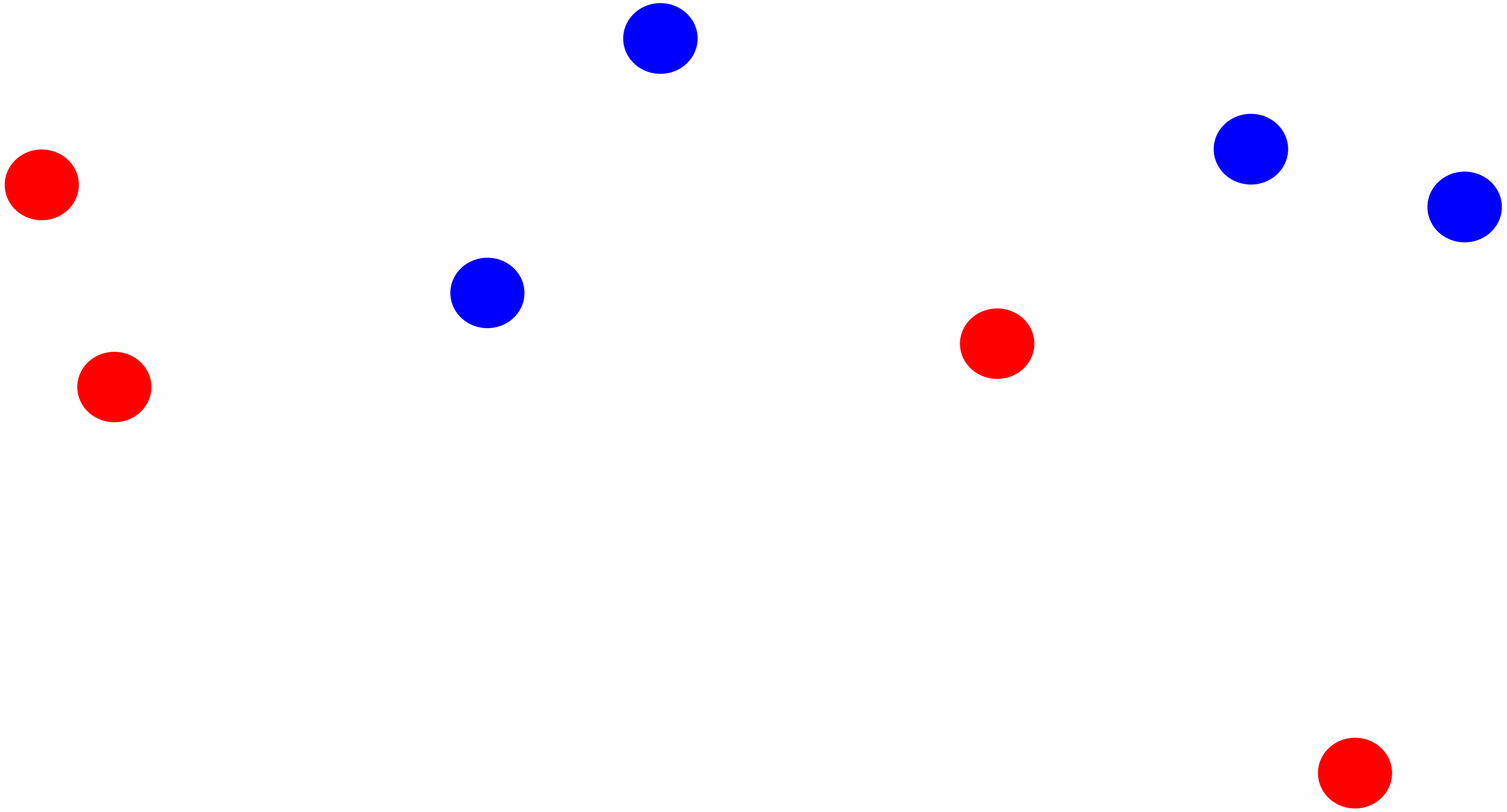
- Some machine learning approaches make strong assumptions about the data
  - If the assumptions are true this can often lead to better performance
  - If the assumptions aren’t true, they can fail miserably

- Other approaches don’t make many assumptions about the data
  - This can allow us to learn from more varied data
  - But, they are more prone to overfitting
  - and generally require more training data

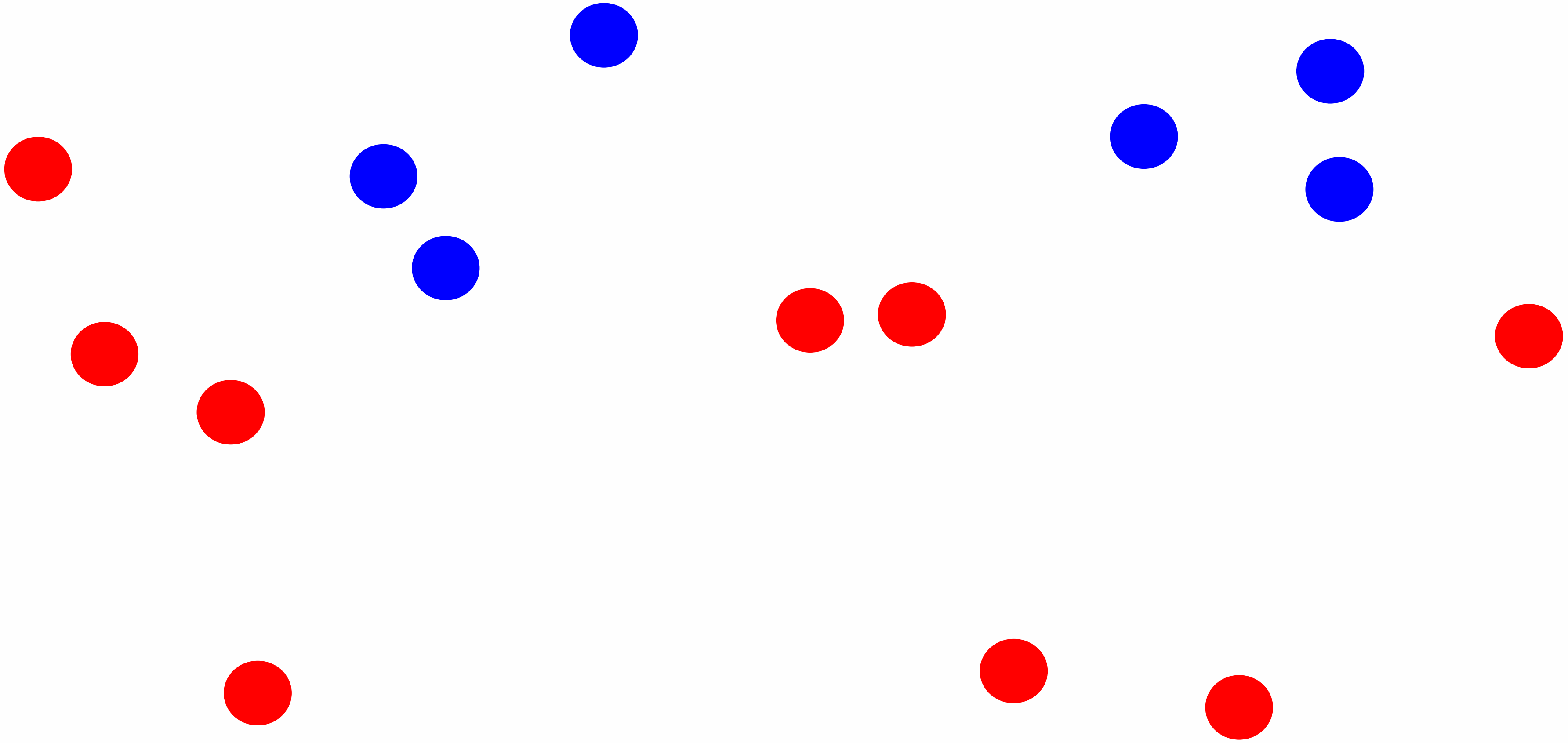
What is the data generating distribution?



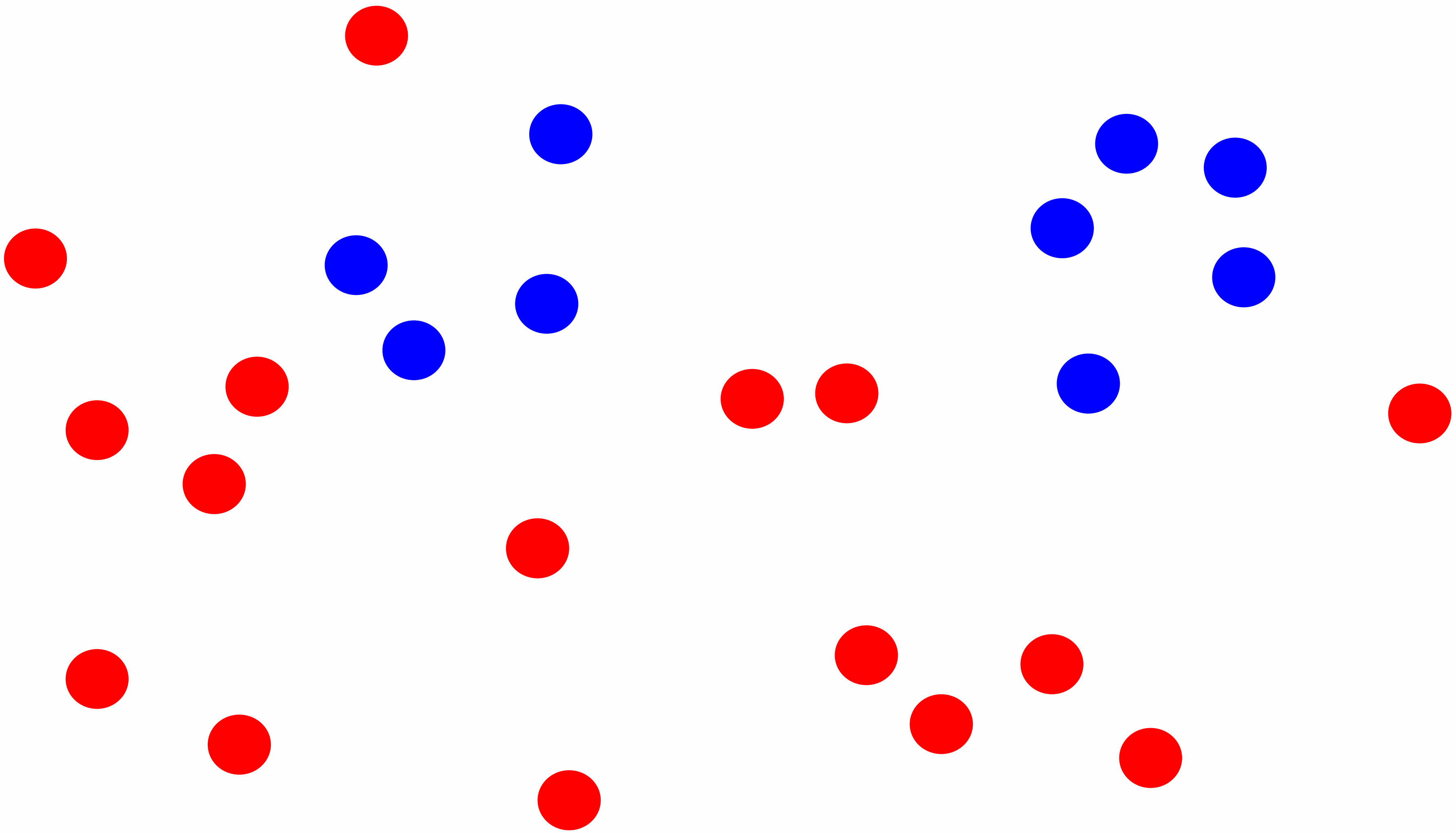
What is the data generating distribution?



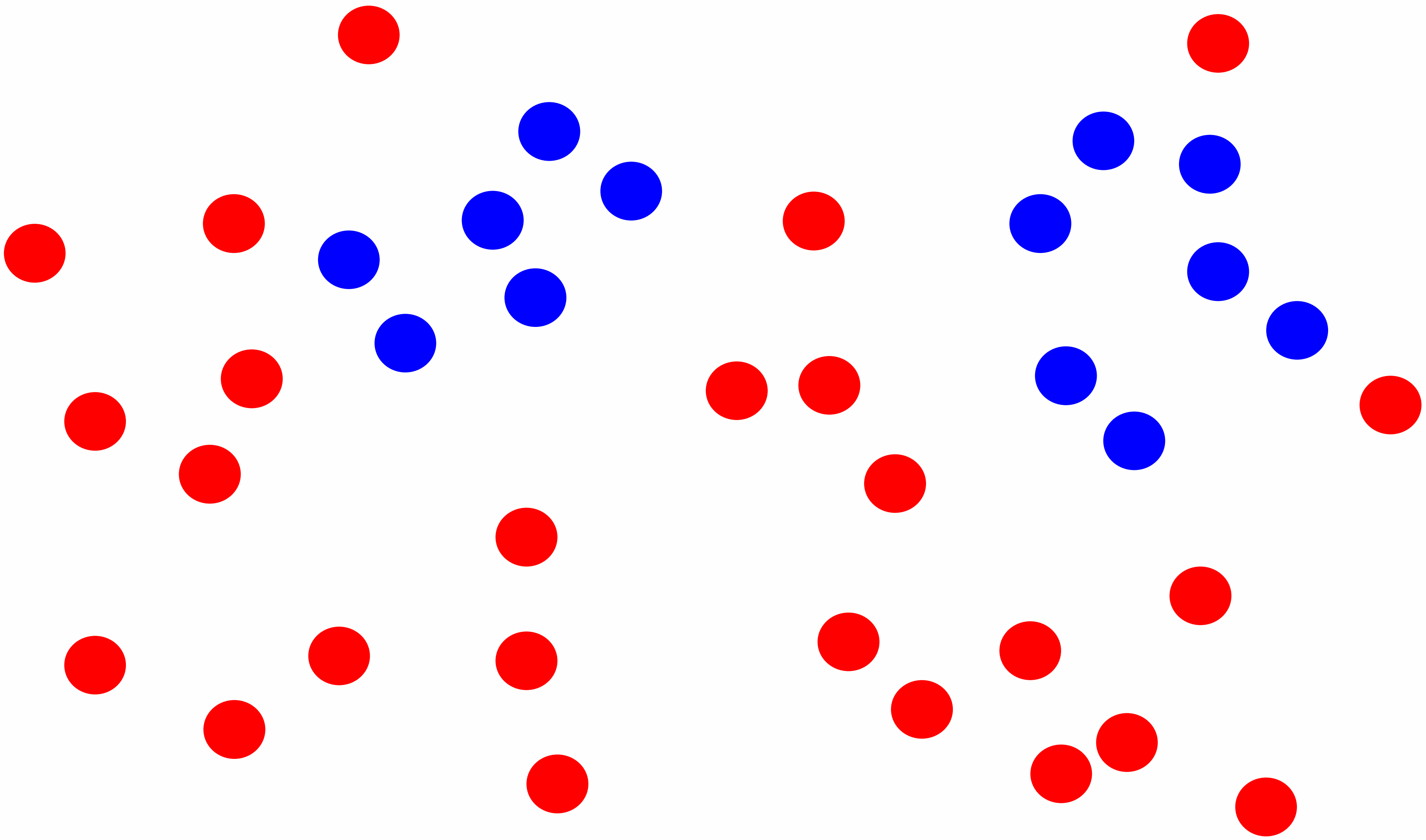
What is the data generating distribution?



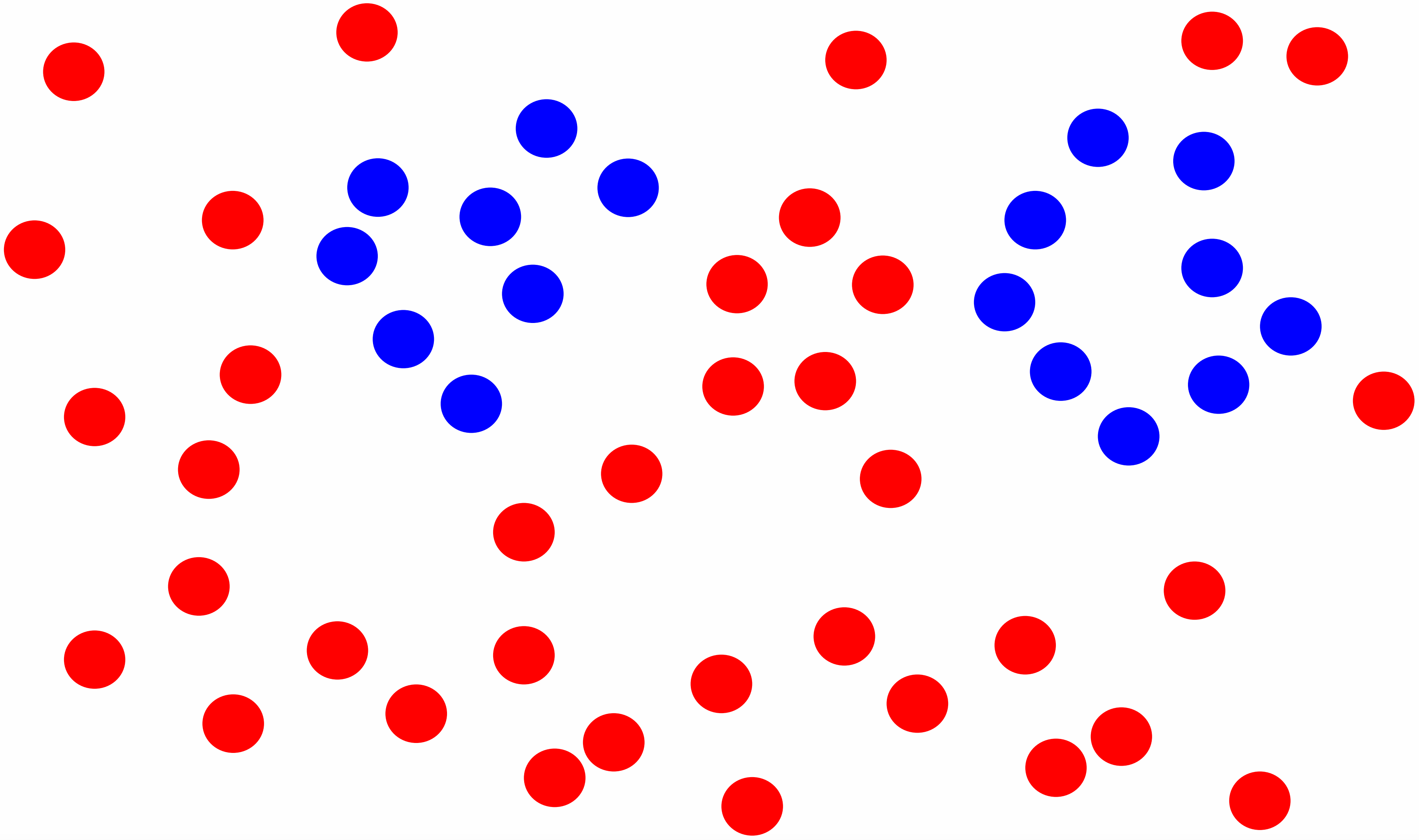
What is the data generating distribution?



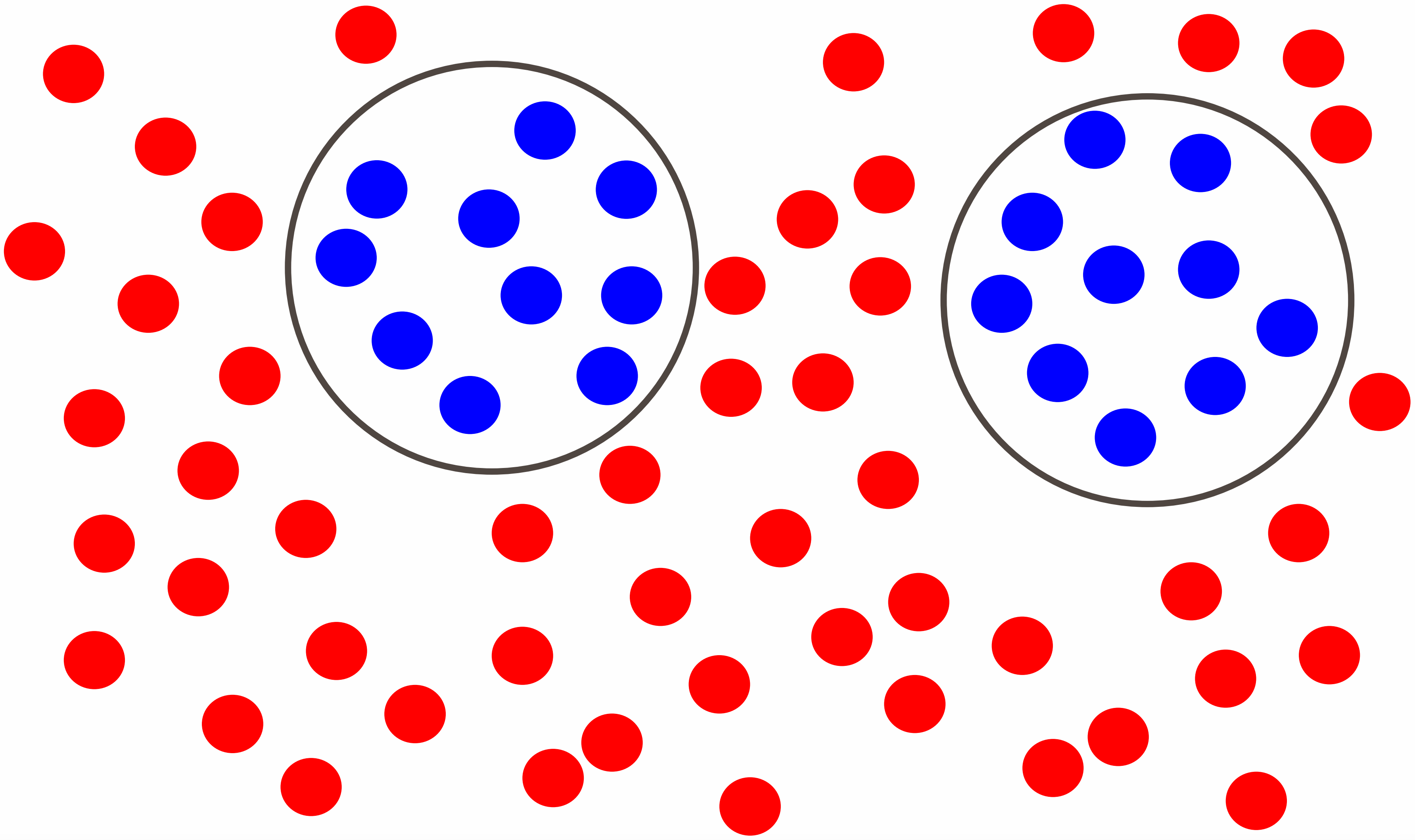
What is the data generating distribution?



What is the data generating distribution?



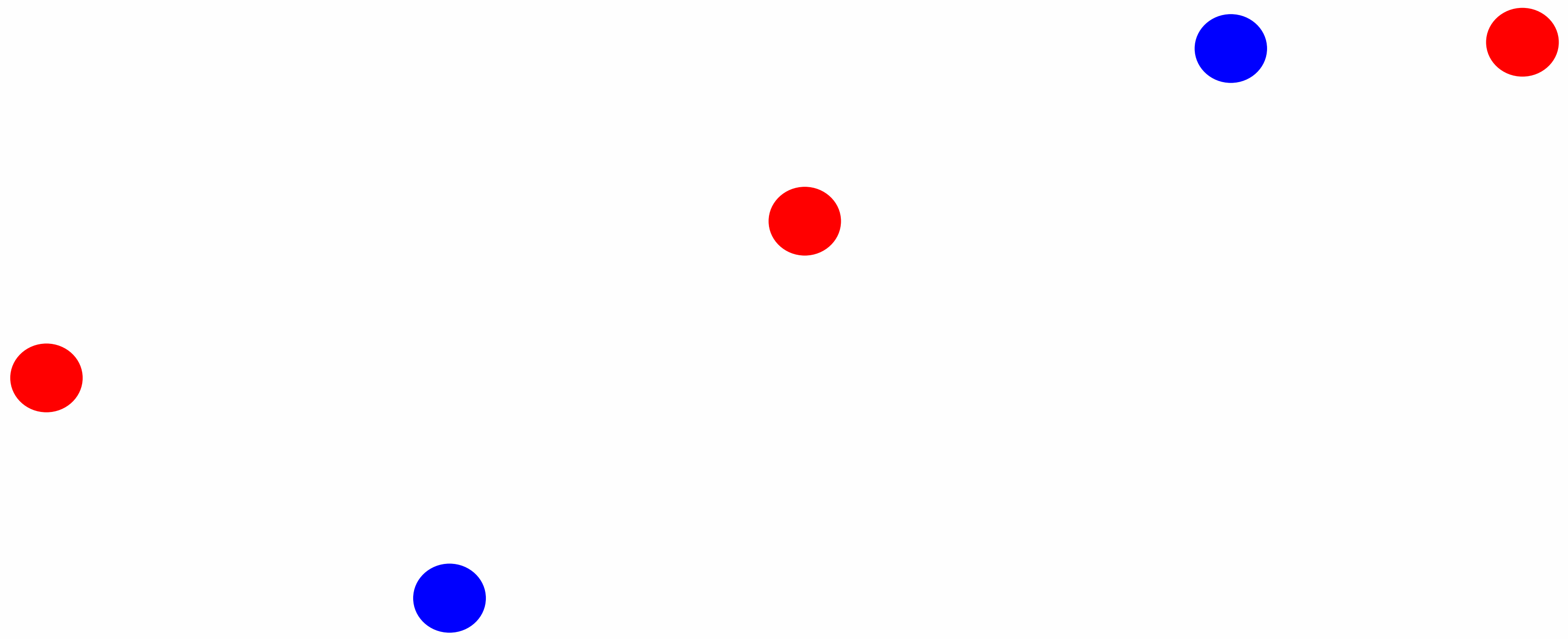
Actual model



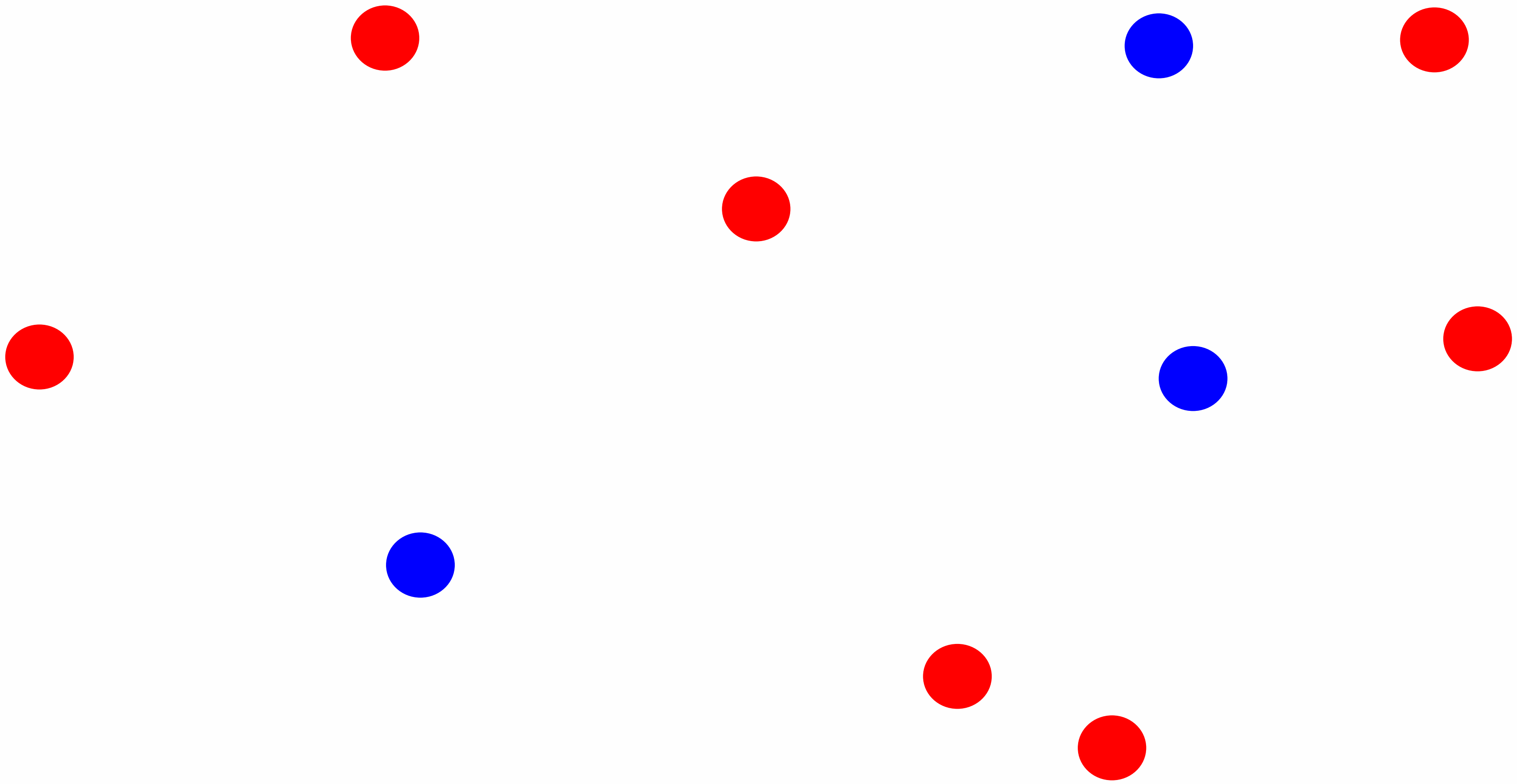
Model assumptions

- If you don’t have strong assumptions about the model, it can take you a longer to learn
- Assume now that our model of the blue class is two circles

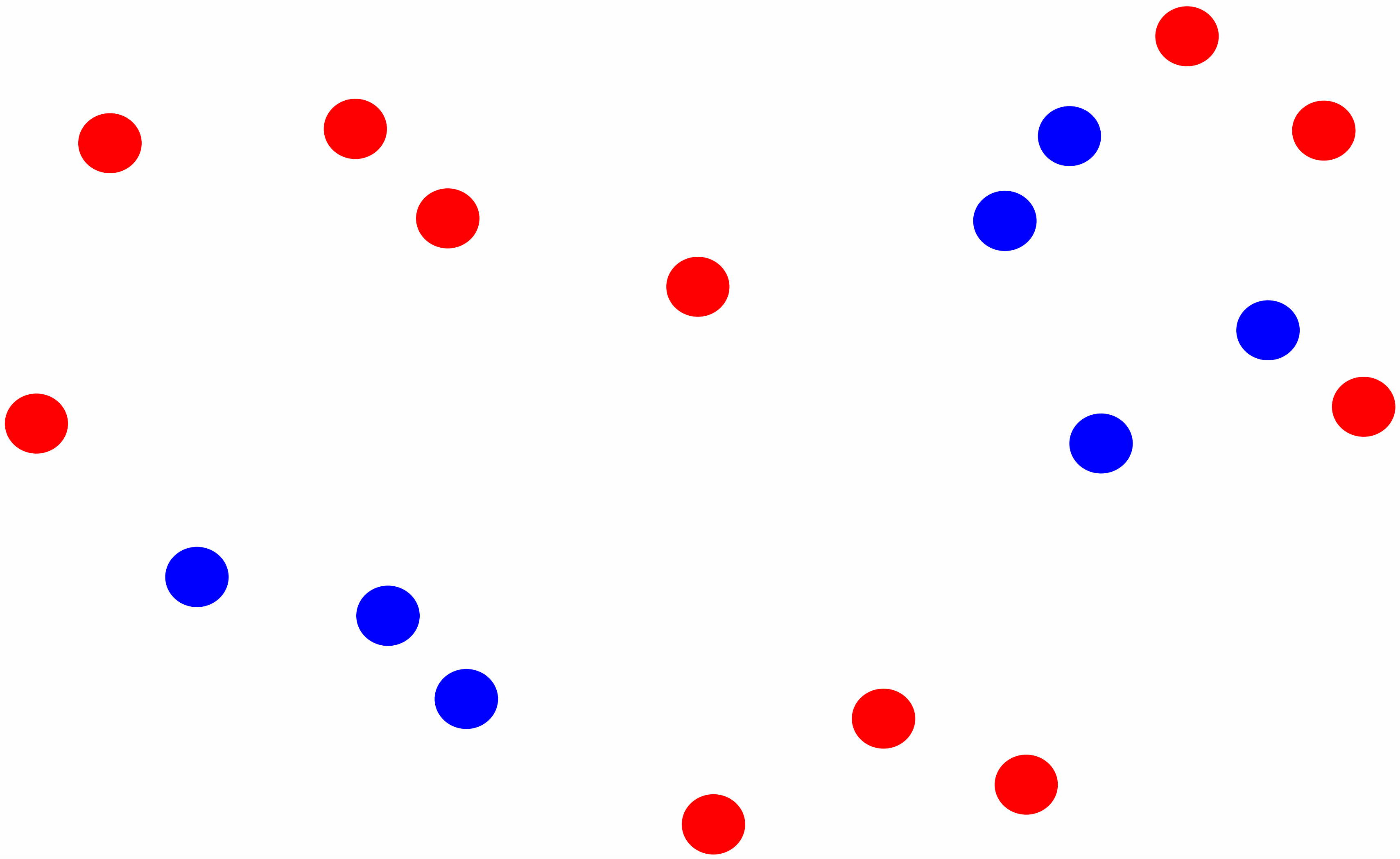
What is the data generating distribution?



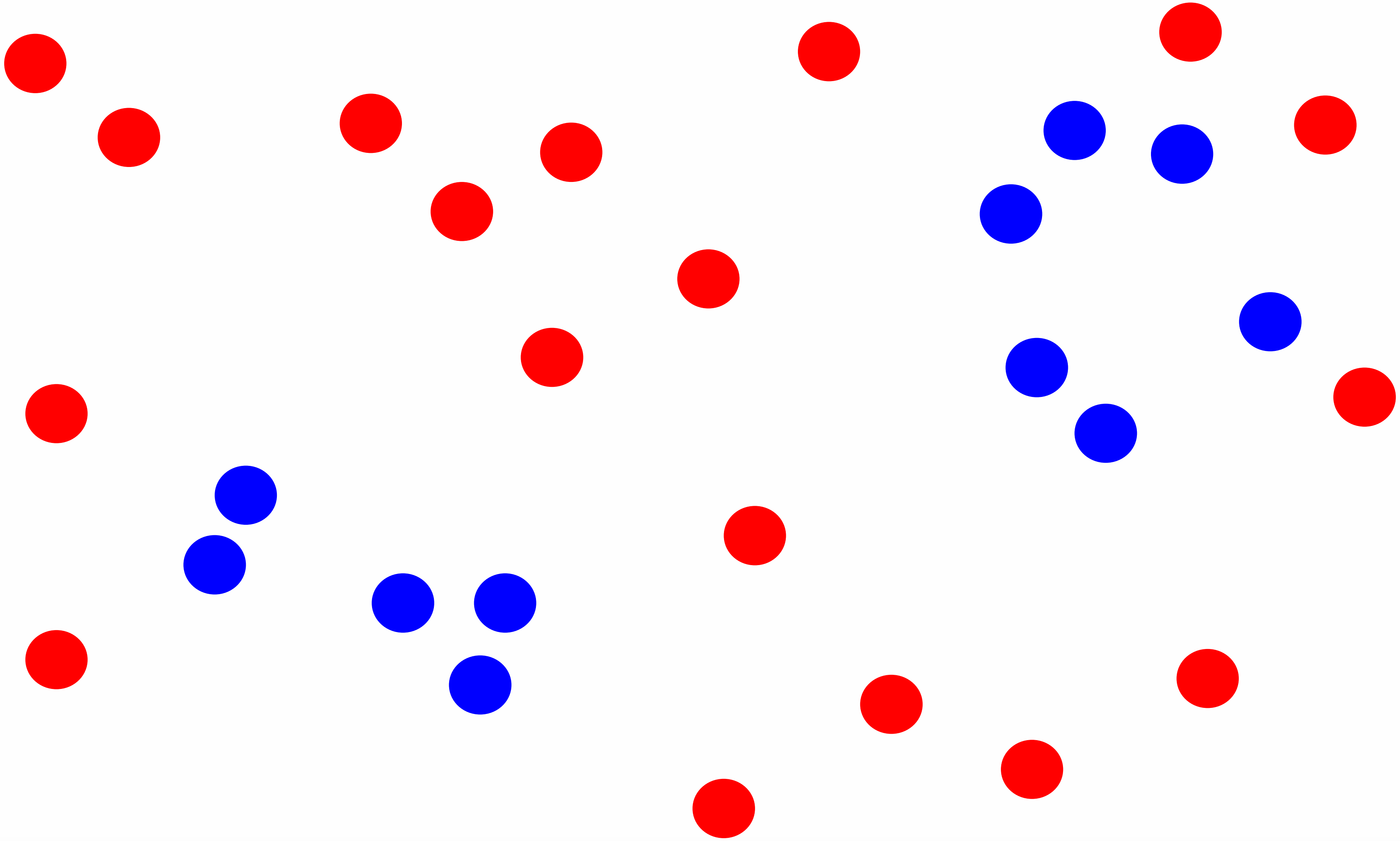
What is the data generating distribution?



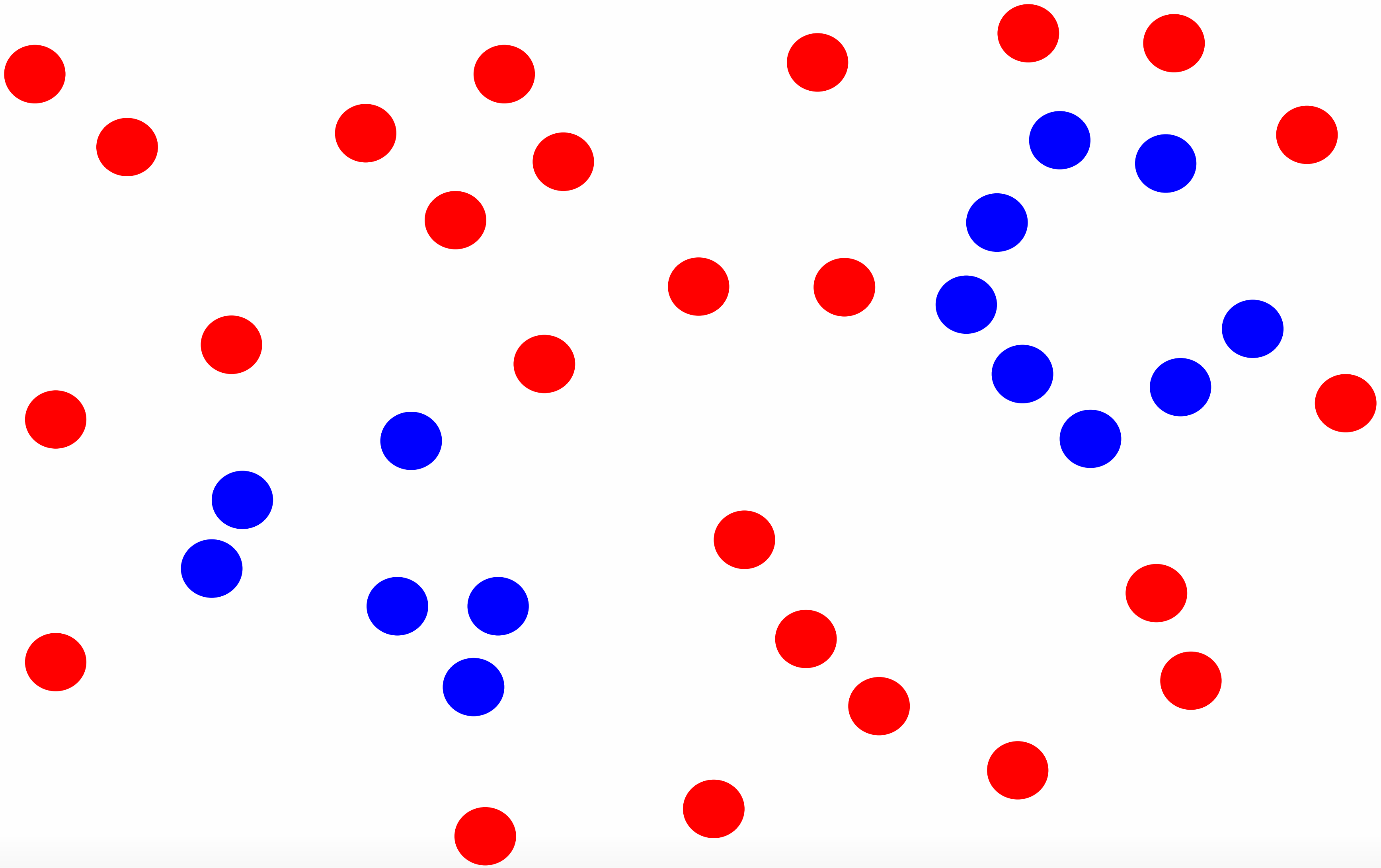
What is the data generating distribution?



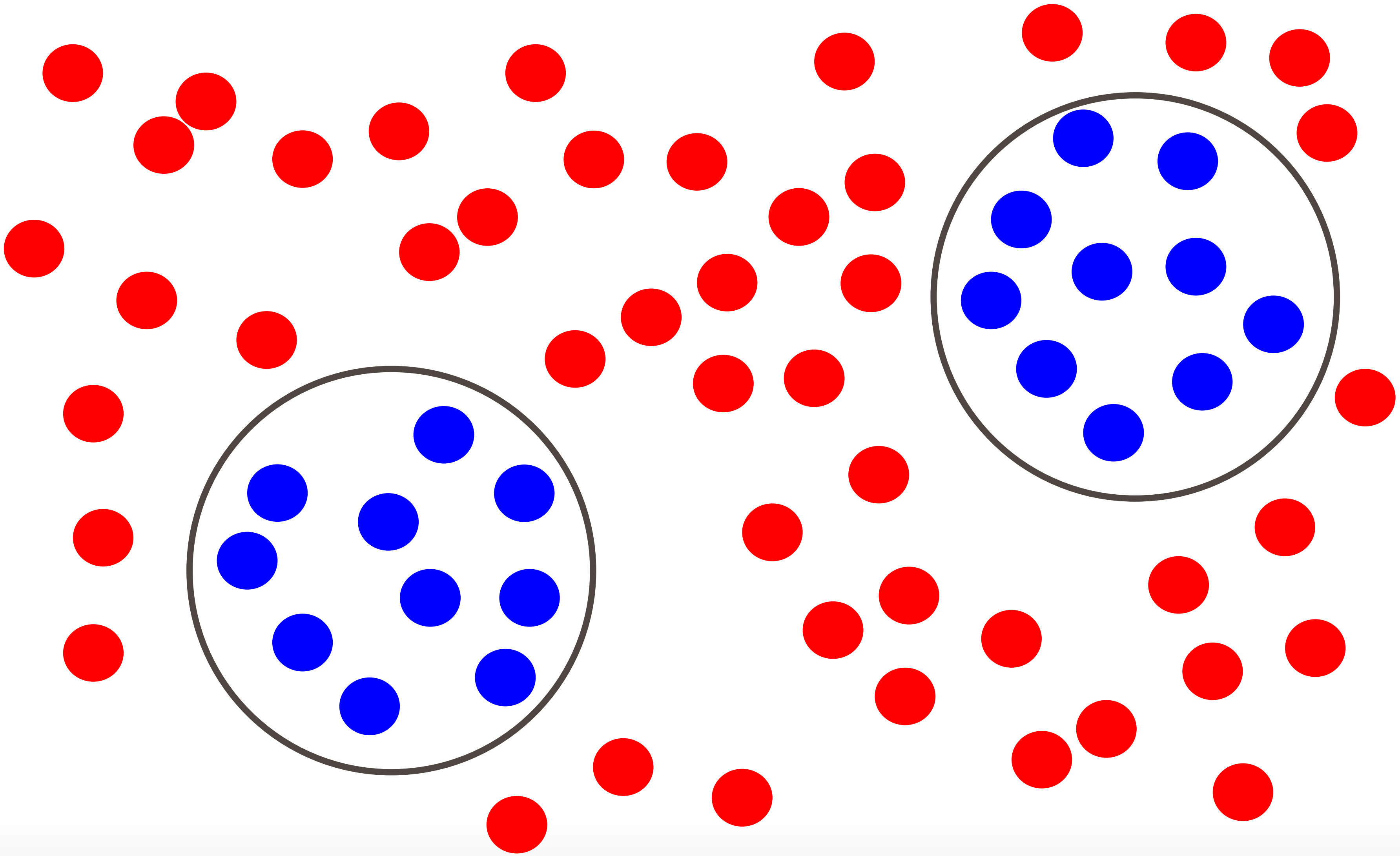
What is the data generating distribution?



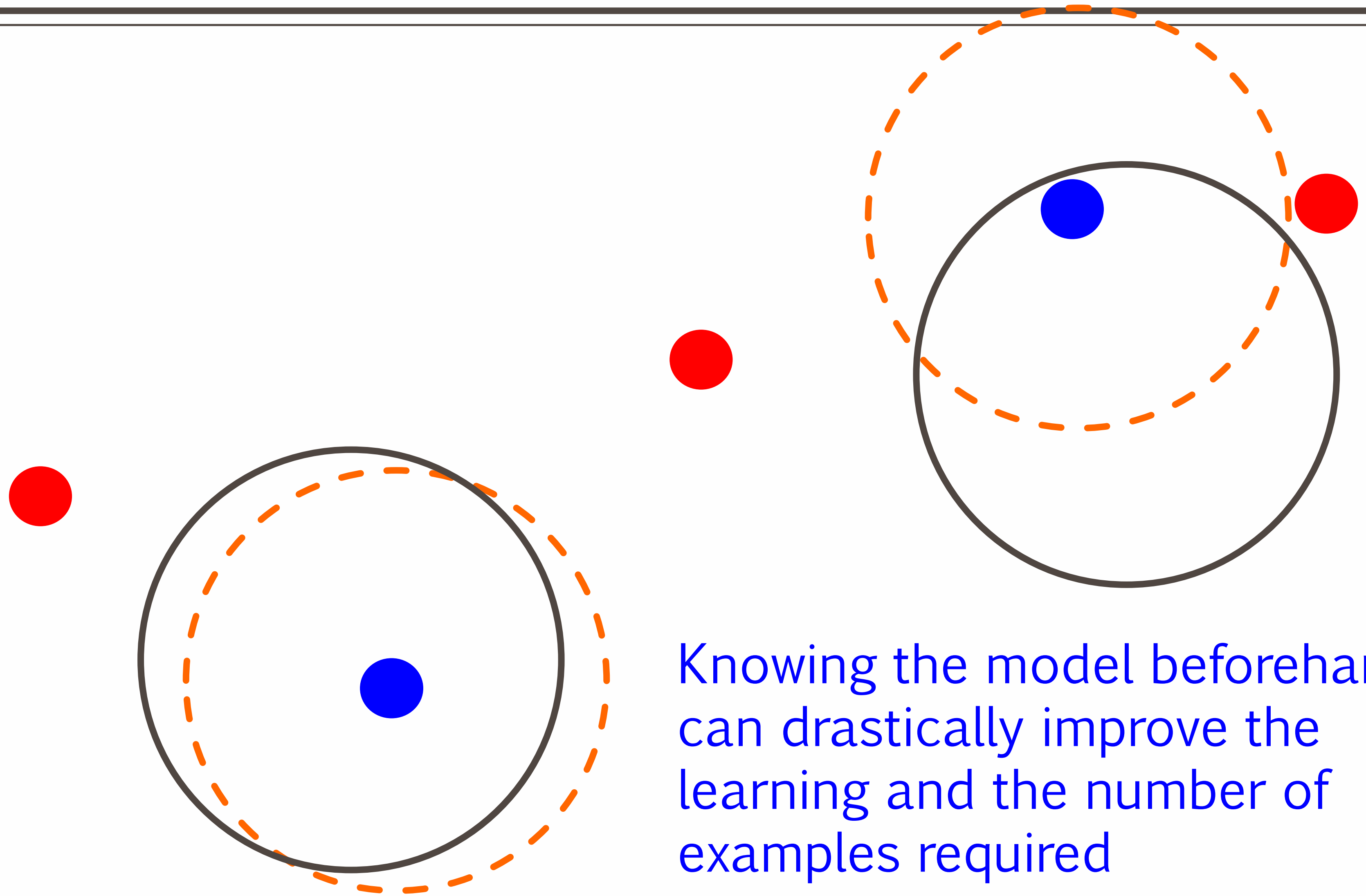
What is the data generating distribution?



Actual model

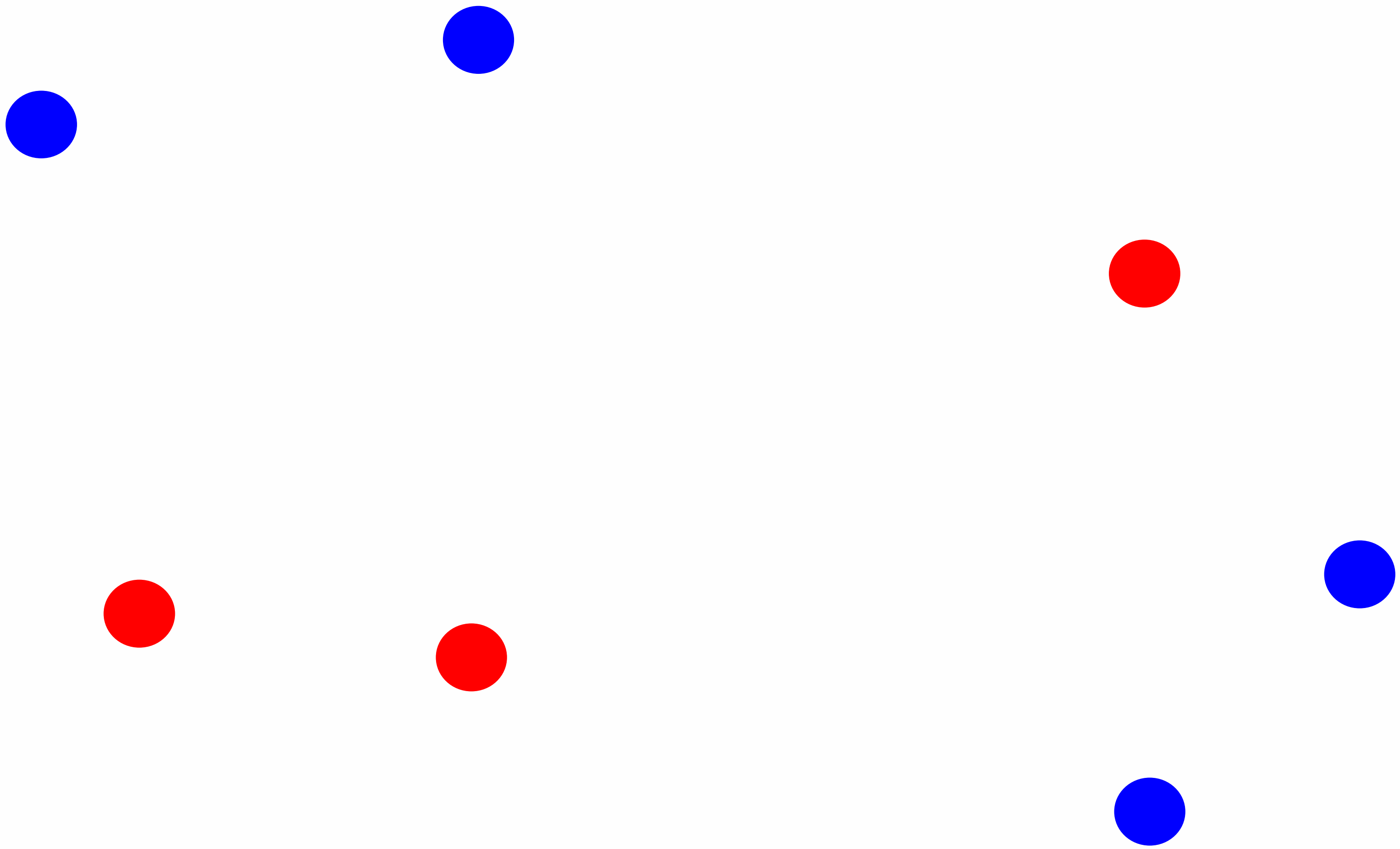


What is the data generating distribution?

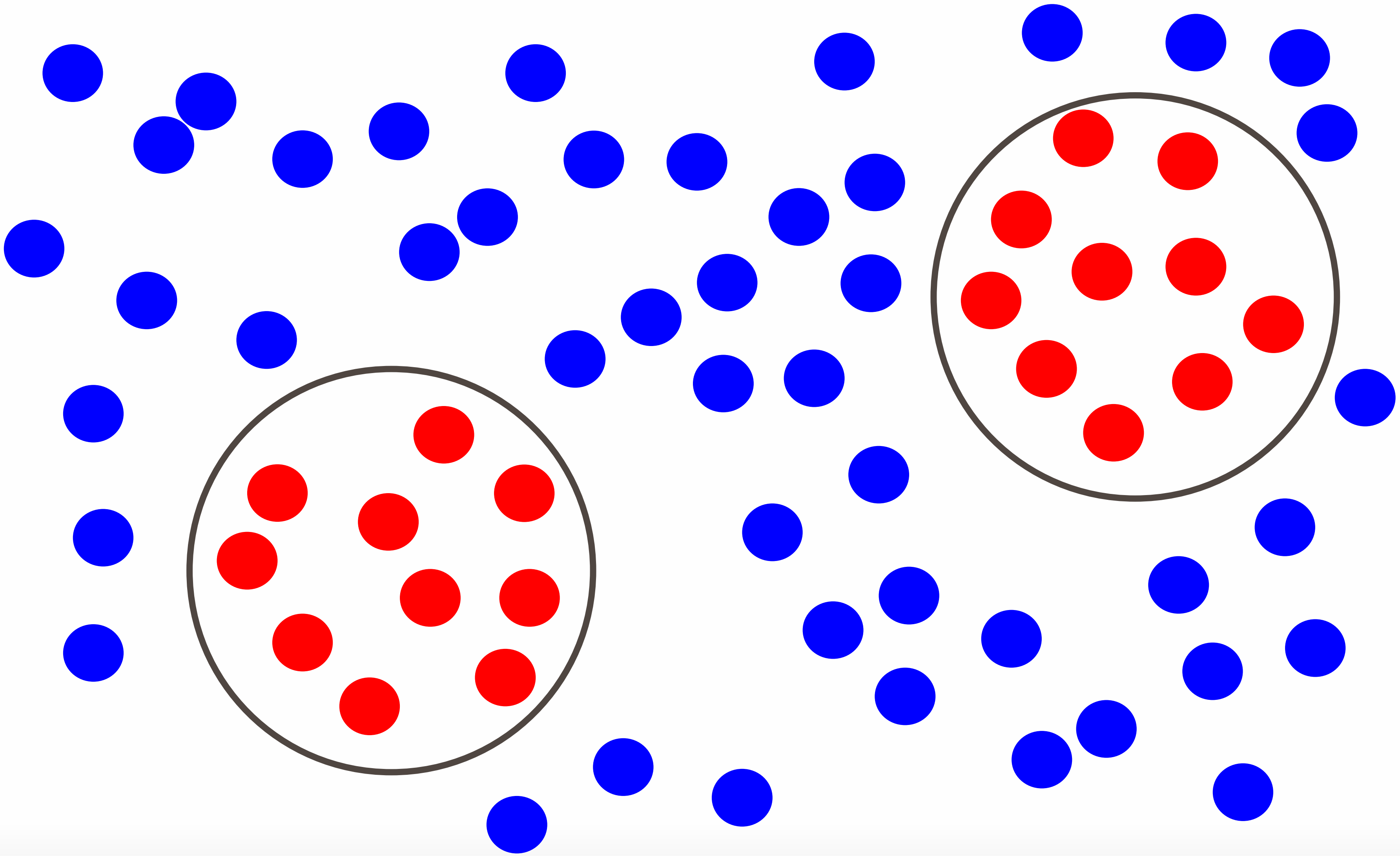


Knowing the model beforehand can drastically improve the learning and the number of examples required

What is the data generating distribution?



Make sure your assumption is correct

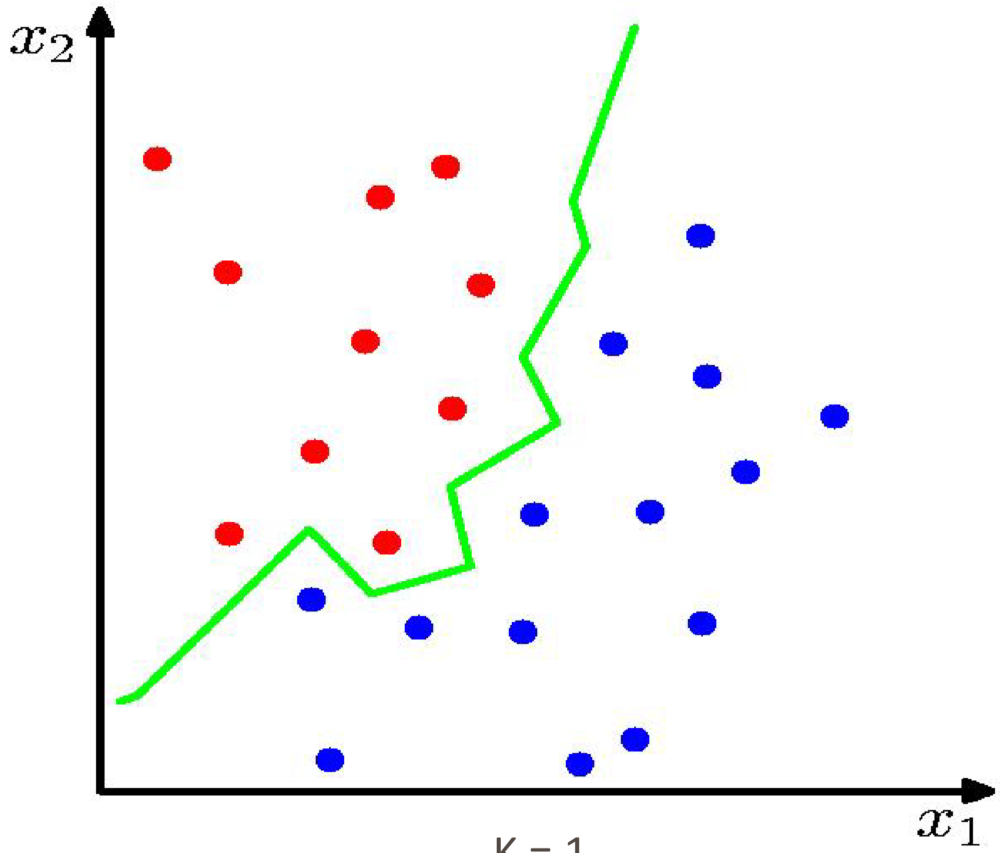


LEARNABLE LINEAR CLASSIFIER

Machine learning models

- What were the model assumptions (if any) that k-NN and decision trees make about the data?
- Are there data sets that could never be learned correctly by either?

k-NN model



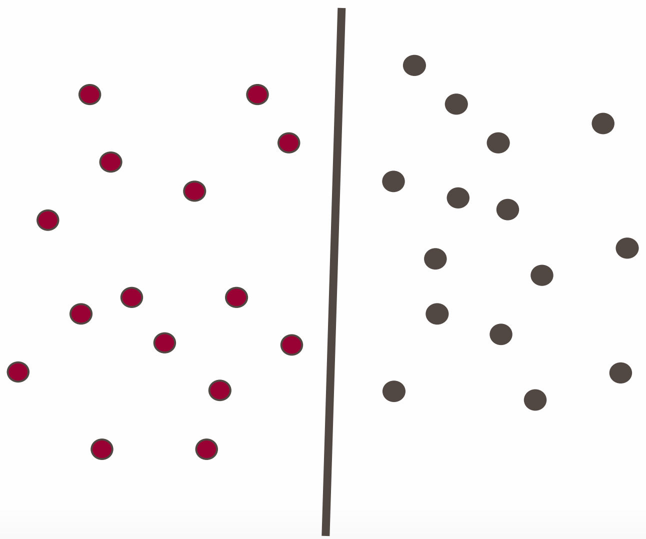
K = 1

Bias

- The “bias” of a model is how strong the model assumptions are.
  - low-bias classifiers make minimal assumptions about the data (k-NN and DT are generally considered low bias)
  - high-bias classifiers make strong assumptions about the data

Linear models

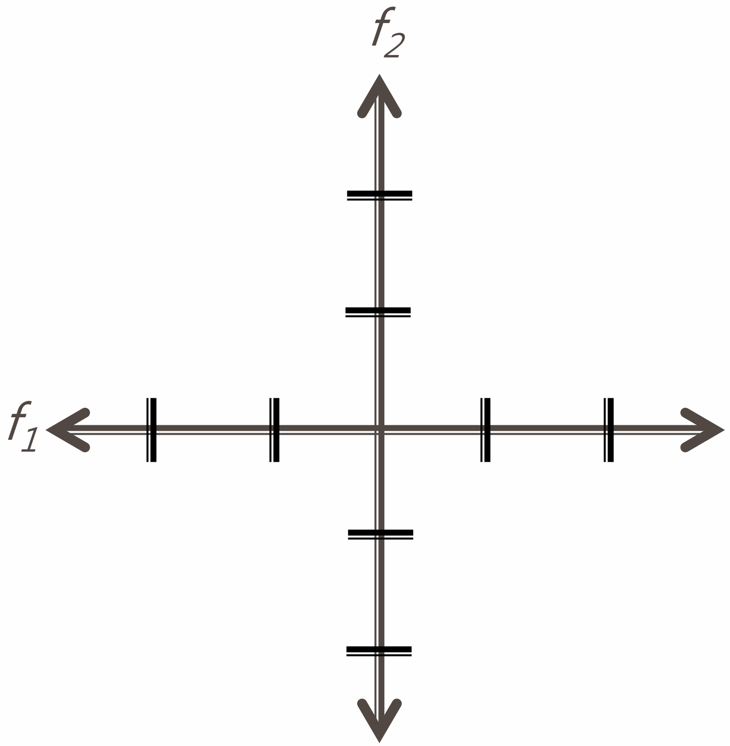
- A strong high-bias assumption is linear separability:
  - in 2 dimensions, can separate classes by a line
  - in higher dimensions, need hyperplanes
- A linear model is a model that assumes the data is linearly separable



Defining a line

Any pair of values (w_1, w_2) defines a line through the origin:

$$0 = w_1 f_1 + w_2 f_2$$



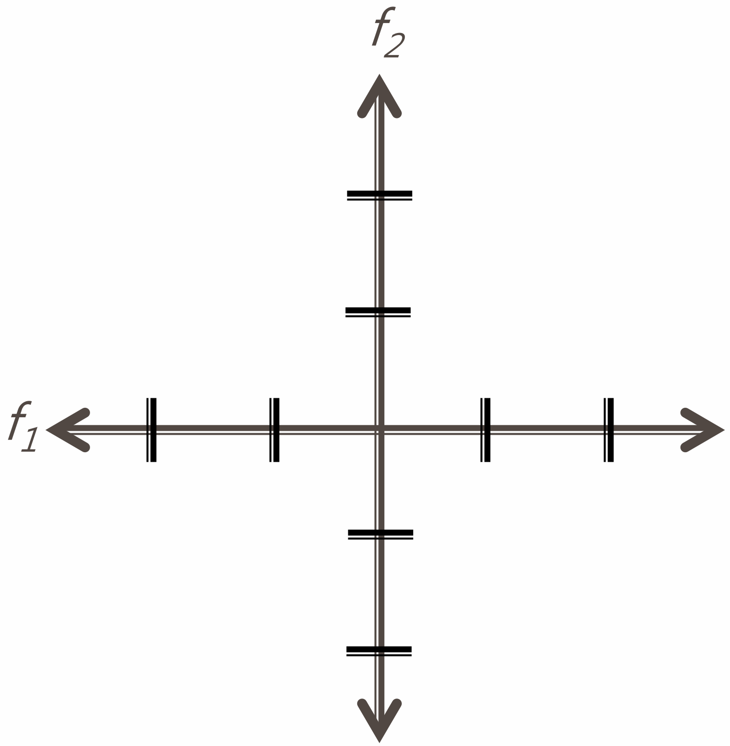
Defining a line

Any pair of values (w_1, w_2) defines a line through the origin:

$$0 = w_1 f_1 + w_2 f_2$$

$$0 = 1 f_1 + 2 f_2$$

-2	1
-1	0.5
0	0
1	-0.5
2	-1



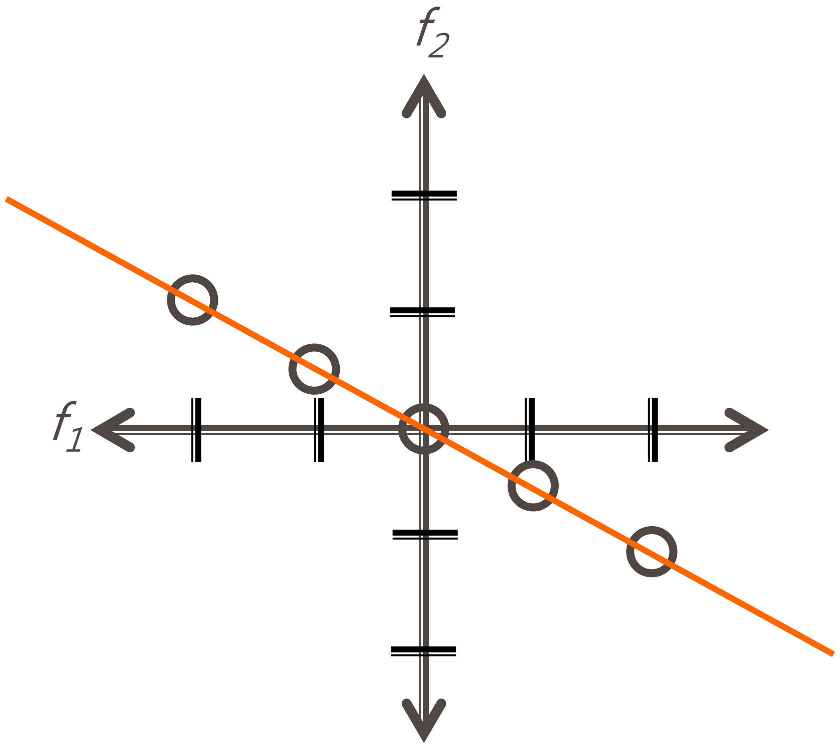
Defining a line

Any pair of values (w_1, w_2) defines a line through the origin:

$$0 = w_1 f_1 + w_2 f_2$$

$$0 = 1 f_1 + 2 f_2$$

-2	1
-1	0.5
0	0
1	-0.5
2	-1



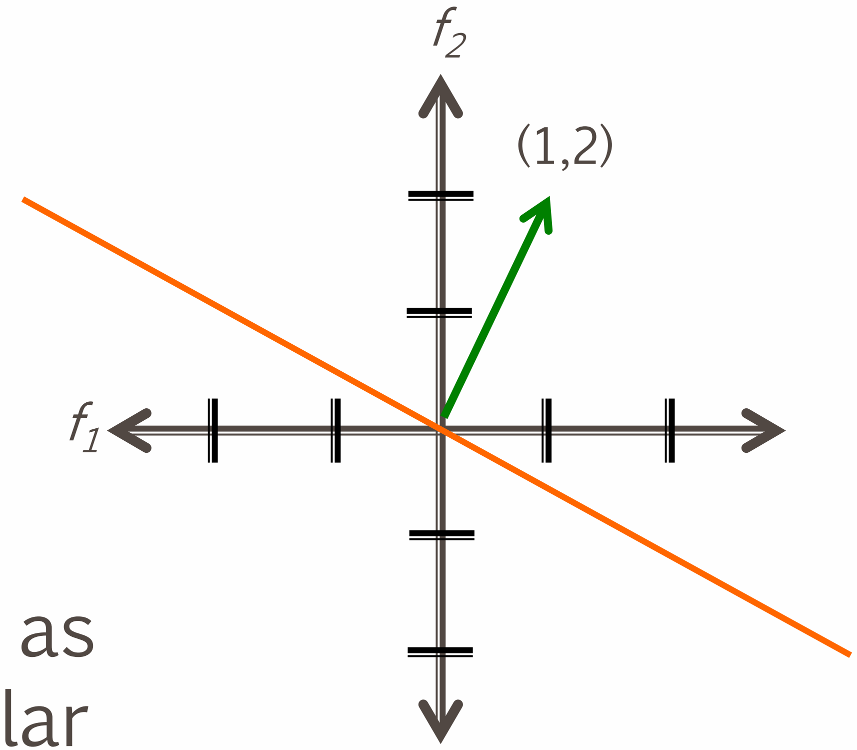
Defining a line

Any pair of values (w_1,w_2) defines a line through the origin:

$$0 = w_1 f_1 + w_2 f_2$$

$$0 = 1 f_1 + 2 f_2$$

w=(1,2)

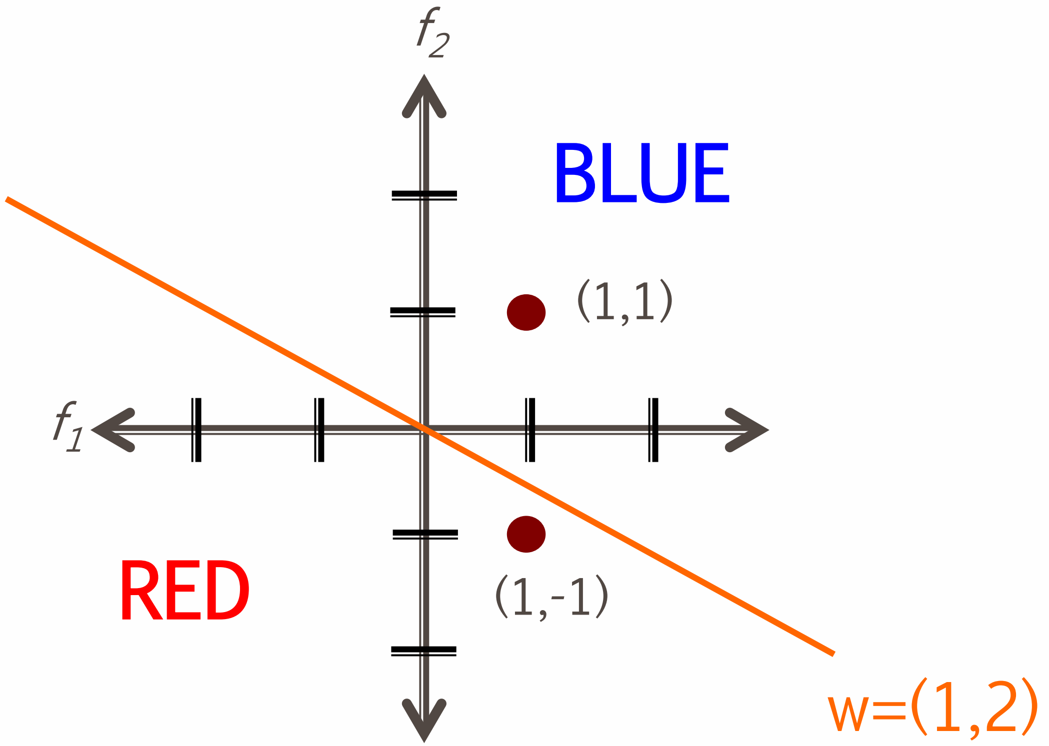


We can also view it as the line perpendicular to the *weight vector*

Classifying with a line

Mathematically, how can we classify points based on a line?

$$0 = 1f_1 + 2f_2$$



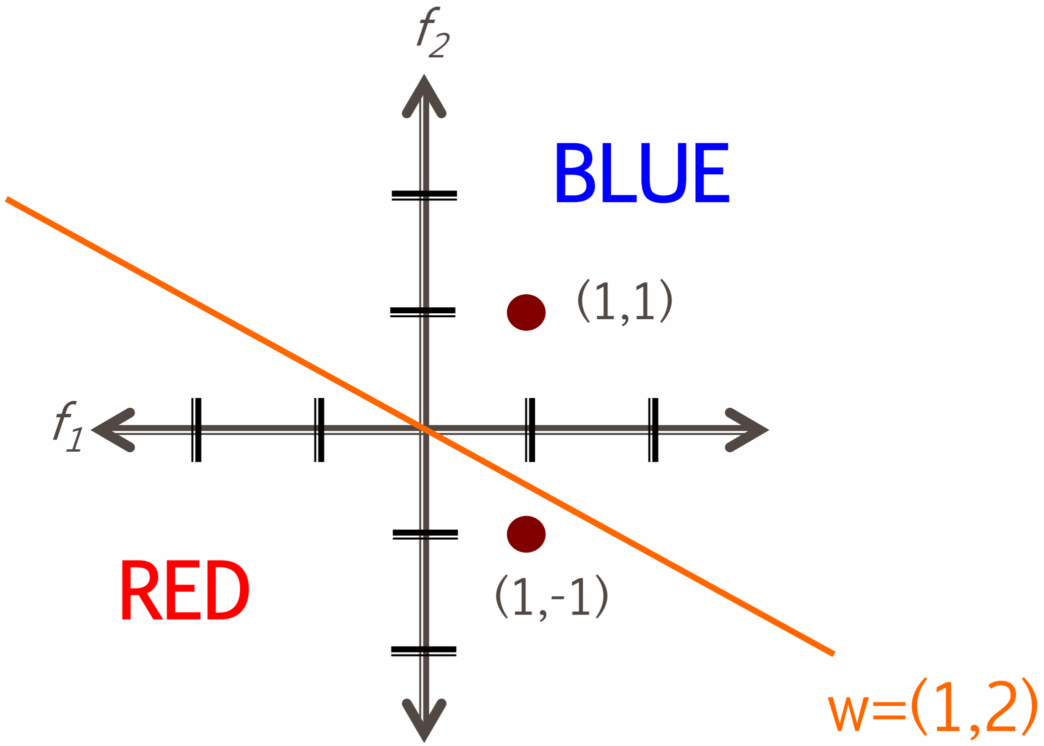
Classifying with a line

Mathematically, how can we classify points based on a line?

$$0 = 1f_1 + 2f_2$$

(1,1): $1*1+2*1=3$

(1,-1): $1*1+2*-1=-1$



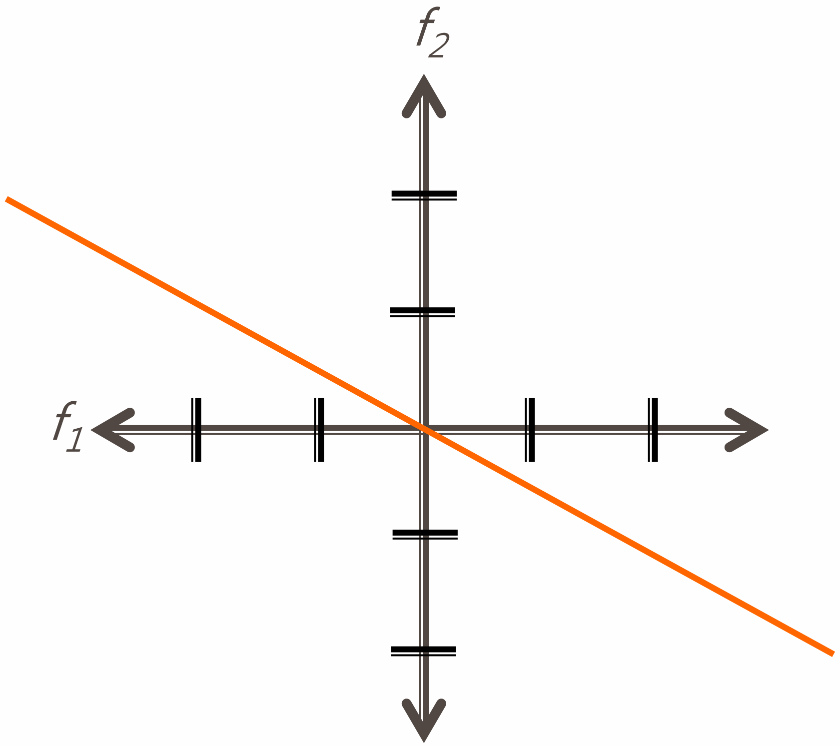
The sign indicates which side of the line

Defining a line

Any pair of values (w_1, w_2) defines a line through the origin:

$$0 = w_1 f_1 + w_2 f_2$$

$$0 = 1 f_1 + 2 f_2$$



How do we move the line off of the origin?

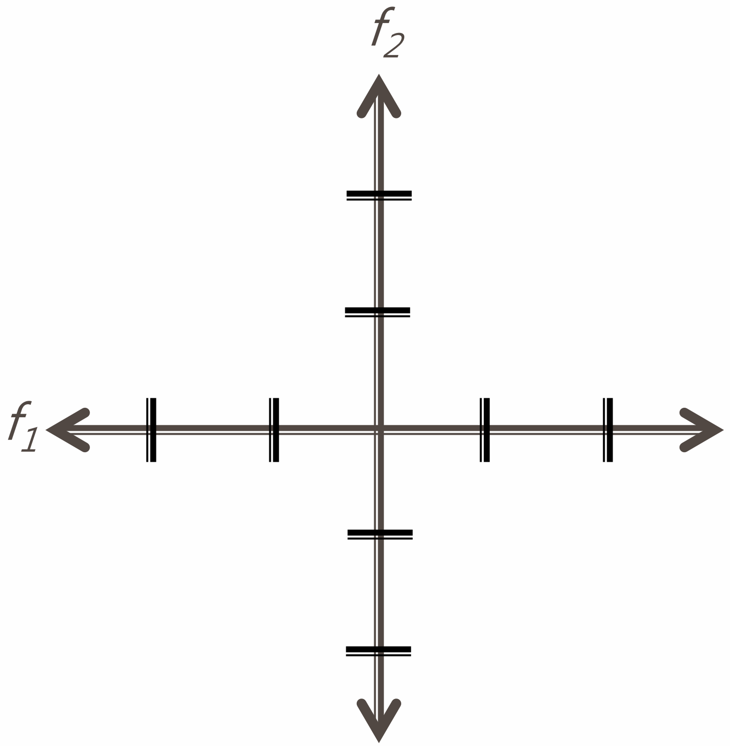
Defining a line

Any pair of values (w_1,w_2) defines a line through the origin:

$$a = w_1 f_1 + w_2 f_2$$

$$-1 = 1 f_1 + 2 f_2$$

-2
-1
0
1
2



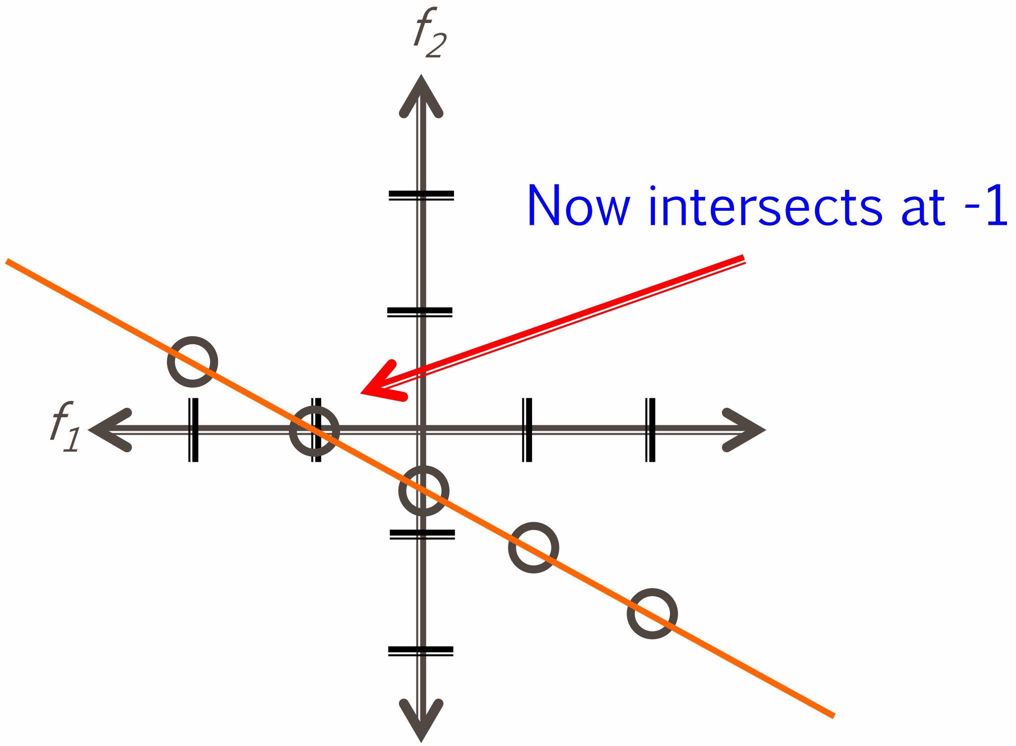
Defining a line

Any pair of values (w_1, w_2) defines a line through the origin:

$$a = w_1 f_1 + w_2 f_2$$

$$-1 = 1 f_1 + 2 f_2$$

-2	0.5
-1	0
0	-0.5
1	-1
2	-1.5



Linear models

- A linear model in n-dimensional space (i.e. n features) is define by n+1 weights:
- In two dimensions, a line:
- In three dimensions, a plane:

$$0 = w_1 f_1 + w_2 f_2 + b$$ (where b = -a)

- In n-dimensions, a hyperplane

$$0 = w_1 f_1 + w_2 f_2 + w_3 f_3 + b$$

$$0 = b + \sum_{i=1}^{n} w_i f_i$$

Classifying with a linear model

- We can classify with a linear model by checking the sign:

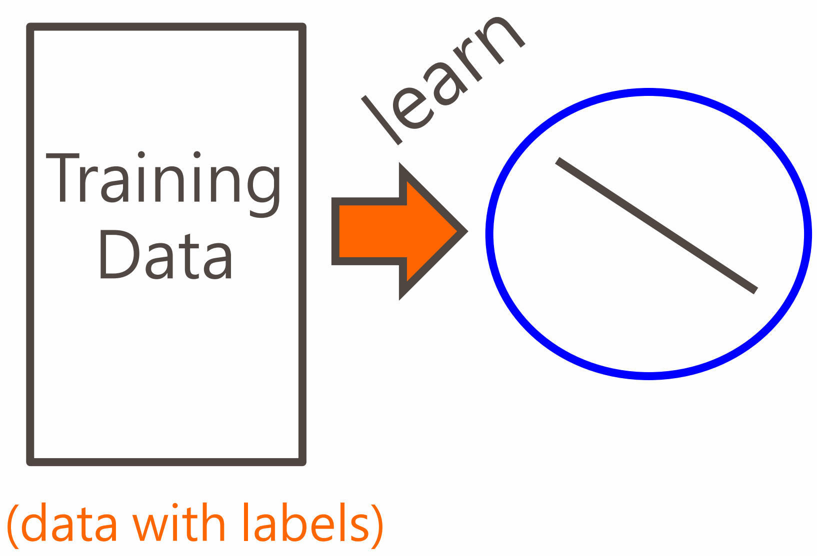
$f_1, f_2, \dots, f_n$ → classifier

$$b + \sum_{i=1}^{n} w_i f_i > 0$$ Positive example

$$b + \sum_{i=1}^{n} w_i f_i < 0$$ Negative example

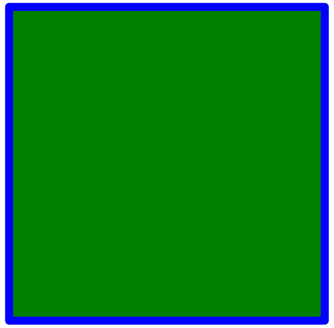
Learning a linear model

- Geometrically, we know what a linear model represents
- Given a linear model (i.e. a set of weights and b) we can classify examples



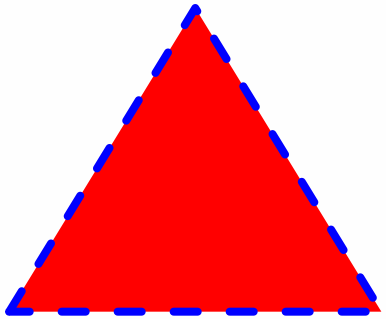
How do we learn
a linear model?

Positive or negative?



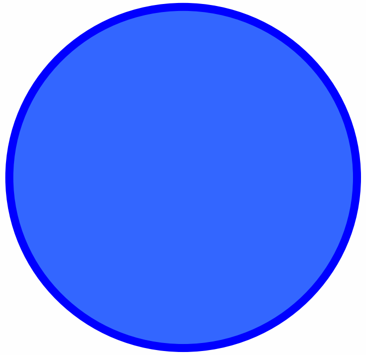
NEGATIVE

Positive or negative?



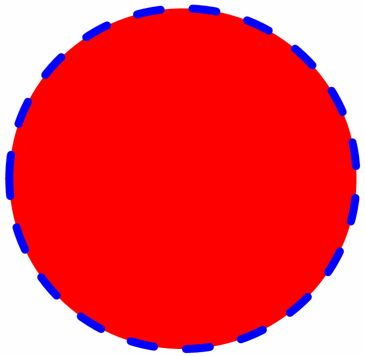
NEGATIVE

Positive or negative?



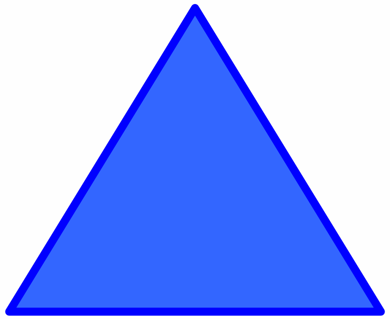
POSITIVE

Positive or negative?



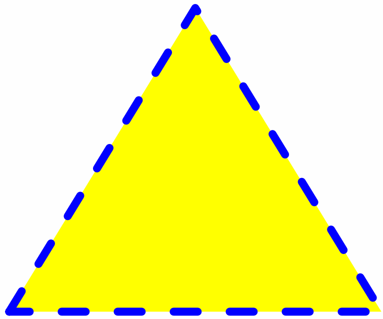
NEGATIVE

Positive or negative?



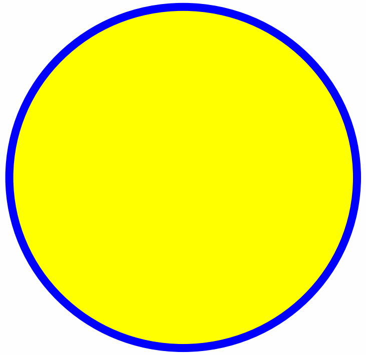
POSITIVE

Positive or negative?



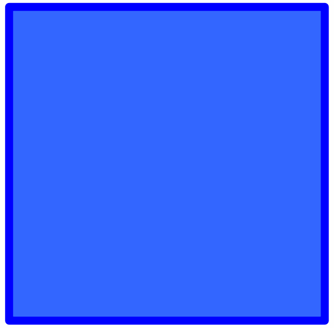
POSITIVE

Positive or negative?



NEGATIVE

Positive or negative?



POSITIVE

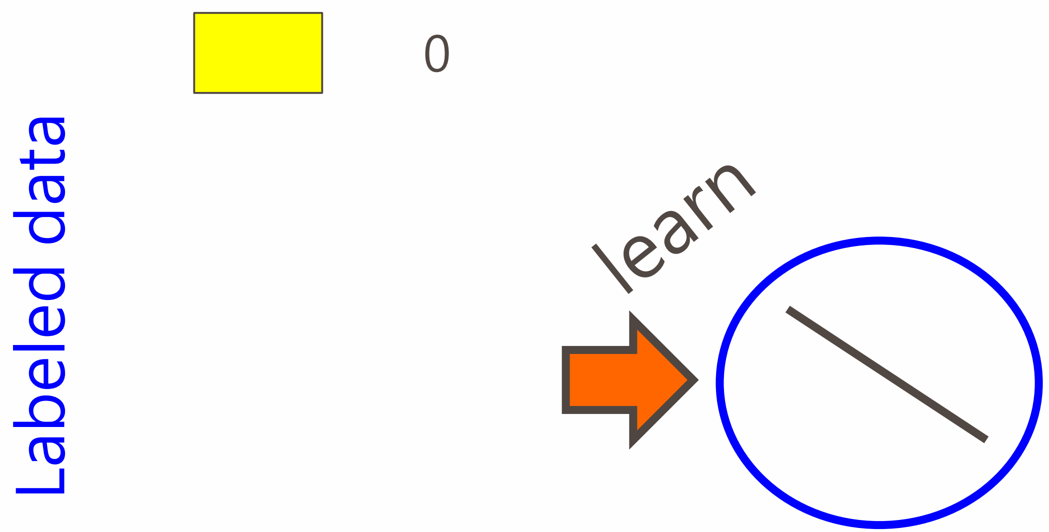
A method to the madness

- blue = positive
- yellow triangles = positive
- all others negative

How is this learning setup different than the learning we' ve done before?

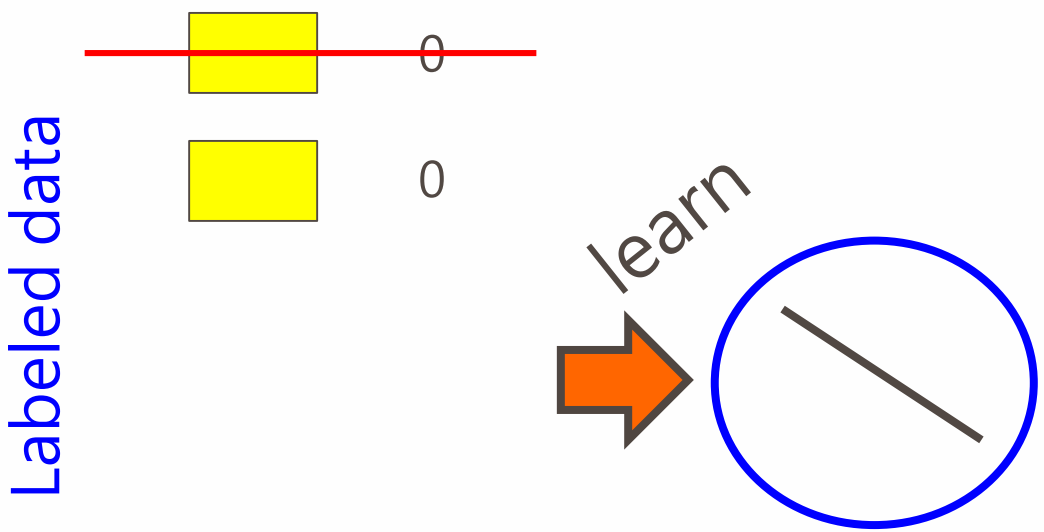
When might this arise?

Online learning algorithm



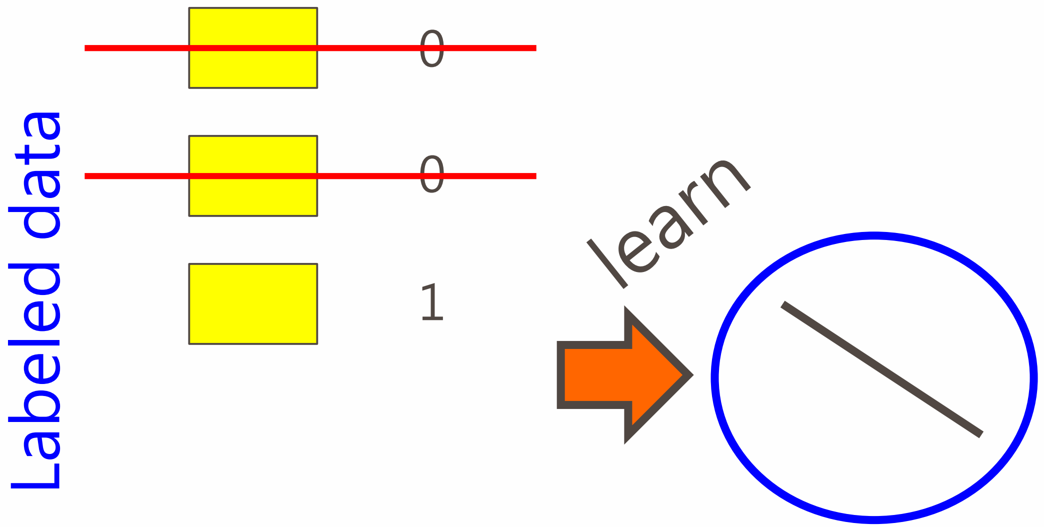
Only get to see one example at a time!

Online learning algorithm



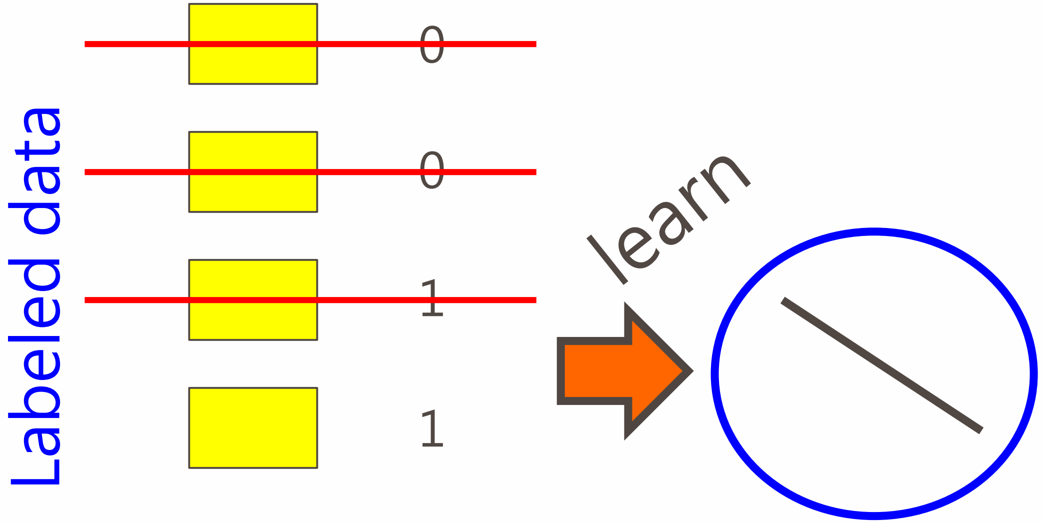
Only get to see one example at a time!

Online learning algorithm



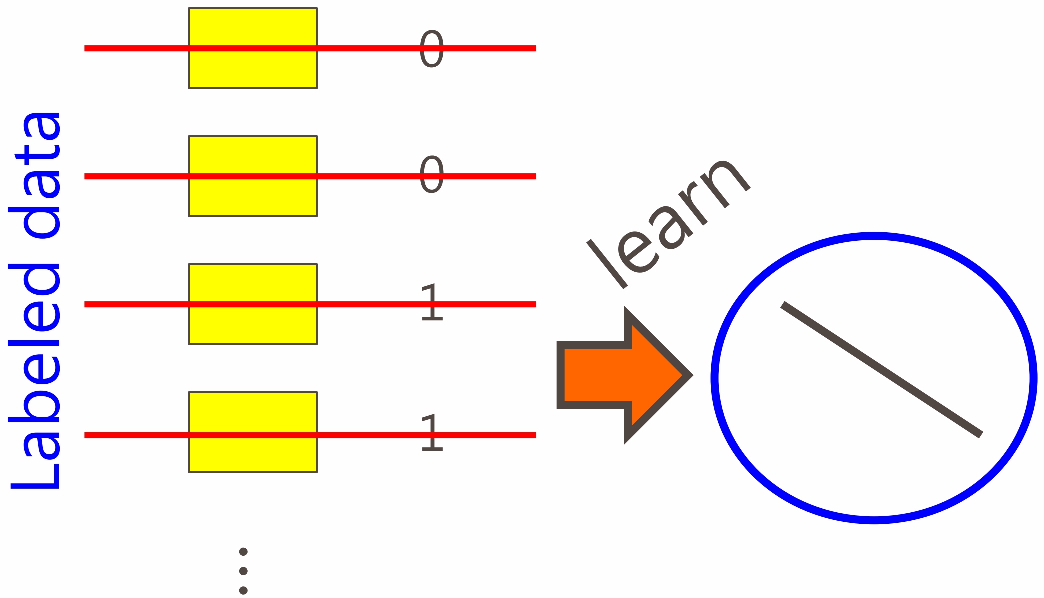
Only get to see one example at a time!

Online learning algorithm



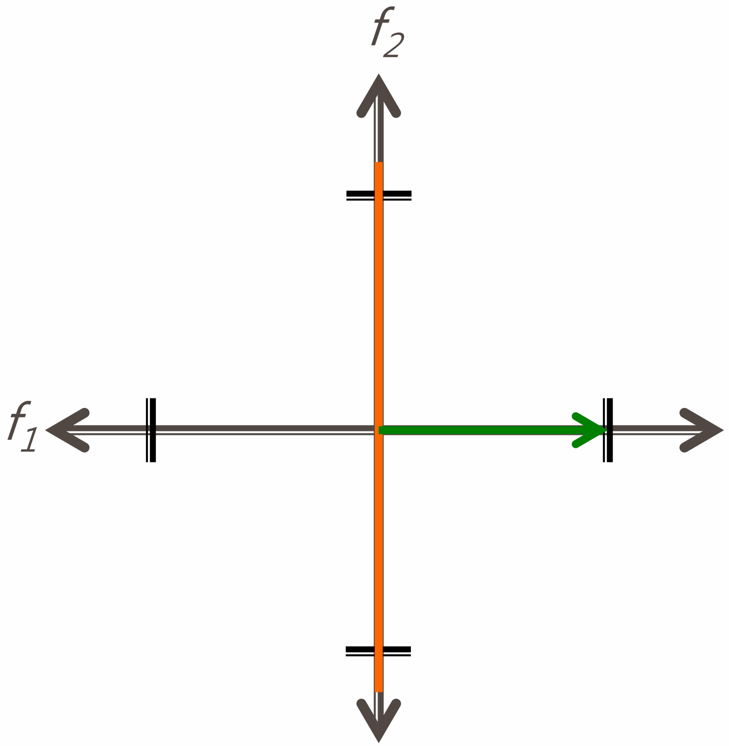
Only get to see one example at a time!

Online learning algorithm



Only get to see one example at a time!

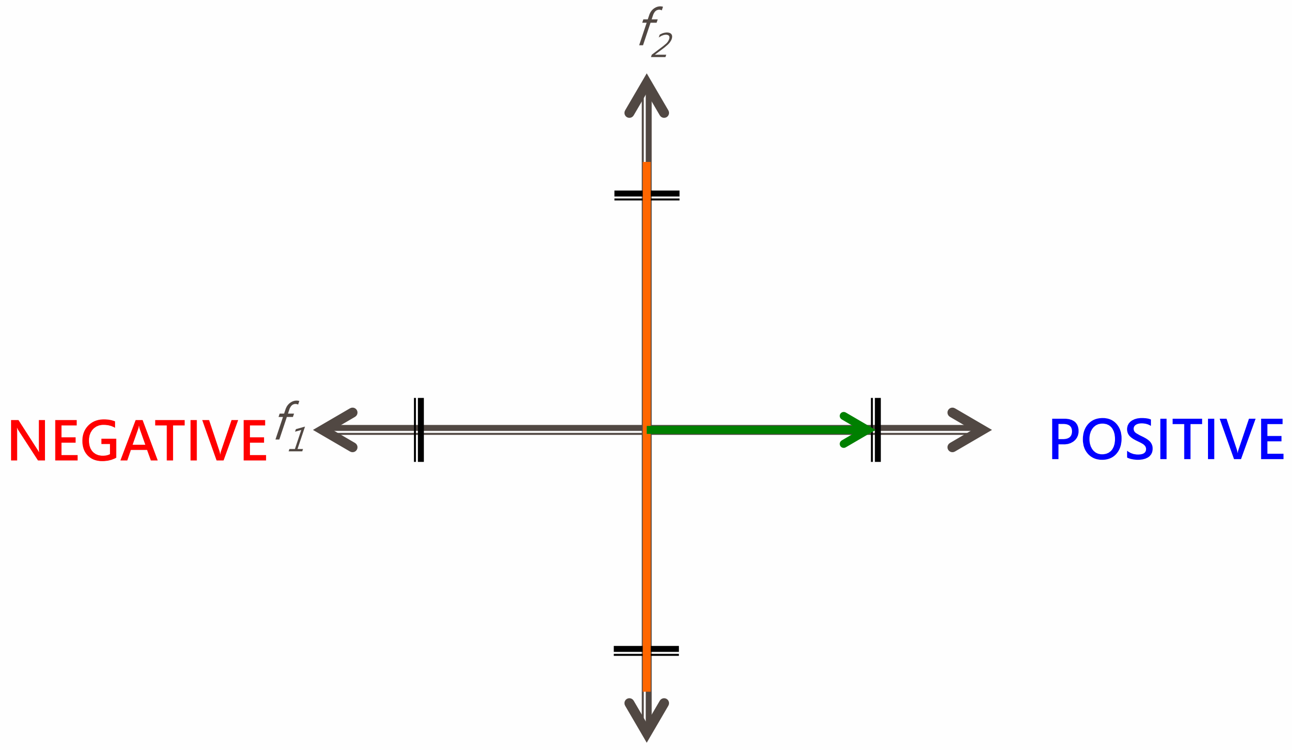
Learning a linear classifier



What does this model currently say?

w=(1,0)

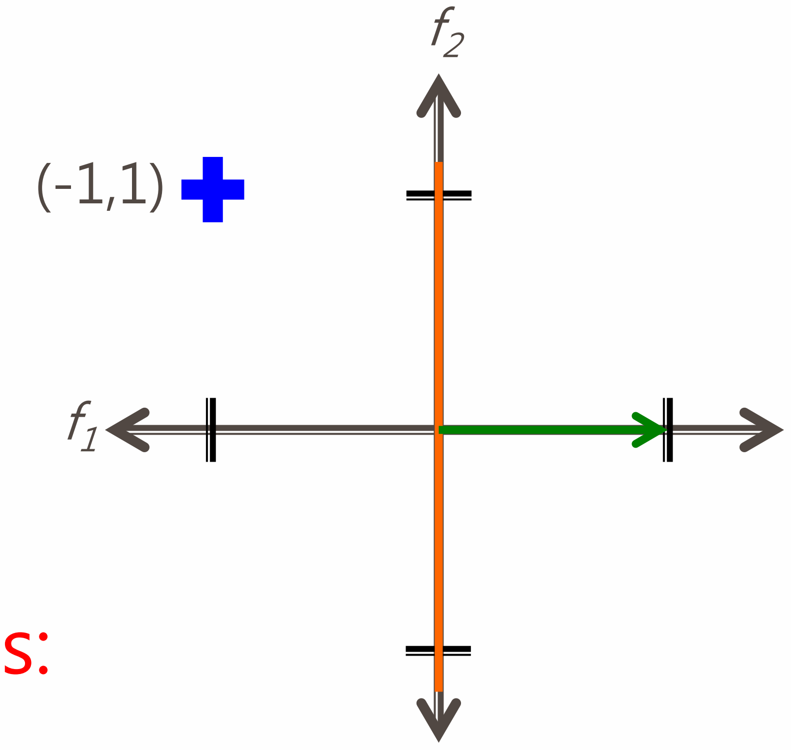
Learning a linear classifier



w=(1,0)

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$



Is our current guess:
right or wrong?

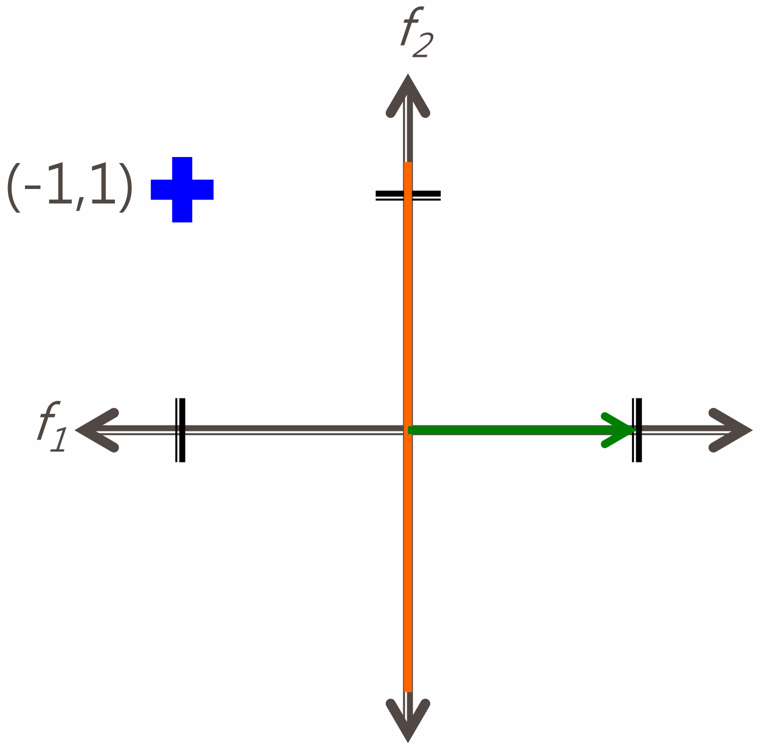
w=(1,0)

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$

$$1 * f_1 + 0 * f_2 =$$

$$1 * -1 + 0 * 1 = -1$$



predicts negative, wrong

How should we update the model?

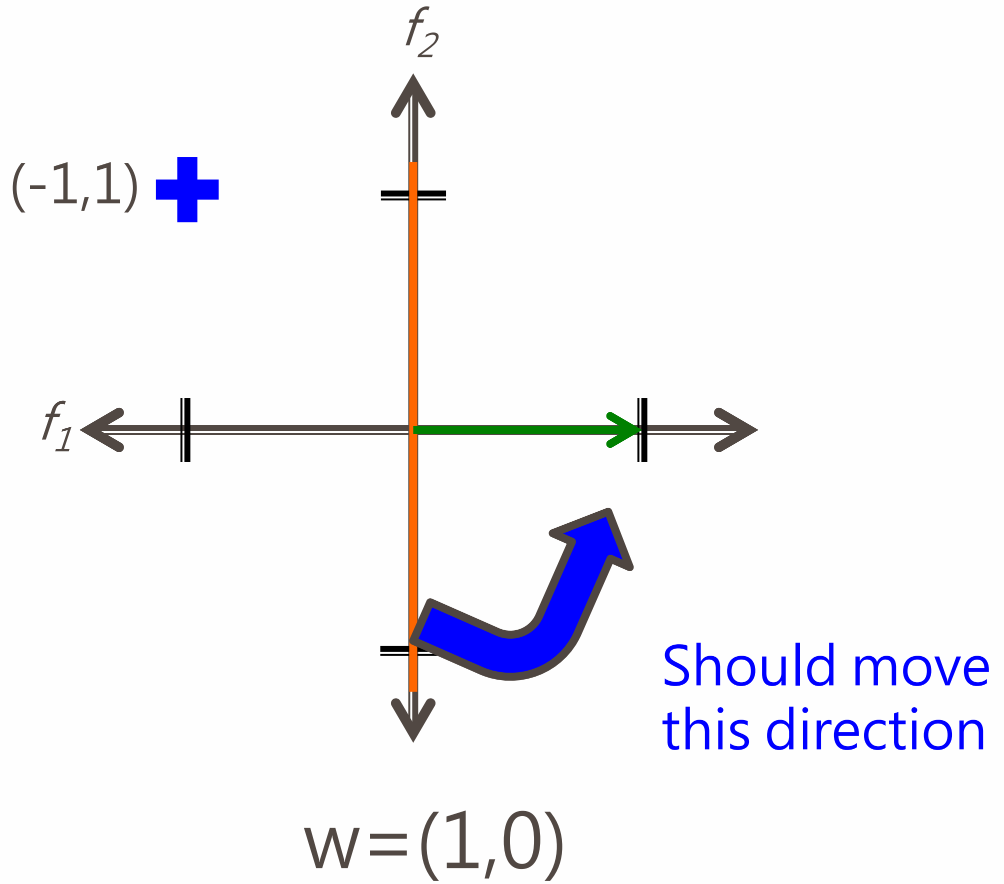
w=(1,0)

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$

$$1 * f_1 + 0 * f_2 =$$

$$1 * -1 + 0 * 1 = -1$$



A closer look at why we got it wrong

w_1 w_2

(-1, 1, positive)

$$1 * f_1 + 0 * f_2 =$$

$$\underbrace{1 * -1 + 0 * 1}_{} = -1$$

We’d like this value to be positive since it’s a positive value

Which of these contributed to the mistake?

A closer look at why we got it wrong

w_1 w_2

(-1, 1, positive)

$$1 * f_1 + 0 * f_2 =$$

$$1 * -1 + 0 * 1 = -1$$

We' d like this value to be positive since it' s a positive value

contributed in the wrong direction

could have contributed (positive feature), but didn' t

How should we change the weights?

A closer look at why we got it wrong

w_1 w_2

(-1, 1, positive)

$$1*f_1+0*f_2=$$

$$1*-1+0*1=-1$$

We'd like this value to be positive since it's a positive value

contributed in the wrong direction

could have contributed (positive feature), but didn't

decrease

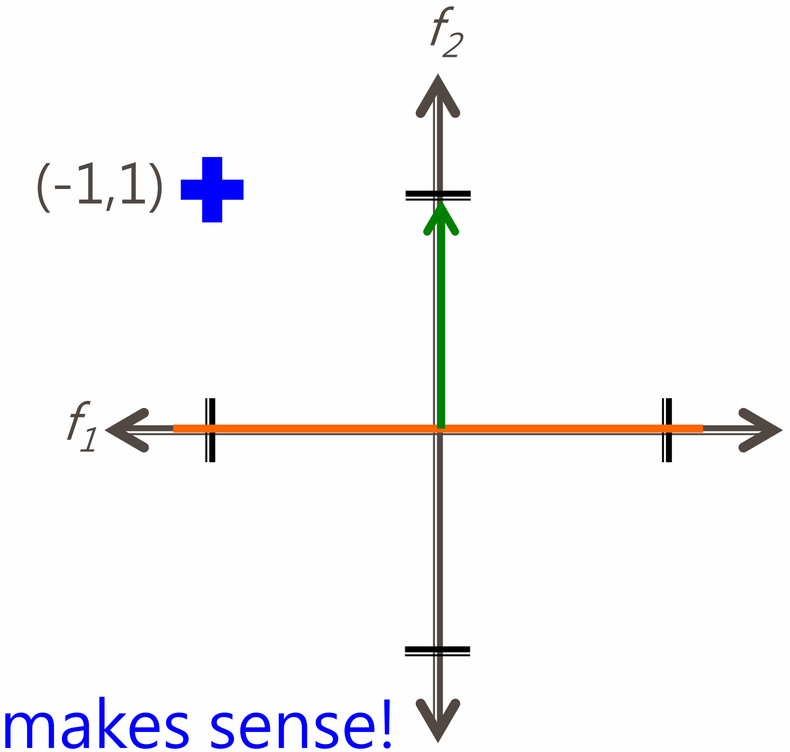
1 -> 0

increase

0 -> 1

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$

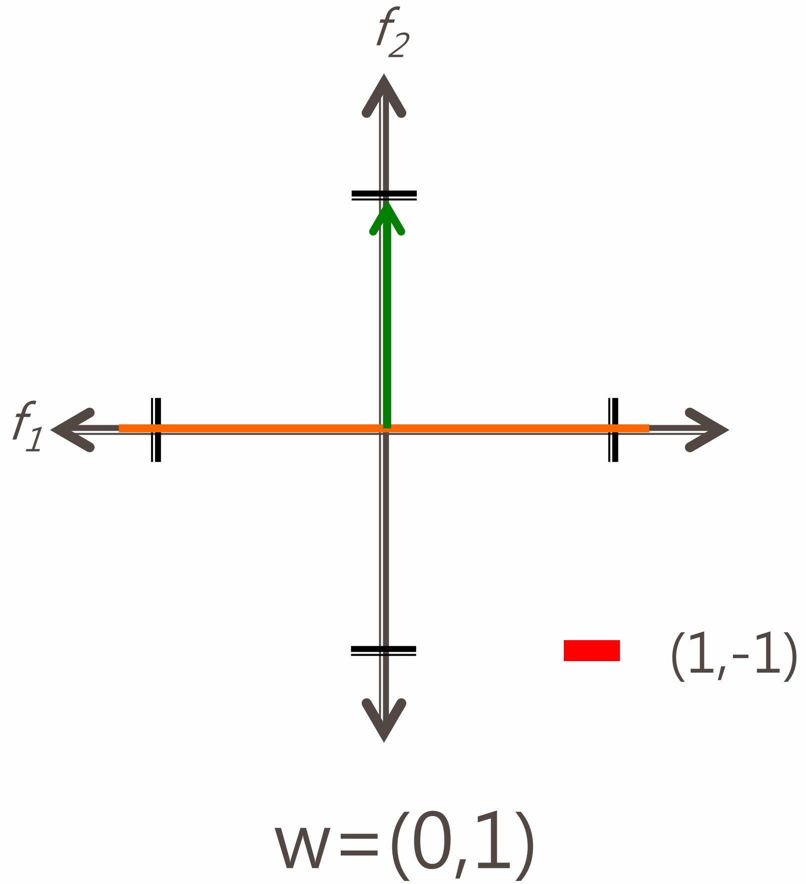


Graphically, this also makes sense!

w=(0,1)

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$



Is our current guess:
right or wrong?

Learning a linear classifier

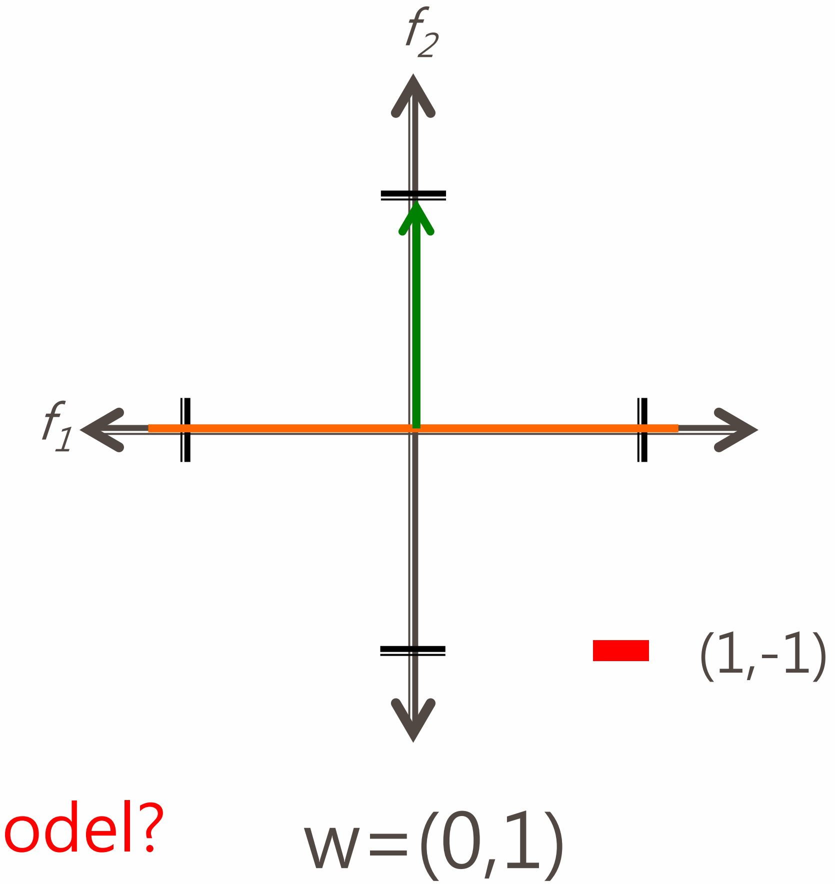
$$0 = w_1 f_1 + w_2 f_2$$

$$0 * f_1 + 1 * f_2 =$$

$$0 * 1 + 1 * -1 = -1$$

predicts negative, correct

How should we update the model?



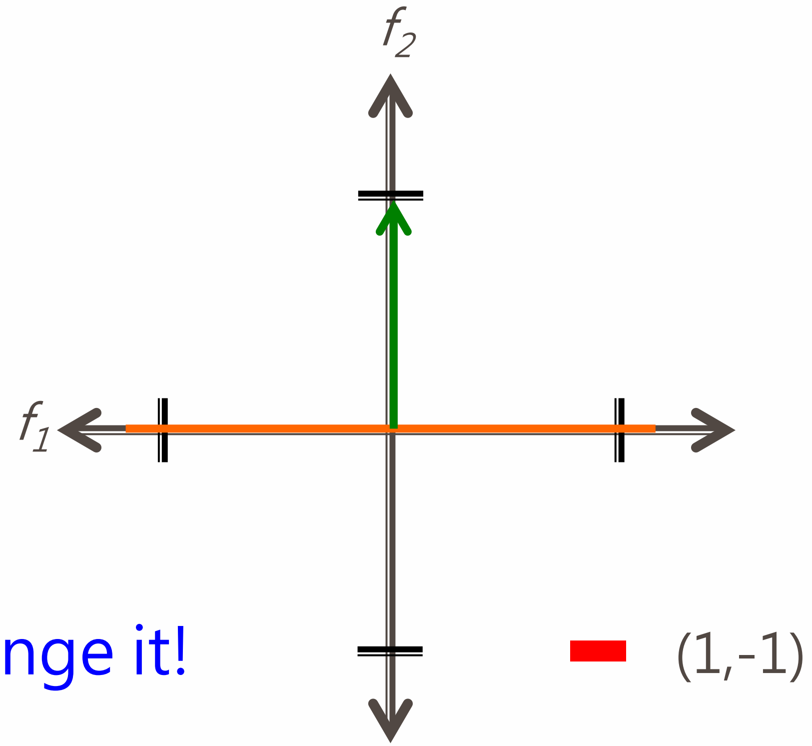
Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$

$$0 * f_1 + 1 * f_2 =$$

$$0 * 1 + 1 * -1 = -1$$

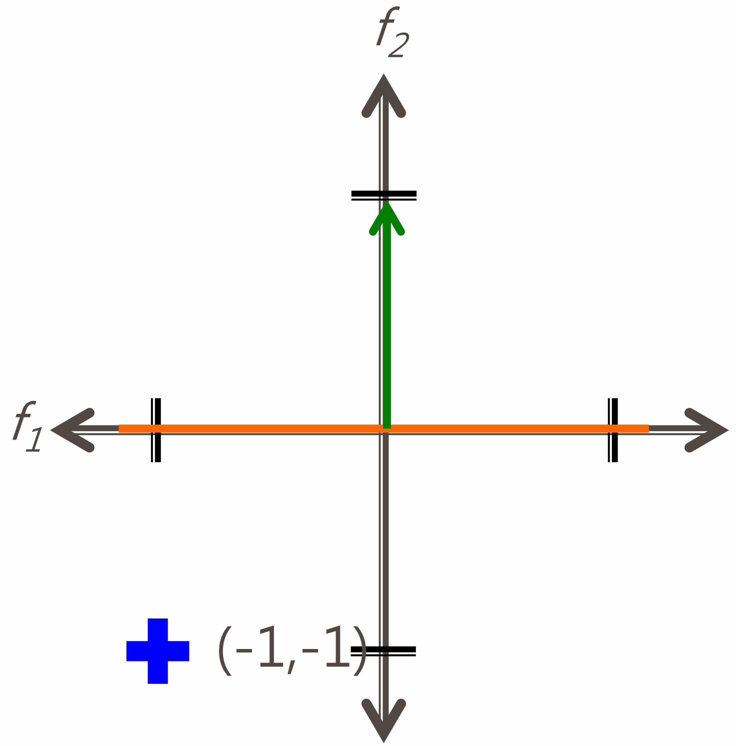
Already correct... don't change it!



w=(0,1)

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$



Is our current guess:
right or wrong?

w=(0,1)

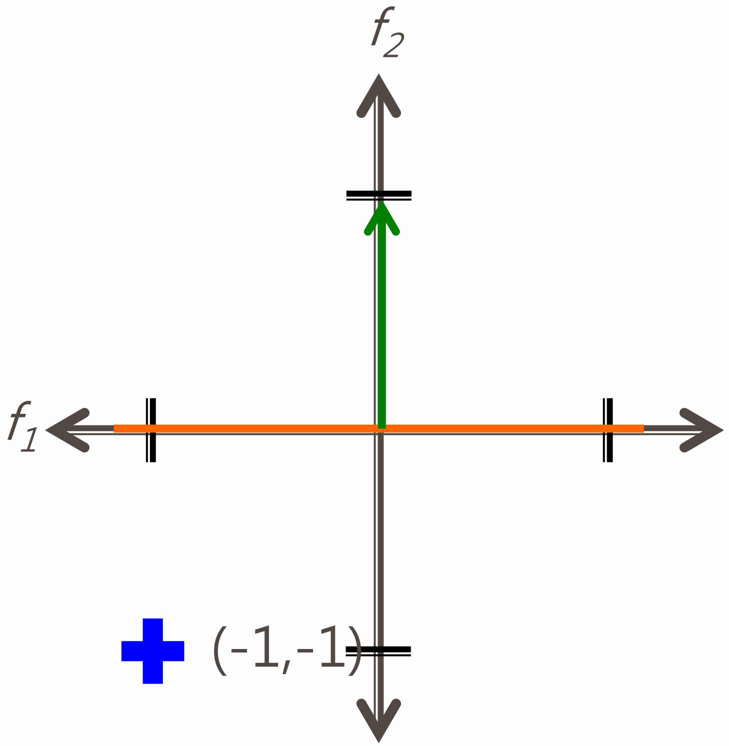
Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$

$$0 * f_1 + 1 * f_2 =$$

$$0 * -1 + 1 * -1 = -1$$

predicts negative, wrong

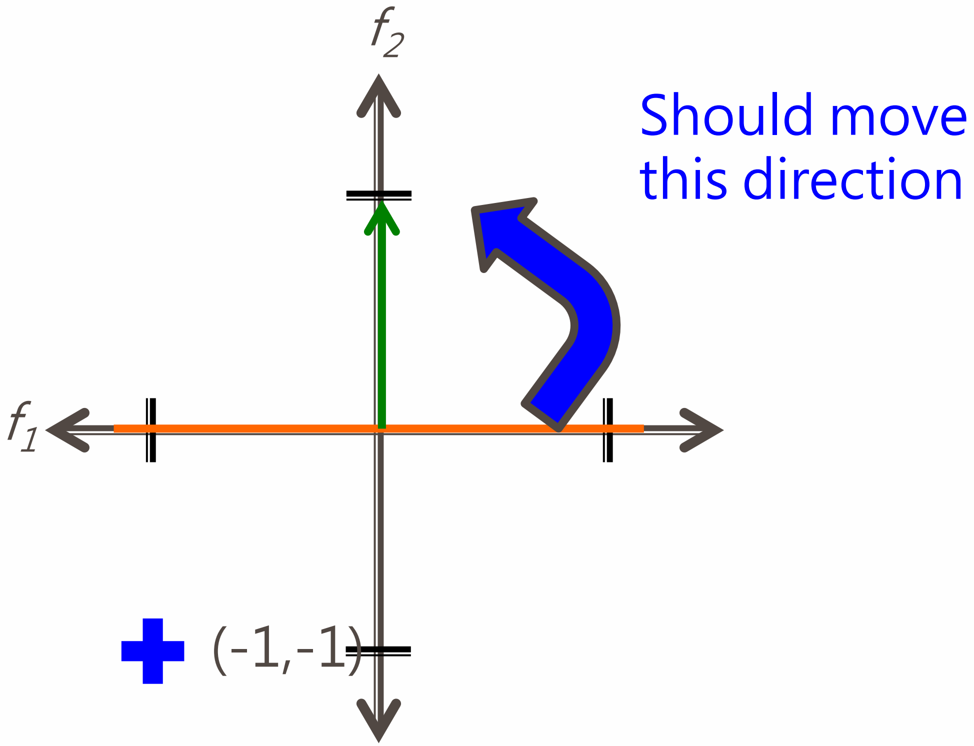


How should we update the model?

w=(0,1)

Learning a linear classifier

$$0 = w_1 f_1 + w_2 f_2$$



w=(0,1)

A closer look at why we got it wrong

$w_1 \quad w_2$

(-1, -1, positive)

$$0 * f_1 + 1 * f_2 =$$

$$\underbrace{0 * -1 + 1 * -1}_{} = -1$$

We'd like this value to be positive since it's a positive value

Which of these contributed to the mistake?

A closer look at why we got it wrong

w_1 w_2

$$0 * f_1 + 1 * f_2 =$$

$$0 * -1 + 1 * -1 = -1$$

(-1, -1, positive)

We'd like this value to be positive since it's a positive value

didn't contribute, but could have

contributed in the wrong direction

How should we change the weights?

A closer look at why we got it wrong

w_1 w_2

$$0 * f_1 + 1 * f_2 =$$

$$0 * -1 + 1 * -1 = -1$$

(-1, -1, positive)

We'd like this value to be positive since it's a positive value

didn't contribute, but could have

decrease

0 -> -1

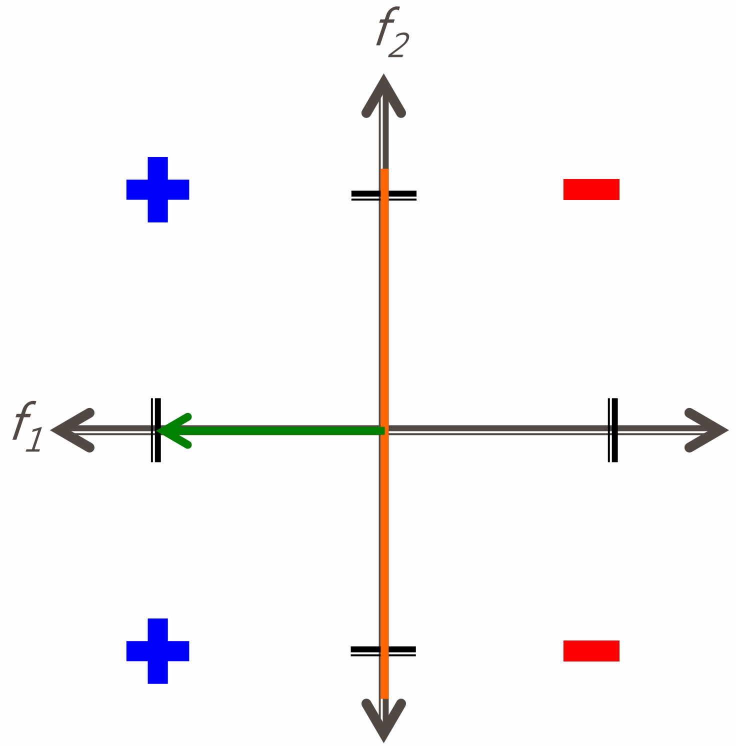
contributed in the wrong direction

decrease

1 -> 0

Learning a linear classifier

f_1, f_2, *label*
-1,-1, positive
-1, 1, positive
1, 1, negative
1,-1, negative



w=(-1,0)

How to formula it?

- Define a cost function

$$MSE = \frac{1}{N}\sum_{i=1}^{N}||p-y||$$

- $$where\ p = predicted\ and\ y = label$$

- Where we know that $p = w^T F$

$$w^* = argmin\frac{1}{N}\sum_{i=1}^{N}||w^T F - y||$$

ADVANCED
PERCEPTRON LEARNING

Linear models

- A linear model in n-dimensional space (i.e. n features) is define by $n+1$ weights:
- In two dimensions, a line:

$$0 = w_1 f_1 + w_2 f_2 + b \quad \text{(where b = -a)}$$

- In three dimensions, a plane:

$$0 = w_1 f_1 + w_2 f_2 + w_3 f_3 + b$$

- In n-dimensions, a hyperplane

$$0 = b + \sum_{i=1}^{n} w_i f_i$$

Perceptron learning algorithm

- repeat until convergence (or for some # of iterations):
- for each training example ($f_1, f_2, \ldots, f_n, label$):
- check if it's correct based on the current model $w_i f_i$

if not correct, update all the weights:
  if label positive and feature positive:
    increase weight (increase weight = predict more positive)
  if label positive and feature negative:
    decrease weight (decrease weight = predict more positive)
  if label negative and feature positive:
    decrease weight (decrease weight = predict more negative)
  if label negative and negative weight:
    increase weight (increase weight = predict more negative)

A trick…

Let positive label = 1 and negative label = -1

label * f_i

```
if not correct, update all the weights:
    if label positive and feature positive:
        increase weight (increase weight = predict more positive)
    if label positive and feature negative:
        decrease weight (decrease weight = predict more positive)
    if label negative and feature positive:
        decrease weight (decrease weight = predict more negative)
    if label negative and negative weight:
        increase weight (increase weight = predict more negative)
```

$1*1=\boxed{1}$

$1*-1=\boxed{-1}$

$-1*1=\boxed{-1}$

$-1*-1=\boxed{1}$

Perceptron learning algorithm

repeat until convergence (or for some # of iterations):

for each training example ($f_1, f_2, \ldots, f_n, label$):

check if it's correct based on the current model

if not correct, update all the weights:

for each w_i:

$w_i = w_i + f_i$*label

$b = b$ + label

How do we check if it's correct?

Perceptron learning algorithm

repeat until convergence (or for some # of iterations):

for each training example ($f_1, f_2, \ldots, f_n, label$):

$$prediction = b + \sum_{i=1}^{n} w_i f_i$$

if $prediction * label \leq 0$: // they don't agree

for each w_i:

$w_i = w_i + f_i * label$

$b = b + label$

Think about: why b is updated by adding $label$ directly?

Your turn ☺

repeat until convergence (or for some # of iterations):

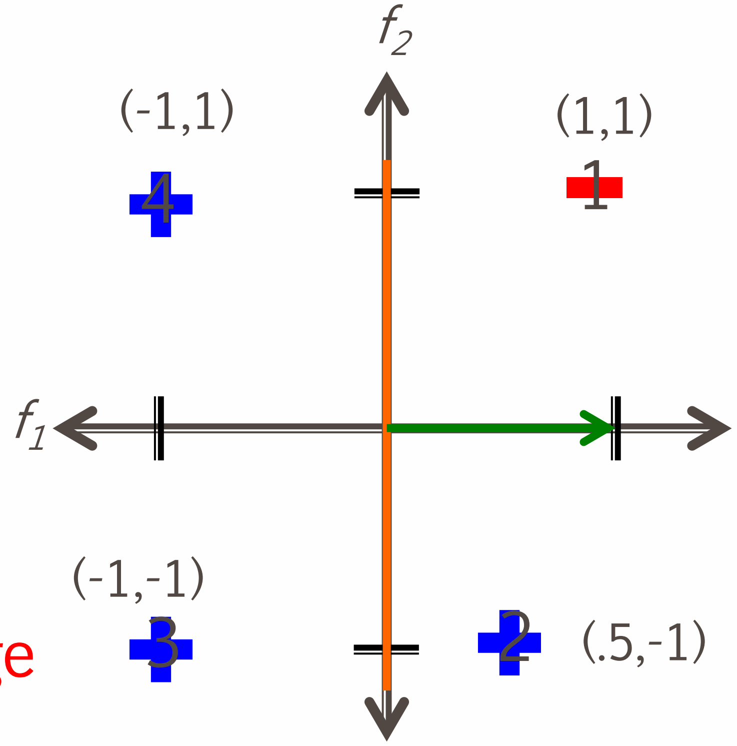
for each training example ($f_1, f_2, \ldots, f_n, label$):

$$prediction = \sum_{i=1}^{n} w_i f_i$$

if $prediction * label \leq 0$: // they don't agree

for each w_i:

$w_i = w_i + f_i * label$



- Repeat until convergence
- Keep track of w_1, w_2 as they change
- Redraw the line after each step

w = (1, 0)

Your turn ☺

repeat until convergence (or for some # of iterations):

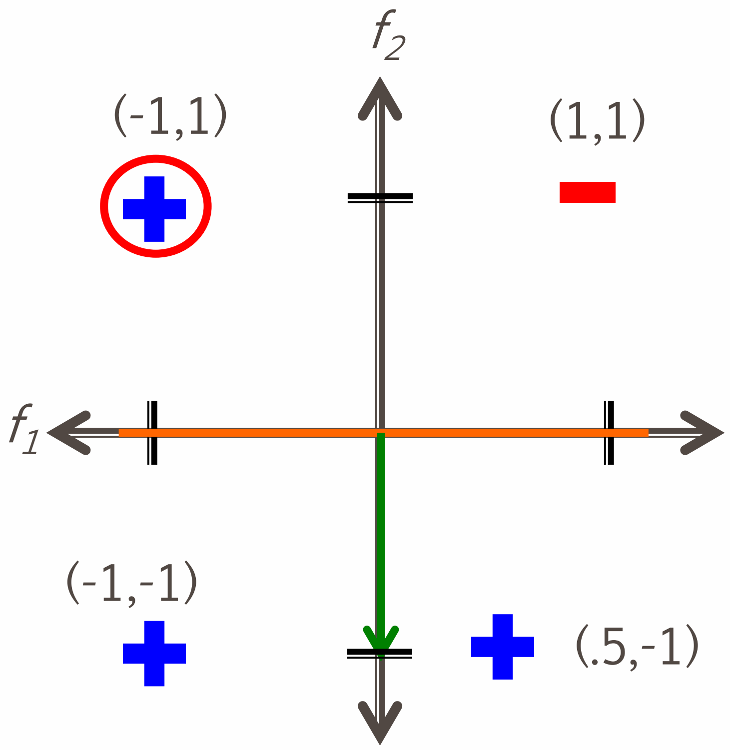
for each training example ($f_1, f_2, \ldots, f_n, label$):

$$prediction = \sum_{i=1}^{n} w_i f_i$$

if $prediction * label \leq 0$: // they don't agree

for each w_i:

$w_i = w_i + f_i * label$



w = (0, -1)

Your turn ☺

repeat until convergence (or for some # of iterations):

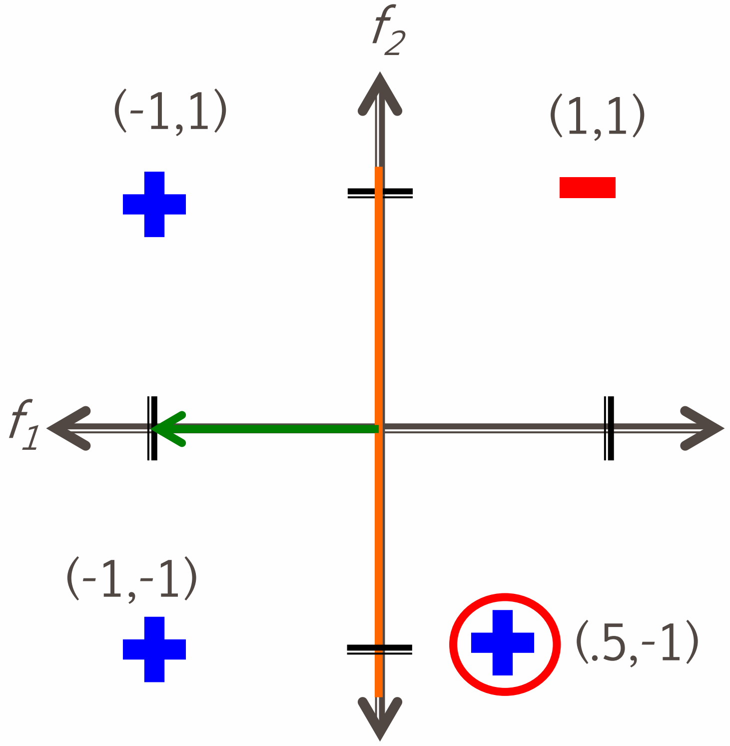
for each training example ($f_1, f_2, \ldots, f_n, label$):

$$prediction = \sum_{i=1}^{n} w_i f_i$$

if $prediction * label \leq 0$: // they don't agree

for each w_i:

$w_i = w_i + f_i * label$



w = (-1, 0)

Your turn ☺

repeat until convergence (or for some # of iterations):

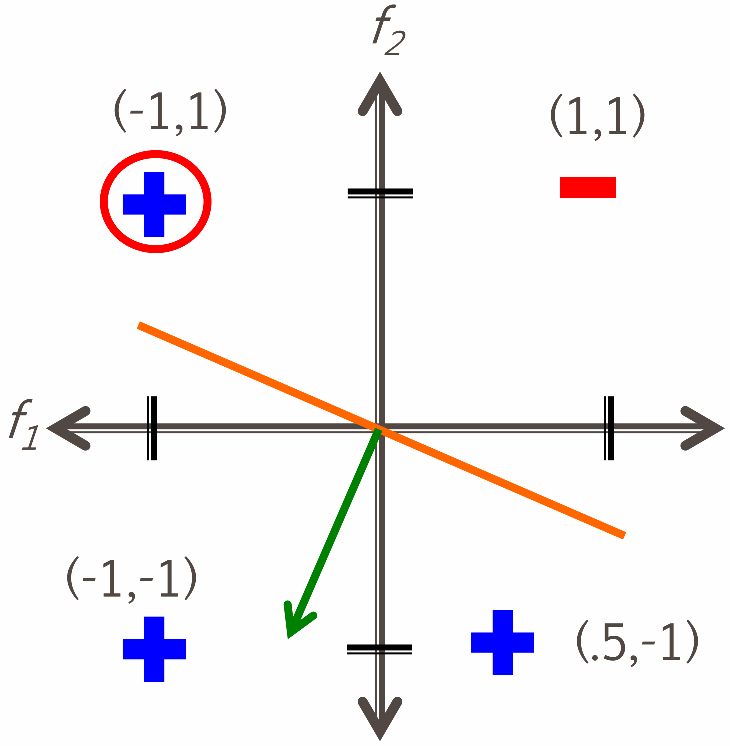
for each training example $(f_1, f_2, \ldots, f_n, label)$:

$$prediction = \sum_{i=1}^{n} w_i f_i$$

if $prediction * label \leq 0$: // they don't agree

for each w_i:

$w_i = w_i + f_i * label$



w = (-.5, -1)

Your turn ☺

repeat until convergence (or for some # of iterations):

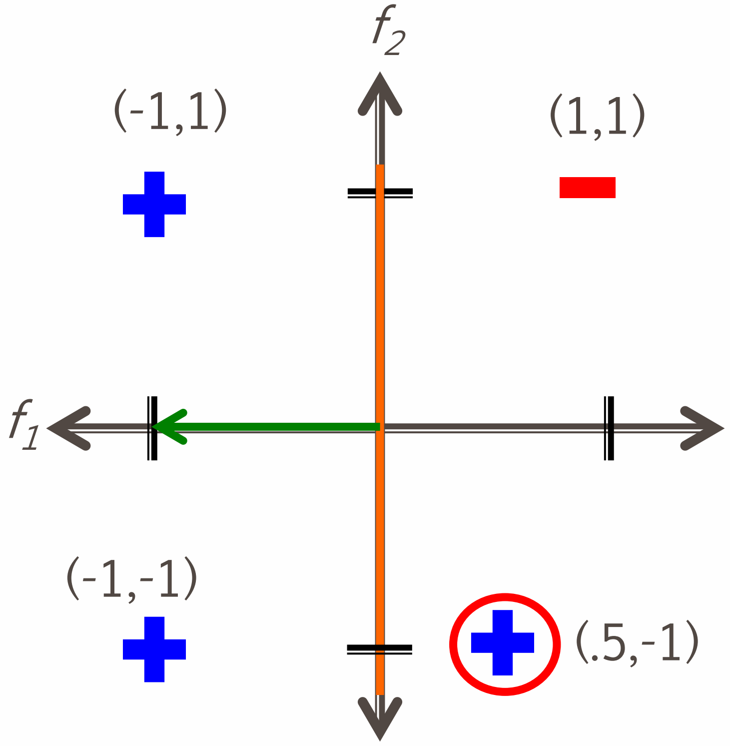
for each training example ($f_1, f_2, \ldots, f_n, label$):

$$prediction = \sum_{i=1}^{n} w_i f_i$$

if $prediction * label \leq 0$: // they don't agree

for each w_i:

$w_i = w_i + f_i * label$



w = (-1.5, 0)

Your turn ☺

repeat until convergence (or for some # of iterations):

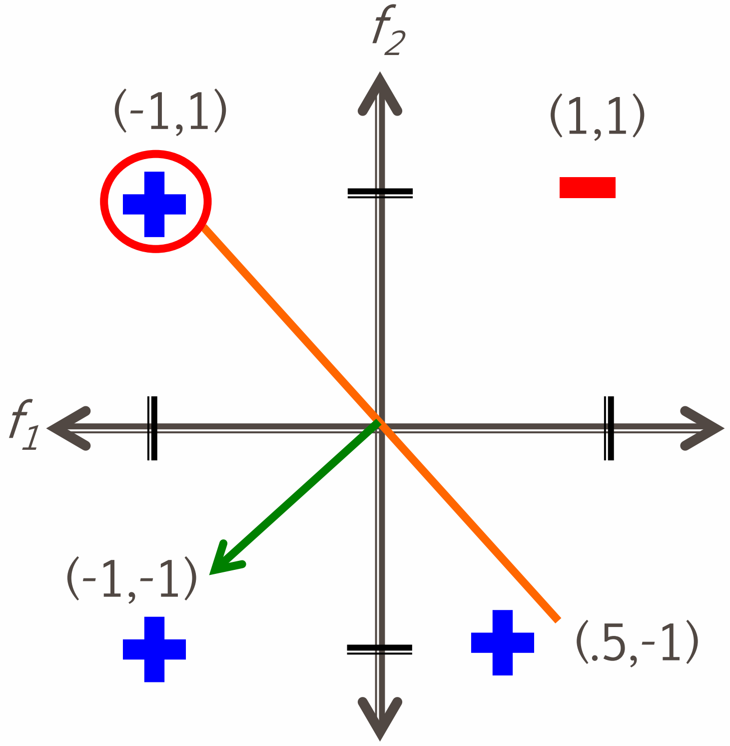
for each training example ($f_1, f_2, \ldots, f_n, label$):

$$prediction = \sum_{i=1}^{n} w_i f_i$$

if $prediction * label \leq 0$: // they don't agree

for each w_i:

$w_i = w_i + f_i * label$



w = (-1, -1)

Your turn ☺

repeat until convergence (or for some # of iterations):

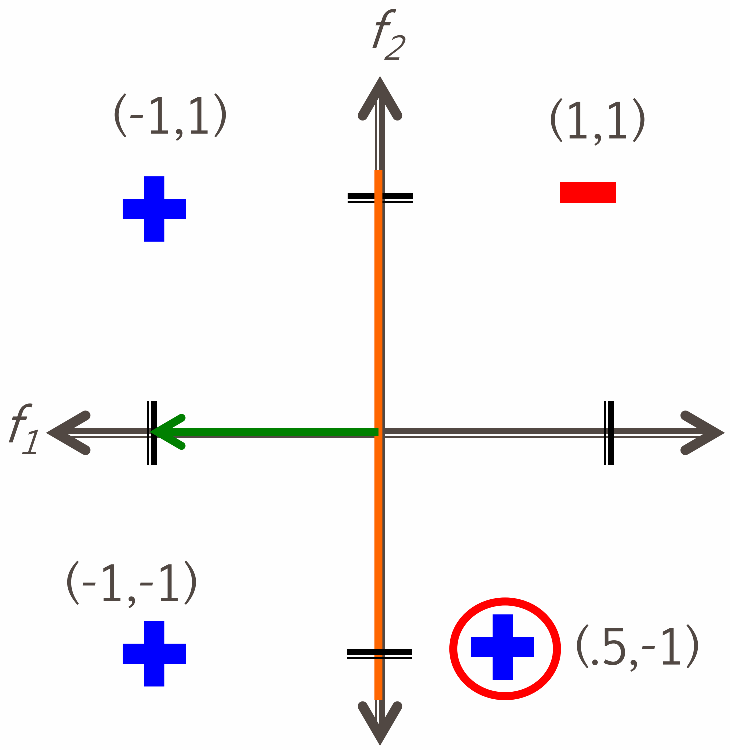
for each training example ($f_1, f_2, \ldots, f_n, label$):

$$prediction = \sum_{i=1}^{n} w_i f_i$$

if $prediction * label \leq 0$: // they don't agree

for each w_i:

$w_i = w_i + f_i * label$



w = (-2, 0)

Your turn ☺

repeat until convergence (or for some # of iterations):

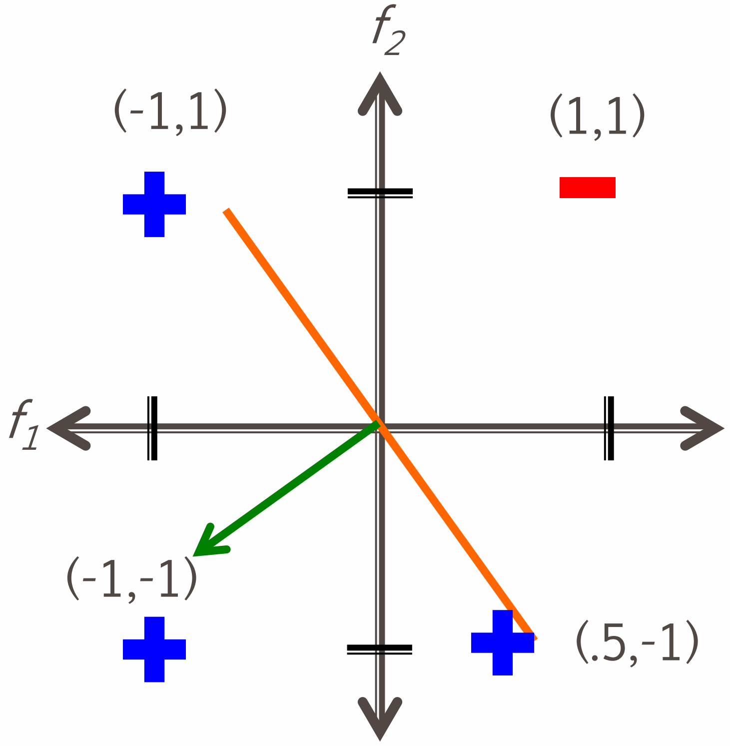
for each training example ($f_1, f_2, \ldots, f_n, label$):

$$prediction = \sum_{i=1}^{n} w_i f_i$$

if $prediction * label \leq 0$: // they don't agree

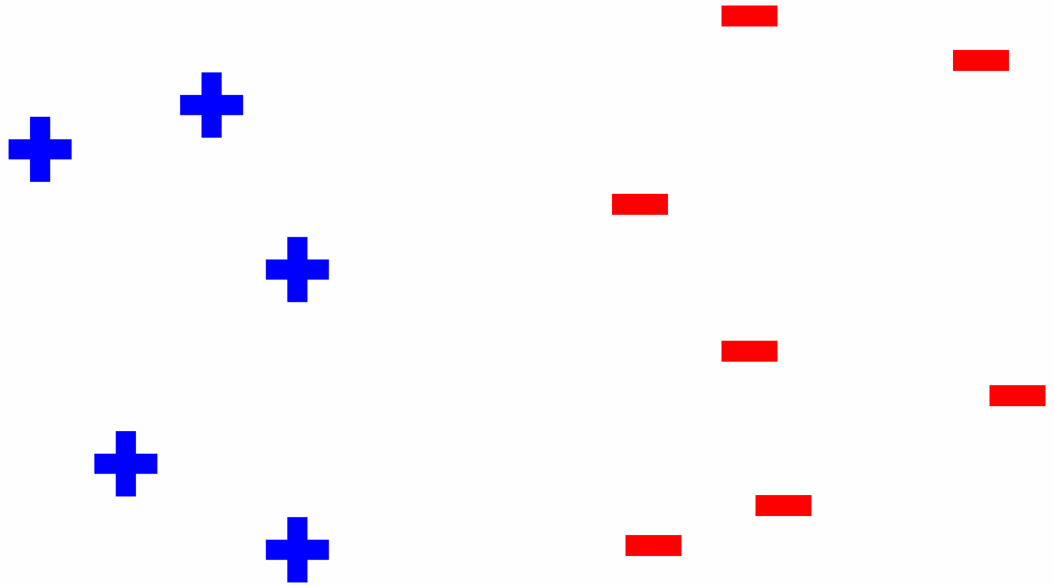
for each w_i:

$w_i = w_i + f_i * label$

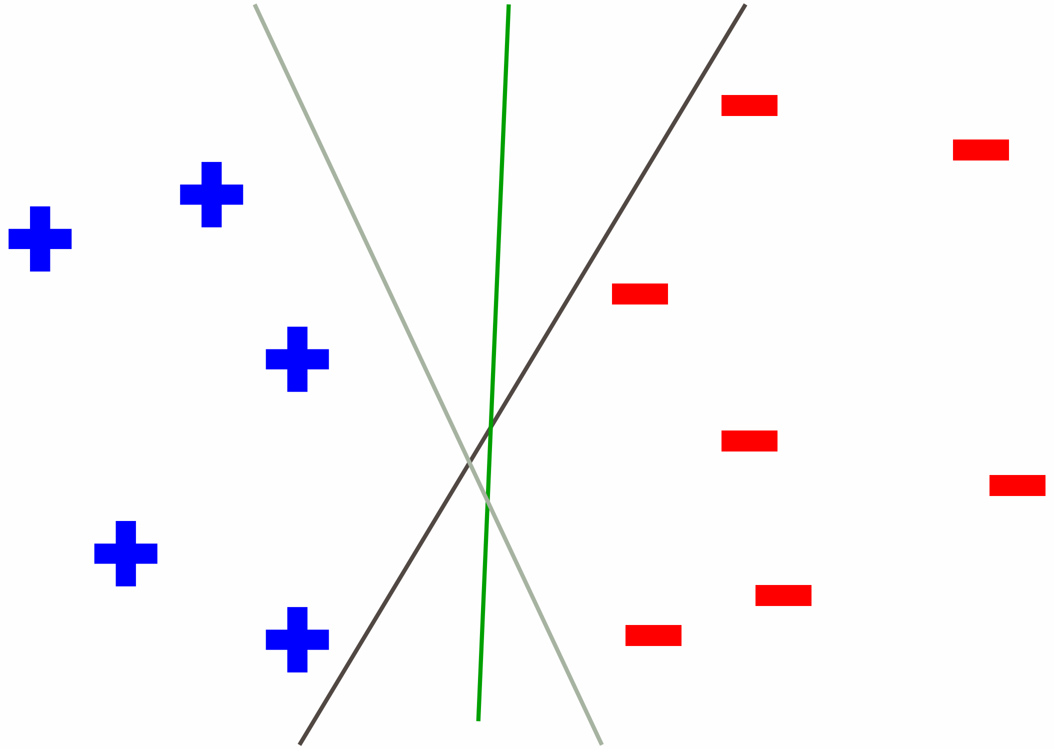


w = (-1.5, -1)

Which line will it find?



Which line will it find?



Only guaranteed to find ***some*** line that separates the data

Convergence

repeat until convergence (or for some # of iterations):

for each training example ($f_1, f_2, \ldots, f_n$, label):

$$prediction = b + \sum_{i=1}^{n} w_i f_i$$

if $prediction * label \leq 0$: // they don't agree

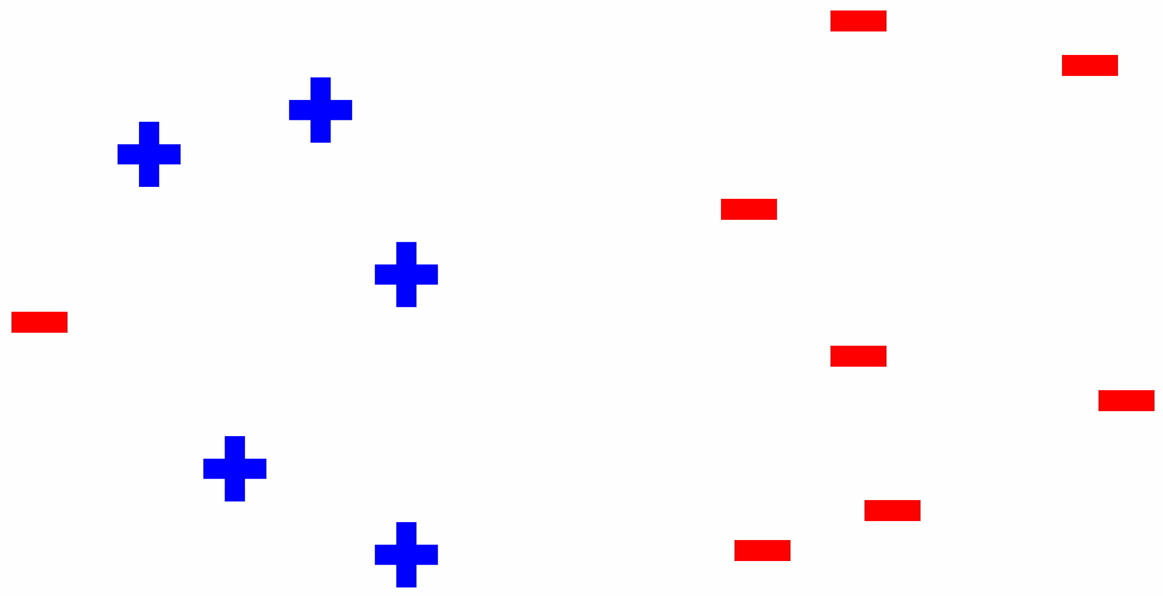
for each w_i:

$w_i = w_i + f_i$*label

$b = b$ + label

Why do we also have the "some # iterations" check?

Handling non-separable data



If we ran the algorithm on this it would never converge!

Convergence

repeat until convergence (or for some # of iterations):

for each training example ($f_1, f_2, \ldots, f_n, label$):

$$prediction = b + \sum_{i=1}^{n} w_i f_i$$

if $prediction * label \leq 0$: // they don't agree

for each w_i:

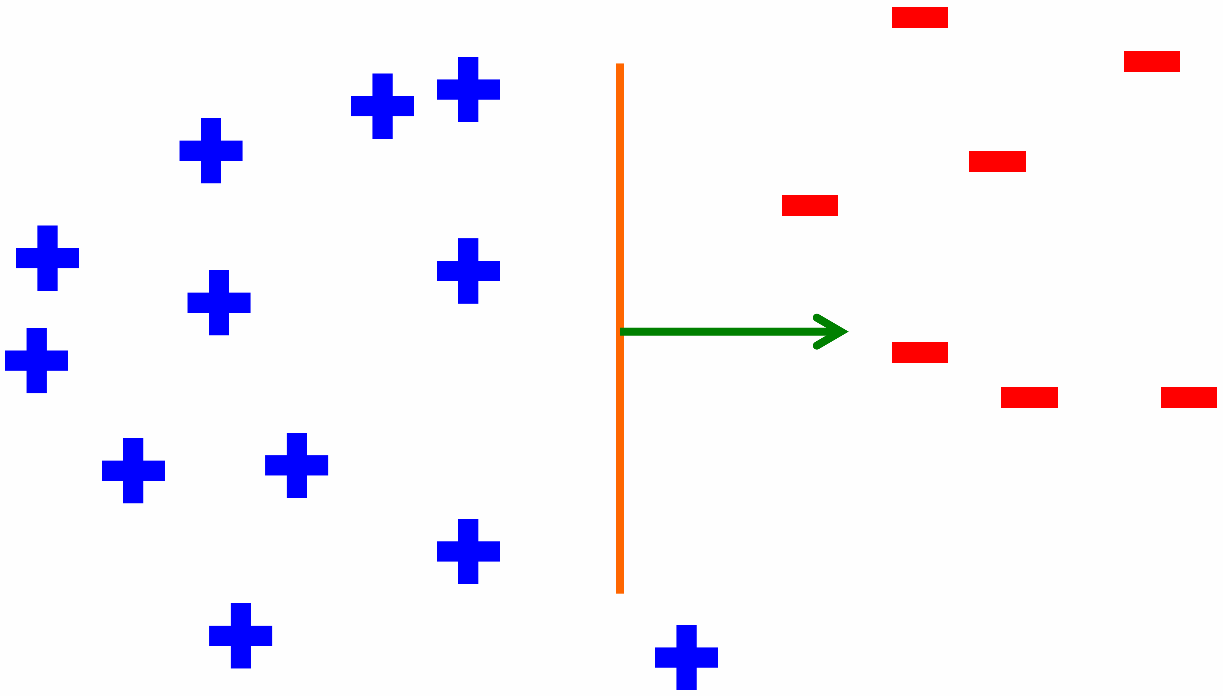
$w_i = w_i + f_i * label$

$b = b + label$

Also helps avoid overfitting!
(This is harder to see in 2-D examples, though)

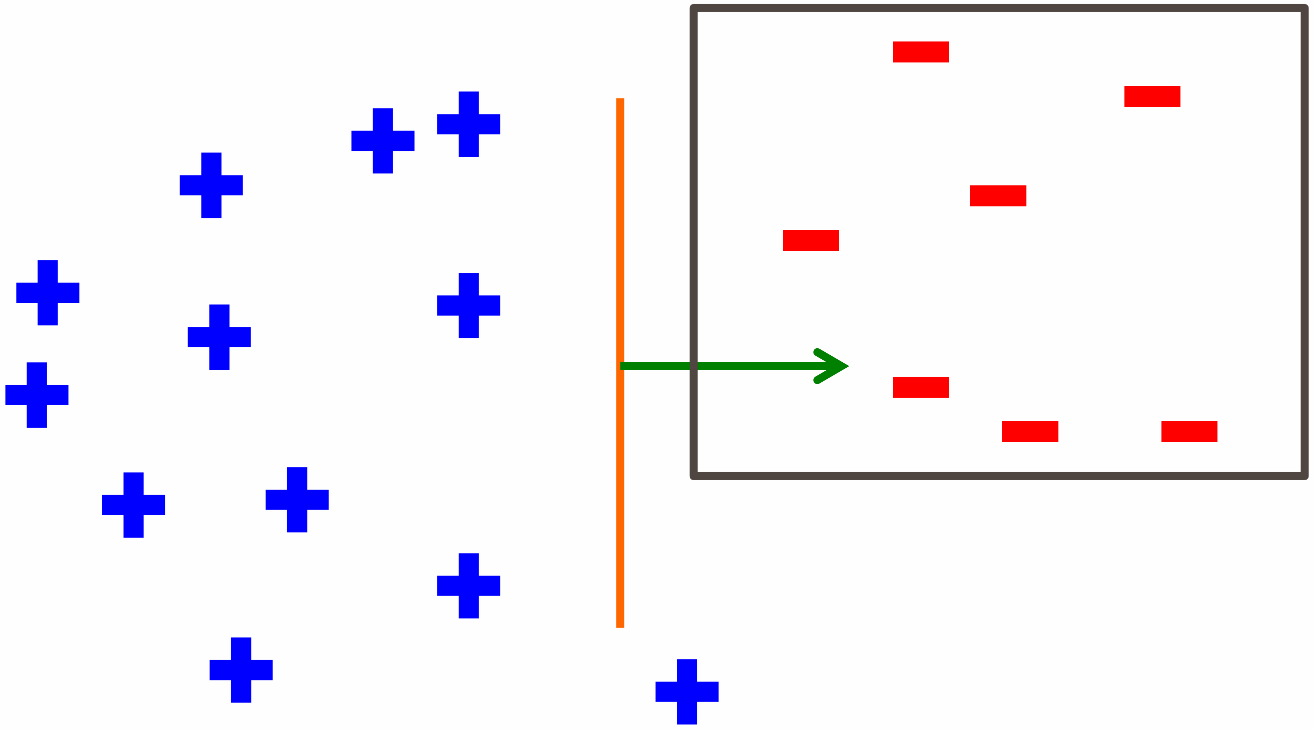
What order should we traverse the examples?
Does it matter?

Order matters

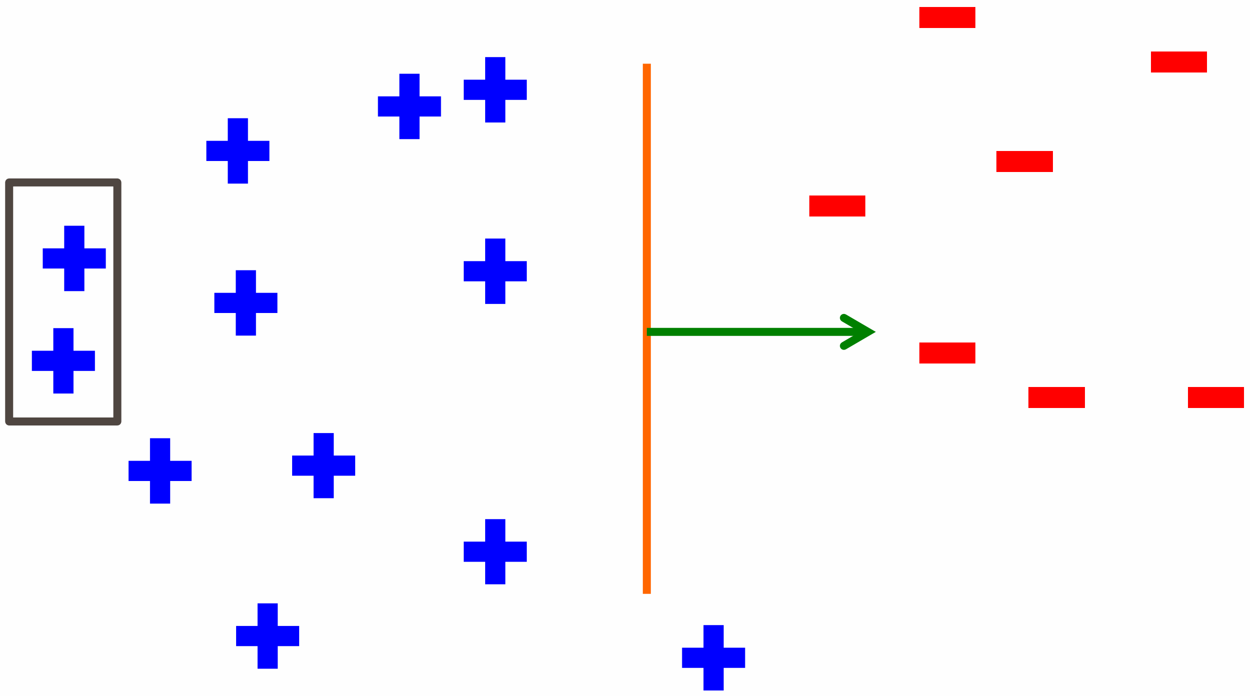


What would be a good/bad order?

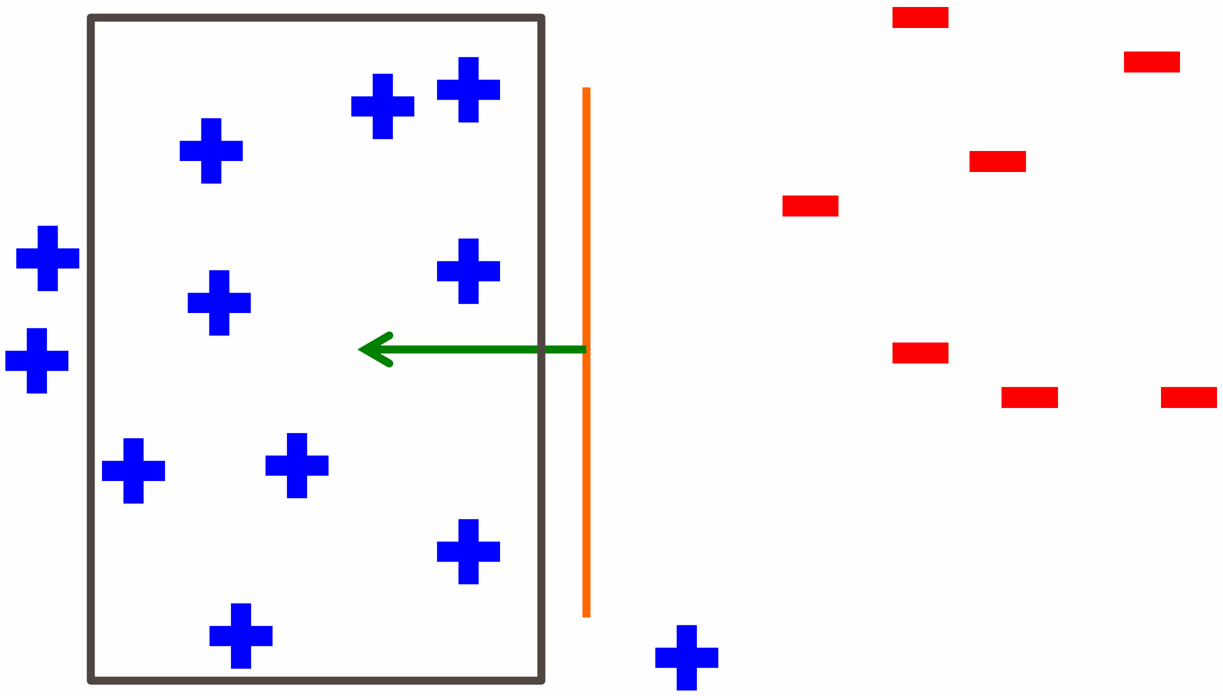
Order matters: a bad order



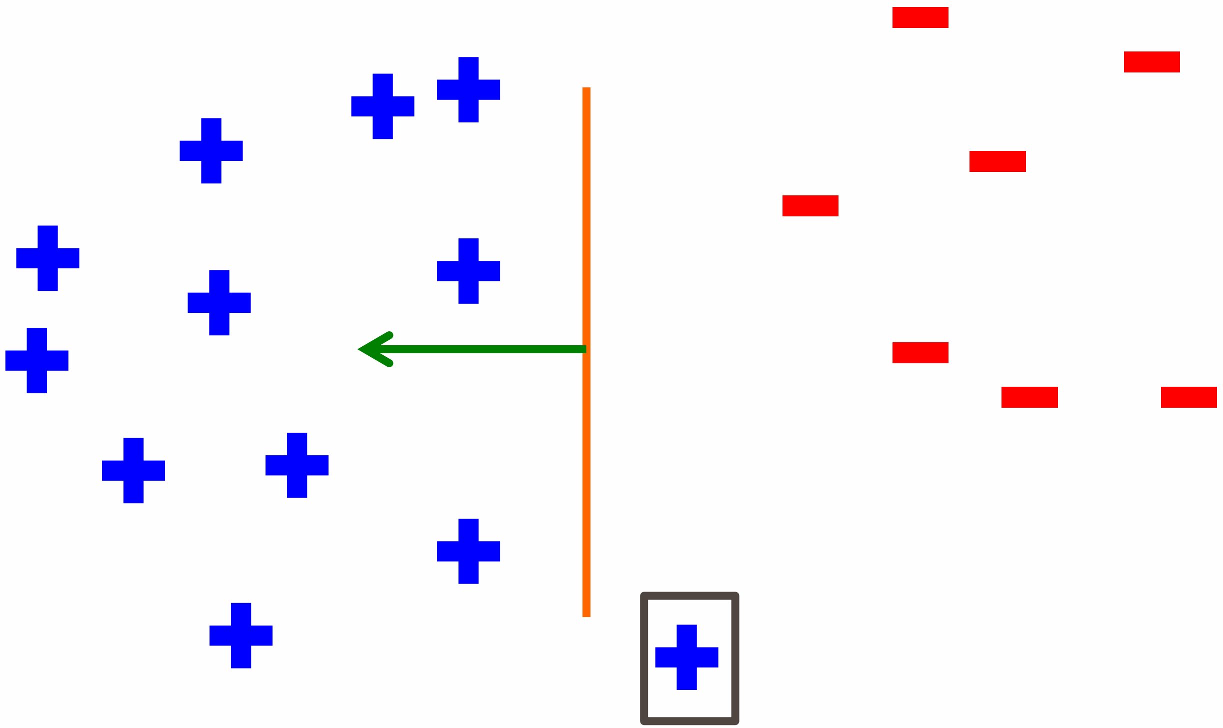
Order matters: a bad order



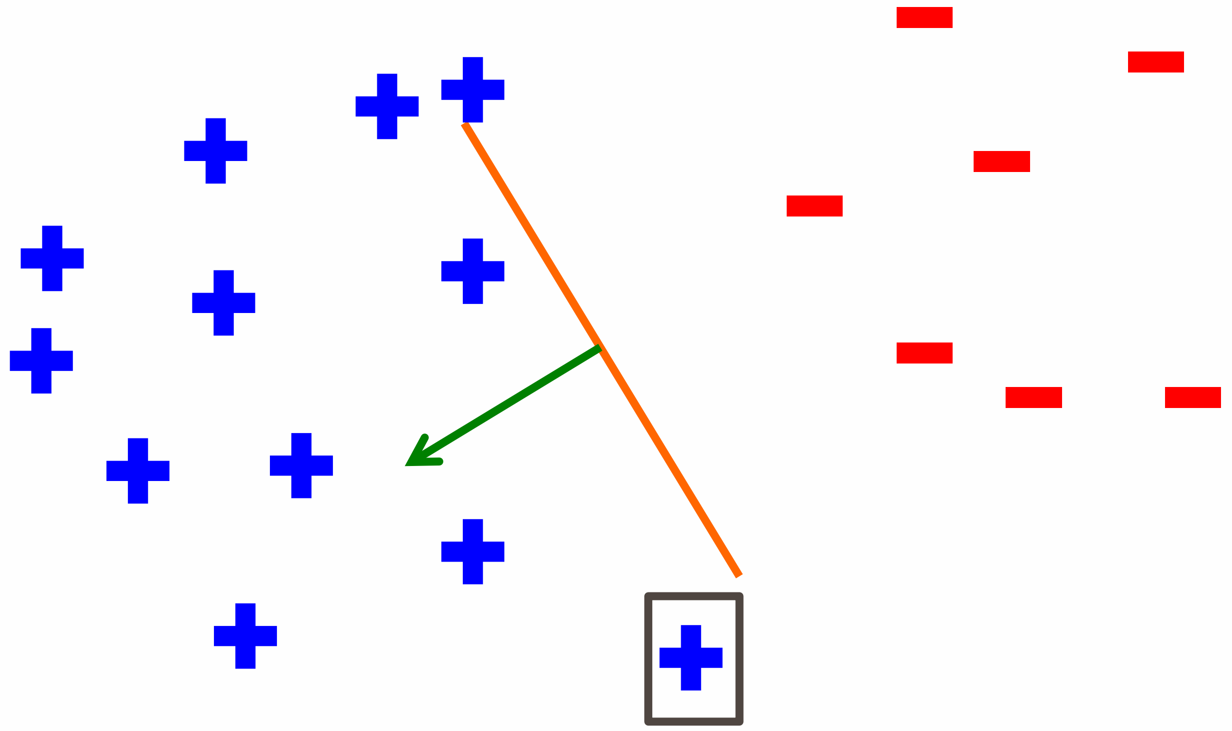
Order matters: a bad order



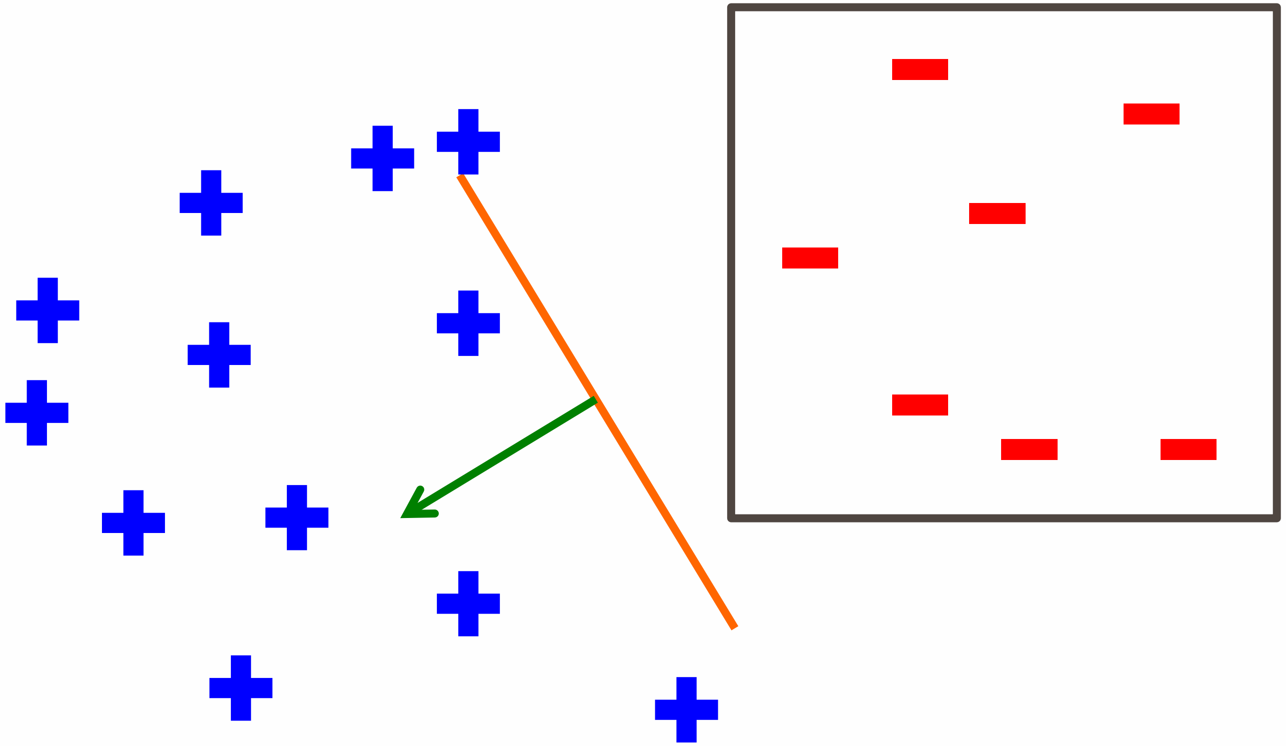
Order matters: a bad order



Order matters: a bad order



Order matters: a bad order



Solution?

Ordering

repeat until convergence (or for some # of iterations):

randomize order or training examples

for each training example ($f_1, f_2, \ldots, f_n, label$):

$$prediction = b + \sum_{i=1}^{n} w_i f_i$$

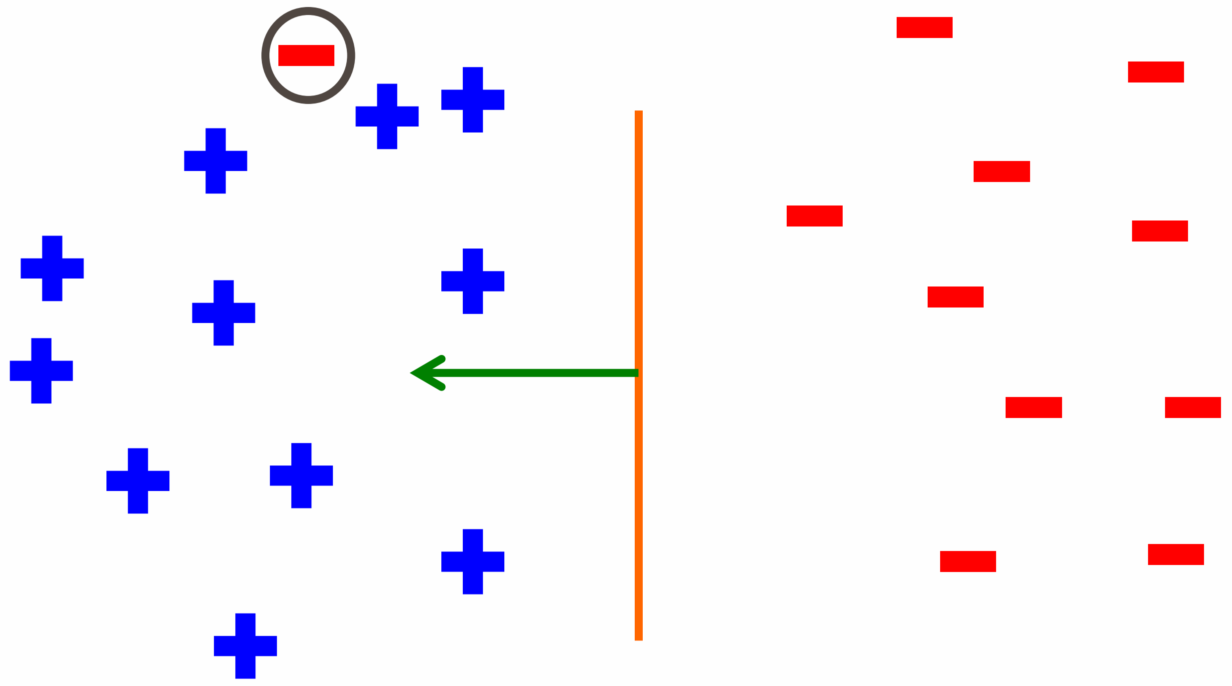
if $prediction * label \leq 0$: // they don't agree

for each w_i:

$w_i = w_i + f_i * label$

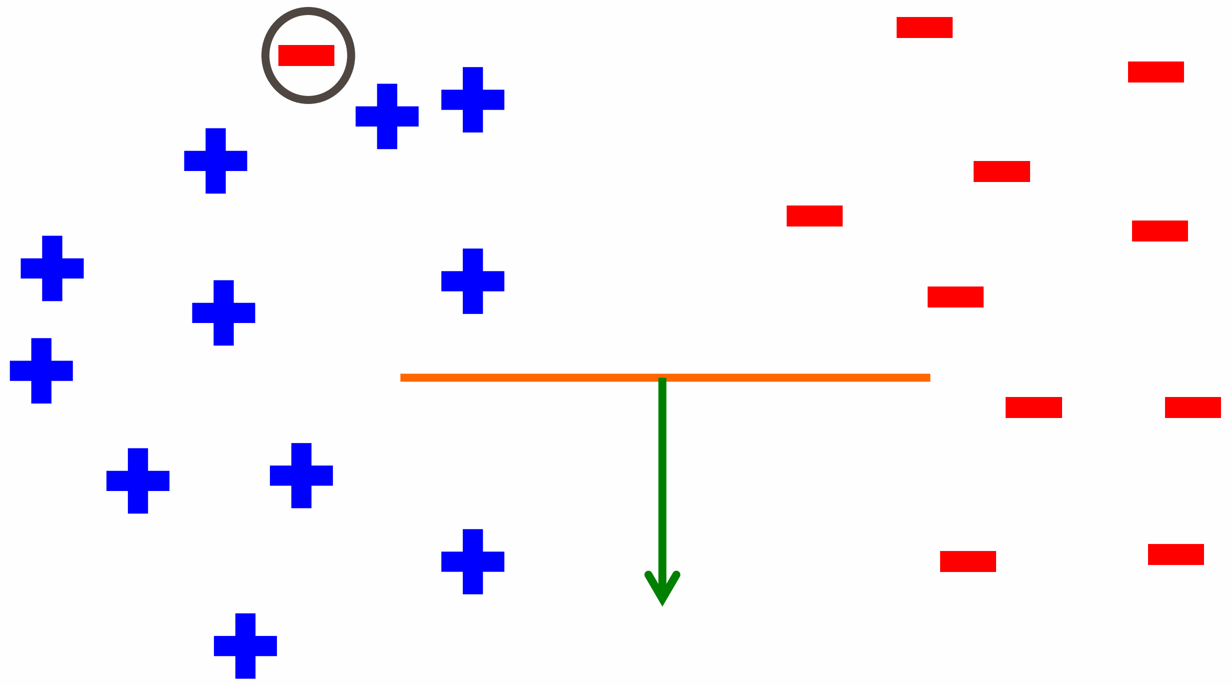
$b = b + label$

Improvements



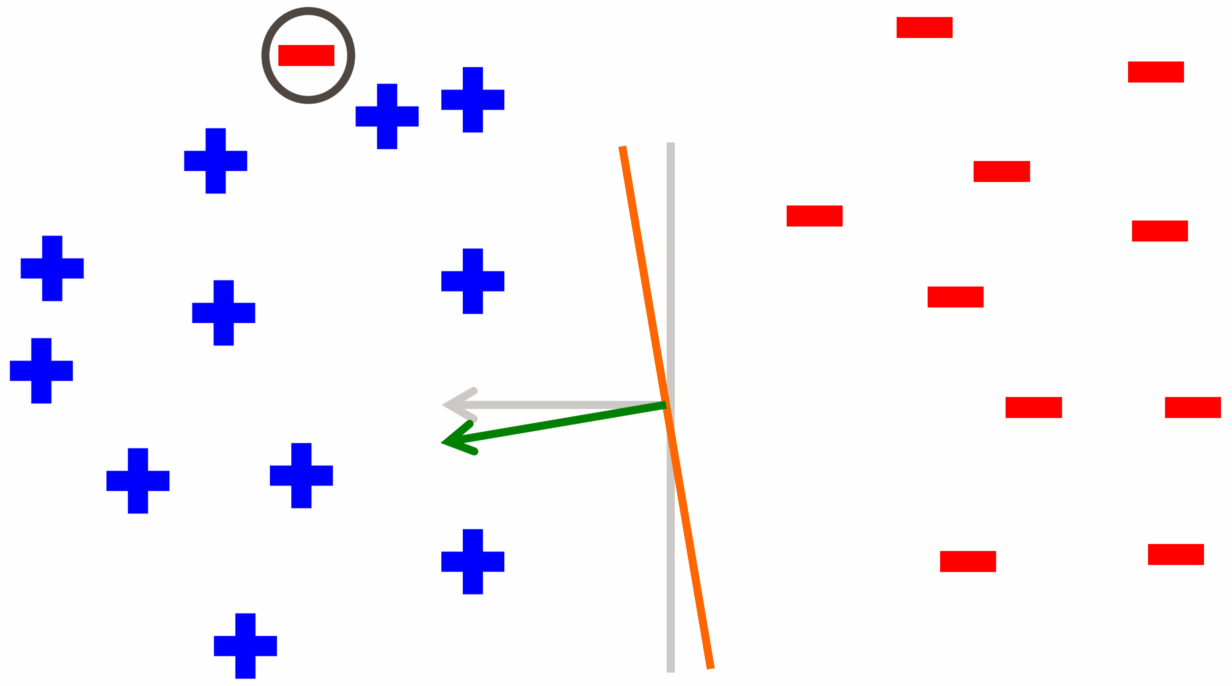
What will happen when we examine this example?

Improvements



Does this make sense? What if we had previously gone through ALL of the other examples correctly?

Improvements



Maybe just move it slightly in the direction of correction

Learning rate: Move slightly

repeat until convergence (or for some # of iterations):

randomize order or training examples

for each training example ($f_1, f_2, \ldots, f_n, label$):

$$prediction = b + \sum_{i=1}^{n} w_i f_i$$

if $prediction * label \leq 0$: // they don't agree

for each w_i:

$w_i = w_i + lr * (f_i * label)$

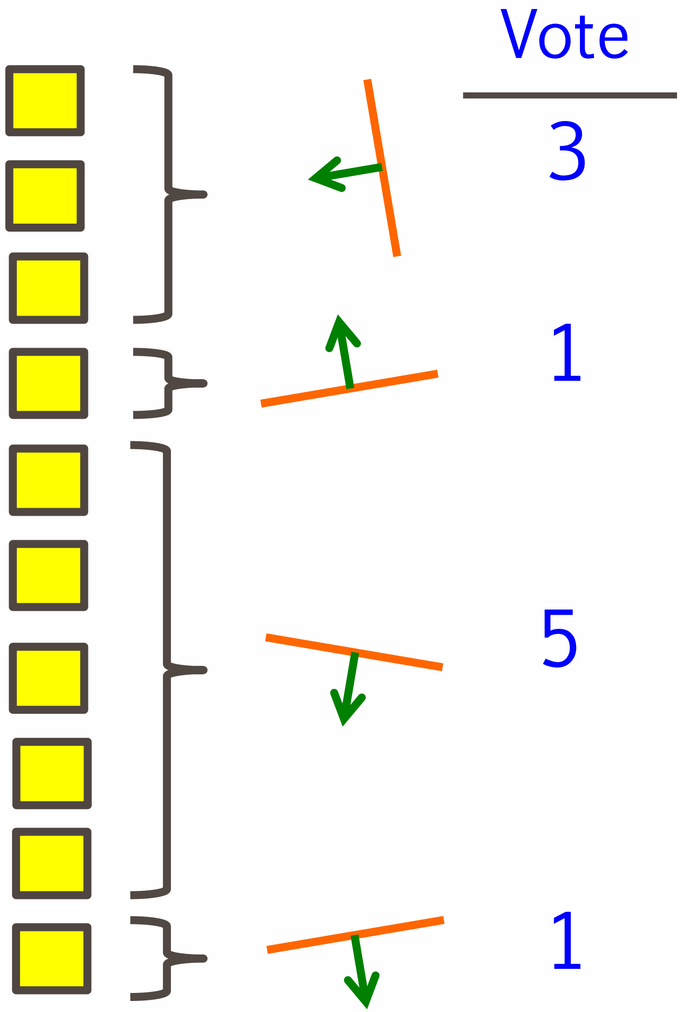
$b = b + label$

VOTED PERCEPTRON

Voted perceptron learning

- Training
  - every time a mistake is made on an example:
    - store the weights (i.e. before changing for current example)
    - store the number of examples that set of weights got correct
- Classify
  - calculate the prediction from ALL saved weights
  - multiply each prediction by the number it got correct (i.e a weighted vote) and take the sum over all predictions
  - said another way: pick whichever prediction has the most votes

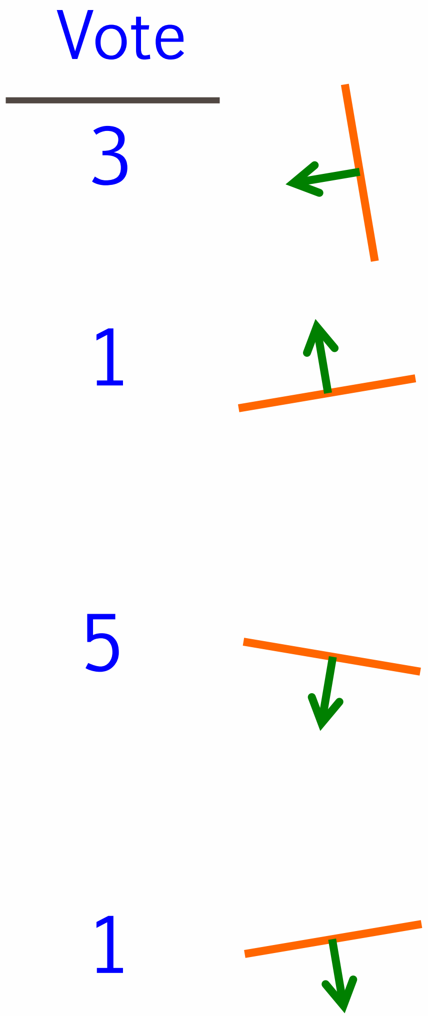
Voted perceptron learning



Training
every time a mistake is made on an example:

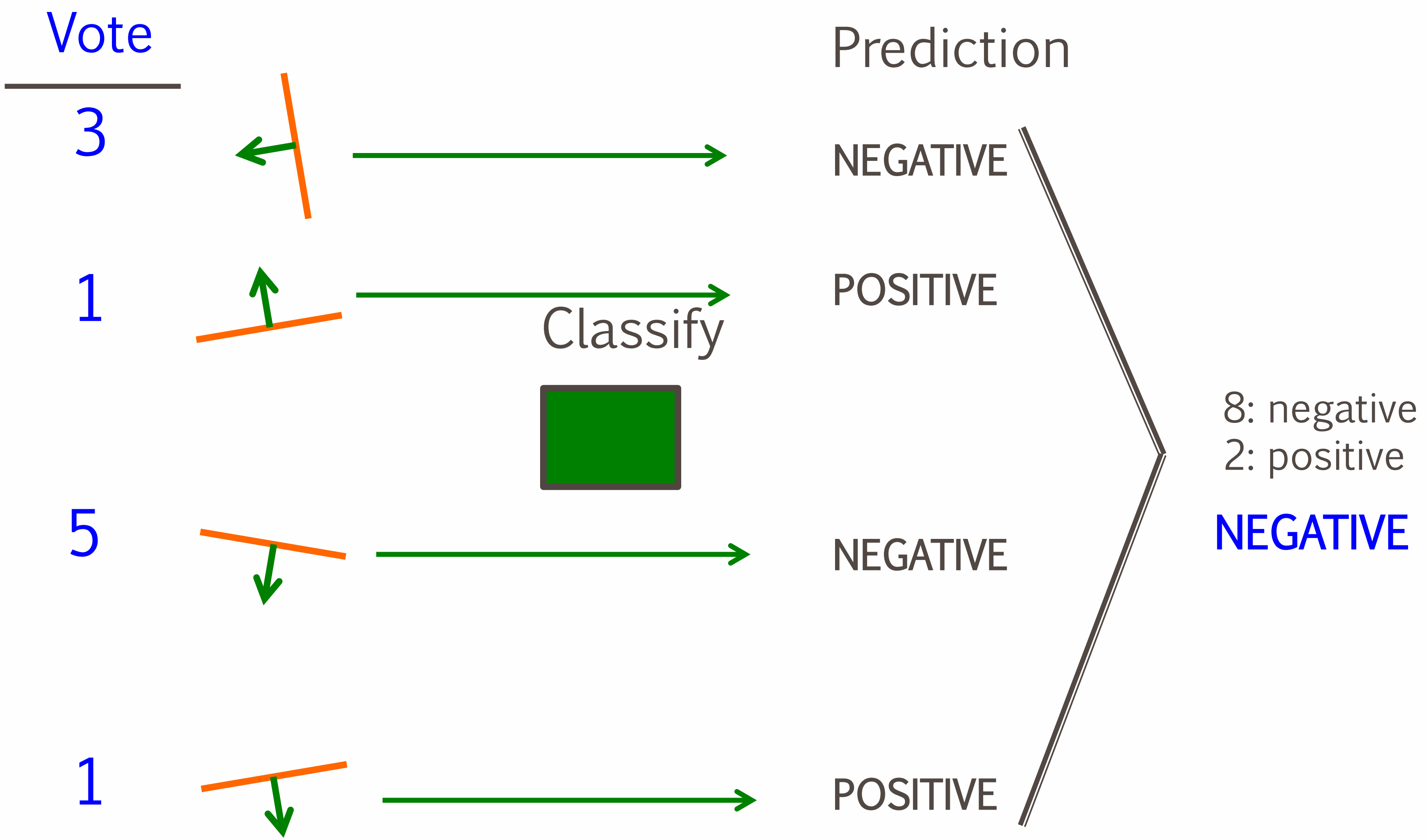
- store the weights
- store the number of examples that set of weights got correct

Voted perceptron learning



Classify

Voted perceptron learning



Voted perceptron learning

- Works much better in practice
- Avoids overfitting, though it can still happen
- Avoids big changes in the result by examples examined at the end of training

Voted perceptron learning

- Training
  - every time a mistake is made on an example:
    - store the weights (i.e. before changing for current example)
    - store the number of examples that set of weights got correct
- Classify
  - calculate the prediction from ALL saved weights
  - multiply each prediction by the number it got correct (i.e a weighted vote) and take the sum over all predictions
  - said another way: pick whichever prediction has the most votes

Any issues/concerns?

Voted perceptron learning

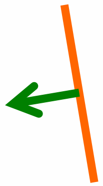
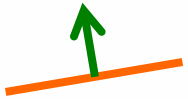
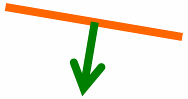
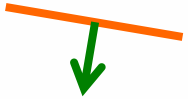
- Training
- every time a mistake is made on an example:
  - store the weights (i.e. before changing for current example)
  - store the number of examples that set of weights got correct

- Classify
  - calculate the prediction from ALL saved weights
  - multiply each prediction by the number it got correct (i.e a weighted vote) and take the sum over all predictions
  - said another way: pick whichever prediction has the most votes

1. Can require a lot of storage
2. Classifying becomes very, very expensive

AVERAGING PERCEPTRON

Average perceptron

Vote		
3		$w_1^1, w_2^1, \ldots, w_n^1, b^1$
1		$w_1^2, w_2^2, \ldots, w_n^2, b^2$
5		$w_1^3, w_2^3, \ldots, w_n^3, b^3$
1		$w_1^4, w_2^4, \ldots, w_n^4, b^4$

$$\overline{w_i} = \frac{3w_i^1 + 1w_i^2 + 5w_i^3 + 1w_i^4}{10}$$

The final weights are the *weighted average* of the previous weights

How does this help us?

Average perceptron

Vote	Weights
3	$w_1^1, w_2^1, \ldots, w_n^1, b^1$
1	$w_1^2, w_2^2, \ldots, w_n^2, b^2$
5	$w_1^3, w_2^3, \ldots, w_n^3, b^3$
1	$w_1^4, w_2^4, \ldots, w_n^4, b^4$

$$\overline{w_i} = \frac{3w_i^1 + 1w_i^2 + 5w_i^3 + 1w_i^4}{10}$$

The final weights are the *weighted average* of the previous weights

Can just keep a running average!

Perceptron learning algorithm

repeat until convergence (or for some # of iterations):

for each training example ($f_1, f_2, \ldots, f_n, label$):

$$prediction = b + \sum_{i=1}^{n} w_i f_i$$

if $prediction * label \leq 0$: // they don't agree

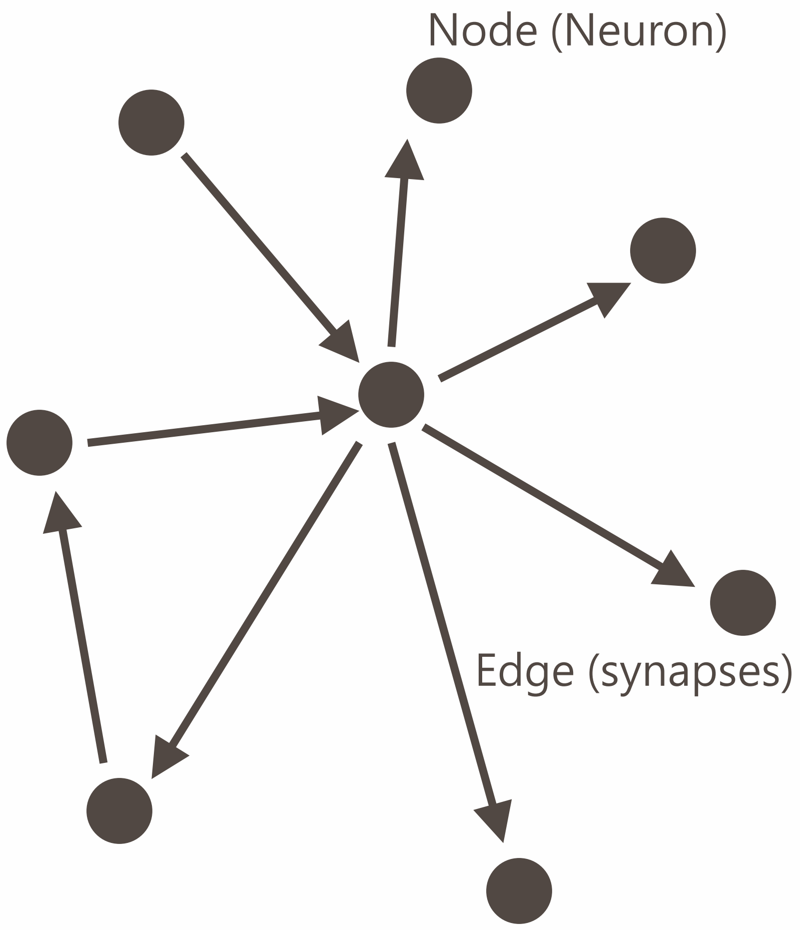
for each w_i:

$w_i = w_i + f_i * label$

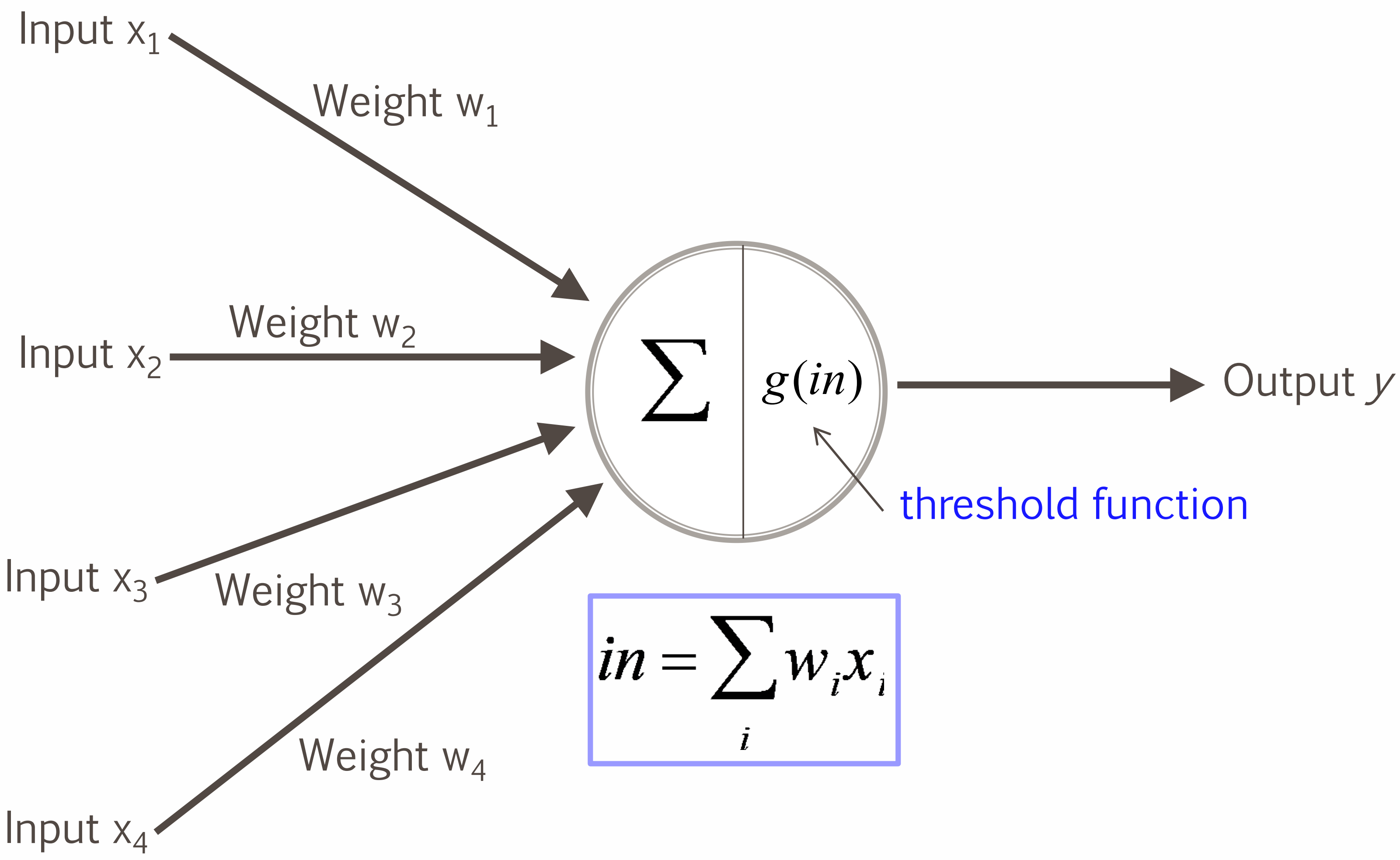
$b = b + label$

Why is it called the "perceptron" learning algorithm if what it learns is a line? Why not "line learning" algorithm?

Neural Networks



A Single Neuron/Perceptron

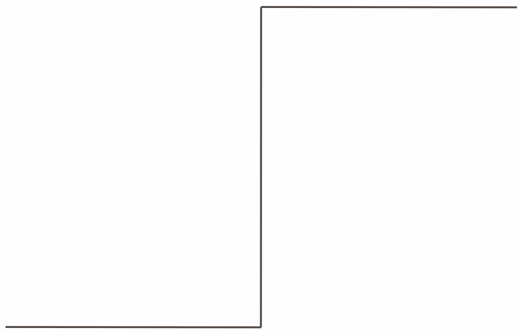


$$\boxed{in = \sum_i w_i x_i}$$

Possible threshold functions

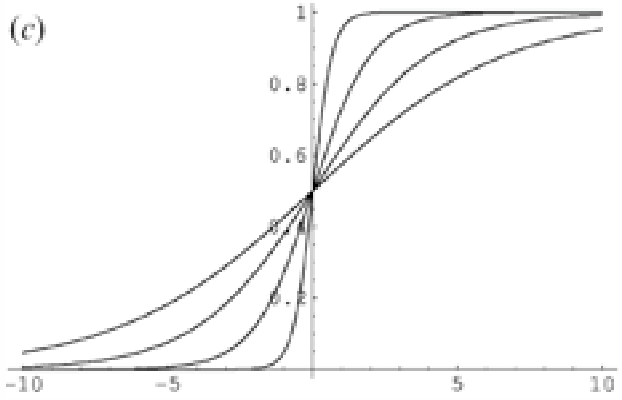
hard threshold:

if *in* (the sum of weights) >= *threshold* 1, 0 otherwise

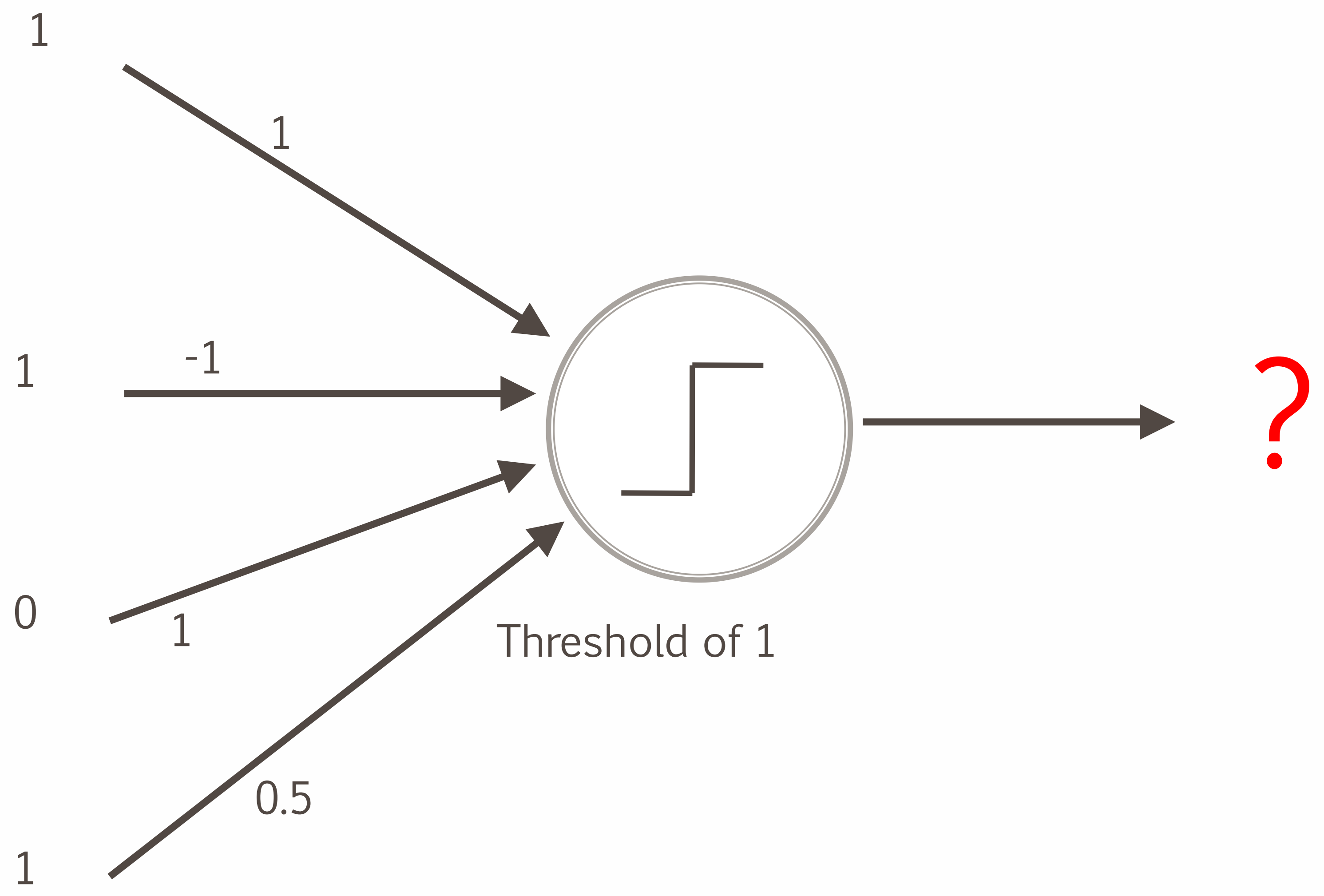


Sigmoid

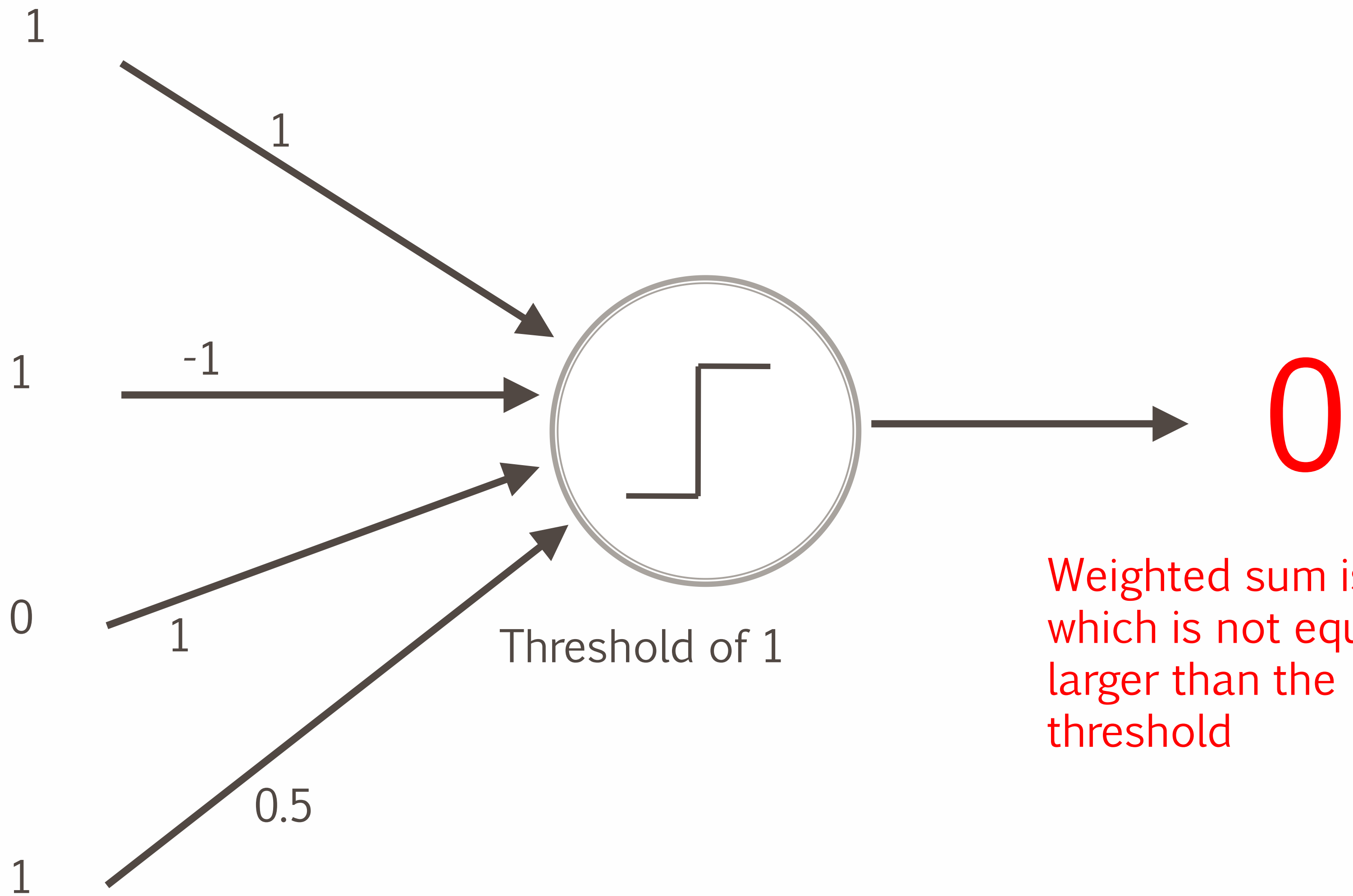
$$g(x) = \frac{1}{1 + e^{-ax}}$$



A Single Neuron/Perceptron

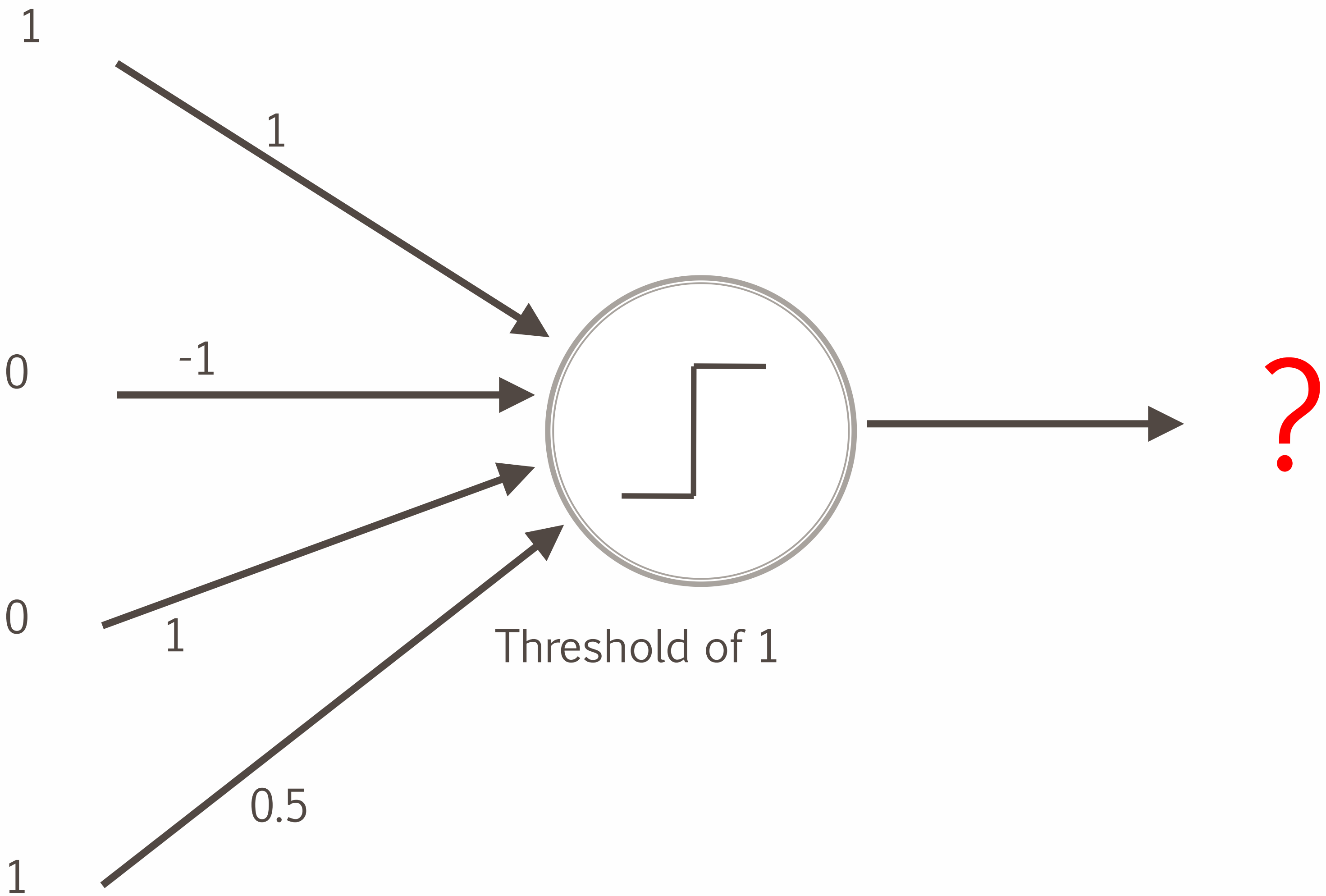


A Single Neuron/Perceptron

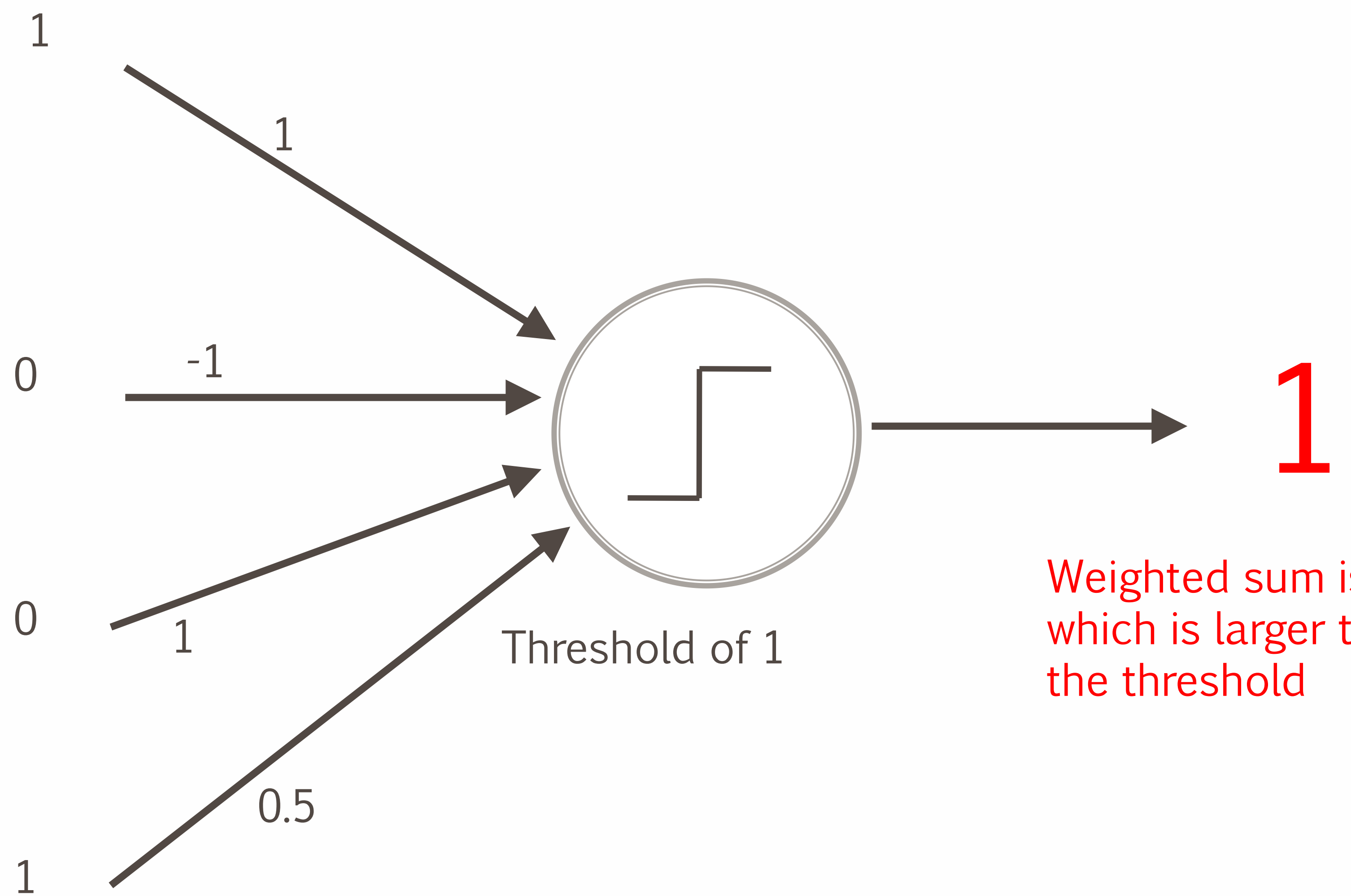


Weighted sum is 0.5, which is not equal or larger than the threshold

A Single Neuron/Perceptron

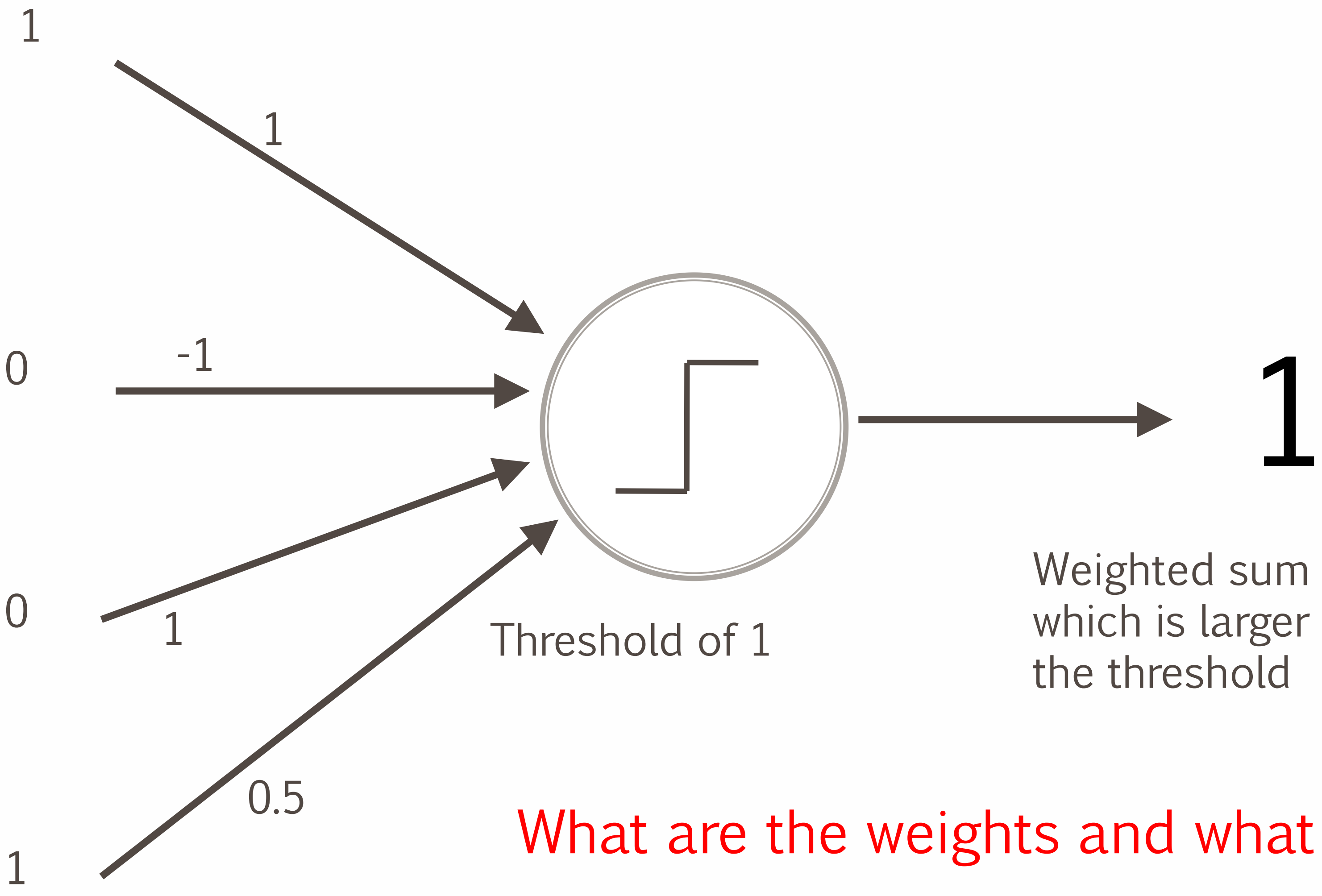


A Single Neuron/Perceptron



Weighted sum is 1.5, which is larger than the threshold

A Single Neuron/Perceptron



Weighted sum is 1.5, which is larger than the threshold

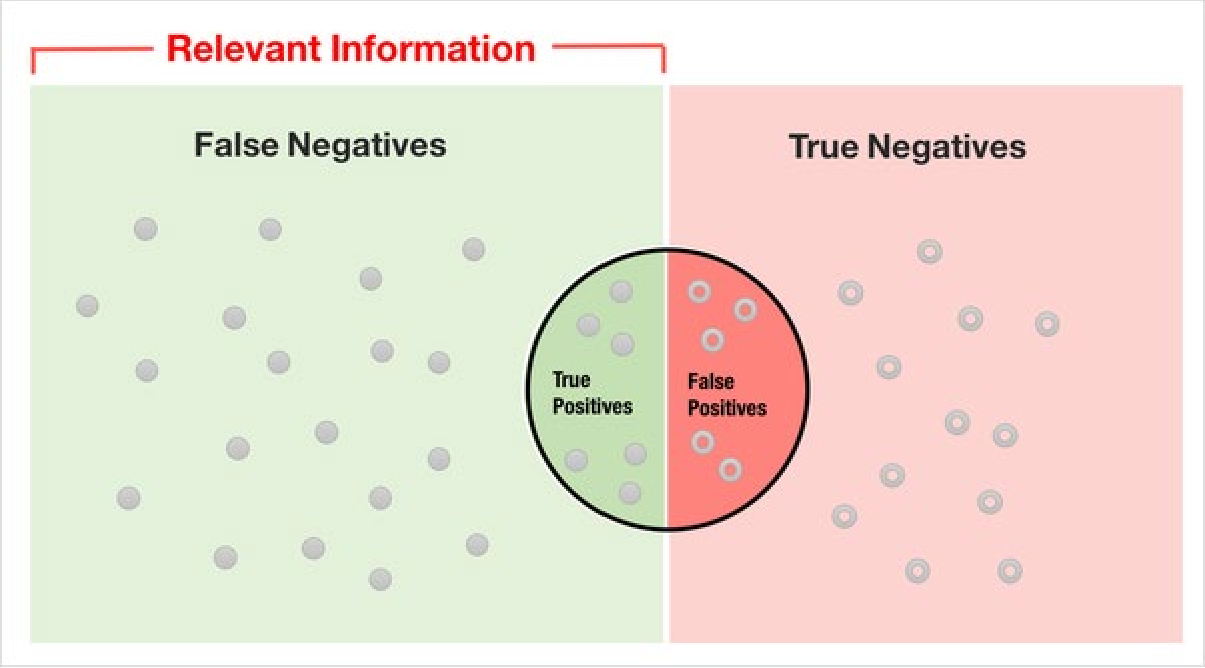
What are the weights and what is b?

PERFORMANCE EVALUATION

Performance Evaluation

Precision	$\frac{\mathrm{TP}}{\mathrm{TP+FP}}$
Accuracy	$\frac{\mathrm{TP+TN}}{\mathrm{TP+TN+FP+FN}}$

		Predicted class P	Predicted class N
Actual Class	P	True Positives (TP)	False Negatives (FN)
Actual Class	N	False Positives (FP)	True Negatives (TN)

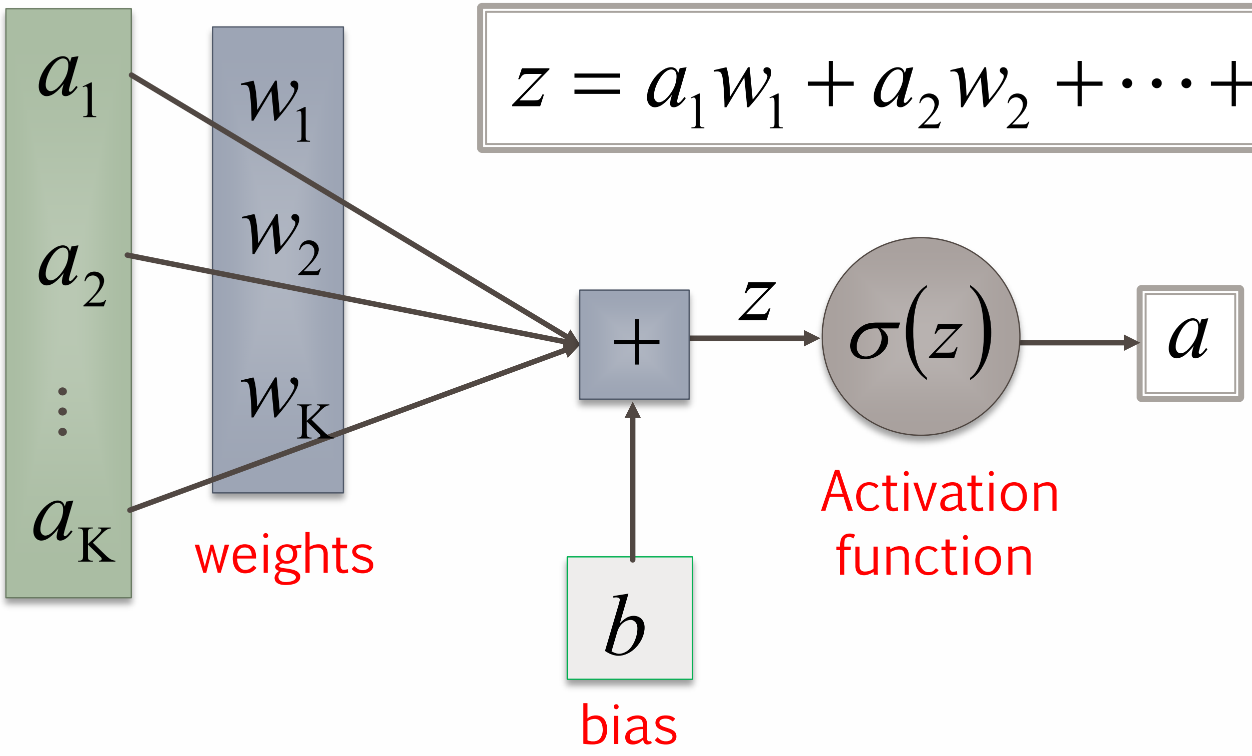


NEURAL NETWORKS

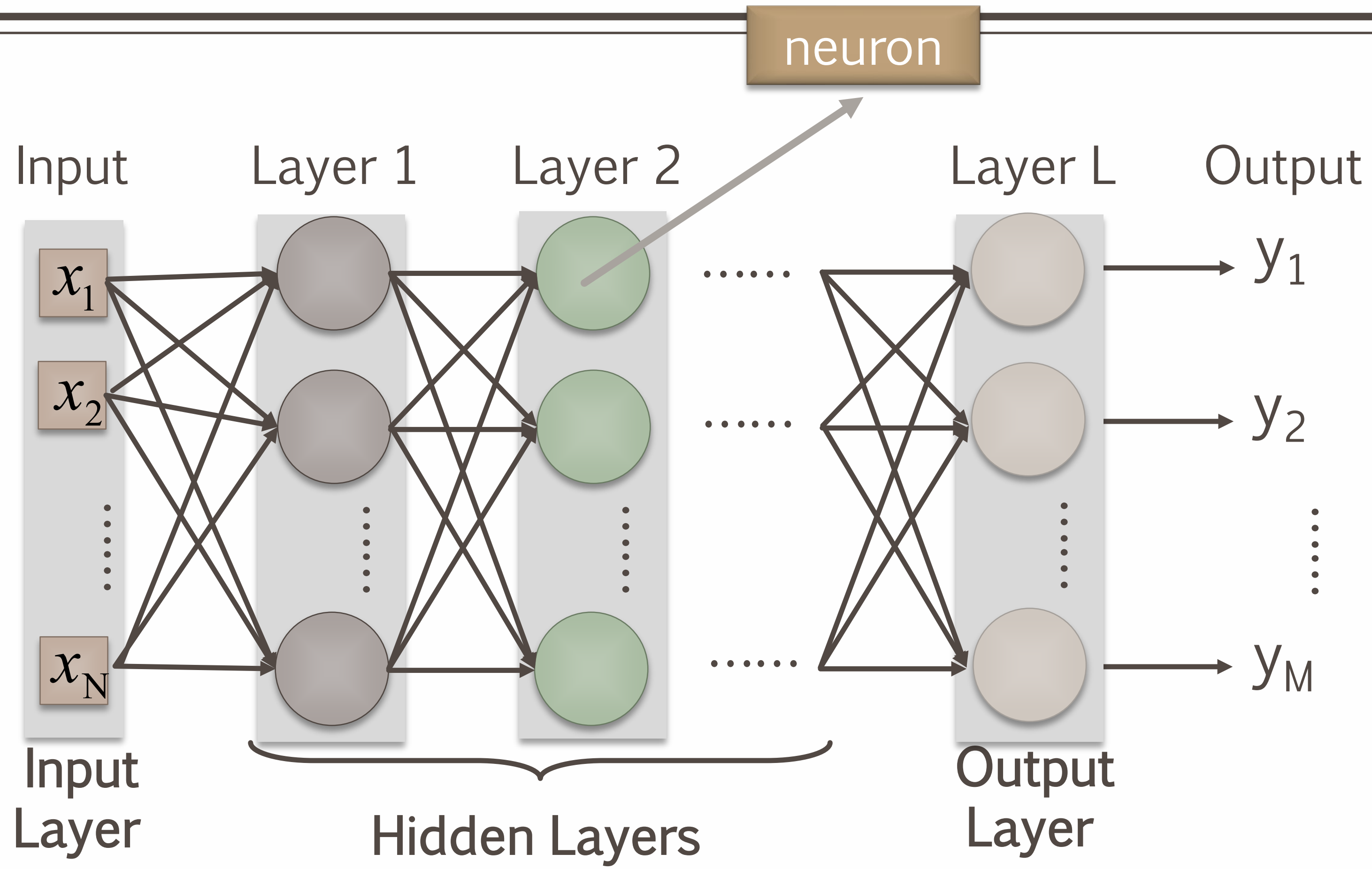
Element of Neural Network

Neuron $f: R^K \to R$

$$z = a_1 w_1 + a_2 w_2 + \cdots + a_K w_K + b$$

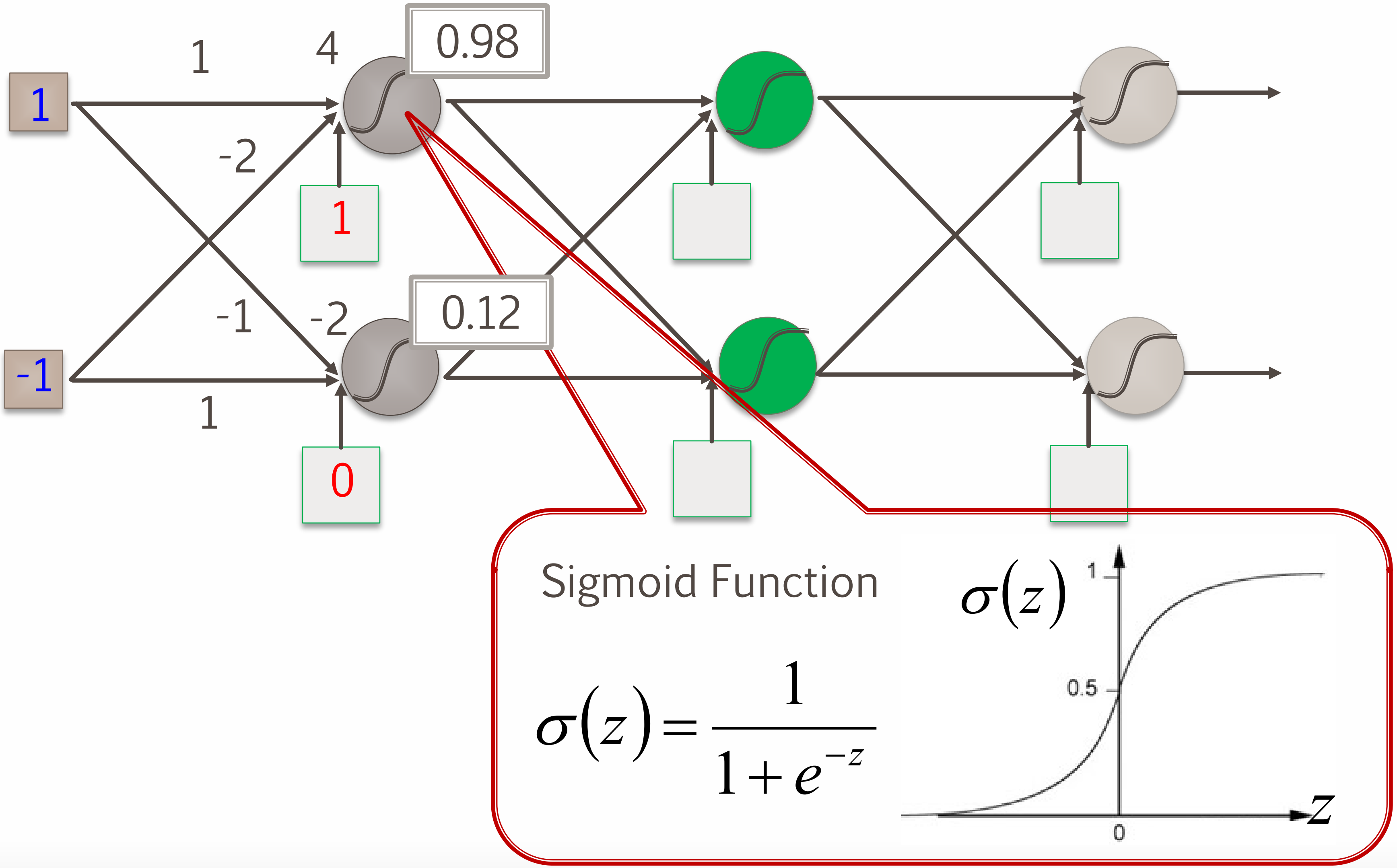


Neural Network



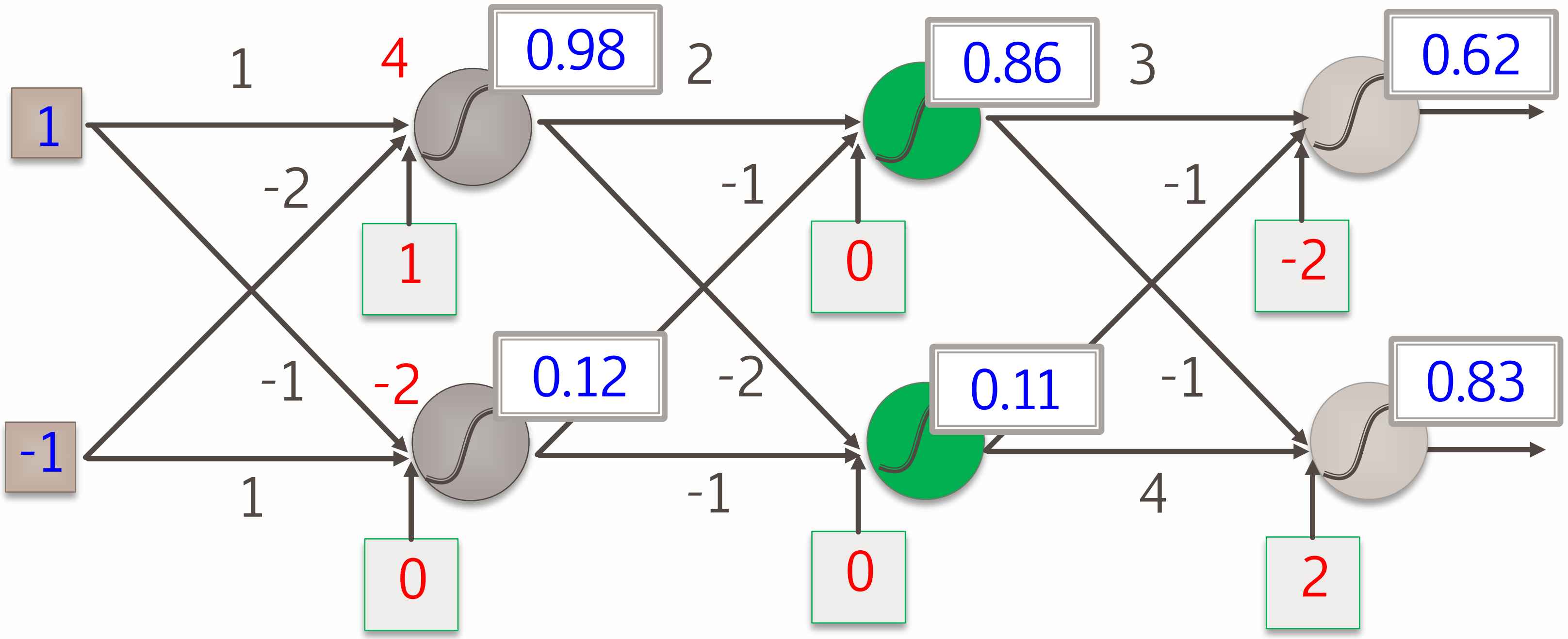
Deep means many hidden layers

Activation function

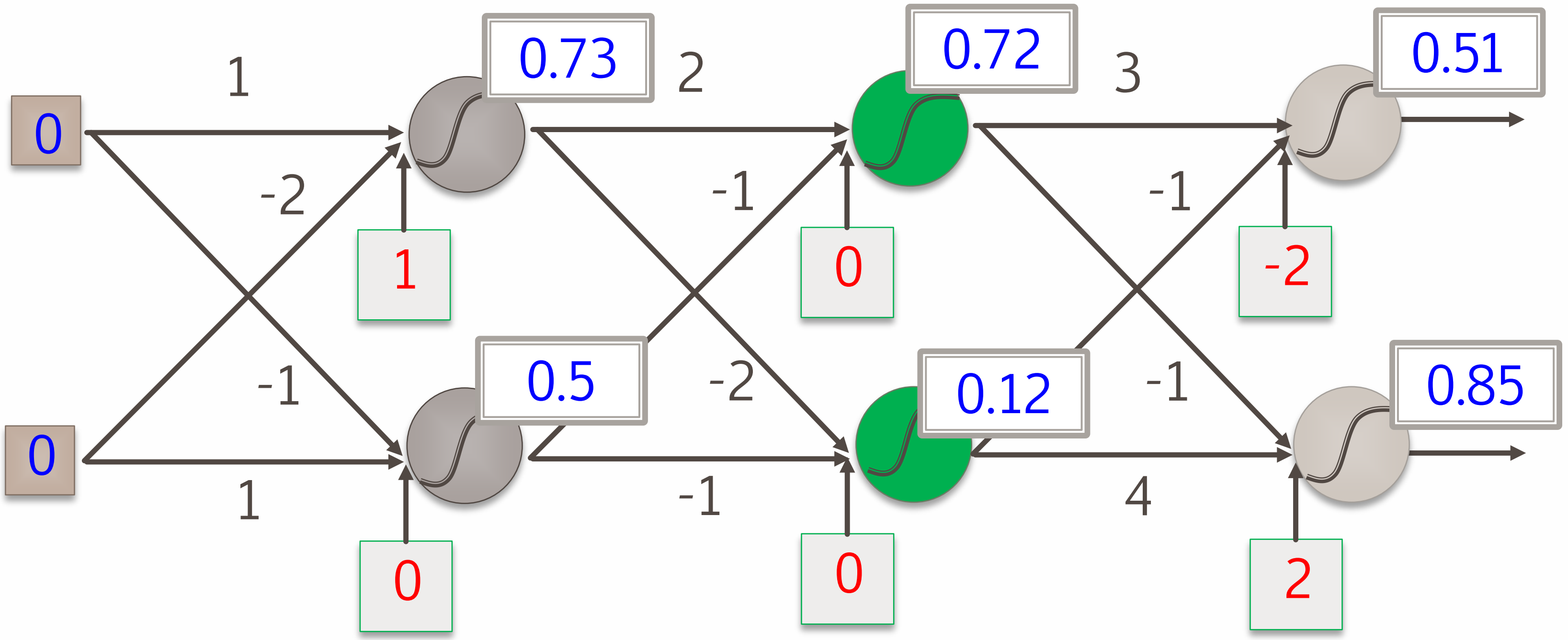


$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Weight/Bias



Weight/Bias (cont.)

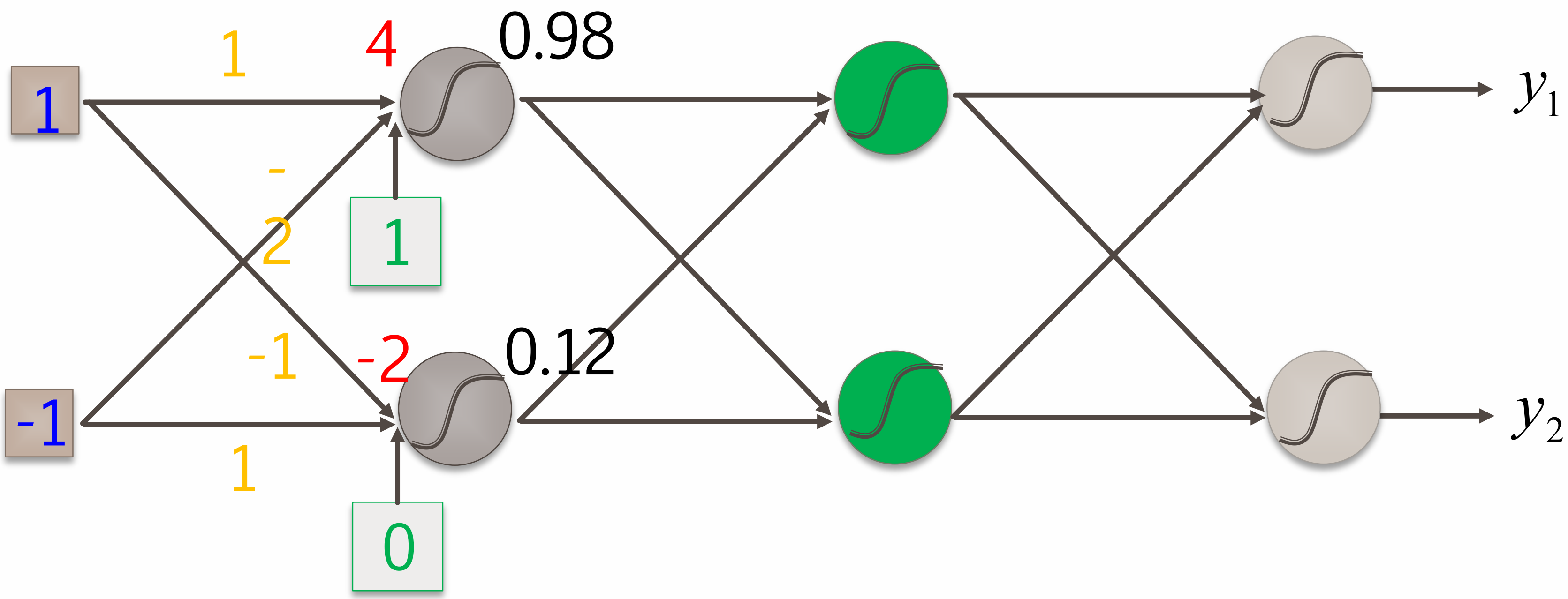


$$f: R^2 \to R^2$$

$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right) = \begin{bmatrix}0.62\\0.83\end{bmatrix} \quad f\left(\begin{bmatrix}0\\0\end{bmatrix}\right) = \begin{bmatrix}0.51\\0.85\end{bmatrix}$$

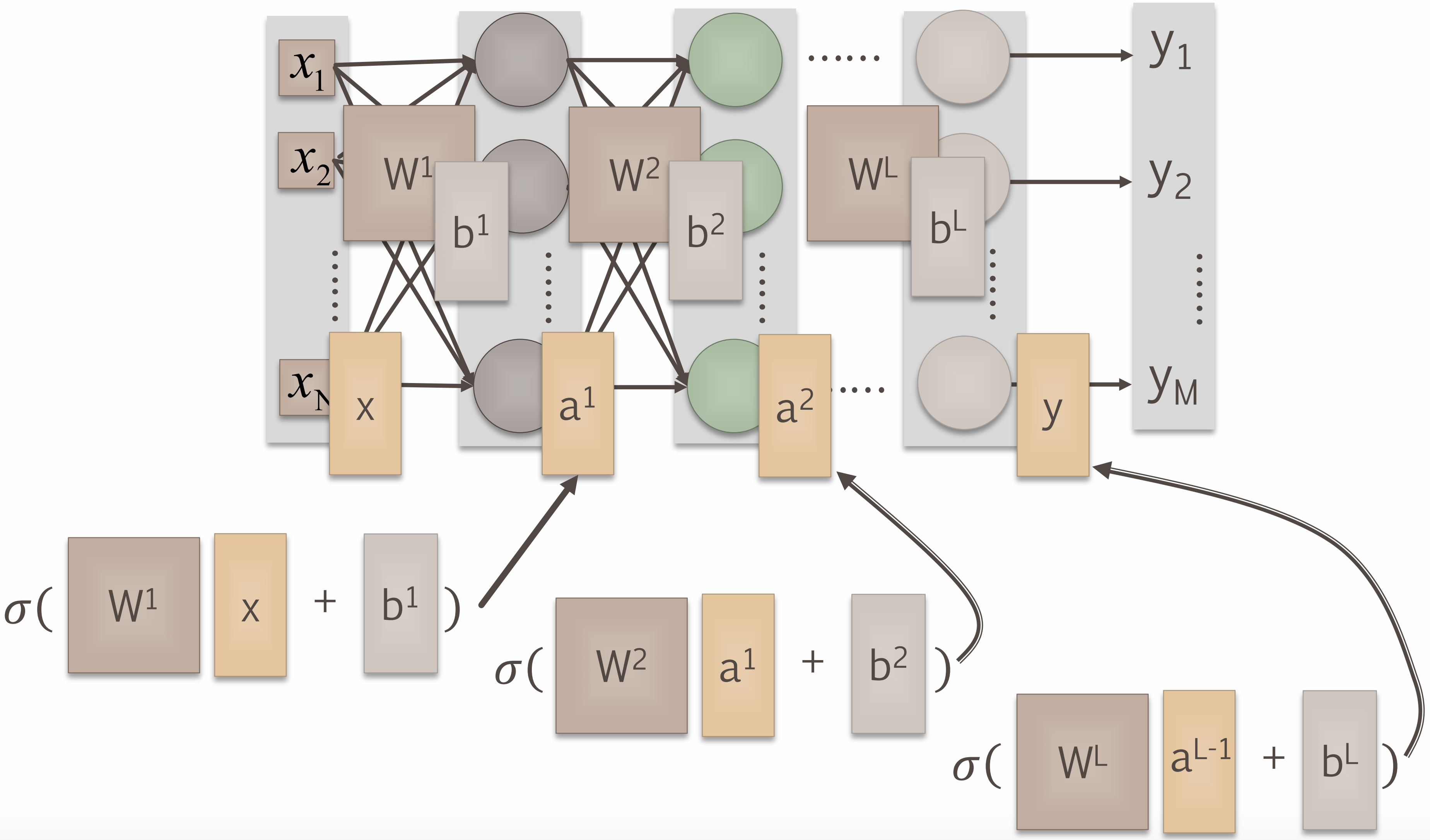
Different parameters define different function

Matrix Operation

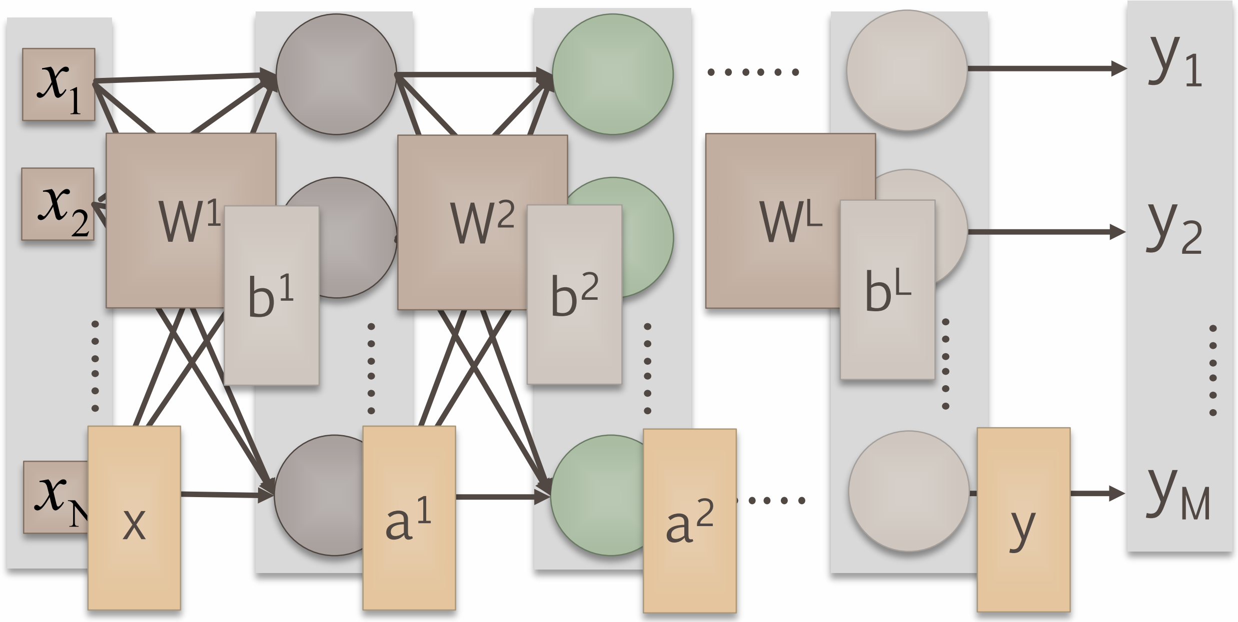


$$\sigma\left(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}}\right) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

How to form the Weight/Bias



How to form the Weight/Bias



$$y = f(x)$$

Using parallel computing techniques to speed up matrix operation

$$= \sigma(W^L \cdots \sigma(W^2 \, \sigma(W^1 x + b^1) + b^2) \cdots + b^L) \qquad x$$

Summary

- We have learned
  - From simple “linear classifier” to “perceptron”
  - Adding the “nonlinear” operation to perceptron becomes neural network
    - Deep learning!!
- All about
  - How to update weights
  - How good weights are
  - What the most efficient/effective way

Softmax Layer

- Softmax layer as the output layer
  - Softmax will be used on the final output
    - Sum of all softmax outputs is 1

Ordinary Layer

$$z_1 \rightarrow \sigma \rightarrow y_1 = \sigma(z_1)$$

$$z_2 \rightarrow \sigma \rightarrow y_2 = \sigma(z_2)$$

$$z_3 \rightarrow \sigma \rightarrow y_3 = \sigma(z_3)$$

In general, the output of network can be any value.
May not be easy to interpret

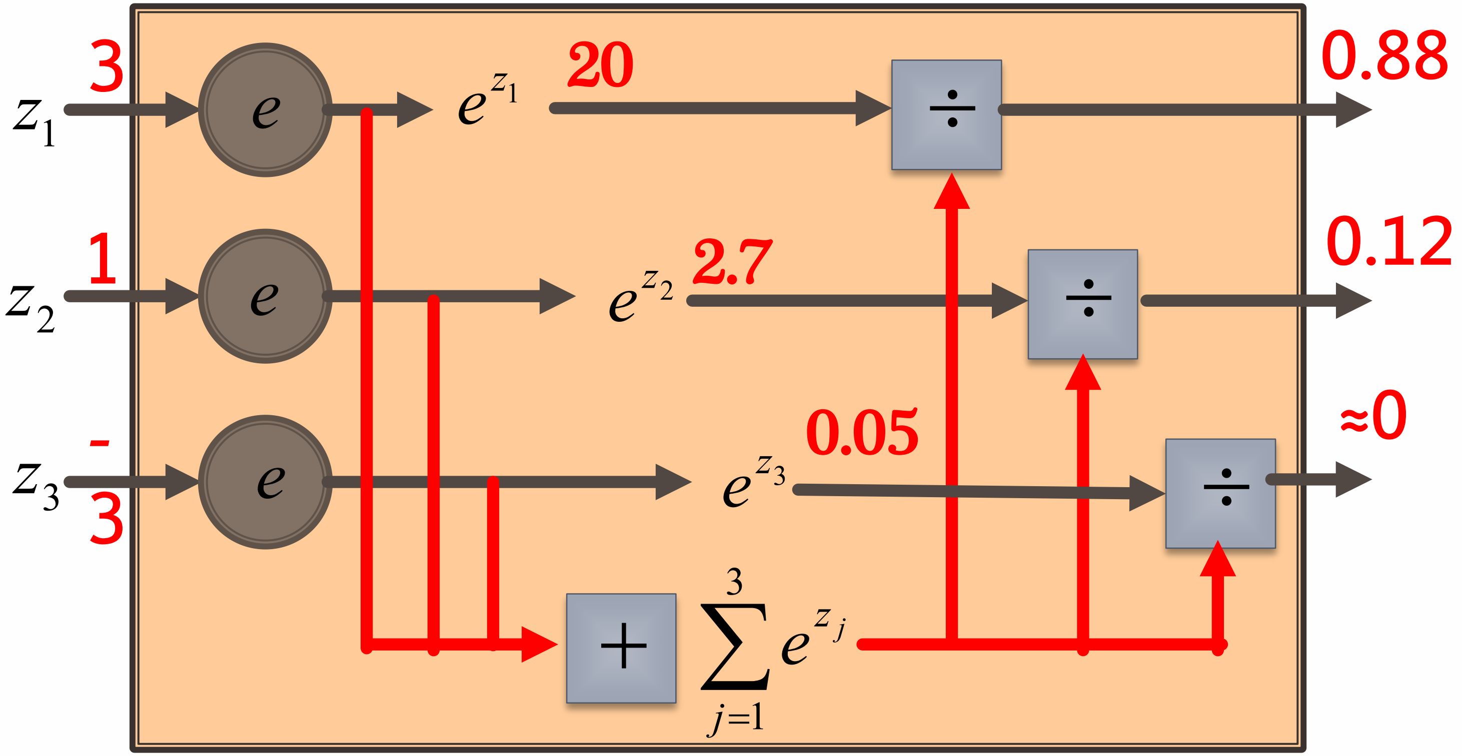
Softmax Layer

- Softmax layer as the output layer

***Probability*:**

- $1 > y_i > 0$
- $\sum_i y_i = 1$

Softmax Layer



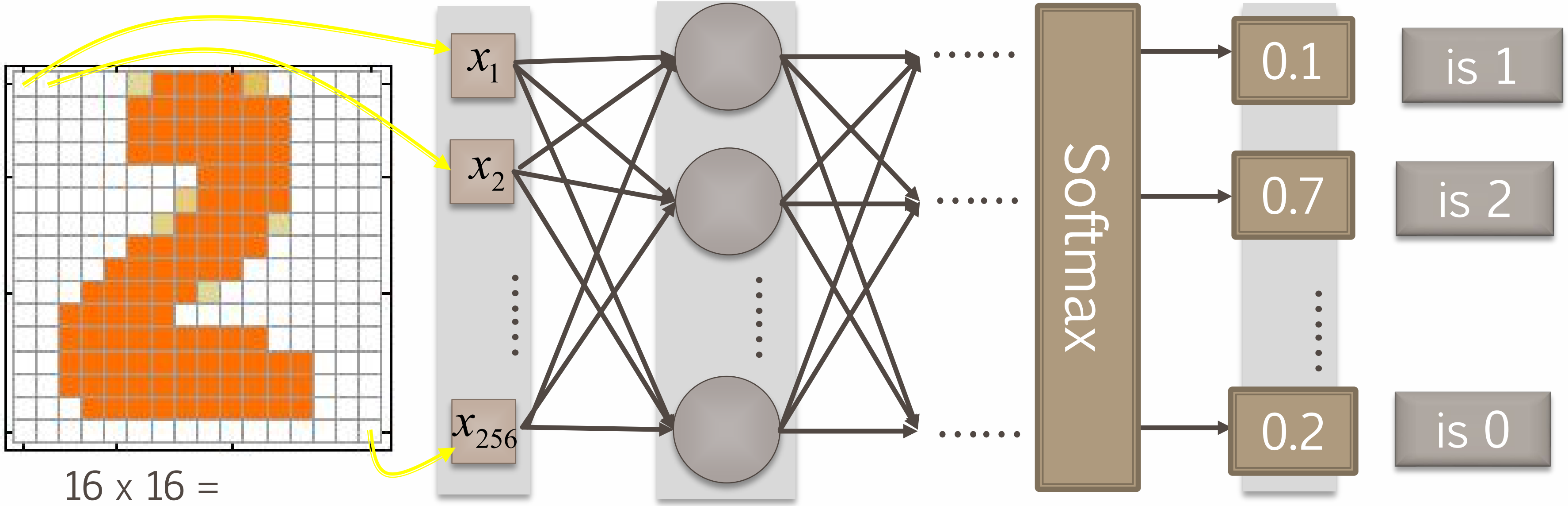
$$y_1 = e^{z_1} \Big/ \sum_{j=1}^{3} e^{z_j}$$

$$y_2 = e^{z_2} \Big/ \sum_{j=1}^{3} e^{z_j}$$

$$y_3 = e^{z_3} \Big/ \sum_{j=1}^{3} e^{z_j}$$

Learning Parameters

$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$



16 x 16 = 256

Ink → 1

No ink → 0

Set the network parameters θ such that ……

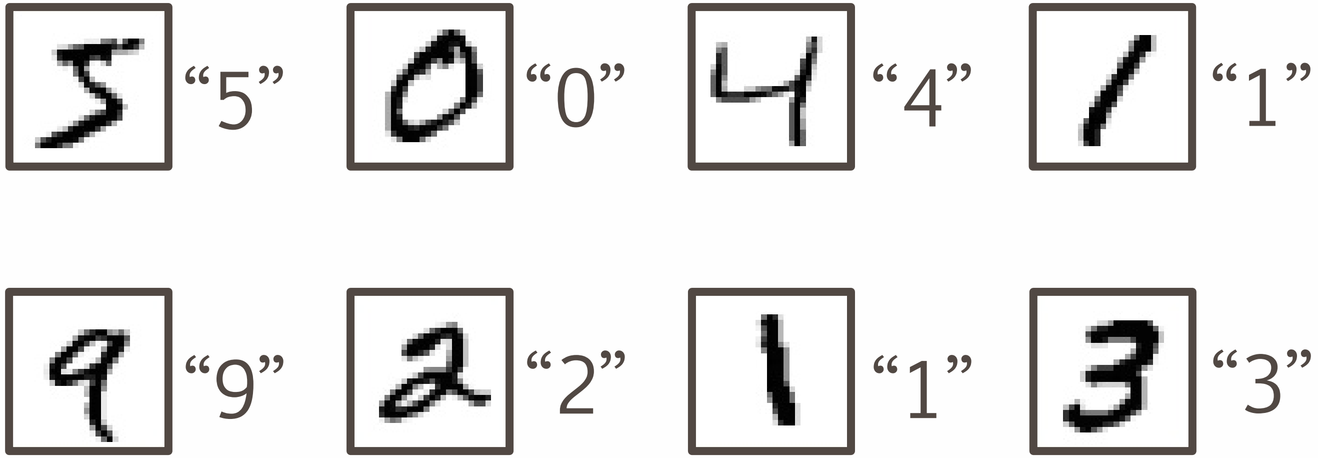
Input … value

Back-propagation!!

Input:  y_2 has the maximum value

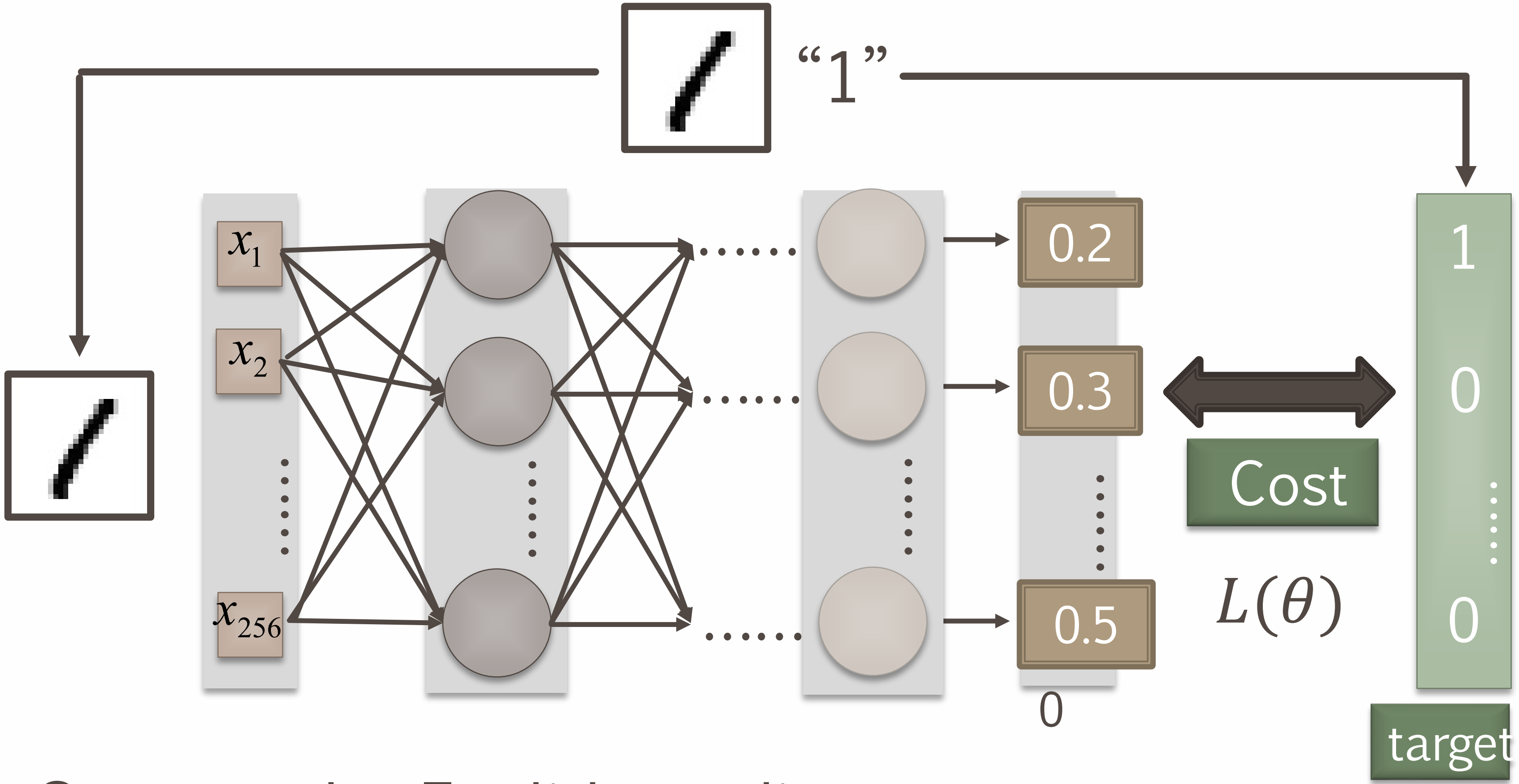
An Example

- Preparing training data: images and their labels
  - Training samples & their labels



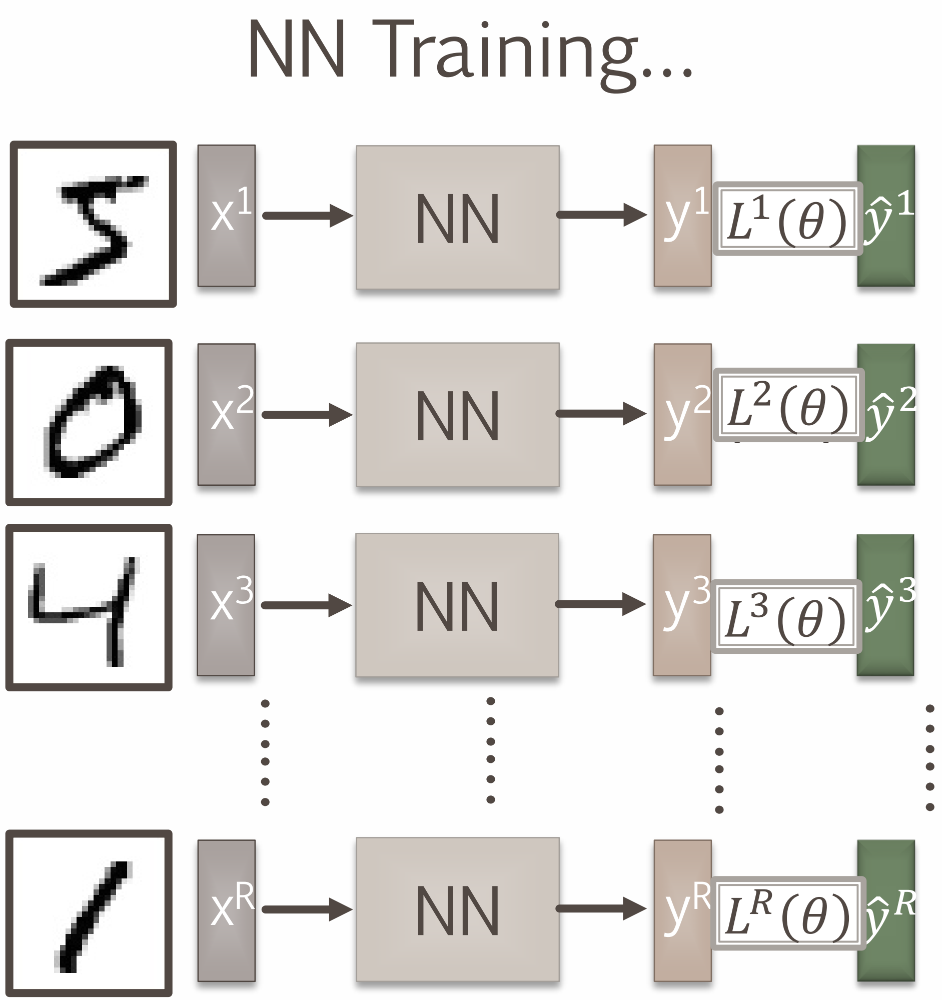
Using the training data to find the network parameters.

Cost function (Answer is different from predicted)



Cost can be Euclidean distance or cross entropy of the network output and target

Define the Total Cost Function!



Total Cost:

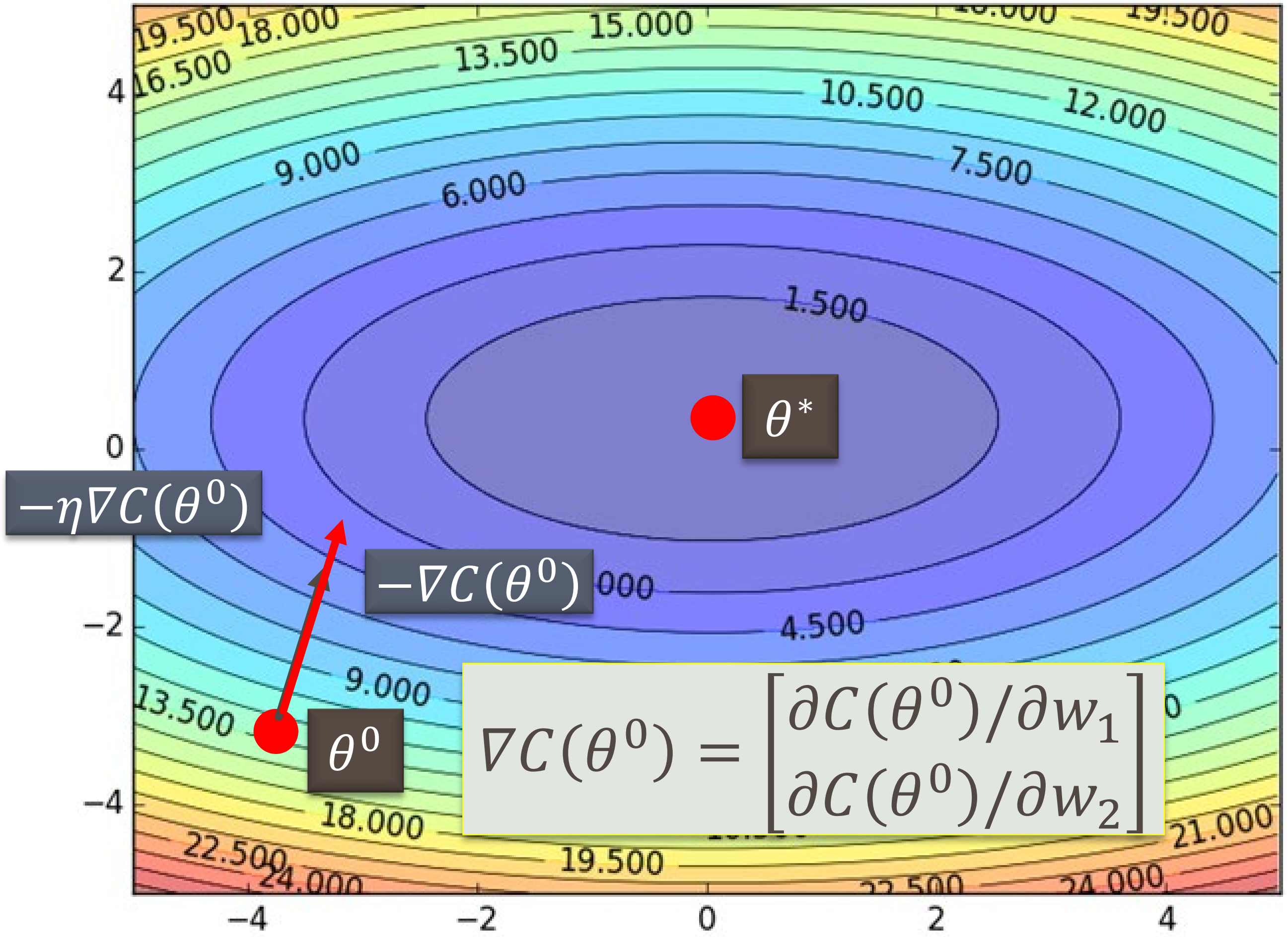
$$C(\theta) = \sum_{r=1}^{R} L^r(\theta)$$

How bad the network parameters θ is on this task

NN parameters θ^* will be updated by the iterative learning

Gradient Descent: A way to learn parameters

Error Surface



$$\nabla C(\theta^0) = \begin{bmatrix} \partial C(\theta^0)/\partial w_1 \\ \partial C(\theta^0)/\partial w_2 \end{bmatrix}$$

The colors represent the value of C.

Assumed 2-dim W

$\theta = \{w_1, w_2\}$

Randomly pick a starting point θ^0

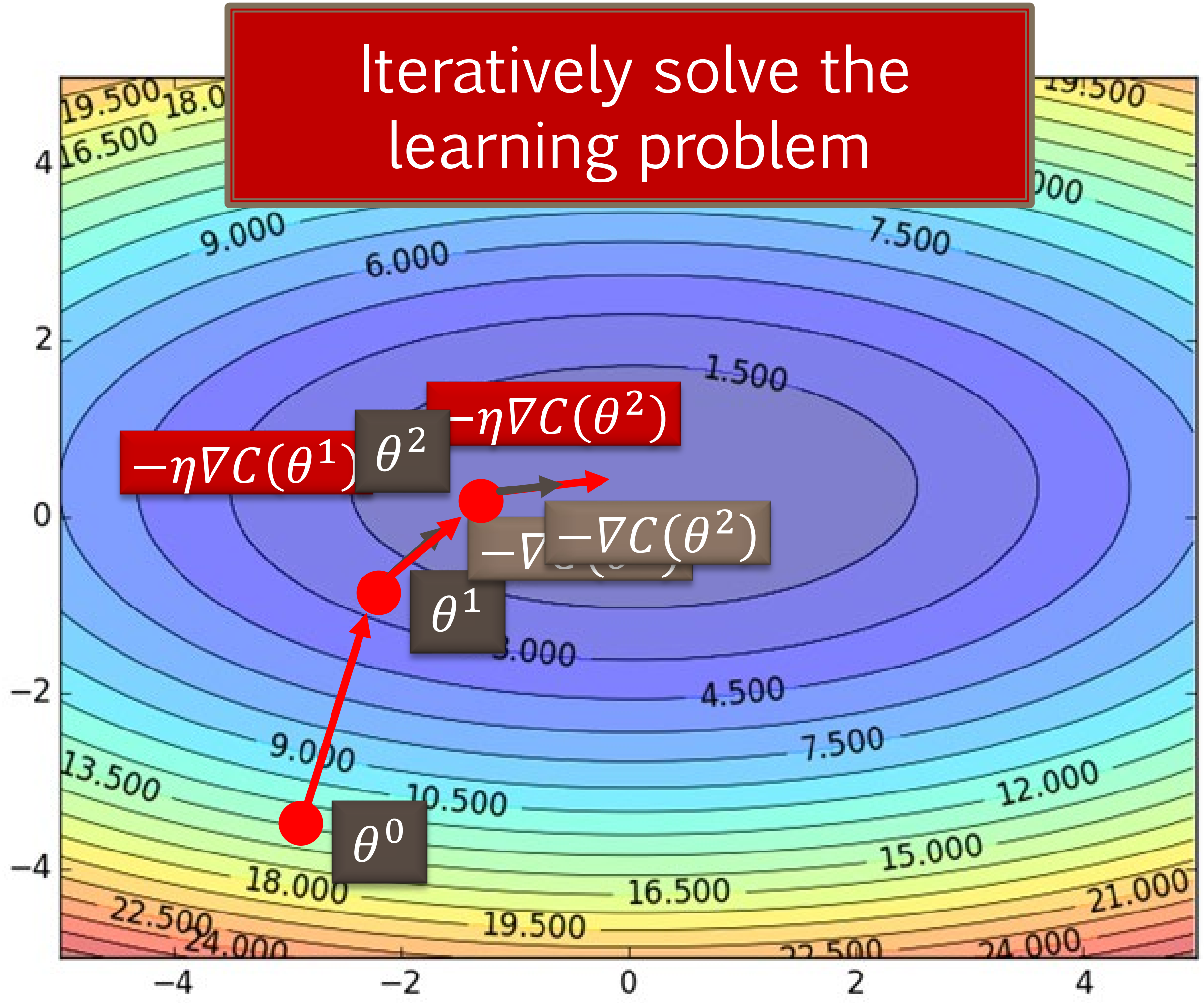
Compute the negative gradient at θ^0

$\Rightarrow -\nabla C(\theta^0)$

Times the learning rate η

$\Rightarrow -\eta \nabla C(\theta^0)$

Gradient Descent: A way to learn parameters



Randomly pick a starting point θ^0

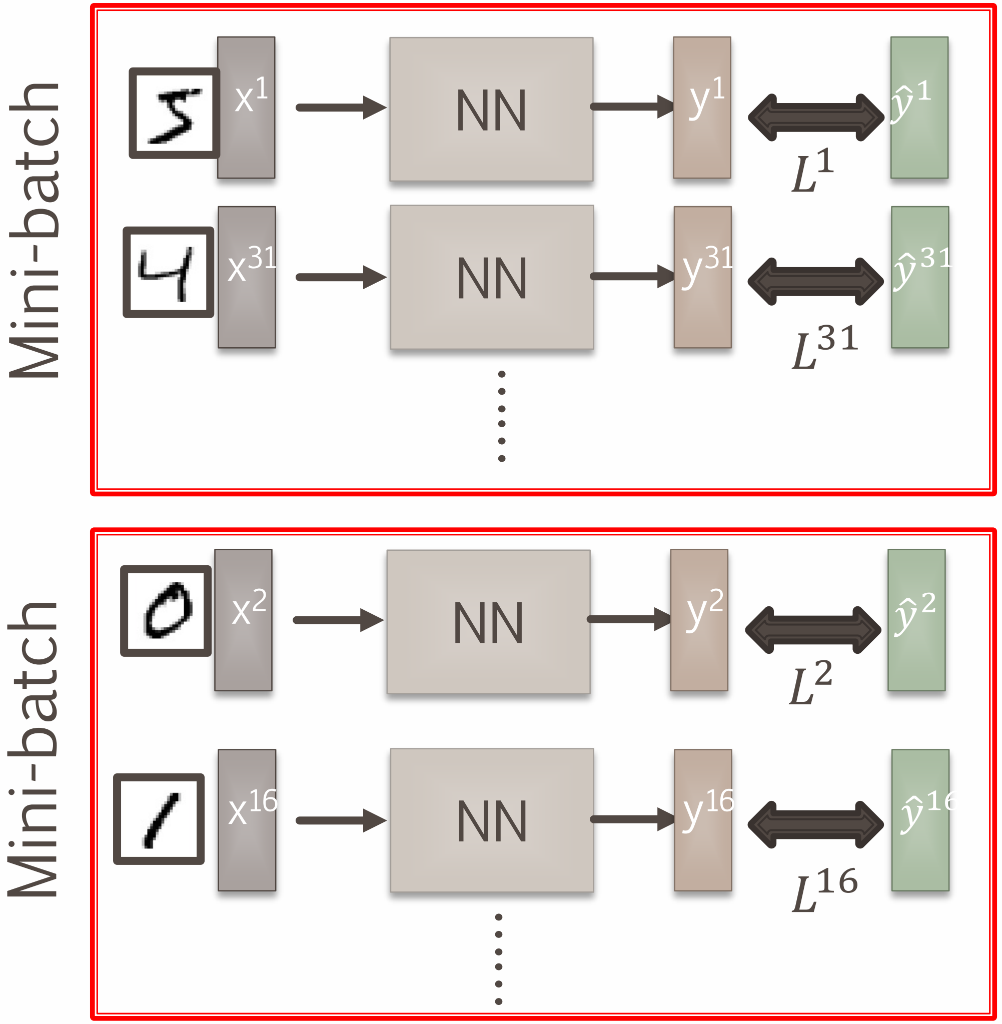
Compute the negative gradient at θ^0

$\Rightarrow -\nabla C(\theta^0)$

Times the learning rate η

$\Rightarrow -\eta\nabla C(\theta^0)$

Mini-batch Gradient Descent



- Randomly initialize θ^0
- Pick the 1st batch

$$C = L^1 + L^{31} + \cdots$$

$$\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$$

- Pick the 2nd batch

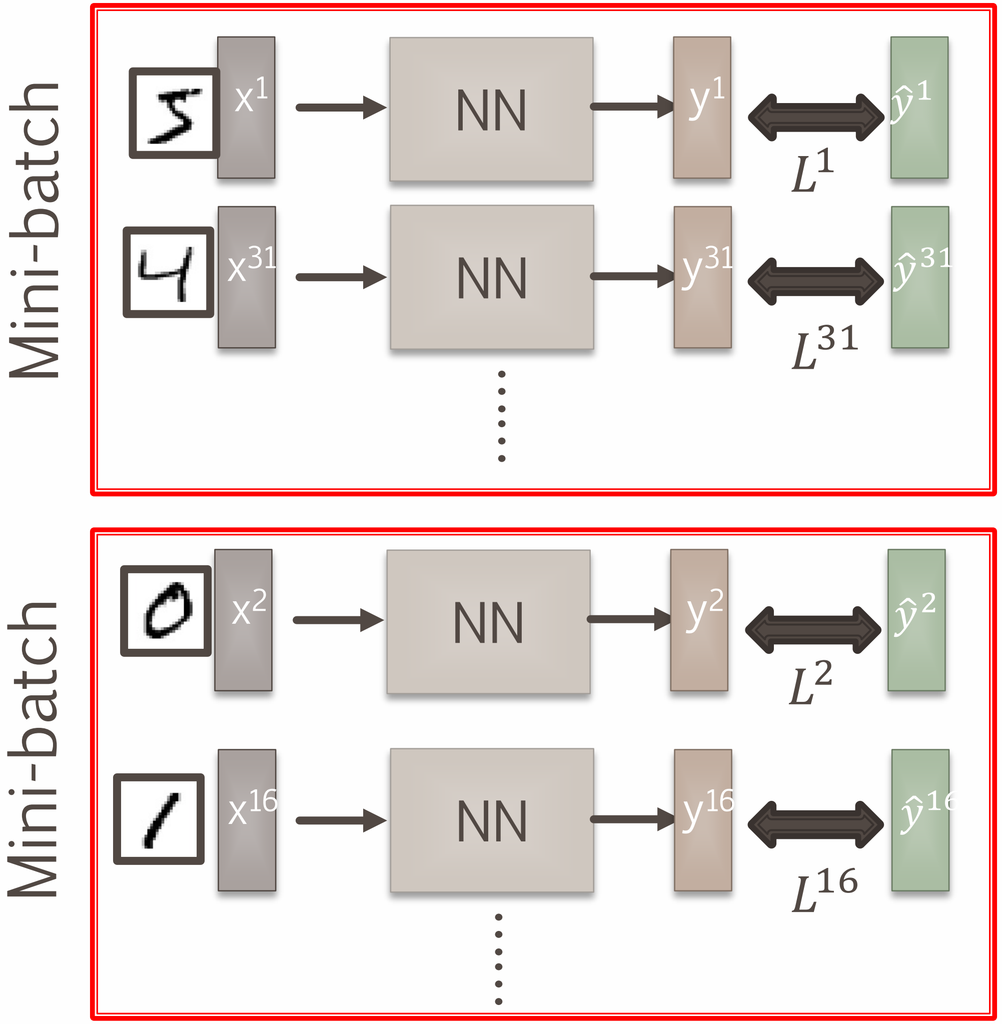
$$C = L^2 + L^{16} + \cdots$$

$$\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$$

$$\vdots$$

Cost will be different for different mini-batch

Mini-batch Advantages



- Randomly initialize θ^0
- Pick the 1st batch

  $C = C^1 + C^{31} + \cdots$

  $\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$
- Pick the 2nd batch

  $C = C^2 + C^{16} + \cdots$

  $\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$

  $\vdots$
- Until all mini-batches have been picked

Faster

one epoch

Repeat the above process