EE-559 – Deep learning

3a. Linear classifiers, perceptron

François Fleuret

https://fleuret.org/dlc/

[version of: March 12, 2018]

idiap RESEARCH INSTITUTE

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

A bit of history, the perceptron

The first mathematical model for a neuron was the Threshold Logic Unit, with Boolean inputs and outputs:

$$f(x) = \mathbf{1}_{\left\{w \sum_i x_i + b \geq 0\right\}}.$$

The first mathematical model for a neuron was the Threshold Logic Unit, with Boolean inputs and outputs:

$$f(x) = \mathbf{1}_{\{w \sum_i x_i + b \geq 0\}}.$$

It can in particular implement

$$\begin{aligned} or(u, v) &= \mathbf{1}_{\{u+v-0.5 \geq 0\}} && (w = 1, b = -0.5) \\ and(u, v) &= \mathbf{1}_{\{u+v-1.5 \geq 0\}} && (w = 1, b = -1.5) \\ not(u) &= \mathbf{1}_{\{-u+0.5 \geq 0\}} && (w = -1, b = 0.5) \end{aligned}$$

The first mathematical model for a neuron was the Threshold Logic Unit, with Boolean inputs and outputs:

$$f(x) = \mathbf{1}_{\{w \sum_i x_i + b \geq 0\}}.$$

It can in particular implement

$$\begin{aligned} or(u, v) &= \mathbf{1}_{\{u+v-0.5 \geq 0\}} && (w = 1, b = -0.5) \\ and(u, v) &= \mathbf{1}_{\{u+v-1.5 \geq 0\}} && (w = 1, b = -1.5) \\ not(u) &= \mathbf{1}_{\{-u+0.5 \geq 0\}} && (w = -1, b = 0.5) \end{aligned}$$

Hence, **any Boolean function can be build with such units.**

(McCulloch and Pitts, 1943)

The perceptron is very similar

$$f(x) = \begin{cases} 1 & \text{if} \quad \sum_i w_i \, x_i + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

but the inputs are real values and the weights can be different.

The perceptron is very similar

$$f(x) = \begin{cases} 1 & \text{if} \quad \sum_i w_i x_i + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

but the inputs are real values and the weights can be different.

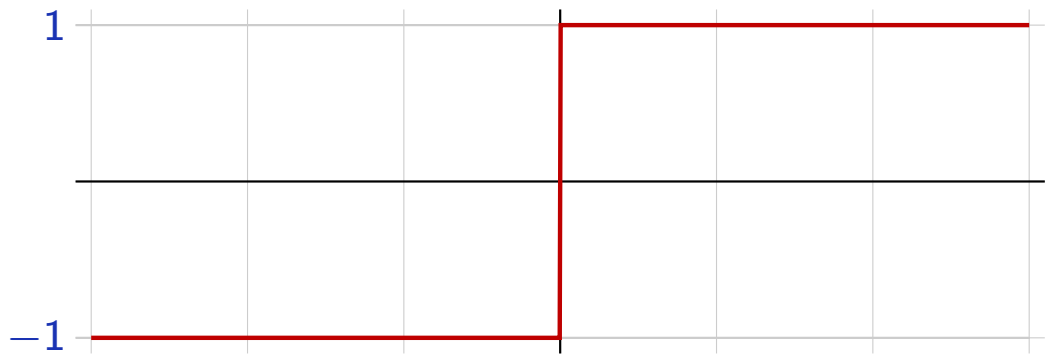
This model was originally motivated by biology, with w_i being the *synaptic weights*, and x_i and f firing rates.

It is a (very) crude biological model.

(Rosenblatt, 1957)

To make things simpler we take responses ± 1. Let

$$\sigma(x) = \begin{cases} 1 & \text{if} \quad x \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

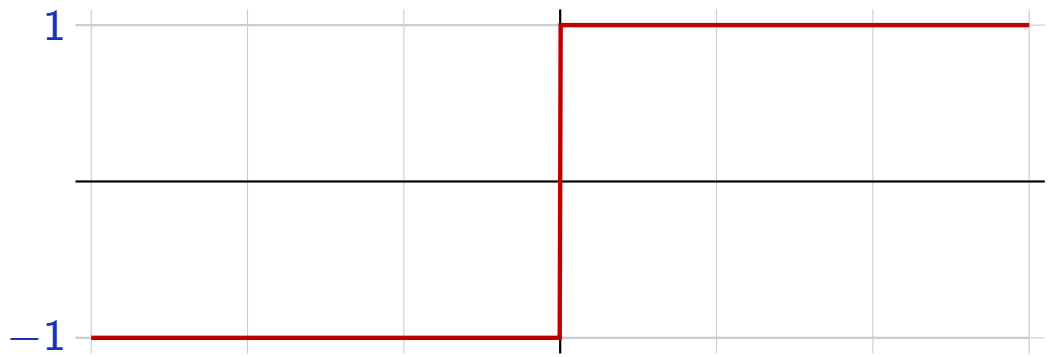


The perceptron classification rule boils down to

$$f(x) = \sigma(w \cdot x + b).$$

To make things simpler we take responses ± 1. Let

$$\sigma(x) = \begin{cases} 1 & \text{if} \quad x \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

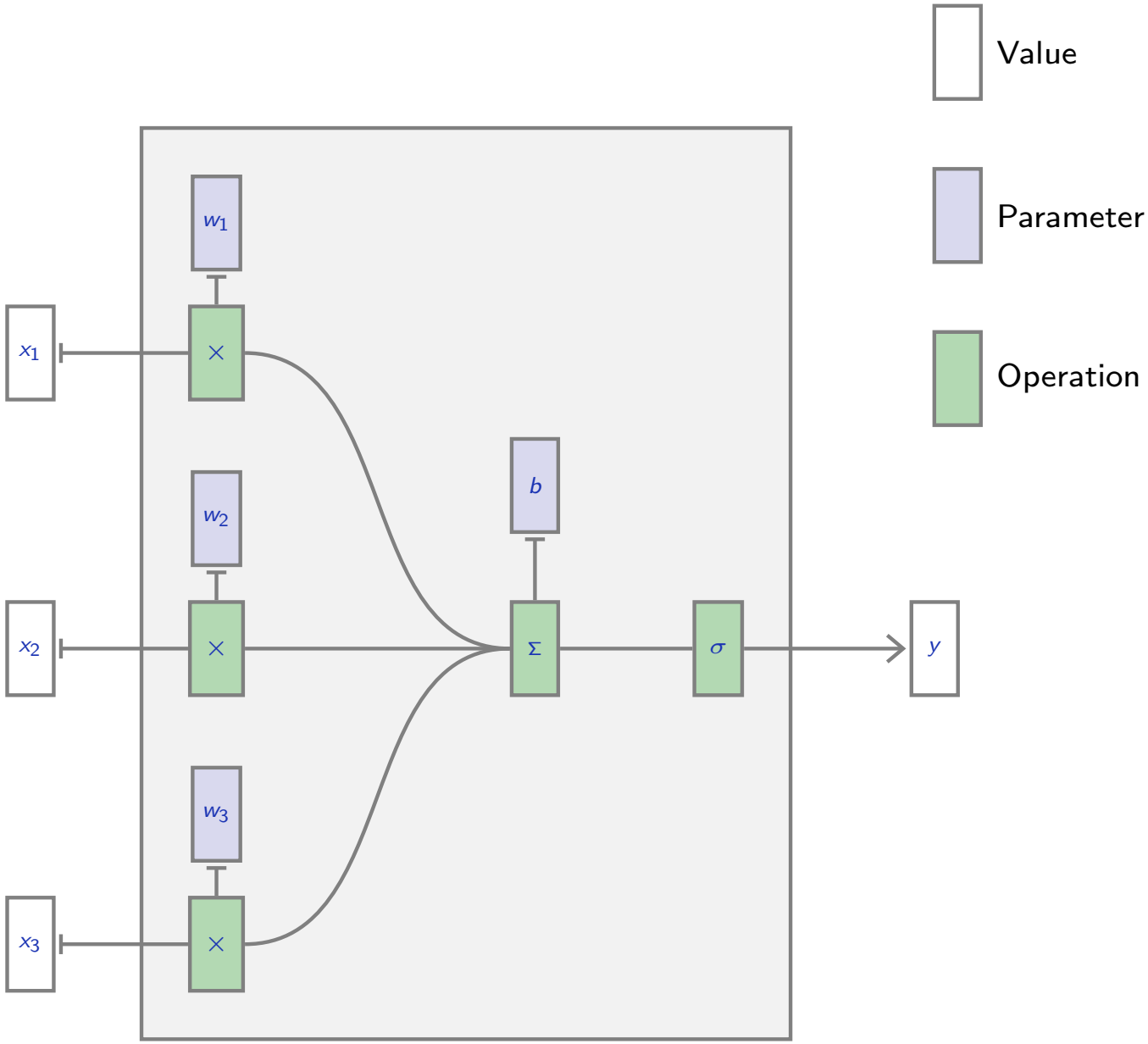


The perceptron classification rule boils down to

$$f(x) = \sigma(w \cdot x + b).$$

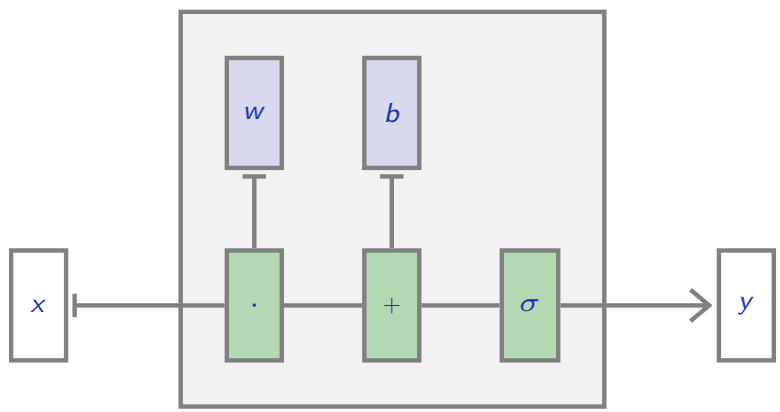
For neural networks, the function σ that follows a linear operator is called the **activation function**.

We can represent this "neuron" as follows:



We can also use tensor operations, as in

$$f(x) = \sigma(w \cdot x + b).$$



Given a training set

$$(x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}, \quad n = 1, \ldots, N,$$

a very simple scheme to train such a linear operator for classification is the **perceptron algorithm:**

Given a training set

$$(x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}, \quad n = 1, \ldots, N,$$

a very simple scheme to train such a linear operator for classification is the **perceptron algorithm:**

1. Start with $w^0 = 0$,
2. while $\exists n_k$ s.t. $y_{n_k} \left(w^k \cdot x_{n_k} \right) \leq 0$, update $w^{k+1} = w^k + y_{n_k} x_{n_k}$.

Given a training set

$$(x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}, \quad n = 1, \ldots, N,$$

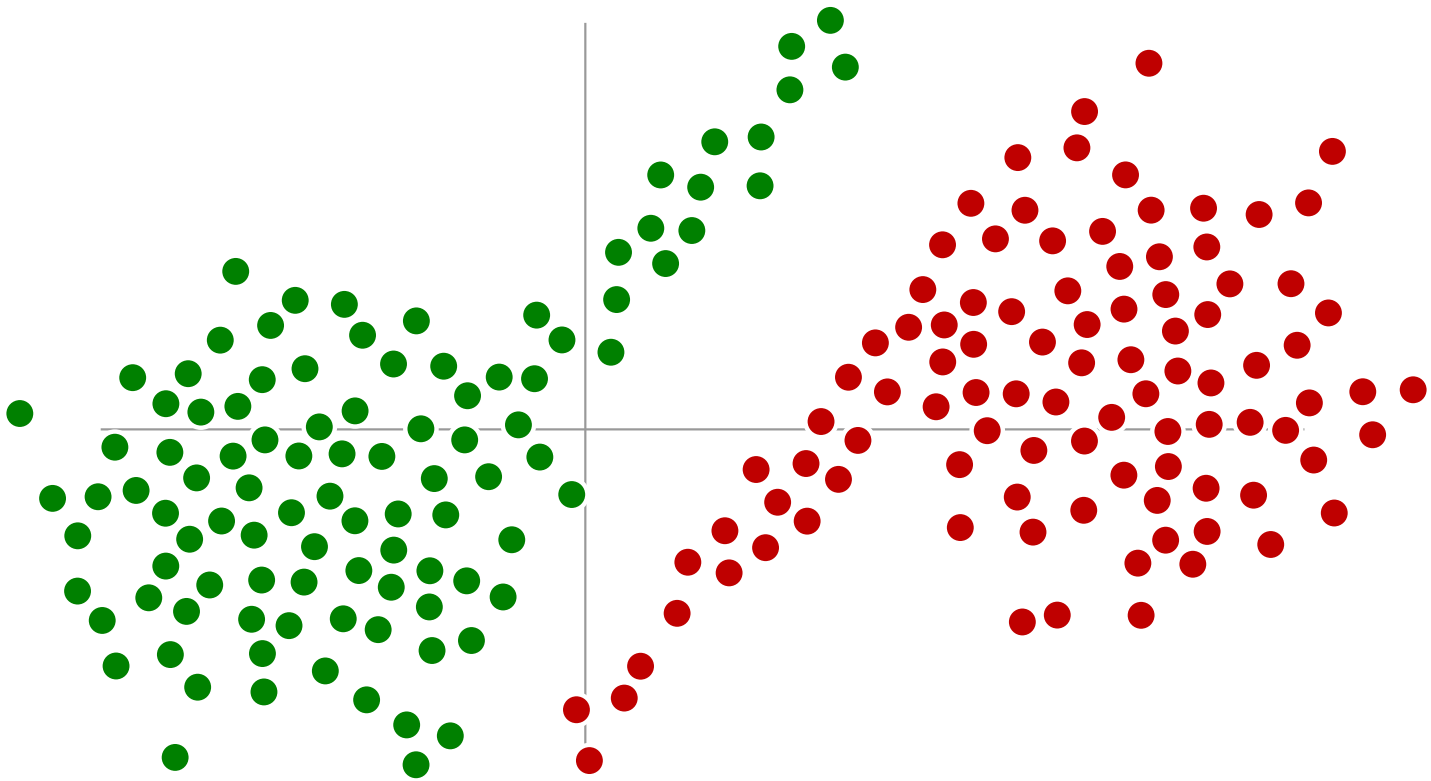
a very simple scheme to train such a linear operator for classification is the **perceptron algorithm:**

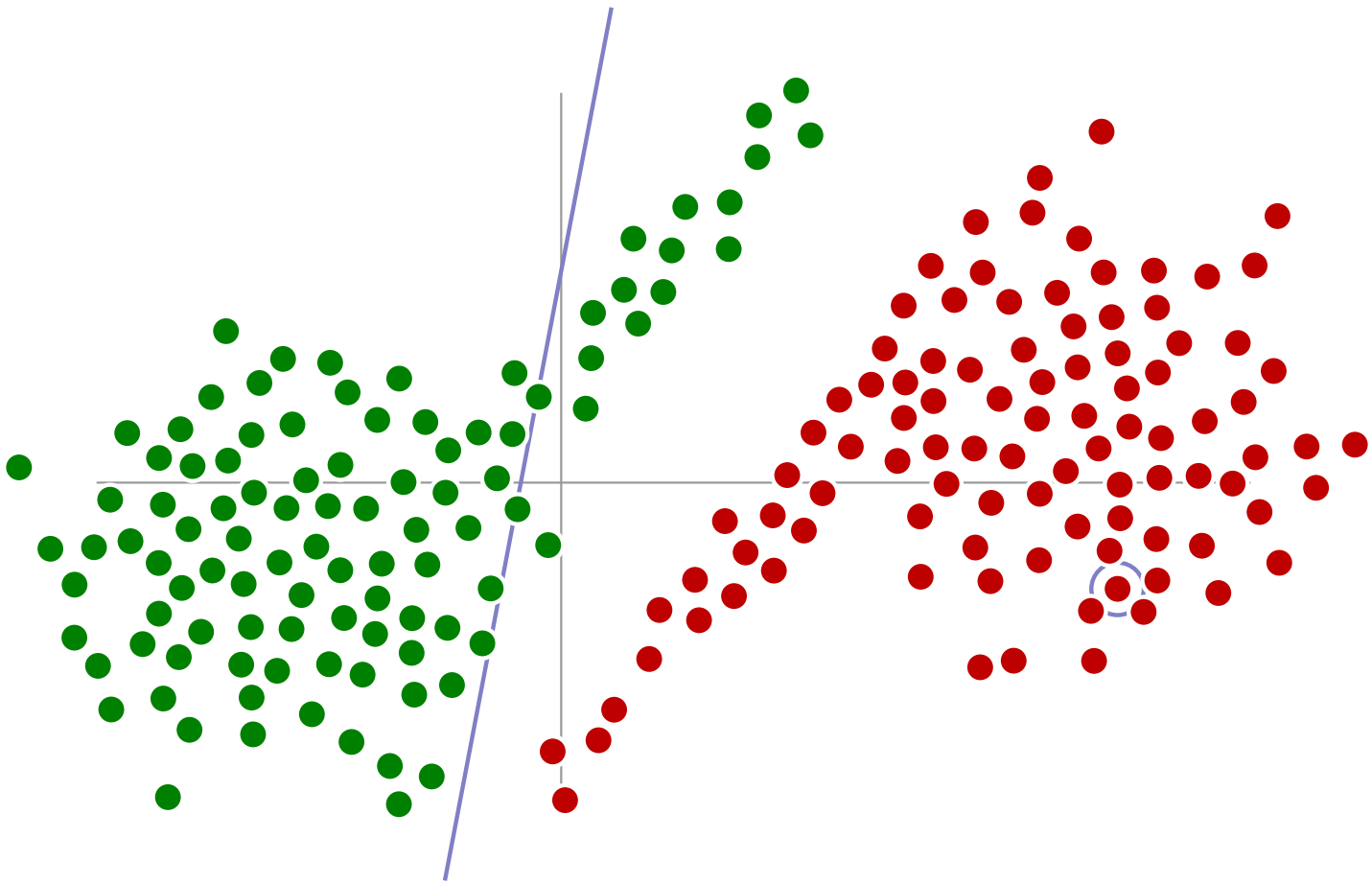
1. Start with $w^0 = 0$,
2. while $\exists n_k$ s.t. $y_{n_k} \left(w^k \cdot x_{n_k} \right) \leq 0$, update $w^{k+1} = w^k + y_{n_k} x_{n_k}$.

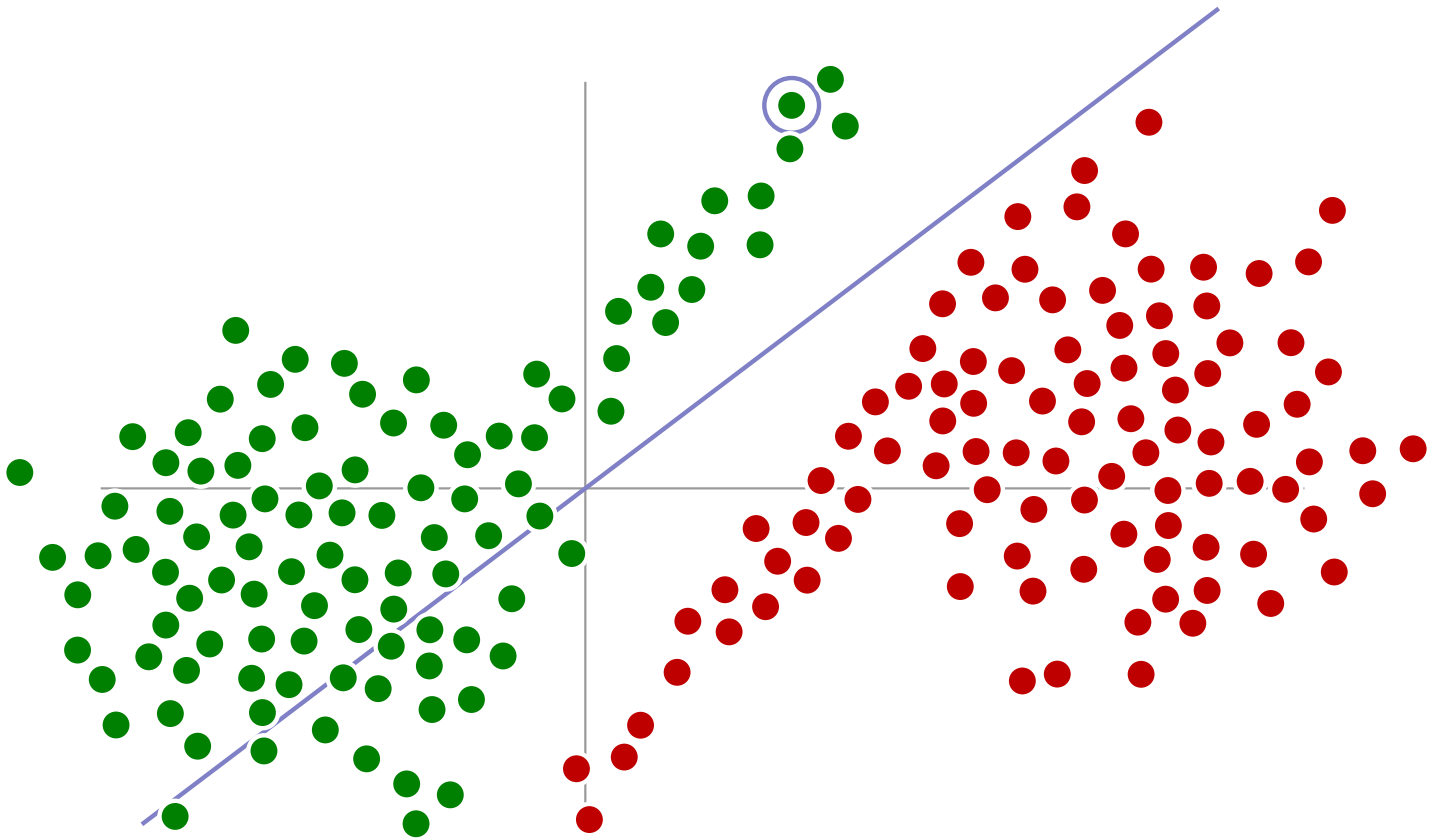
The bias b can be introduced as one of the ws by adding a constant component to x equal to 1.

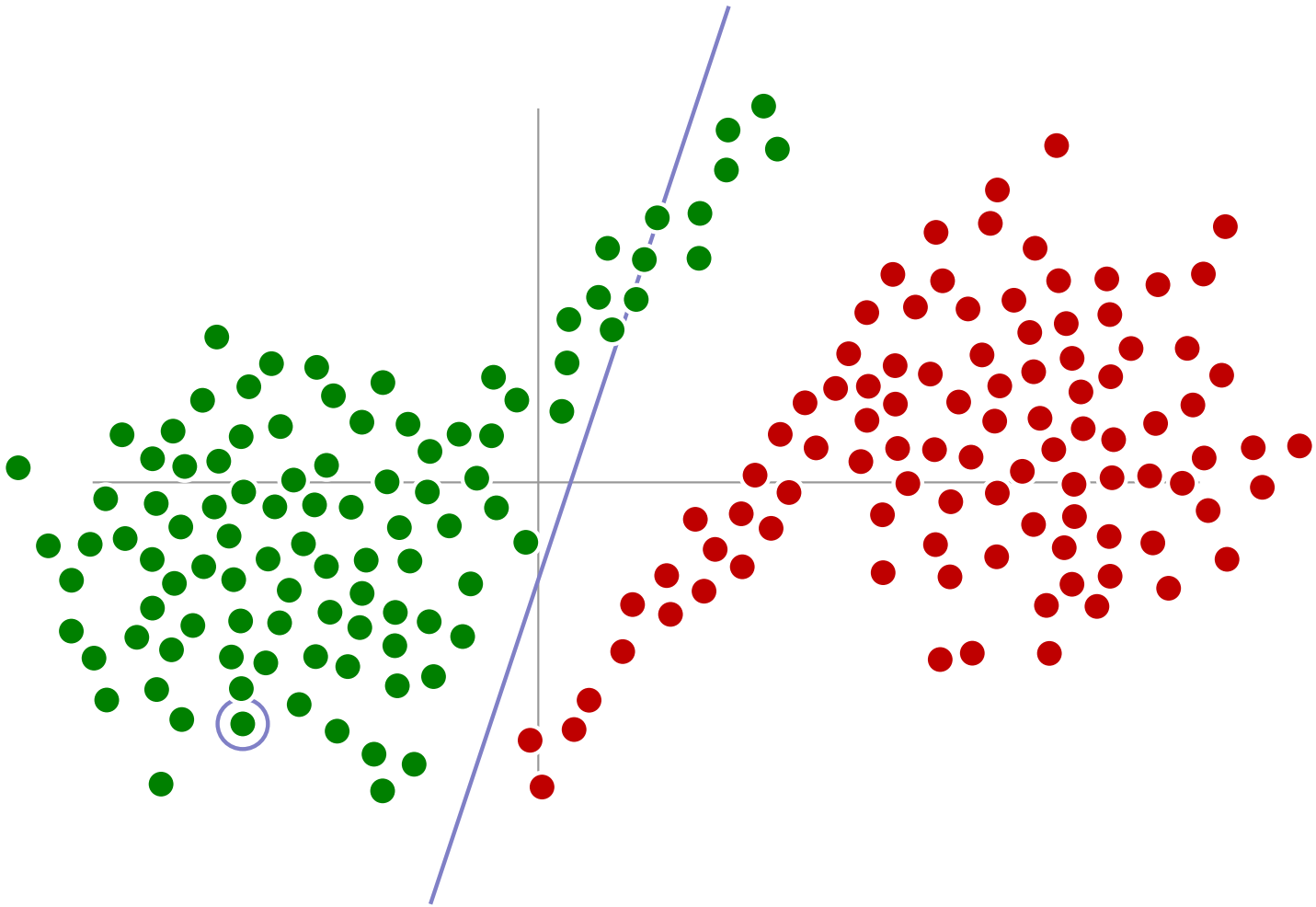
(Rosenblatt, 1957)

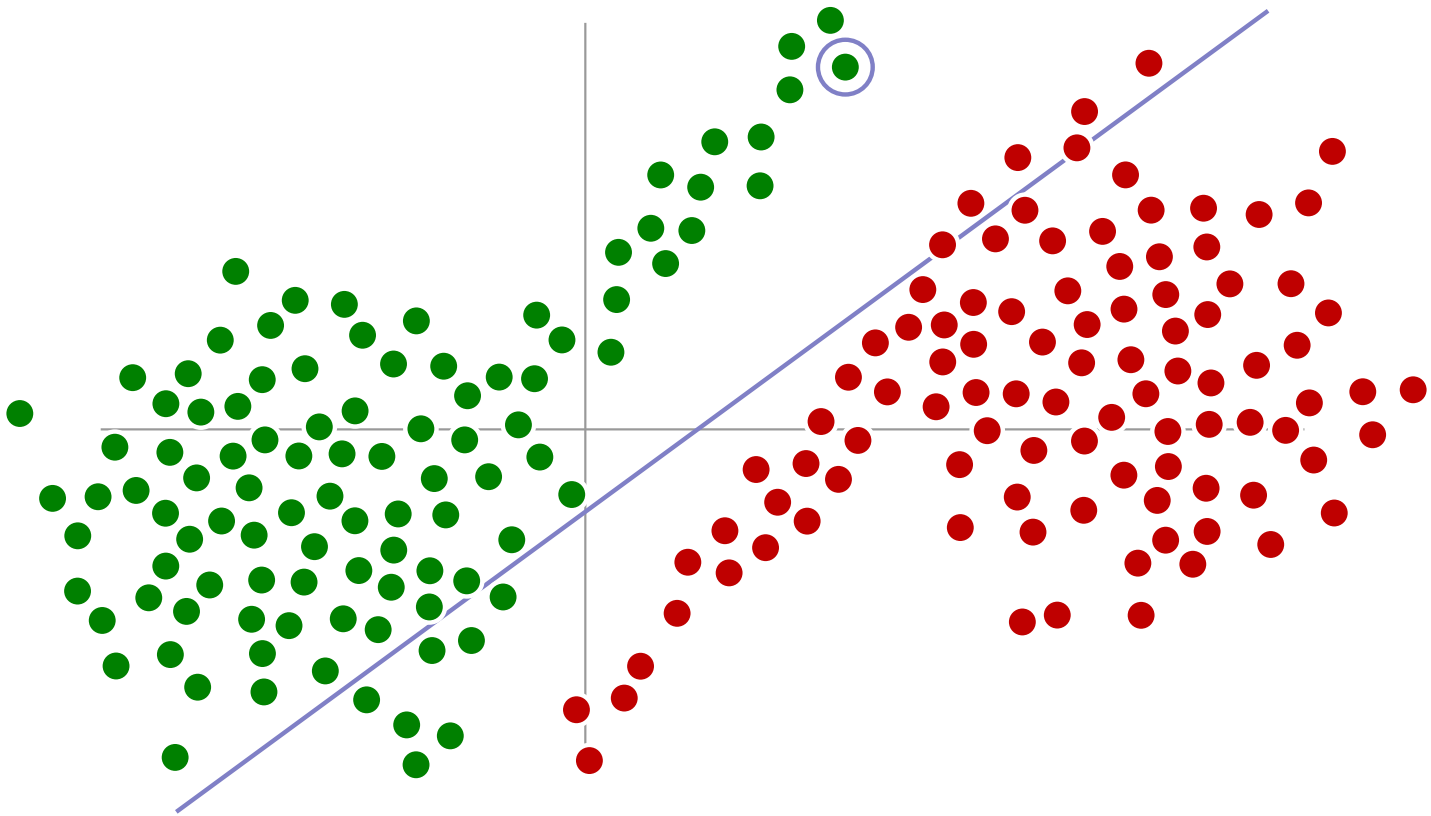
```
def train_perceptron(x, y, nb_epochs_max):
    w = Tensor(x.size(1)).zero_()

    for e in range(nb_epochs_max):
        nb_changes = 0
        for i in range(x.size(0)):
            if x[i].dot(w) * y[i] <= 0:
                w = w + y[i] * x[i]
                nb_changes = nb_changes + 1
        if nb_changes == 0: break;

    return w
```

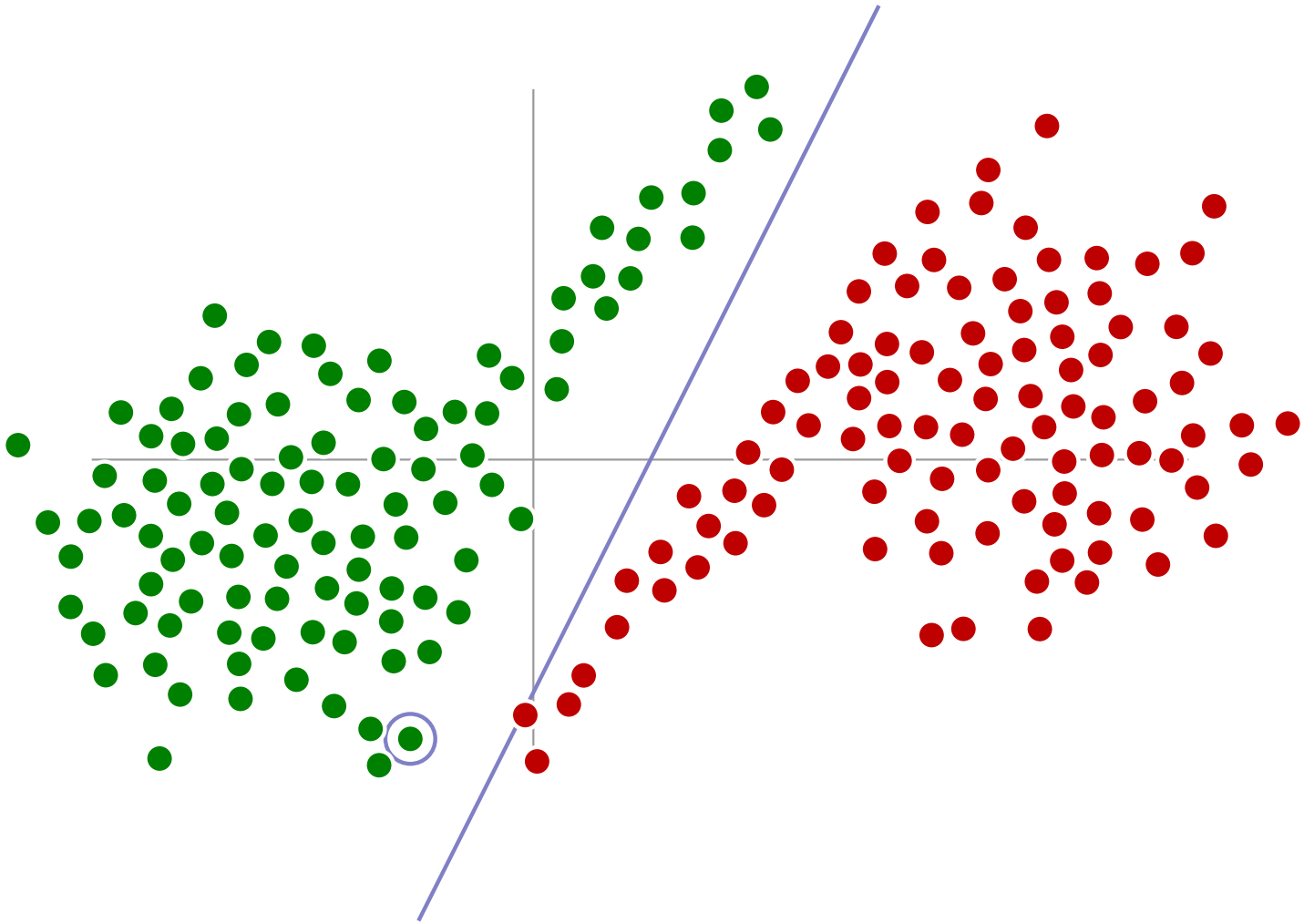




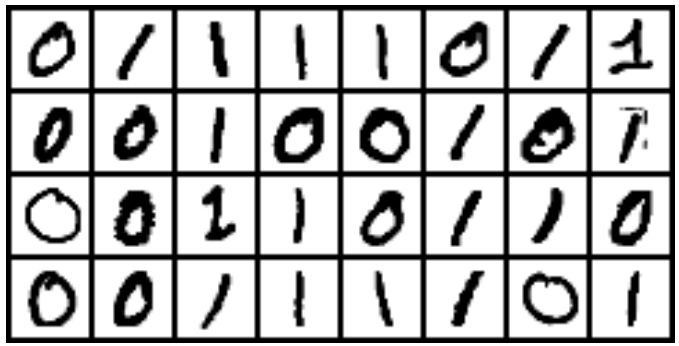




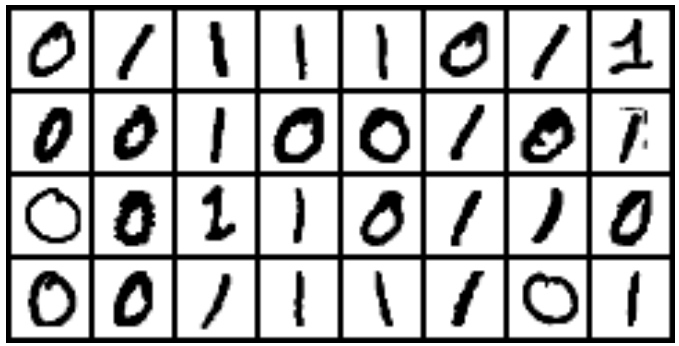




This crude algorithm works often surprisingly well. With MNIST's "0"s as negative class, and "1"s as positive one.

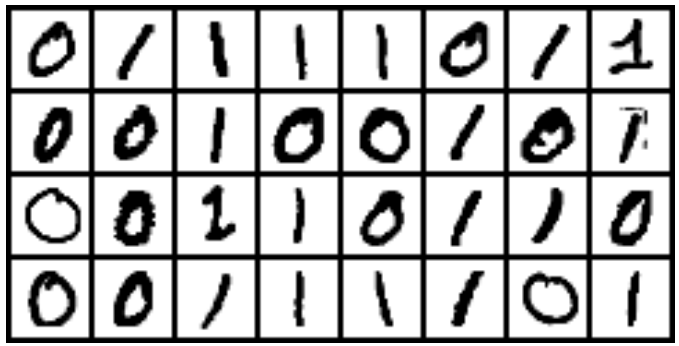


This crude algorithm works often surprisingly well. With MNIST's "0"s as negative class, and "1"s as positive one.

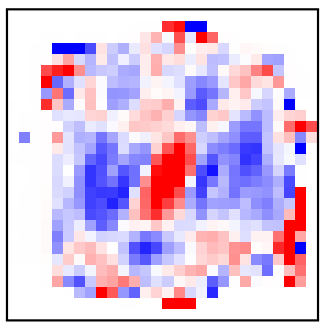


```
epoch 0 nb_changes 64 train_error 0.23% test_error 0.19%
epoch 1 nb_changes 24 train_error 0.07% test_error 0.00%
epoch 2 nb_changes 10 train_error 0.06% test_error 0.05%
epoch 3 nb_changes 6 train_error 0.03% test_error 0.14%
epoch 4 nb_changes 5 train_error 0.03% test_error 0.09%
epoch 5 nb_changes 4 train_error 0.02% test_error 0.14%
epoch 6 nb_changes 3 train_error 0.01% test_error 0.14%
epoch 7 nb_changes 2 train_error 0.00% test_error 0.14%
epoch 8 nb_changes 0 train_error 0.00% test_error 0.14%
```

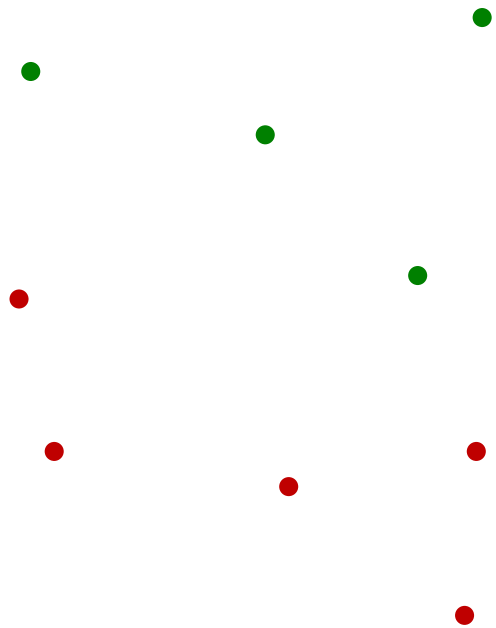
This crude algorithm works often surprisingly well. With MNIST's "0"s as negative class, and "1"s as positive one.



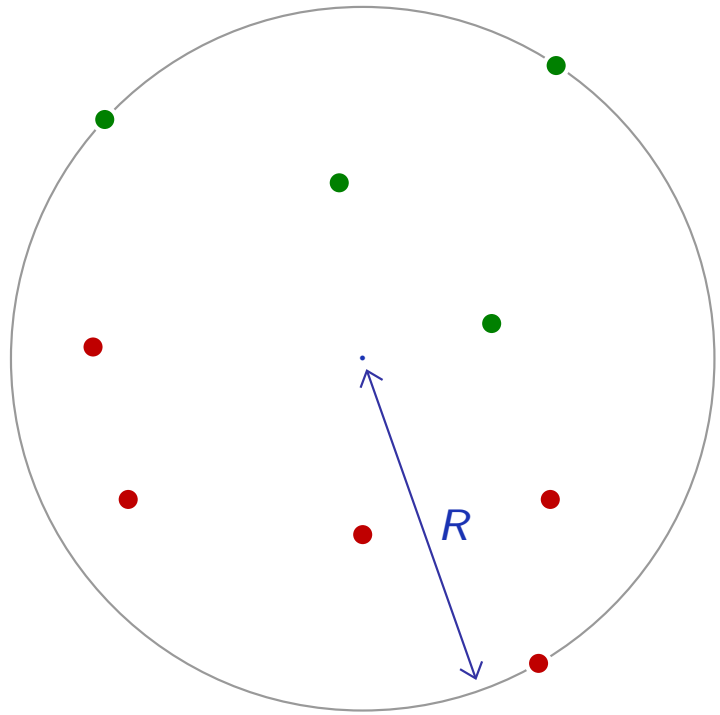
```
epoch 0 nb_changes 64 train_error 0.23% test_error 0.19%
epoch 1 nb_changes 24 train_error 0.07% test_error 0.00%
epoch 2 nb_changes 10 train_error 0.06% test_error 0.05%
epoch 3 nb_changes 6 train_error 0.03% test_error 0.14%
epoch 4 nb_changes 5 train_error 0.03% test_error 0.09%
epoch 5 nb_changes 4 train_error 0.02% test_error 0.14%
epoch 6 nb_changes 3 train_error 0.01% test_error 0.14%
epoch 7 nb_changes 2 train_error 0.00% test_error 0.14%
epoch 8 nb_changes 0 train_error 0.00% test_error 0.14%
```



We can get a convergence result under two assumptions:

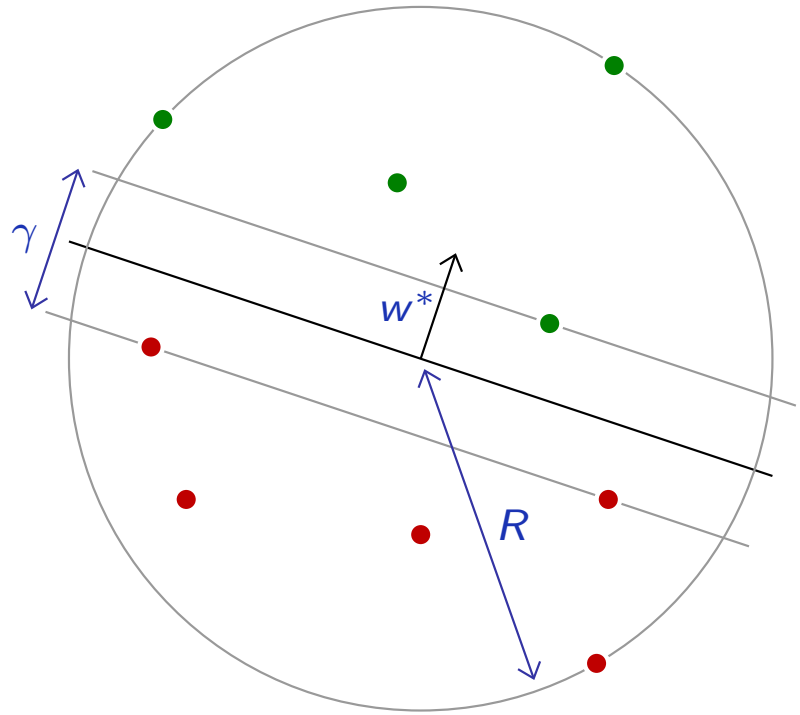


We can get a convergence result under two assumptions:



1. The x_n are in a sphere of radius R:
$$\exists R > 0,\ \forall n,\ \|x_n\| \leq R.$$

We can get a convergence result under two assumptions:



1. The x_n are in a sphere of radius R:
$$\exists R > 0,\ \forall n,\ \|x_n\| \leq R.$$
2. The two populations can be separated with a margin $\gamma > 0$.
$$\exists w^*,\ \|w^*\| = 1,\ \exists \gamma > 0,\ \forall n,\ y_n\,(x_n \cdot w^*) \geq \gamma/2.$$

To prove the convergence, let us make the assumption that there still is a misclassified sample at iteration k, and w^{k+1} is the weight vector updated with it. We have

$$\begin{aligned} w^{k+1} \cdot w^* &= \left(w^k + y_{n_k} x_{n_k}\right) \cdot w^* \\ &= w^k \cdot w^* + y_{n_k} \left(x_{n_k} \cdot w^*\right) \\ &\geq w^k \cdot w^* + \gamma/2 \\ &\geq (k+1)\,\gamma/2. \end{aligned}$$

To prove the convergence, let us make the assumption that there still is a misclassified sample at iteration k, and w^{k+1} is the weight vector updated with it. We have

$$\begin{aligned} w^{k+1} \cdot w^* &= \left(w^k + y_{n_k} x_{n_k}\right) \cdot w^* \\ &= w^k \cdot w^* + y_{n_k} \left(x_{n_k} \cdot w^*\right) \\ &\geq w^k \cdot w^* + \gamma/2 \\ &\geq (k+1)\,\gamma/2. \end{aligned}$$

Since

$$\|w^k\| \|w^*\| \geq w^k \cdot w^*,$$

we get

$$\begin{aligned} \|w^k\|^2 &\geq \left(w^k \cdot w^*\right)^2 / \|w^*\|^2 \\ &\geq k^2 \gamma^2 / 4. \end{aligned}$$

And

$$
\begin{aligned}
\|w^{k+1}\|^2 &= w^{k+1} \cdot w^{k+1} \\
&= \left(w^k + y_{n_k} x_{n_k}\right) \cdot \left(w^k + y_{n_k} x_{n_k}\right) \\
&= w^k \cdot w^k + 2 \underbrace{y_{n_k}\, w^k \cdot x_{n_k}}_{\leq 0} + \underbrace{\|x_{n_k}\|^2}_{\leq R^2} \\
&\leq \|w^k\|^2 + R^2 \\
&\leq (k+1)\, R^2.
\end{aligned}
$$

Putting these two results together, we get

$$k^2\gamma^2/4 \leq \|w^k\|^2 \leq \; k\,R^2$$

hence

$$k \leq 4R^2/\gamma^2,$$

hence no misclassified sample can remain after $\lfloor 4R^2/\gamma^2 \rfloor$ iterations.

Putting these two results together, we get

$$k^2\gamma^2/4 \leq \|w^k\|^2 \leq \; k\,R^2$$

hence

$$k \leq 4R^2/\gamma^2,$$

hence no misclassified sample can remain after $\lfloor 4R^2/\gamma^2 \rfloor$ iterations.

This result makes sense:

- The bound does not change if the population is scaled, and
- the larger the margin, the more quickly the algorithm classifies all the samples correctly.

The perceptron stops as soon as it finds a separating boundary.

Other algorithms maximize the distance of samples to the decision boundary, which improves robustness to noise.

The perceptron stops as soon as it finds a separating boundary.

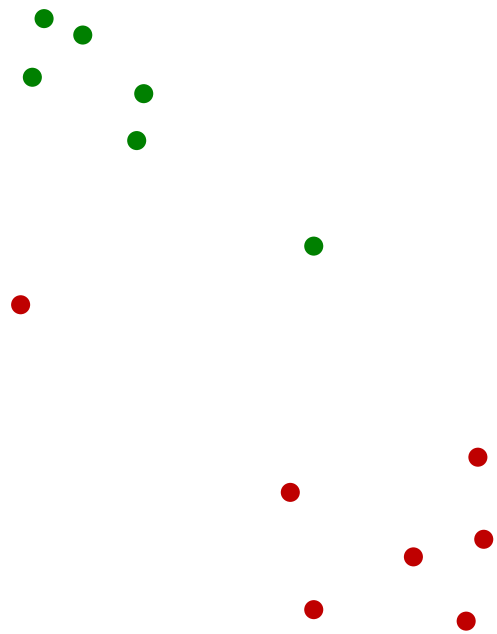
Other algorithms maximize the distance of samples to the decision boundary, which improves robustness to noise.

Support Vector Machines (SVM) achieve this by minimizing

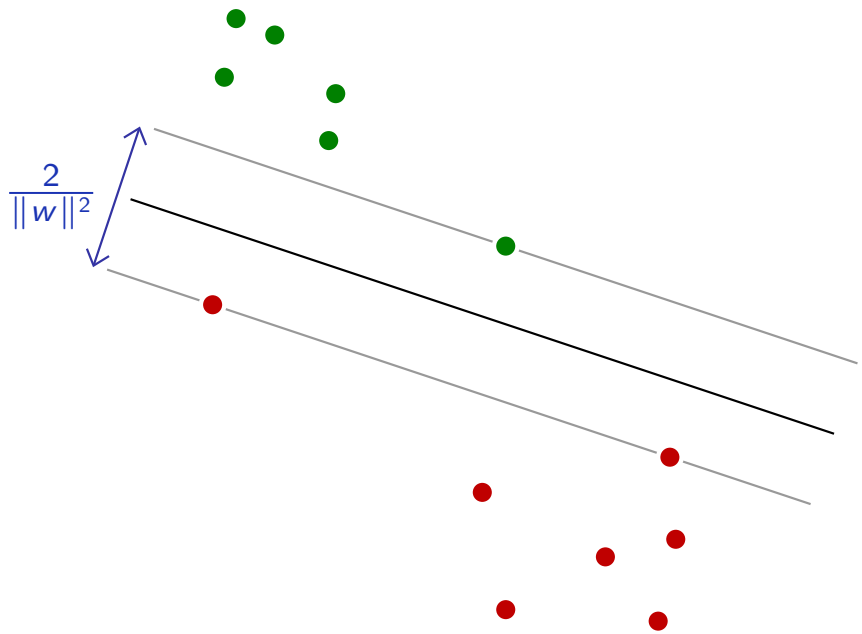
$$\mathscr{L}(w, b) = \lambda \|w\|^2 + \frac{1}{N} \sum_n \max(0, 1 - y_n(w \cdot x_n + b)),$$

which is convex and has a global optimum.

$$\mathscr{L}(w, b) = \lambda \|w\|^2 + \frac{1}{N} \sum_n \max(0, 1 - y_n(w \cdot x_n + b))$$

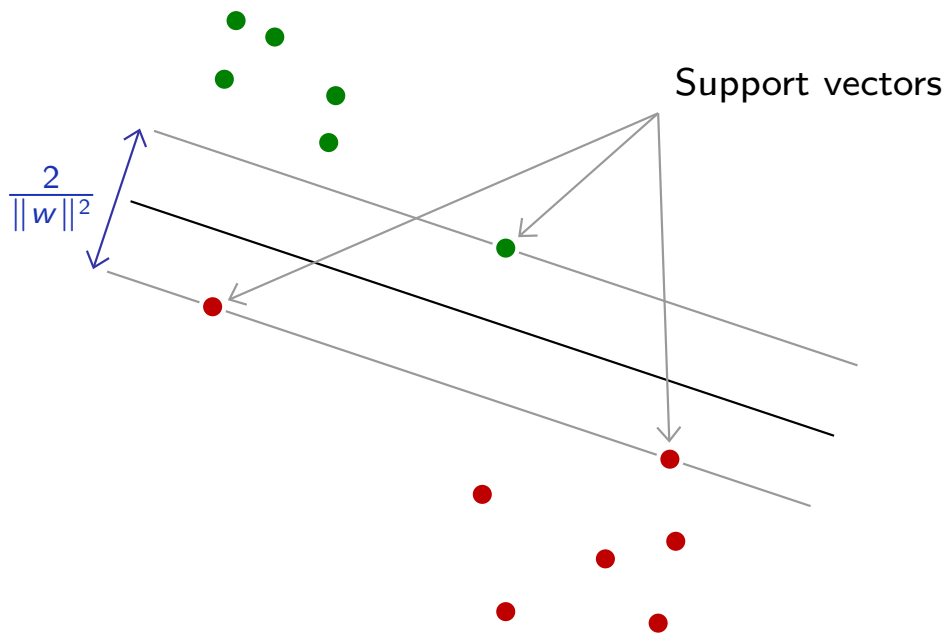


$$\mathscr{L}(w, b) = \lambda \|w\|^2 + \frac{1}{N} \sum_n \max(0, 1 - y_n(w \cdot x_n + b))$$



Minimizing $\max(0, 1 - y_n(w \cdot x_n + b))$ pushes the nth sample beyond the plane $w \cdot x + b = y_n$, and minimizing $\|w\|^2$ increases the distance between the $w \cdot x + b = \pm 1$.

$$\mathscr{L}(w, b) = \lambda \|w\|^2 + \frac{1}{N} \sum_n \max(0, 1 - y_n(w \cdot x_n + b))$$



Minimizing $\max(0, 1 - y_n(w \cdot x_n + b))$ pushes the nth sample beyond the plane $w \cdot x + b = y_n$, and minimizing $\|w\|^2$ increases the distance between the $w \cdot x + b = \pm 1$.

At convergence, only a small number of samples matter, the "support vectors".

The term

$$\max(0, 1 - \alpha)$$

is the so called "hinge loss"

Probabilistic interpretation of linear classifiers

The Linear Discriminant Analysis (LDA) algorithm provides a nice bridge between these linear classifiers and probabilistic modeling.

The Linear Discriminant Analysis (LDA) algorithm provides a nice bridge between these linear classifiers and probabilistic modeling.

Consider the following class populations

$$\forall y \in \{0,1\}, x \in \mathbb{R}^D,$$

$$\mu_{X|Y=y}(x) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp\left(-\frac{1}{2}(x - m_y)\Sigma^{-1}(x - m_y)^T\right).$$

That is, they are Gaussian with **the same covariance matrix Σ.** This is the **homoscedasticity** assumption.

We have

$$P(Y = 1 \mid X = x)$$

We have

$$P(Y = 1 \mid X = x) = \frac{\mu_{X|Y=1}(x)P(Y = 1)}{\mu_X(x)}$$

We have

$$P(Y = 1 \mid X = x) = \frac{\mu_{X|Y=1}(x)P(Y = 1)}{\mu_X(x)}$$

$$= \frac{\mu_{X|Y=1}(x)P(Y = 1)}{\mu_{X|Y=0}(x)P(Y = 0) + \mu_{X|Y=1}(x)P(Y = 1)}$$

We have

$$
\begin{aligned}
P(Y = 1 \mid X = x) &= \frac{\mu_{X|Y=1}(x) P(Y = 1)}{\mu_X(x)} \\
&= \frac{\mu_{X|Y=1}(x) P(Y = 1)}{\mu_{X|Y=0}(x) P(Y = 0) + \mu_{X|Y=1}(x) P(Y = 1)} \\
&= \frac{1}{1 + \frac{\mu_{X|Y=0}(x)}{\mu_{X|Y=1}(x)} \frac{P(Y=0)}{P(Y=1)}}.
\end{aligned}
$$

We have

$$
\begin{aligned}
P(Y = 1 \mid X = x) &= \frac{\mu_{X|Y=1}(x) P(Y = 1)}{\mu_X(x)} \\
&= \frac{\mu_{X|Y=1}(x) P(Y = 1)}{\mu_{X|Y=0}(x) P(Y = 0) + \mu_{X|Y=1}(x) P(Y = 1)} \\
&= \frac{1}{1 + \frac{\mu_{X|Y=0}(x)}{\mu_{X|Y=1}(x)} \frac{P(Y=0)}{P(Y=1)}}.
\end{aligned}
$$

It follows that, with

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

We have

$$P(Y = 1 \mid X = x) = \frac{\mu_{X|Y=1}(x)P(Y = 1)}{\mu_X(x)}$$
$$= \frac{\mu_{X|Y=1}(x)P(Y = 1)}{\mu_{X|Y=0}(x)P(Y = 0) + \mu_{X|Y=1}(x)P(Y = 1)}$$
$$= \frac{1}{1 + \frac{\mu_{X|Y=0}(x)}{\mu_{X|Y=1}(x)} \frac{P(Y=0)}{P(Y=1)}}.$$

It follows that, with

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

we get

$$P(Y = 1 \mid X = x) = \sigma\left(\log \frac{\mu_{X|Y=1}(x)}{\mu_{X|Y=0}(x)} + \log \frac{P(Y = 1)}{P(Y = 0)} \right).$$

So with our Gaussians $\mu_{X|Y=y}$ of same Σ, we get

$$P(Y = 1 \mid X = x)$$

So with our Gaussians $\mu_{X|Y=y}$ of same Σ, we get

$$P(Y = 1 \mid X = x) = \sigma\left(\log \frac{\mu_{X|Y=1}(x)}{\mu_{X|Y=0}(x)} + \underbrace{\log \frac{P(Y=1)}{P(Y=0)}}_{a} \right)$$

So with our Gaussians $\mu_{X|Y=y}$ of same Σ, we get

$$
\begin{aligned}
&P(Y=1 \mid X=x) \\
&\qquad = \sigma\Bigg(\log \frac{\mu_{X|Y=1}(x)}{\mu_{X|Y=0}(x)} + \underbrace{\log \frac{P(Y=1)}{P(Y=0)}}_{a} \Bigg) \\
&\qquad = \sigma\left(\log \mu_{X|Y=1}(x) - \log \mu_{X|Y=0}(x) + a \right)
\end{aligned}
$$

So with our Gaussians $\mu_{X|Y=y}$ of same Σ, we get

$$
\begin{aligned}
&P(Y=1 \mid X=x) \\
&\qquad = \sigma\Bigg(\log \frac{\mu_{X|Y=1}(x)}{\mu_{X|Y=0}(x)} + \underbrace{\log \frac{P(Y=1)}{P(Y=0)}}_{a} \Bigg) \\
&\qquad = \sigma\left(\log \mu_{X|Y=1}(x) - \log \mu_{X|Y=0}(x) + a \right) \\
&\qquad = \sigma\left(-\frac{1}{2}(x-m_1)\Sigma^{-1}(x-m_1)^T + \frac{1}{2}(x-m_0)\Sigma^{-1}(x-m_0)^T + a \right)
\end{aligned}
$$

So with our Gaussians $\mu_{X|Y=y}$ of same Σ, we get

$$
\begin{aligned}
&P(Y=1 \mid X=x) \\
&\qquad = \sigma\Bigg(\log \frac{\mu_{X|Y=1}(x)}{\mu_{X|Y=0}(x)} + \underbrace{\log \frac{P(Y=1)}{P(Y=0)}}_{a} \Bigg) \\
&\qquad = \sigma\left(\log \mu_{X|Y=1}(x) - \log \mu_{X|Y=0}(x) + a \right) \\
&\qquad = \sigma\left(-\frac{1}{2}(x-m_1)\Sigma^{-1}(x-m_1)^T + \frac{1}{2}(x-m_0)\Sigma^{-1}(x-m_0)^T + a \right) \\
&\qquad = \sigma\Bigg(-\frac{1}{2}x\Sigma^{-1}x^T + m_1\Sigma^{-1}x^T - \frac{1}{2}m_1\Sigma^{-1}m_1^T \\
&\qquad\qquad + \frac{1}{2}x\Sigma^{-1}x^T - m_0\Sigma^{-1}x^T + \frac{1}{2}m_0\Sigma^{-1}m_0^T + a \Bigg)
\end{aligned}
$$

So with our Gaussians $\mu_{X|Y=y}$ of same Σ, we get

$$
\begin{aligned}
&P(Y=1 \mid X=x) \\
&\qquad = \sigma\left(\log \frac{\mu_{X|Y=1}(x)}{\mu_{X|Y=0}(x)} + \underbrace{\log \frac{P(Y=1)}{P(Y=0)}}_{a} \right) \\
&\qquad = \sigma\left(\log \mu_{X|Y=1}(x) - \log \mu_{X|Y=0}(x) + a \right) \\
&\qquad = \sigma\left(-\frac{1}{2}(x-m_1)\Sigma^{-1}(x-m_1)^T + \frac{1}{2}(x-m_0)\Sigma^{-1}(x-m_0)^T + a \right) \\
&\qquad = \sigma\Bigg(-\frac{1}{2}x\Sigma^{-1}x^T + m_1\Sigma^{-1}x^T - \frac{1}{2}m_1\Sigma^{-1}m_1^T \\
&\qquad\qquad + \frac{1}{2}x\Sigma^{-1}x^T - m_0\Sigma^{-1}x^T + \frac{1}{2}m_0\Sigma^{-1}m_0^T + a \Bigg) \\
&\qquad = \sigma\Bigg(\underbrace{(m_1-m_0)\Sigma^{-1}}_{w} x^T + \underbrace{\frac{1}{2}\left(m_0\Sigma^{-1}m_0^T - m_1\Sigma^{-1}m_1^T \right) + a}_{b} \Bigg)
\end{aligned}
$$

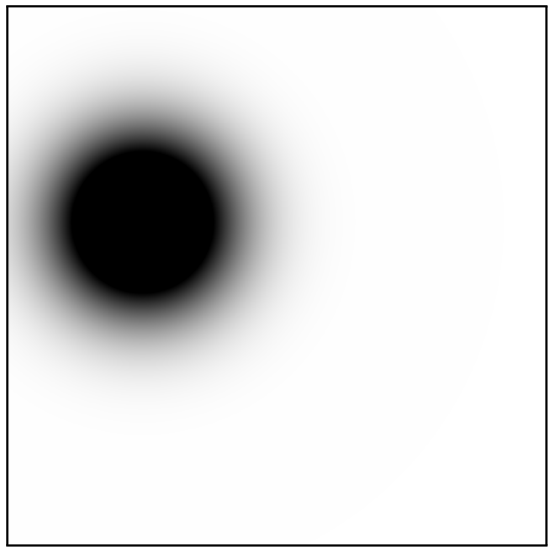
So with our Gaussians $\mu_{X|Y=y}$ of same Σ, we get

$$
\begin{aligned}
&P(Y=1 \mid X=x) \\
&\qquad = \sigma\Bigg(\log \frac{\mu_{X|Y=1}(x)}{\mu_{X|Y=0}(x)} + \underbrace{\log \frac{P(Y=1)}{P(Y=0)}}_{a} \Bigg) \\
&\qquad = \sigma\left(\log \mu_{X|Y=1}(x) - \log \mu_{X|Y=0}(x) + a \right) \\
&\qquad = \sigma\left(-\frac{1}{2}(x-m_1)\Sigma^{-1}(x-m_1)^T + \frac{1}{2}(x-m_0)\Sigma^{-1}(x-m_0)^T + a \right) \\
&\qquad = \sigma\Bigg(-\frac{1}{2}x\Sigma^{-1}x^T + m_1\Sigma^{-1}x^T - \frac{1}{2}m_1\Sigma^{-1}m_1^T \\
&\qquad\qquad + \frac{1}{2}x\Sigma^{-1}x^T - m_0\Sigma^{-1}x^T + \frac{1}{2}m_0\Sigma^{-1}m_0^T + a \Bigg) \\
&\qquad = \sigma\Bigg(\underbrace{(m_1-m_0)\Sigma^{-1}}_{w} x^T + \underbrace{\frac{1}{2}\left(m_0\Sigma^{-1}m_0^T - m_1\Sigma^{-1}m_1^T \right) + a}_{b} \Bigg) \\
&\qquad = \sigma(w \cdot x + b).
\end{aligned}
$$

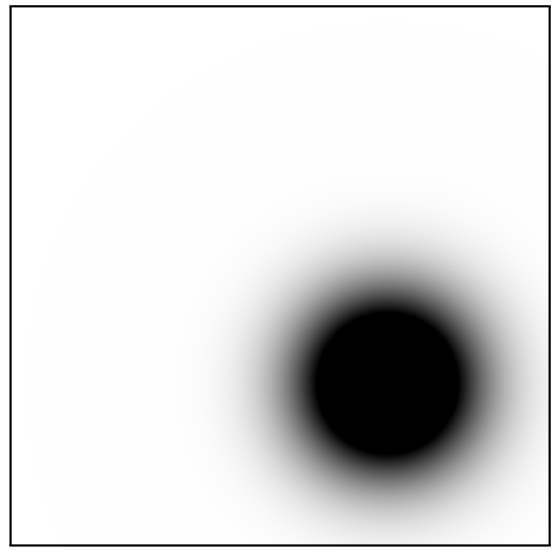
So with our Gaussians $\mu_{X|Y=y}$ of same Σ, we get

$$
\begin{aligned}
&P(Y=1 \mid X=x) \\
&\qquad = \sigma\left(\log \frac{\mu_{X|Y=1}(x)}{\mu_{X|Y=0}(x)} + \underbrace{\log \frac{P(Y=1)}{P(Y=0)}}_{a} \right) \\
&\qquad = \sigma\left(\log \mu_{X|Y=1}(x) - \log \mu_{X|Y=0}(x) + a \right) \\
&\qquad = \sigma\left(-\frac{1}{2}(x-m_1)\Sigma^{-1}(x-m_1)^T + \frac{1}{2}(x-m_0)\Sigma^{-1}(x-m_0)^T + a \right) \\
&\qquad = \sigma\Bigg(-\frac{1}{2}x\Sigma^{-1}x^T + m_1\Sigma^{-1}x^T - \frac{1}{2}m_1\Sigma^{-1}m_1^T \\
&\qquad\qquad + \frac{1}{2}x\Sigma^{-1}x^T - m_0\Sigma^{-1}x^T + \frac{1}{2}m_0\Sigma^{-1}m_0^T + a \Bigg) \\
&\qquad = \sigma\Bigg(\underbrace{(m_1-m_0)\Sigma^{-1}}_{w} x^T + \underbrace{\frac{1}{2}\left(m_0\Sigma^{-1}m_0^T - m_1\Sigma^{-1}m_1^T \right) + a}_{b} \Bigg) \\
&\qquad = \sigma(w \cdot x + b).
\end{aligned}
$$

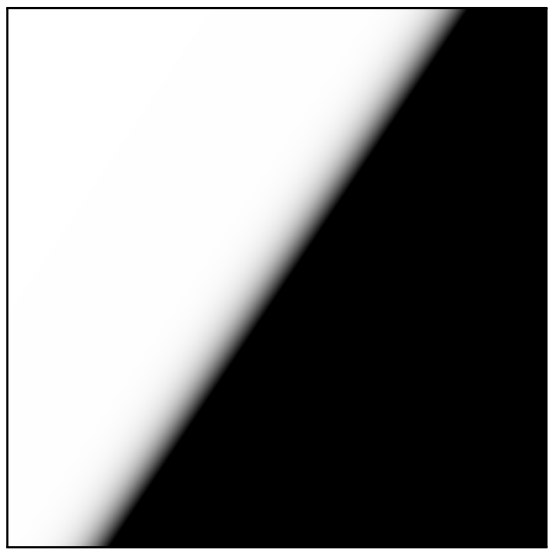
The homoscedasticity makes the second-order terms vanish.



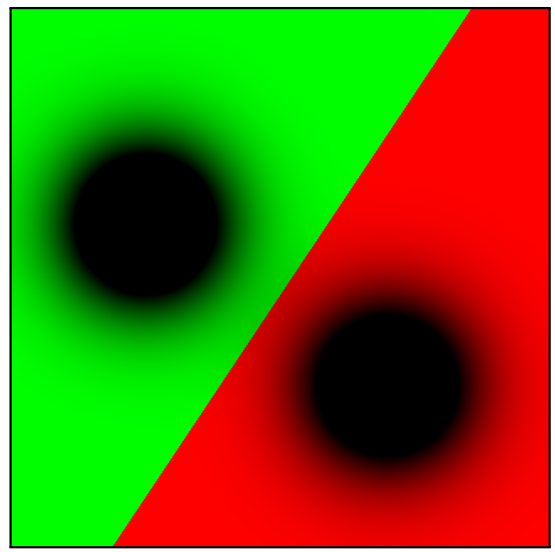
$\mu_{X|Y=0}$

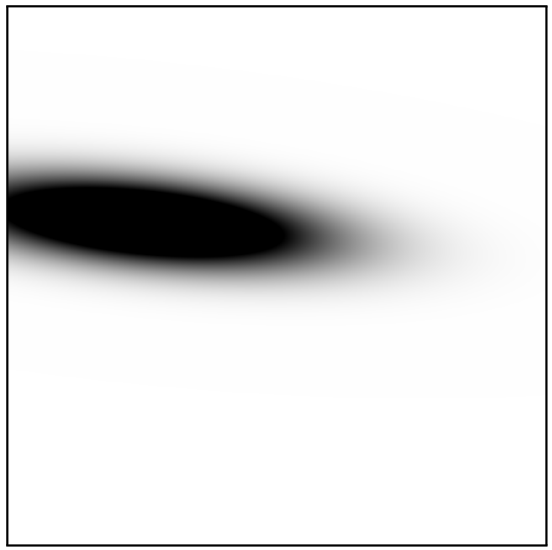


$\mu_{X|Y=1}$

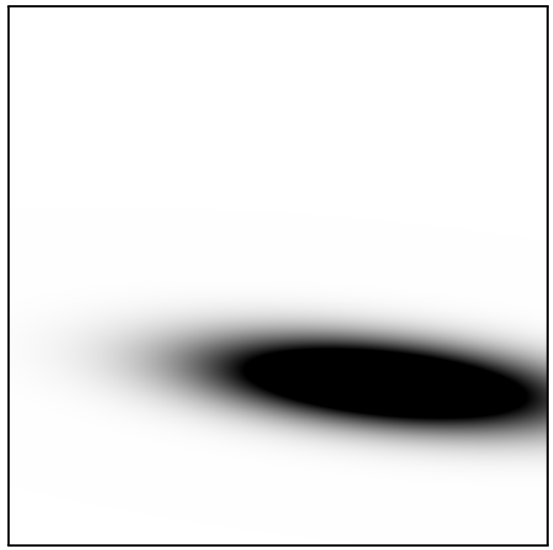


$P(Y = 1 \mid X = x)$

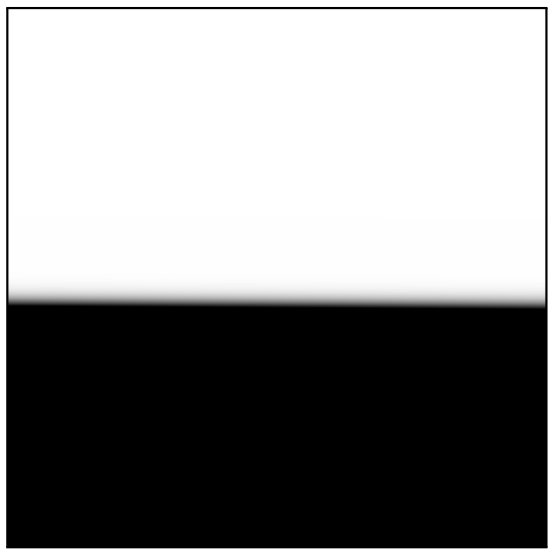




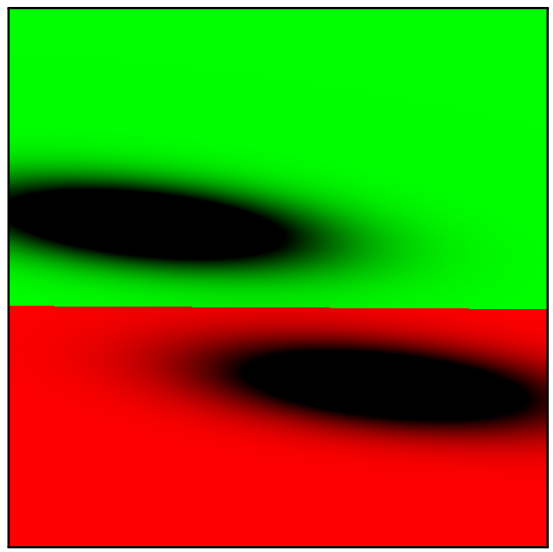
$\mu_{X|Y=0}$

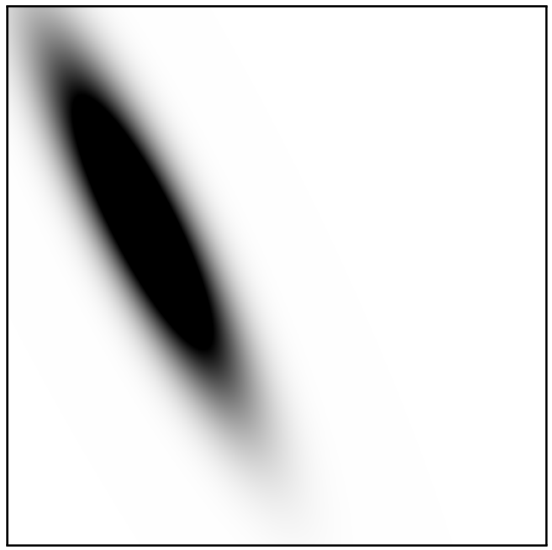


$\mu_{X|Y=1}$



$P(Y = 1 \mid X = x)$

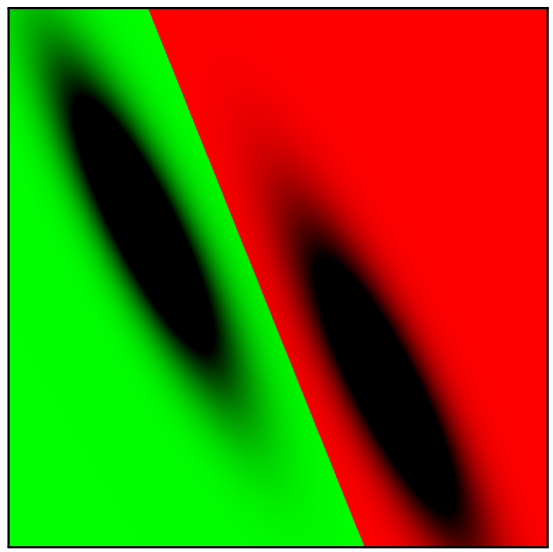




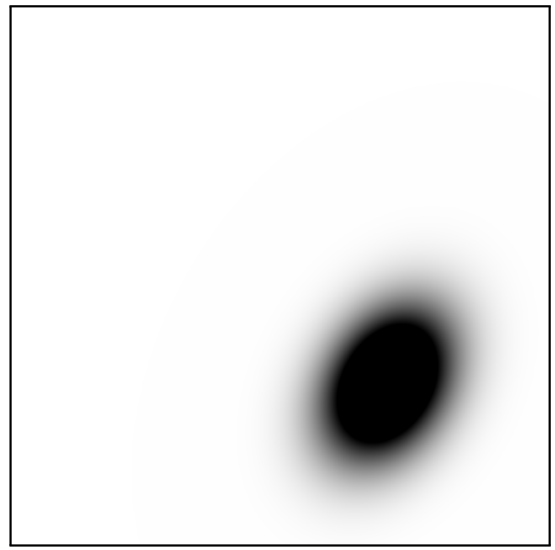
$\mu_{X|Y=0}$

$\mu_{X|Y=1}$

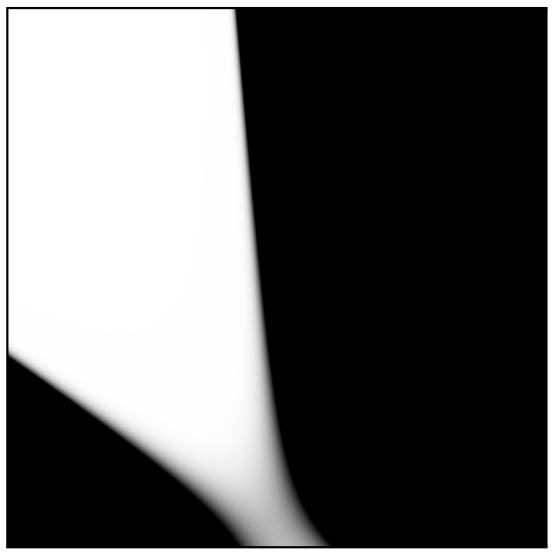
$P(Y = 1 \mid X = x)$



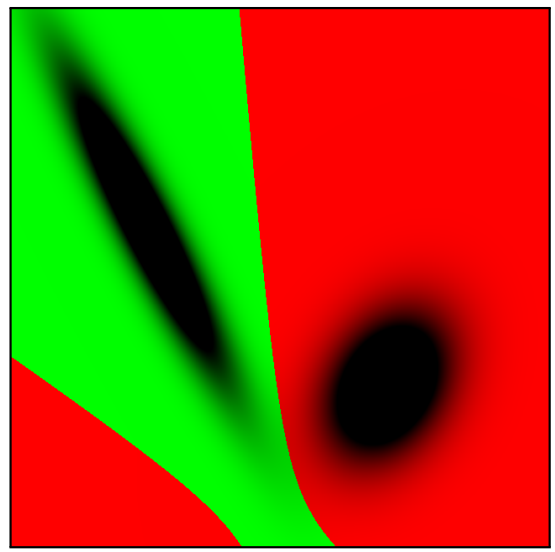
$\mu_{X|Y=0}$



$\mu_{X|Y=1}$



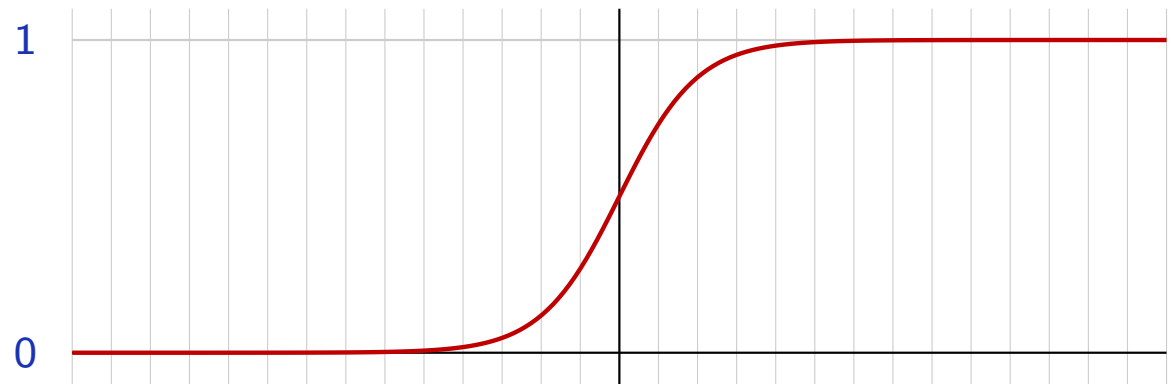
$P(Y = 1 \mid X = x)$



Note that the (logistic) sigmoid function

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

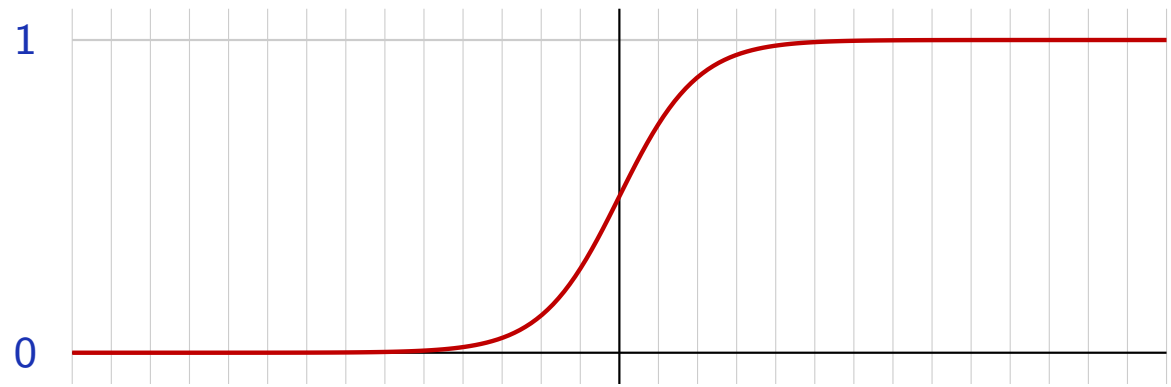
looks like a "soft heavyside"



Note that the (logistic) sigmoid function

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

looks like a "soft heavyside"



So the overall model

$$f(x; w, b) = \sigma(w \cdot x + b)$$

looks very similar to the perceptron.

We can use the model from LDA

$$f(x; w, b) = \sigma(w \cdot x + b)$$

but instead of modeling the densities and derive the values of w and b, directly compute them by maximizing their probability given the training data.

We can use the model from LDA

$$f(x; w, b) = \sigma(w \cdot x + b)$$

but instead of modeling the densities and derive the values of w and b, directly compute them by maximizing their probability given the training data.

First, to simplify the next slide, note that we have

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

hence if Y takes value in $\{-1, 1\}$ then

$$\forall y \in \{-1, 1\}, \quad P(Y = y \mid X = x) = \sigma(y(w \cdot x + b)).$$

We have

$$\begin{aligned}
&\log \mu_{W,B}(w, b \mid \mathscr{D} = \mathbf{d}) \\
&\qquad = \log \frac{\mu_{\mathscr{D}}(\mathbf{d} \mid W = w, B = b)\, \mu_{W,B}(w, b)}{\mu_{\mathscr{D}}(\mathbf{d})} \\
&\qquad = \log \mu_{\mathscr{D}}(\mathbf{d} \mid W = w, B = b) + \log \mu_{W,B}(w, b) - \log Z \\
&\qquad = \sum_n \log \sigma(y_n(w \cdot x_n + b)) + \log \mu_{W,B}(w, b) - \log Z'
\end{aligned}$$

We have

$$\begin{aligned}
&\log \mu_{W,B}(w, b \mid \mathscr{D} = \mathbf{d}) \\
&\qquad = \log \frac{\mu_{\mathscr{D}}(\mathbf{d} \mid W = w, B = b)\, \mu_{W,B}(w, b)}{\mu_{\mathscr{D}}(\mathbf{d})} \\
&\qquad = \log \mu_{\mathscr{D}}(\mathbf{d} \mid W = w, B = b) + \log \mu_{W,B}(w, b) - \log Z \\
&\qquad = \sum_n \log \sigma(y_n(w \cdot x_n + b)) + \log \mu_{W,B}(w, b) - \log Z'
\end{aligned}$$

This is the **logistic regression**, whose loss aims at minimizing

$$-\log \sigma(y_n f(x_n))$$

Although the probabilistic and Bayesian formulations may be helpful in certain contexts, the bulk of deep learning is disconnected from such modeling.

Although the probabilistic and Bayesian formulations may be helpful in certain contexts, the bulk of deep learning is disconnected from such modeling.

We will come back sometime to a probabilistic interpretation, but most of the methods will be envisioned from the signal-processing and optimization angles.

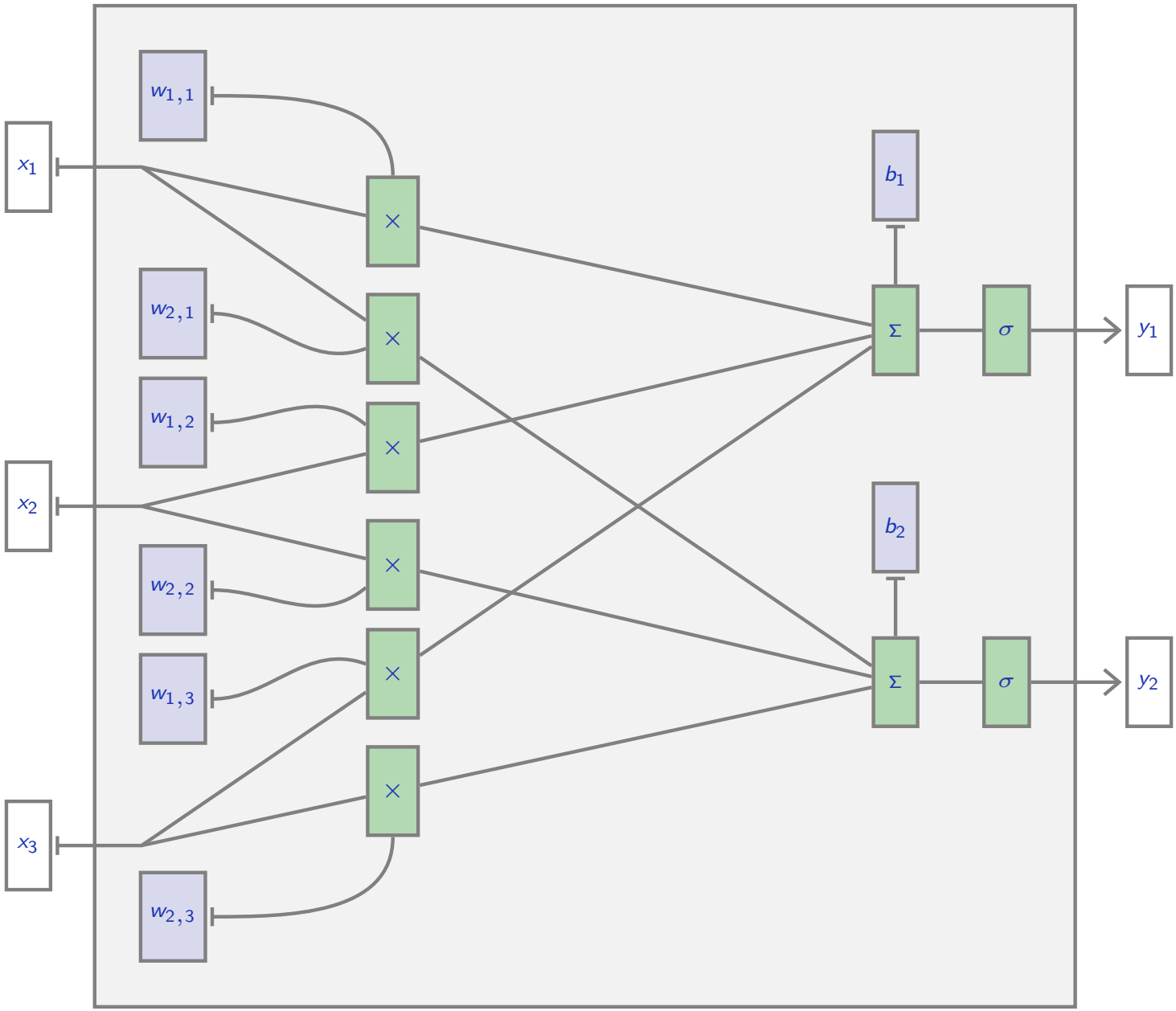
Multi-dimensional output

We can combine multiple linear predictors into a "layer" that takes several inputs and produces several outputs:

$$\forall i = 1, \ldots, M, \; y_i = \sigma \left(\sum_{j=1}^{N} w_{i,j} x_j + b_i \right)$$

where b_i is the "bias" of the i-th unit, and $w_{i,1}, \ldots, w_{i,N}$ are its weights.

With $M = 2$ and $N = 3$, we can picture such a layer as



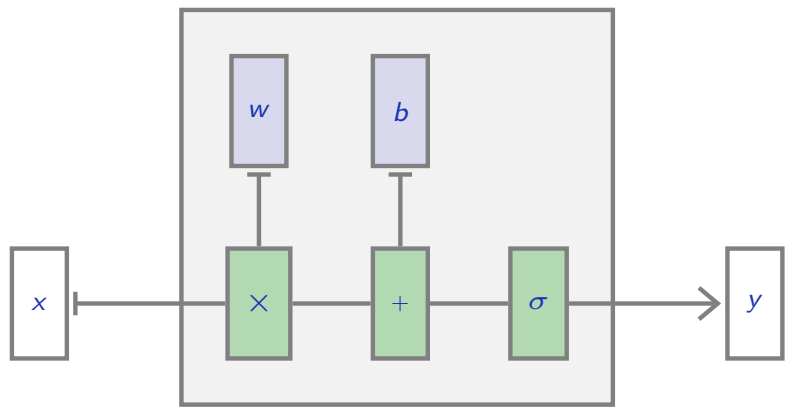
If we forget the historical interpretation as "neurons", we can use a clearer algebraic / tensorial formulation:

$$y = \sigma\left(wx + b\right)$$

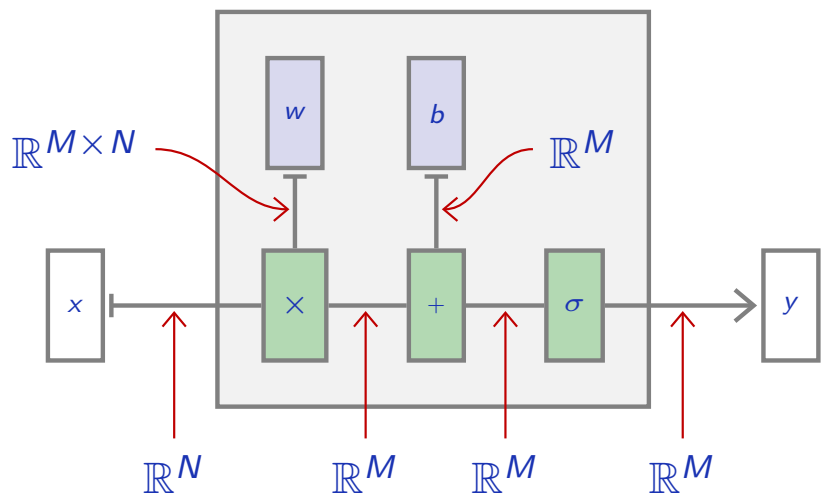
where $x \in \mathbb{R}^N$, $w \in \mathbb{R}^{M \times N}$, $b \in \mathbb{R}^M$, $y \in \mathbb{R}^M$, and σ denotes a component-wise extension of the $\mathbb{R} \to \mathbb{R}$ mapping:

$$\sigma : (y_1, \ldots, y_M) \mapsto (\sigma(y_1), \ldots, \sigma(y_M)).$$

With "$\times$" for the matrix-vector product, the "tensorial block figure" remains almost identical to that of the single neuron.



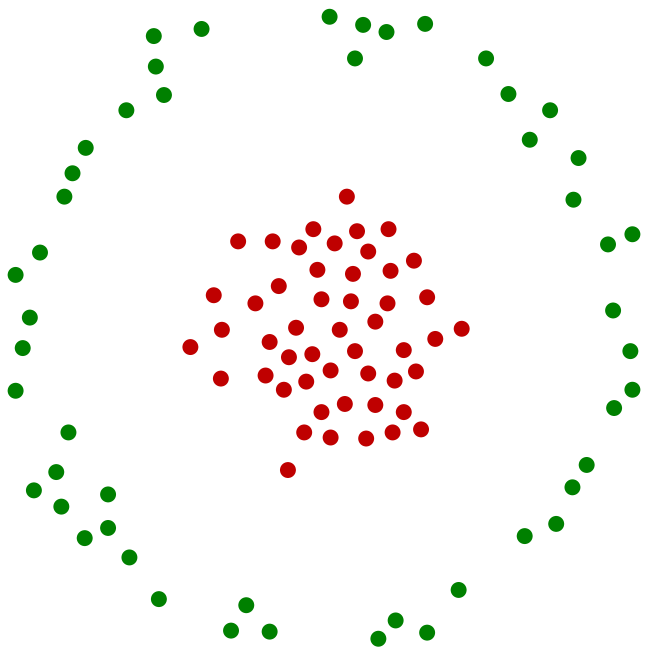
With "$\times$" for the matrix-vector product, the "tensorial block figure" remains almost identical to that of the single neuron.



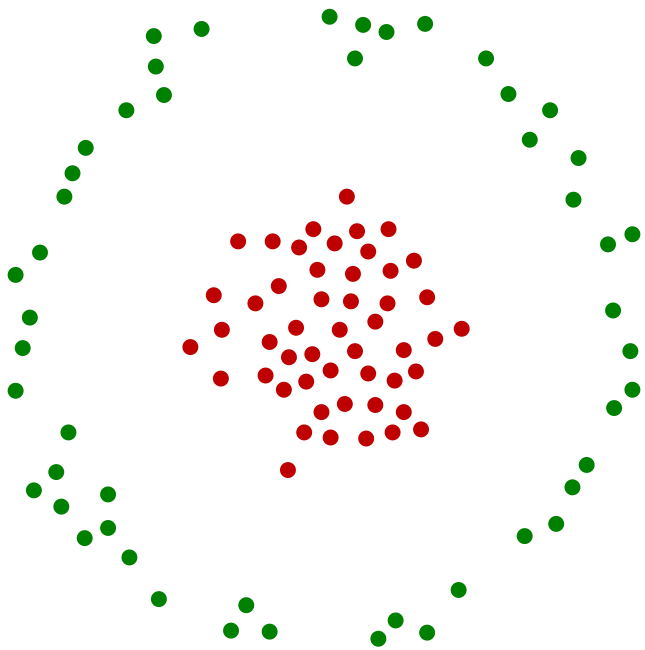
Limitations of linear predictors, feature design

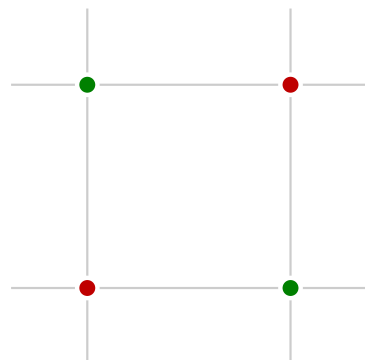
The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be **linearly separable**.

The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be **linearly separable**.



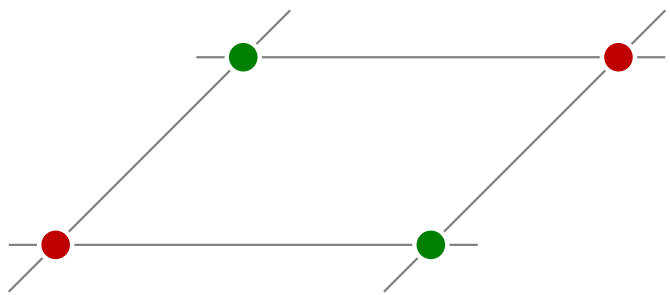
The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be **linearly separable**.





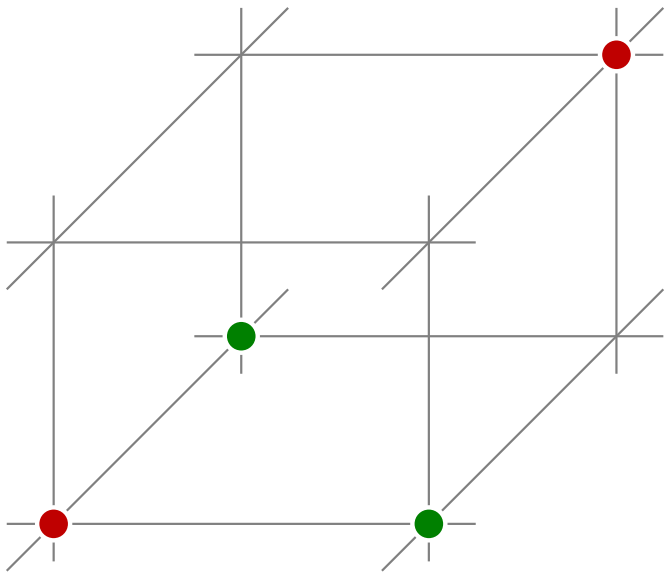
"xor"

The xor example can be solved by pre-processing the data to make the two populations linearly separable:



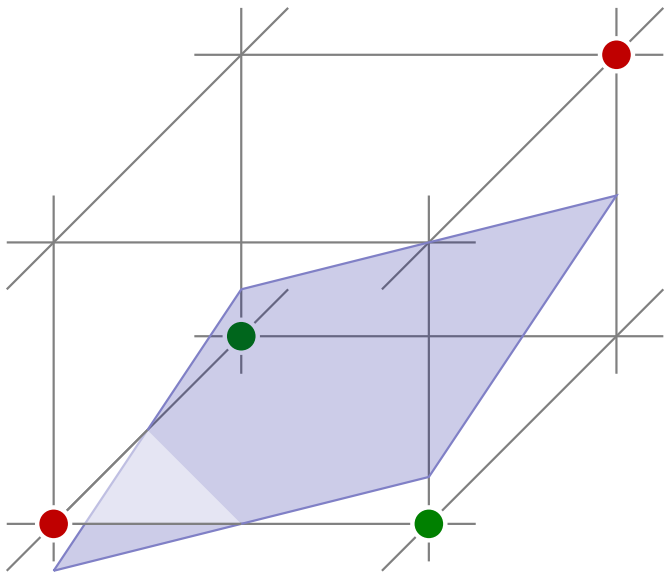
The xor example can be solved by pre-processing the data to make the two populations linearly separable:

$$\Phi : (x_u, x_v) \mapsto (x_u, x_v, x_u x_v).$$



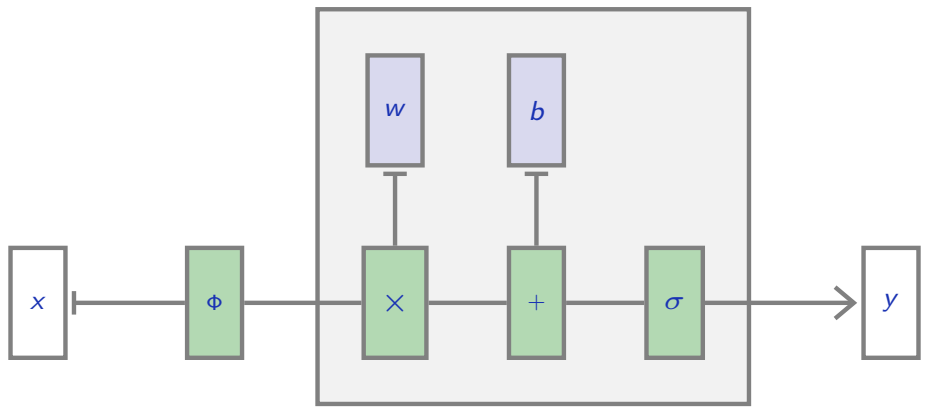
The xor example can be solved by pre-processing the data to make the two populations linearly separable:

$$\Phi : (x_u, x_v) \mapsto (x_u, x_v, x_u x_v).$$



So we can model the xor with

$$f(x) = \sigma(w\,\Phi(x) + b).$$


w
b
x
Φ
×
+
σ
y

This is similar to the polynomial regression. If we have

$$\Phi : x \mapsto (1, x, x^2, \ldots, x^D)$$

and

$$\alpha = (\alpha_0, \ldots, \alpha_D)$$

then

$$\sum_{d=0}^{D} \alpha_d x^d = \alpha \cdot \Phi(x).$$

This is similar to the polynomial regression. If we have

$$\Phi : x \mapsto (1, x, x^2, \ldots, x^D)$$

and

$$\alpha = (\alpha_0, \ldots, \alpha_D)$$

then

$$\sum_{d=0}^{D} \alpha_d x^d = \alpha \cdot \Phi(x).$$

By increasing D, we can approximate any continuous real function on a compact space (Stone-Weierstrass theorem).

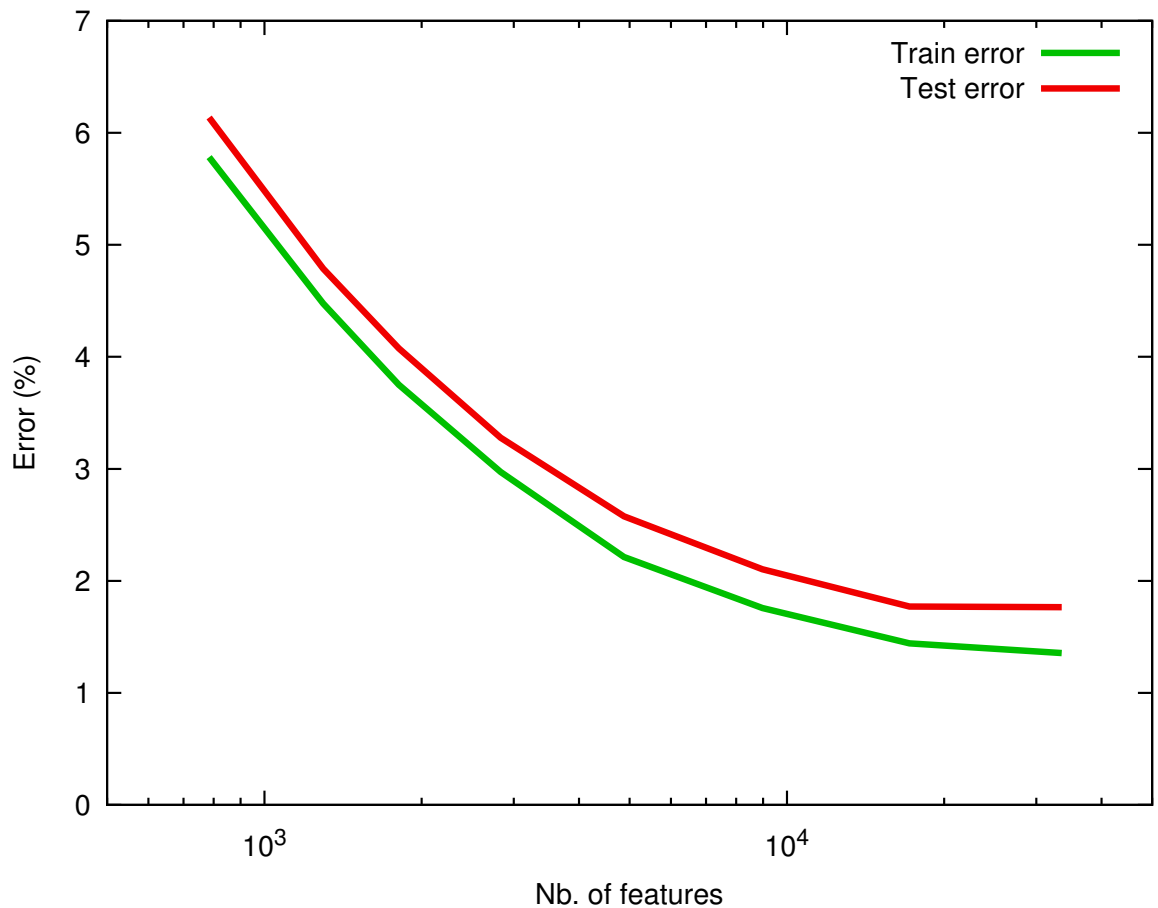
It means that we can make the capacity as high as we want.

We can apply the same to a more realistic binary classification problem: MNIST's "8" vs. the other classes with a perceptron.

The original 28×28 features are supplemented with the products of pairs of features taken at random.

We can apply the same to a more realistic binary classification problem: MNIST's "8" vs. the other classes with a perceptron.

The original 28×28 features are supplemented with the products of pairs of features taken at random.



Remember the bias-variance tradeoff.

$$\mathbb{E}((Y - y)^2) = \underbrace{(\mathbb{E}(Y) - y)^2}_{\text{Bias}} + \underbrace{\mathbb{V}(Y)}_{\text{Variance}}.$$

The right class of models reduces the bias more and increases the variance less.

Remember the bias-variance tradeoff.

$$\mathbb{E}((Y - y)^2) = \underbrace{(\mathbb{E}(Y) - y)^2}_{\text{Bias}} + \underbrace{\mathbb{V}(Y)}_{\text{Variance}}.$$

The right class of models reduces the bias more and increases the variance less.

Beside increasing capacity to reduce the bias, "feature design" may also be a way of reducing capacity without hurting the bias, or with improving it.

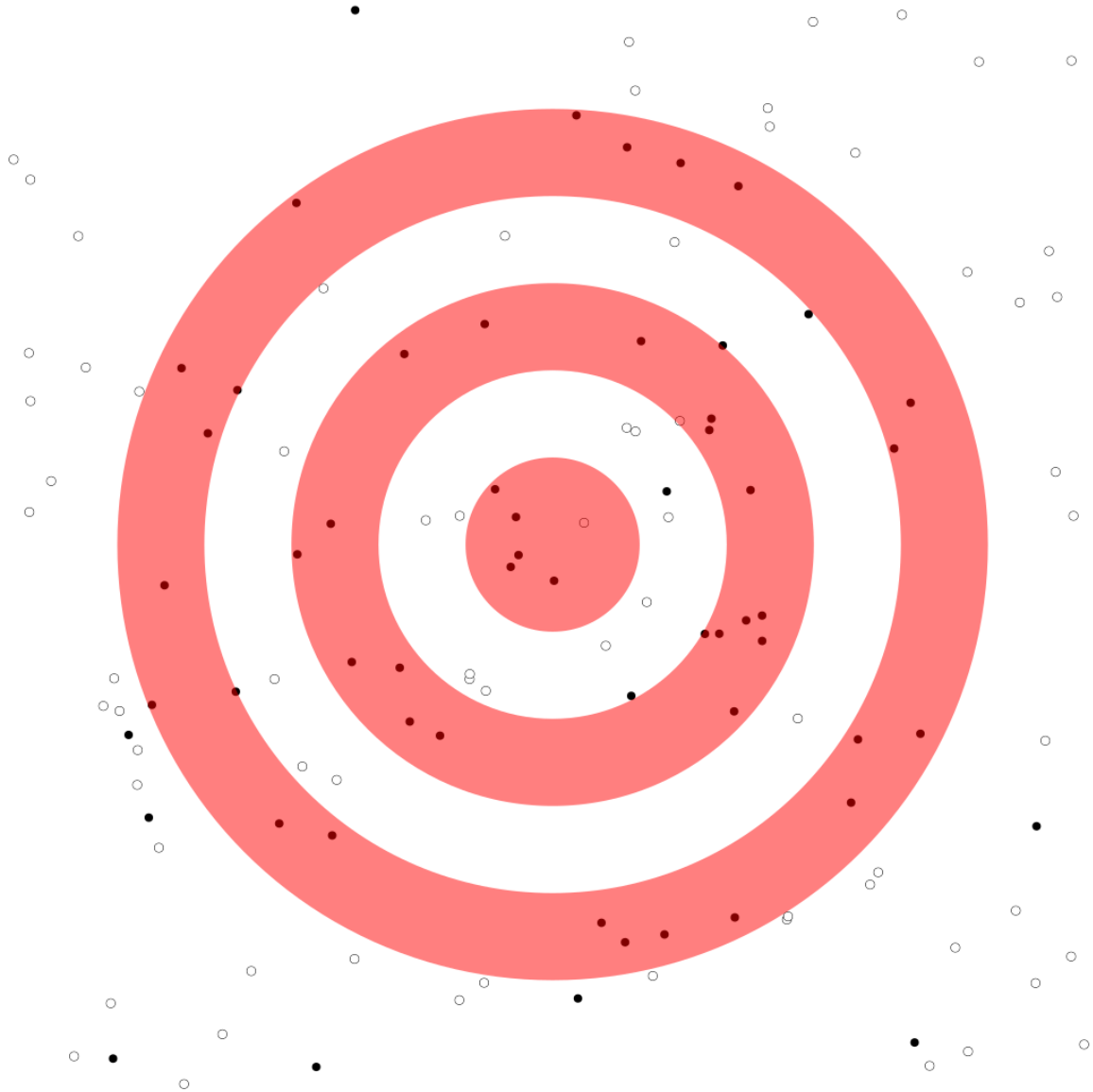
Remember the bias-variance tradeoff.

$$\mathbb{E}((Y - y)^2) = \underbrace{(\mathbb{E}(Y) - y)^2}_{\text{Bias}} + \underbrace{\mathbb{V}(Y)}_{\text{Variance}}.$$

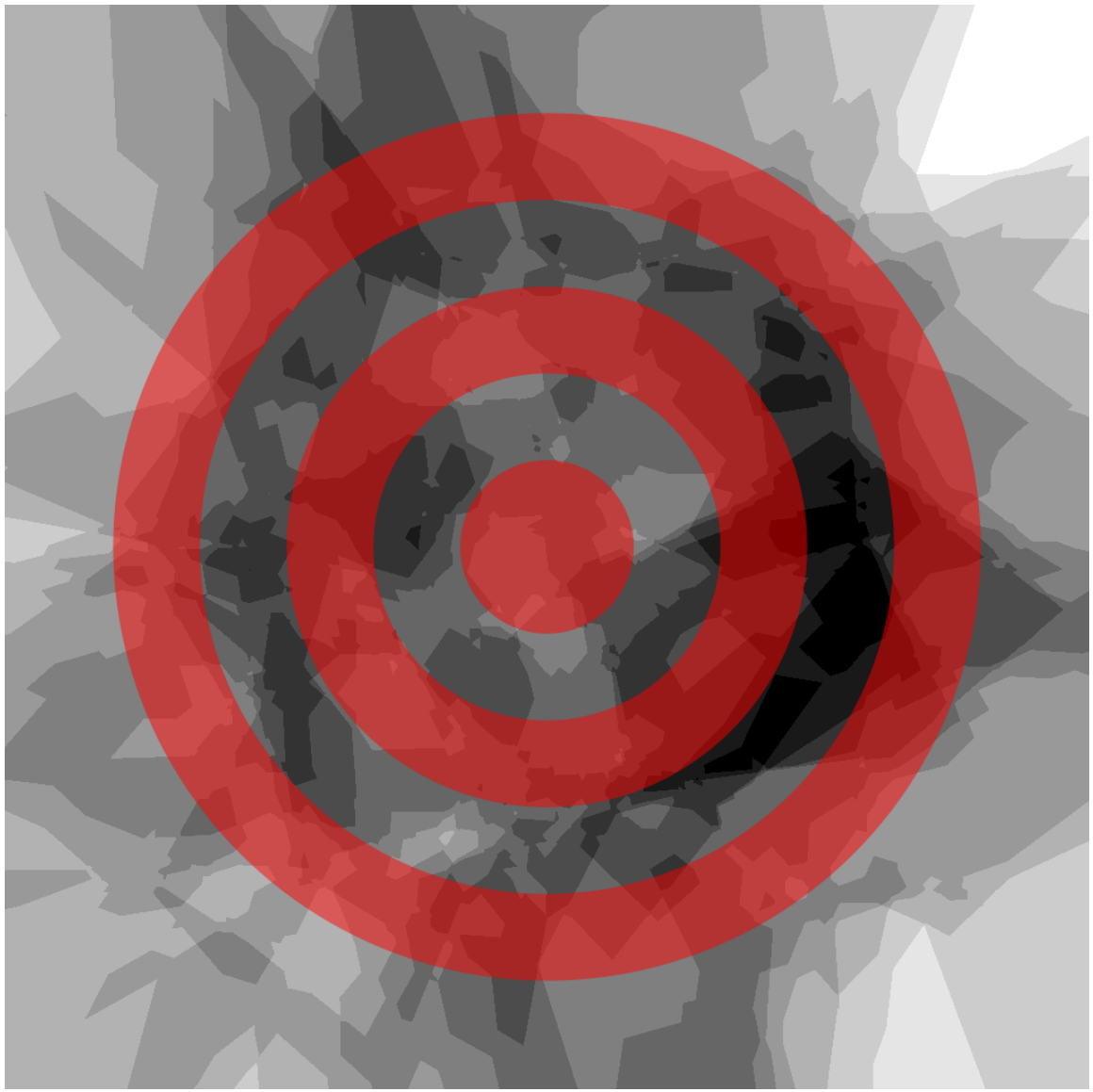
The right class of models reduces the bias more and increases the variance less.

Beside increasing capacity to reduce the bias, "feature design" may also be a way of reducing capacity without hurting the bias, or with improving it.

In particular, good features should be invariant to perturbations of the signal known to keep the value to predict unchanged.

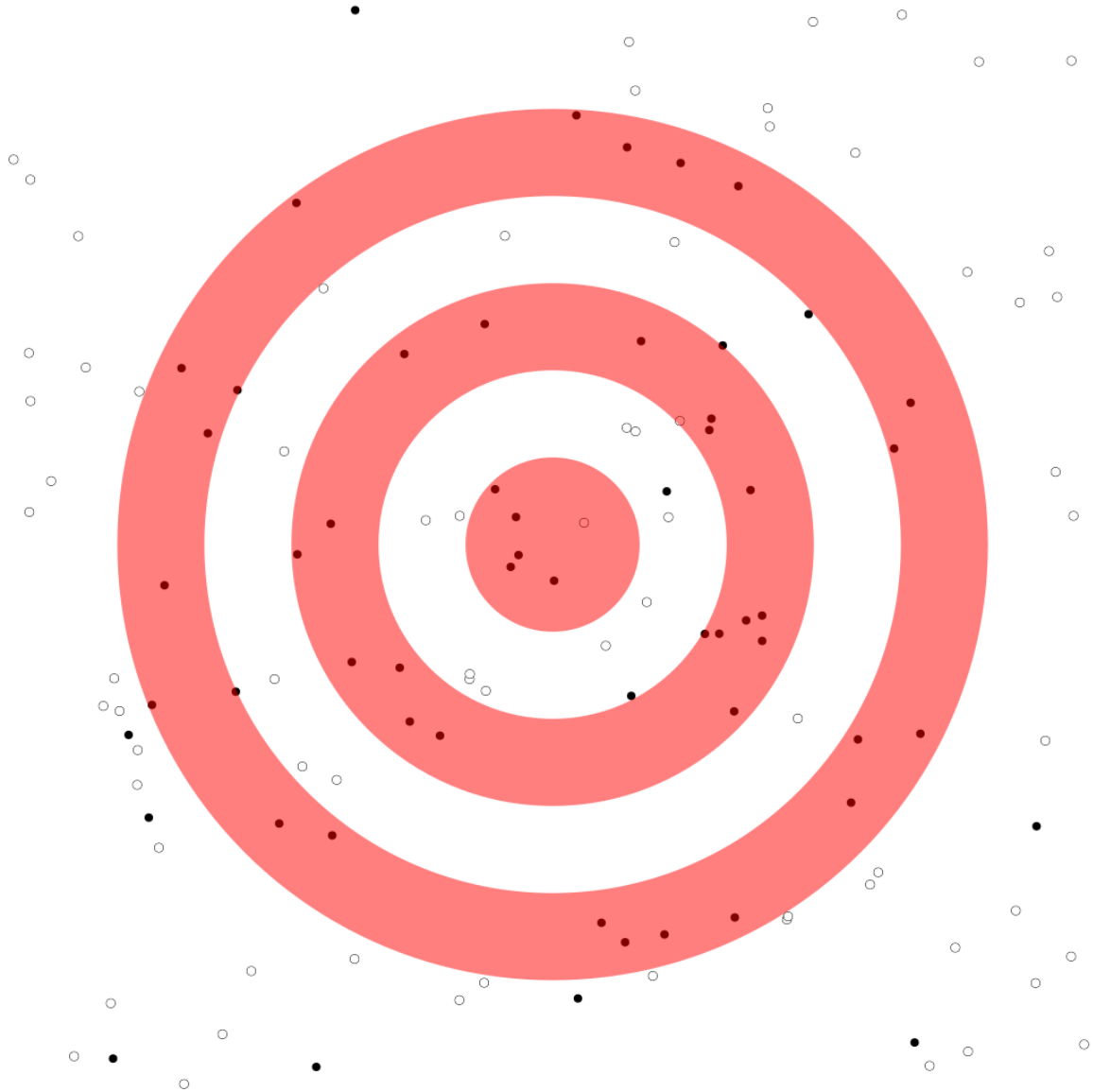


Training points

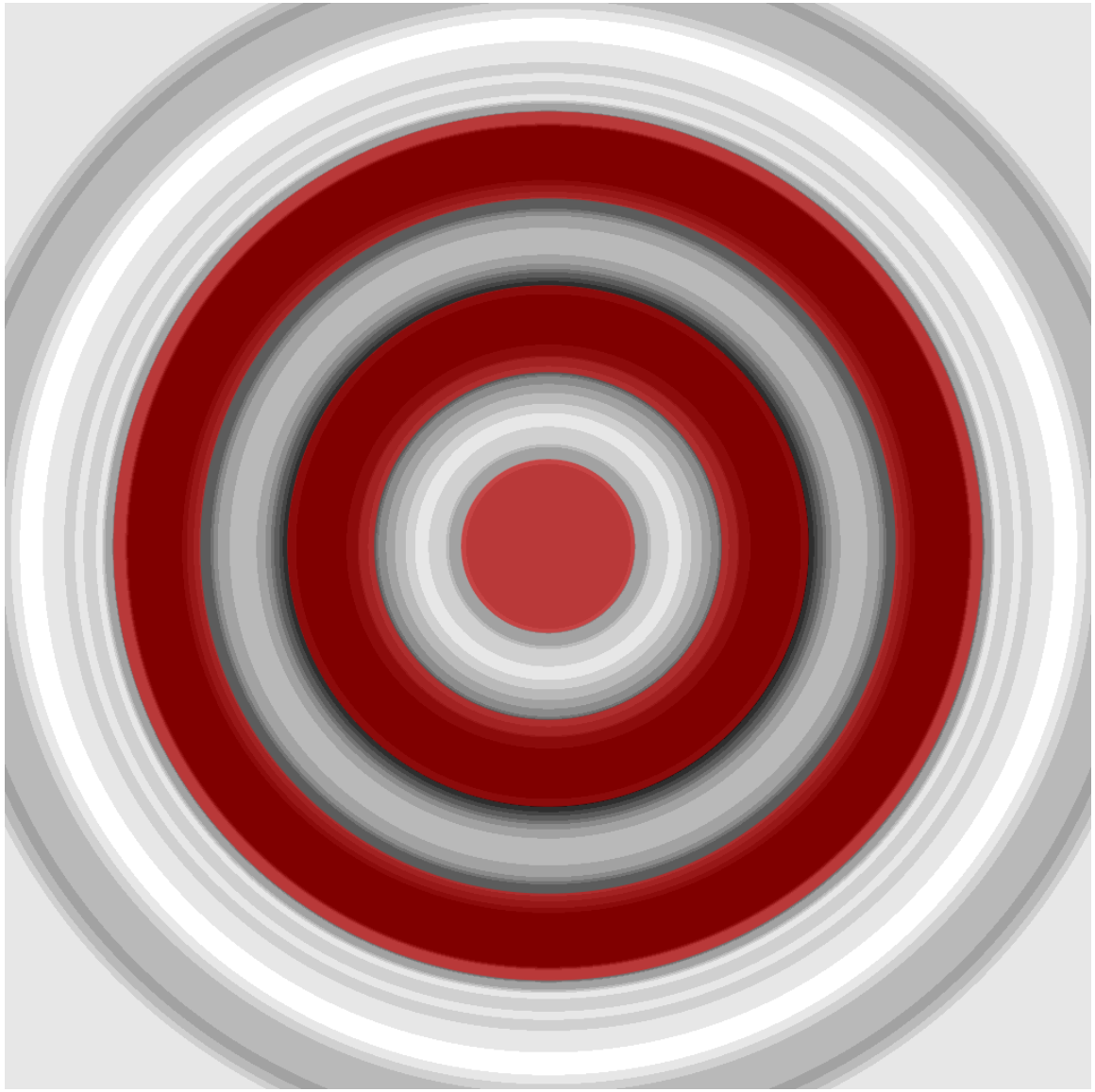


Votes (K=11)

Prediction (K=11)



Training points

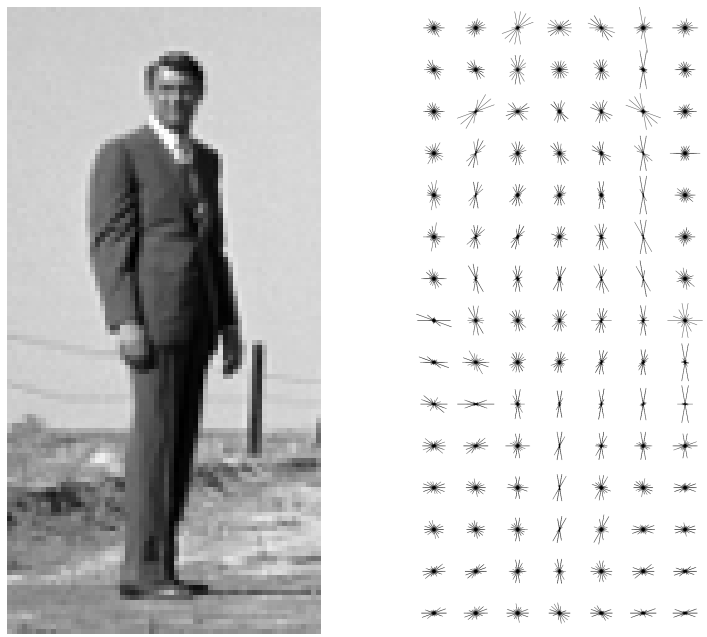


Votes, radial feature (K=11)

Prediction, radial feature (K=11)

A classical example is the "Histogram of Oriented Gradient" descriptors (HOG), initially designed for person detection.

Roughly: divide the image in 8×8 blocks, compute in each the distribution of edge orientations over 9 bins.



Dalal and Triggs (2005) combined them with a SVM, and Dollar et al. (2009) extended them with other modalities into the "channel features".

Many methods (perceptron, SVM, k-means, PCA, etc.) only require to compute $\kappa(x, x') = \Phi(x) \cdot \Phi(x')$ for any (x, x').

So one needs to specify κ alone, and may keep Φ undefined.

Many methods (perceptron, SVM, k-means, PCA, etc.) only require to compute $\kappa(x, x') = \Phi(x) \cdot \Phi(x')$ for any (x, x').

So one needs to specify κ alone, and may keep Φ undefined.

This is the **kernel trick**, which we will not talk about in this course.

Training a model composed of manually engineered features and a parametric model such as logistic regression is now referred to as **"shallow learning"**.

The signal goes through a single processing trained from data.

The end

References

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 886–893, 2005.

P. Dollar, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *British Machine Vision Conference*, pages 91.1–91.11, 2009.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

F. Rosenblatt. The perceptron–A perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.