The Kernel Trick

There are a lot of good presentations of the kernel trick available online. Most of them present it in the context of Support Vector Machines (SVMs), SVMs and the kernel trick are both advances on basic perceptrons and historically came into wide use at the same time. In fact, there probably are not many applications for which one would want to use the kernel trick with an ordinary perceptron rather than an SVM.

However, the idea - the trick - doesn't require the extra complication of SVMs and so this presentation leaves them out.

One excellent online exposition that goes more deeply into the math than these notes do is:

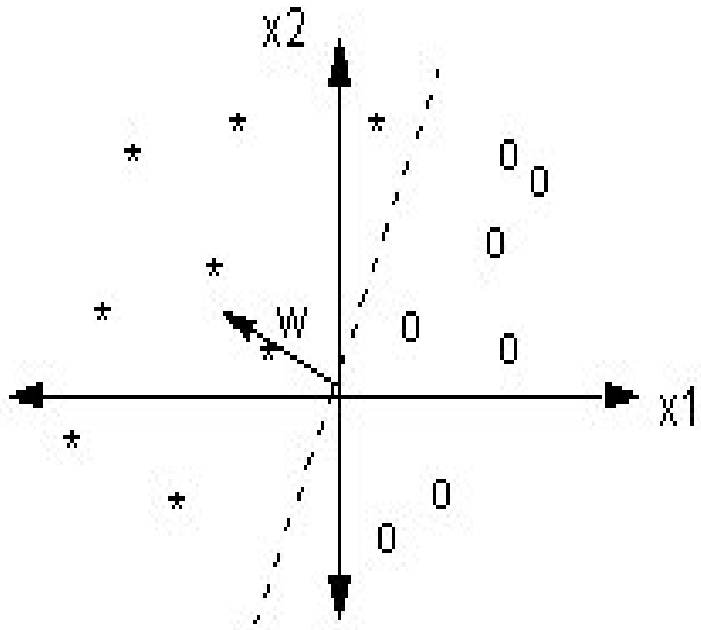
https://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring-2012/lecture-notes/MIT15_097S12_lec13.pdf

The Kernel Trick

OUTLINE

- Limitation: Perceptrons are linear
- Partial fix:
  - Linear isn't the same in all spaces
  - Compute linear operations in space B using non-linear operations in space A to get non-linear boundary in space A
- Historical note

Perceptrons



Inner products are the key operation

Points are classified by taking an inner product with the weight vector *w* and comparing with a threshold.

And in the perceptron learning algorithm, all operations on points in the space can be characterized as inner products.

Perceptrons

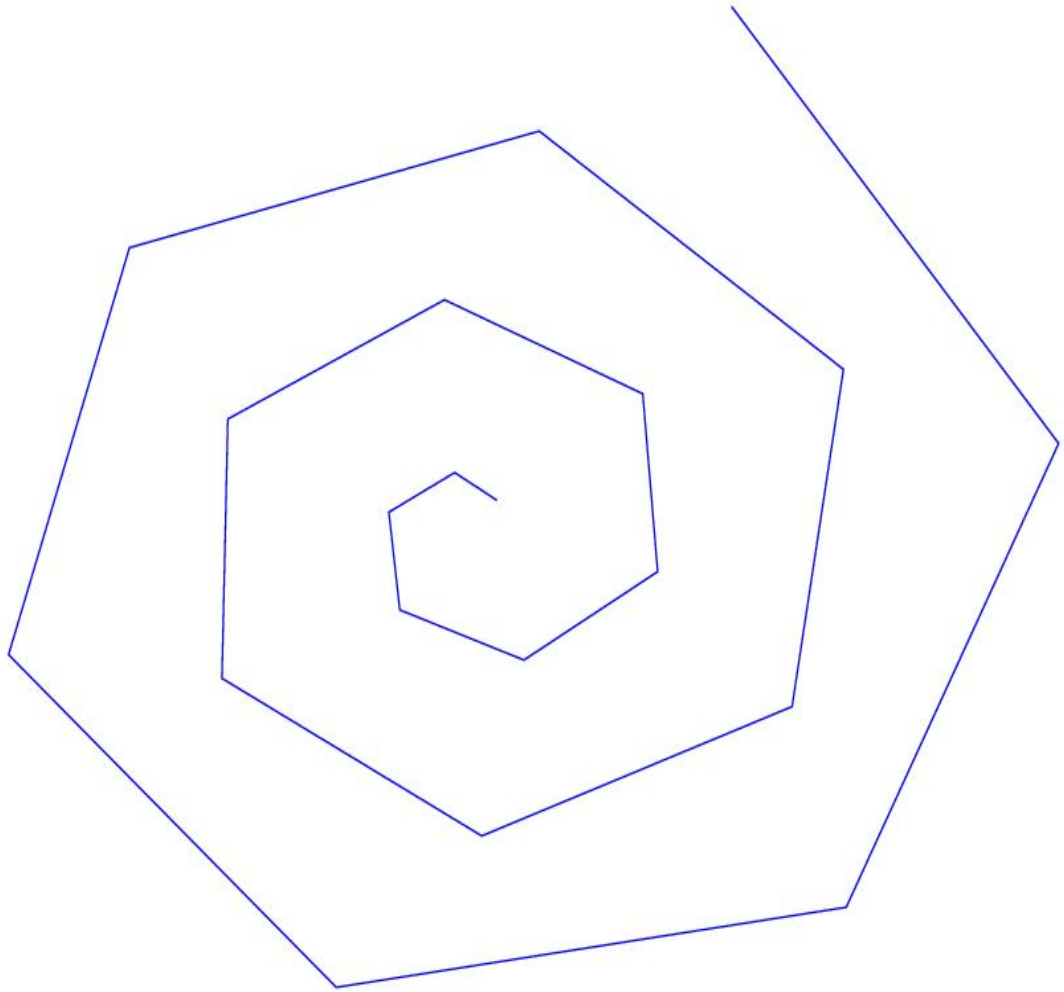
Two suggestions for extensions of the perceptron idea that get us to the same place in the end:

Map the points to another space, e.g., polar coordinates, and learn a perceptron there.

Replace inner products by another function that still permits the proof of the learning algorithm to go through.

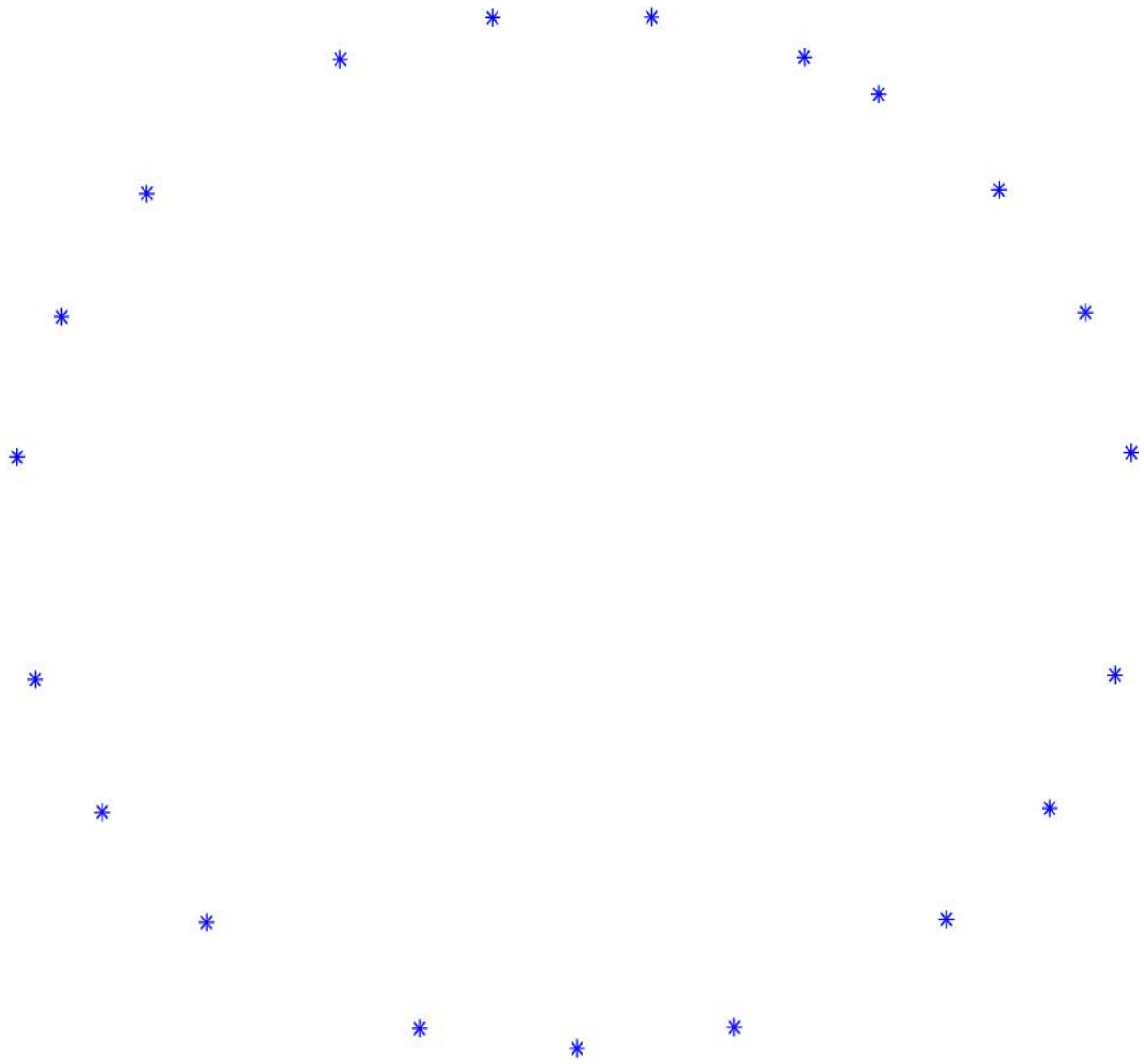
Lines in polar coordinate space

20 points on the “line” $r = \theta$ plotted in x,y coordinates



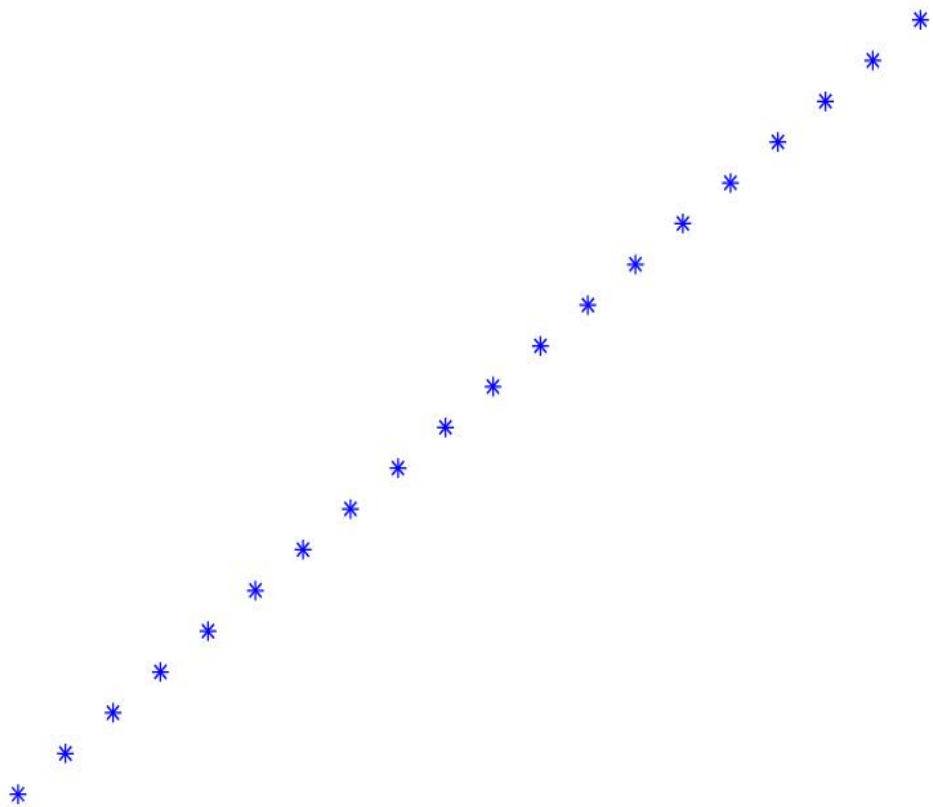
Polar coordinates

20 points on the “line” $r = 1$



Polar coordinates

20 points on the “line” $\theta = \pi/4$



So if we thought that two classes of x,y points could be separated by a polar coordinate “line”, we could map the points to polar coordinates and train a perceptron on the polar coordinate version of the two classes.

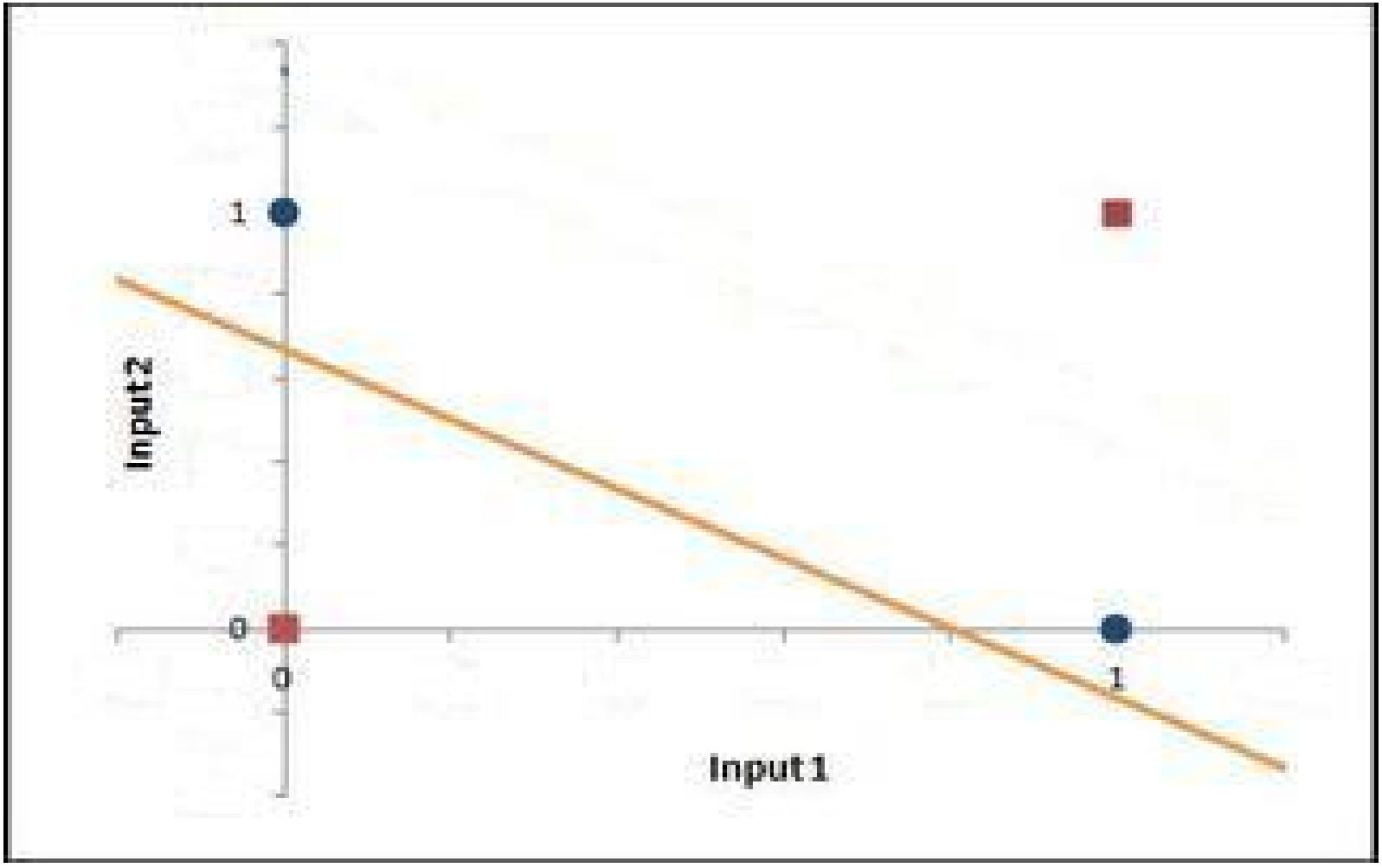
This would involve taking a lot of polar coordinate inner products, the product of the radii plus the product of the angles, a weird sort of operation in rectangular coordinate space.

Instead of choosing a promising looking space and mapping the original data there, the kernel trick stays in the original space and works with a promising looking substitute for the inner product, a *kernel*.

Kernels are “inner product-like” functions that are guaranteed to correspond to actual inner products in *some* space, although we often don’t know, or care, what that space is.

The XOR problem

No way to draw a line that puts both blue points on one side and both red points on the other side.



The XOR problem

We want to distinguish the case where the two inputs are different from the case where they are the same.

Subtracting one from the other, gives 0 if they are the same, and non-0 if they are different, which is a start. And furthermore subtraction can be performed by taking the inner product with $< 1, -1 >$.

The trouble is that the non-0 values are 1 and -1, which can't both be on the same side of a threshold that puts 0 on the other side.

Of course if we *squared* the inner product with $< 1, -1 >$, then the two cases where the inputs are different would both yield 1, and any threshold between 0 and 1 would distinguish the two cases.

The XOR problem

But then we don’t have a perceptron anymore.

Or do we?

The XOR problem

Consider the mapping from 2-D space to 3-D space that maps $< x, y > \mapsto < x^2, -\sqrt{2}xy, y^2 >$

So

$$< 0, 0 > \mapsto < 0, 0, 0 >$$
$$< 0, 1 > \mapsto < 0, 0, 1 >$$
$$< 1, 0 > \mapsto < 1, 0, 0 >$$
$$< 1, 1 > \mapsto < 1, -\sqrt{2}, 1 >$$

The XOR problem

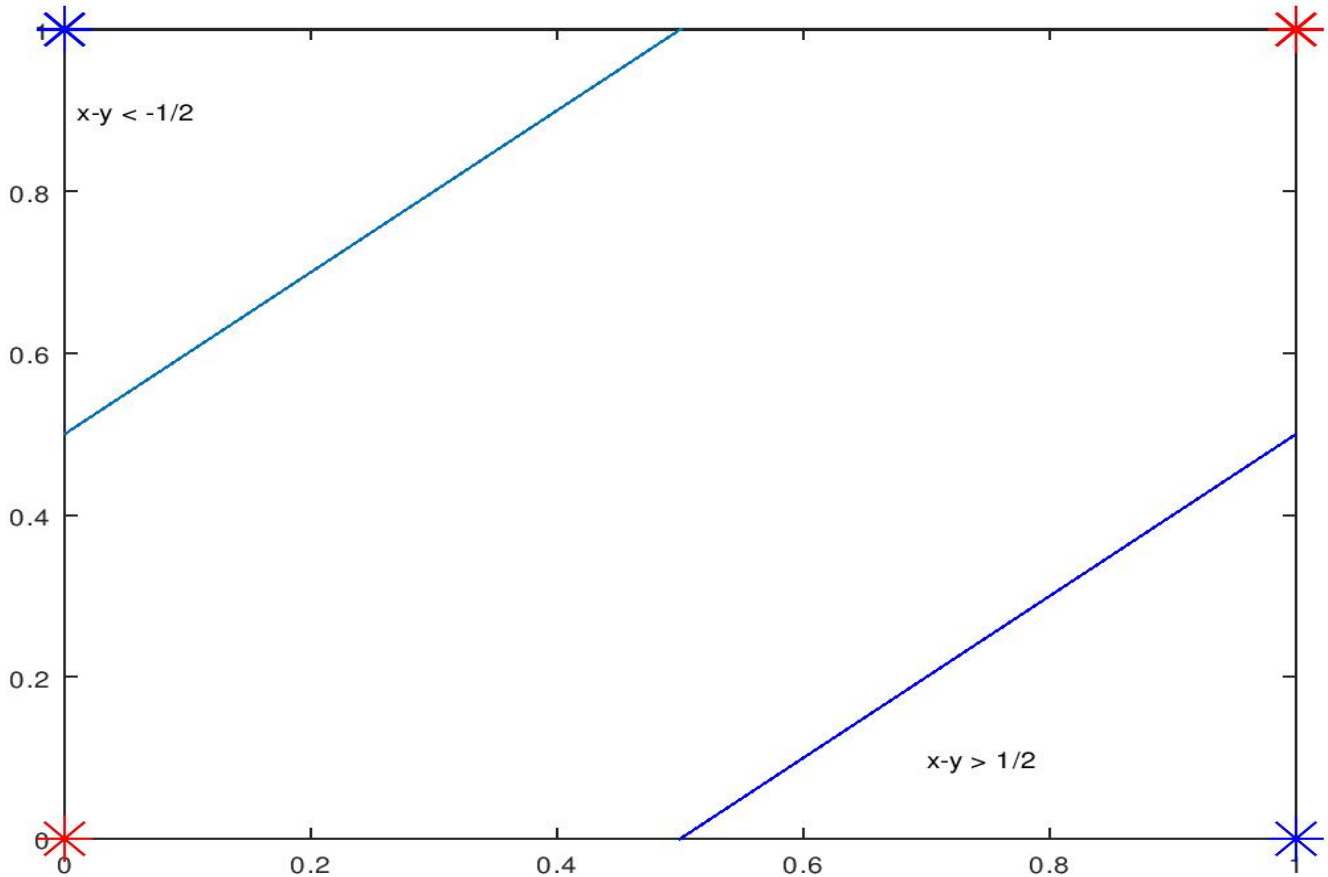
Taking inner products with the 3-D vector $w = < 1, \sqrt{2}, 1 >$, we get

$$w \cdot < 0, 0, 0 >= 0$$
$$w \cdot < 0, 0, 1 >= 1$$
$$w \cdot < 1, 0, 0 >= 1$$
$$w \cdot < 1, -\sqrt{2}, 1 >= 0$$

So w and any threshold between 0 and 1 will compute (the 3-D version of) XOR, and a 3-D perceptron could learn to do it.

The XOR Problem

Taking, say, $1/4$ as threshold, doing a little algebra, and mapping back to 2-D, we find, not surprisingly, that the 3-D computation amounts to $(x-y)^2 > 1/4$ or $|x-y| > 1/2$, which looks like:



The Trick - Staying in 2-D

But so far this isn't different from the polar coordinate case (except that polar coordinates don't help with XOR). We move to a new space and do perceptron learning there, taking inner products in the new space.

But in fact, we don't have to move to the 3-D space, we just have to know that it is there.

The 3-D inner product

$< x^2, -\sqrt{2}xy, y^2 > \cdot < z^2, -\sqrt{2}zw, w^2 >$
is

$x^2z^2 + 2xyzw + y^2w^2 = (xz + yw)^2 = (< x, y > \cdot < z, w >)^2$

The Trick - Staying in 2-D

That means that we can carry out all the perceptron operations in 2-D, but replacing inner product with its square, knowing that it will work because what we are doing is *equivalent* to working with actual inner products in the 3-D space.

So is there something special about squaring, or are there other inner product replacements that will work as well?

Kernels

There is class of functions, called (positive definite, or p.d.) kernels, any of which will work as replacements for the inner product in this way because they are inner products in some other space.

The definition, which is often hard to apply in practice, is:

A function $K(x, y)$ is a (p.d.) kernel iff

K is symmetric: $K(x, y) = K(y, x)$ (which is easy)

K is positive definite (which is not)

Positive Definiteness

A function K from pairs of vectors to real numbers is positive definite iff:

For any set of vectors v_i in the space, the *Gram matrix G*, where

$$G_{i,j} = K(v_i, v_j)$$

is positive definite.

Uh huh

Positive Definiteness

A symmetric nxn matrix M is positive definite iff:

For any non-zero length n vector u,

$$u^T M u > 0$$

We can see that $u^T M v$ is an "inner product-like" operation because, with symmetric M, it is symmetric, and it is linear in its arguments

$$(a_1 u_1 + a_2 u_2)^T M v = a_1 (u_1^T M v) + a_2 (u_2^T M v)$$

But, better still:

Positive Definiteness

Theorem: A positive definite matrix has all positive eigenvalues

Therefore M has a decomposition

$$M = USU^{-1} = (\sqrt{S}U^{-1})^T \sqrt{S}U^{-1}$$

where U is orthonormal and the diagonal matrix S has all positive entries on its diagonal.

Positive Definiteness

So:

Setting

$$V = \sqrt{S}U^{-1}$$

We have

$$M = V^T V$$
$$u^T M v = u^T V^T V v = (Vu)^T Vv$$

So $u^T Mv$ is actually an inner product in the space defined by the change of basis V.

Getting back to kernels, this gives (I hope) some insight into why they compute inner products.

Back to Kernels

Although the definition of a kernel is hard to use directly in deciding whether a given function is a kernel, or trying to construct a new kernel, there are some helpful theorems that let us generate new kernels from ones we already have:

(Thanks to Wikipedia for help in formatting the math)

New Kernels from Old

For p.d. kernels $(K_i)_{i\in\mathbb{N}}$,

The sum $\sum_{i=1}^{n} \lambda_i K_i$ is p.d., given $\lambda_1, \ldots, \lambda_n \geq 0$

The product $K_1^{a_1} \ldots K_n^{a_n}$ is p.d., given $a_1, \ldots, a_n \in \mathbb{N}$

The limit $K = \lim_{n\to\infty} K_n$ is p.d. if the limit exists.

If $(\mathcal{X}_i)_{i=1}^{n}$ is a sequence of sets, and $(K_i)_{i=1}^{n}$, a sequence of p.d. kernels, then both

$K((x_1, \ldots, x_n), (y_1, \ldots, y_n)) = \prod_{i=1}^{n} K_i(x_i, y_i)$
and
$K((x_1, \ldots, x_n), (y_1, \ldots, y_n)) = \sum_{i=1}^{n} K_i(x_i, y_i)$
are p.d. kernels on $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$.

Let $\mathcal{X}_0 \subset \mathcal{X}$. Then the restriction K_0 of K to $\mathcal{X}_0 \times \mathcal{X}_0$ is also a p.d. kernel.

Some Example Kernels

Linear kernel: $K(x, y) = x^T y, \quad x, y \in \mathbb{R}^d$.

Polynomial kernel: $K(x, y) = (x^T y + r)^n, \quad x, y \in \mathbb{R}^d, r >= 0$.

Gaussian kernel (Radial basis function kernel|RBF Kernel):

$$K(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}, \quad x, y \in \mathbb{R}^d, \sigma > 0.$$

Laplacian kernel: $K(x, y) = e^{-\alpha\|x-y\|}, \quad x, y \in \mathbb{R}^d, \alpha > 0$.

The spaces in which these kernels compute the inner product are not necessarily finite dimensional, so that computing the inner product in the space itself might not be feasible.

Nor is it necessarily simple to describe what the space corresponding to a given kernel is. See, e.g., Ingo Steinwart, Don Hush, and Clint Scovel "An explicit description of the reproducing kernel Hilbert spaces of Gaussian RBF kernels" (2005) http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.99.107&rep=rep1&type=pdf

The inner product can be thought of as a similarity measure. It is proportional to the angle between two vectors and is largest when the angle is 0. Rather than trying to understand the space that a kernel maps to, it is probably more enlightening to consider what points are made similar under a given kernel and what sort of contour $K(v, w) > \theta$ gives for fixed w and θ.

When K is the inner product, $K(v, w) > \theta$ gives points on one side of a hyperplane normal to w.

When K is the RBF kernel, $K(v, w) > \theta$ gives points outside of a hypersphere around w.

Historical Note

The basis of the kernel trick appeared, in English translation, in

Aizerman, M. A.; Braverman, Emmanuel M.; Rozoner, L. I. (1964). "Theoretical foundations of the potential function method in pattern recognition learning". Automation and Remote Control. 25: 821-837

five years before the publication of Minsky and Papert's *Perceptrons*.

Automation and Remote Control is a translation of the Russian journal Avtomatika i Telemekhanika, from the Russian Academy of Sciences.

So it was known in 1969, in Russia at least, that perceptrons are not restricted to performing linear separation *in the space in which the data points are presented.*