13 November, 2016

Neural Networks

Course 7: Other activation functions

Overview

- Tanh activation
- Relu activation
- Radial Basis Function Network
- Conclusions

Tanh activation

Tanh activation

- **Problems with the sigmoid function**

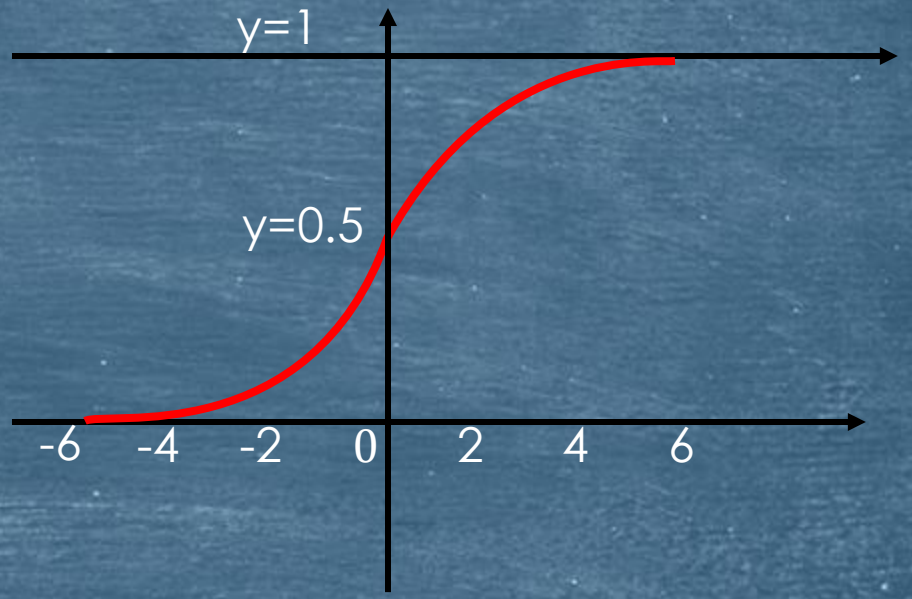
1. The output is always positive

The gradient w.r.t the weights is computed as:

$$\frac{\partial C}{\partial w_{ij}} = \delta_i^l y^{l-1}$$

If all the outputs are positive, that means that all the weights of a neuron will have the same sign $sgn(\delta_i^l)$

If a neuron wants to shift direction, it will take more steps. It is desirable that the average of activations on one layer is zero (including the input layer)



Tanh activation

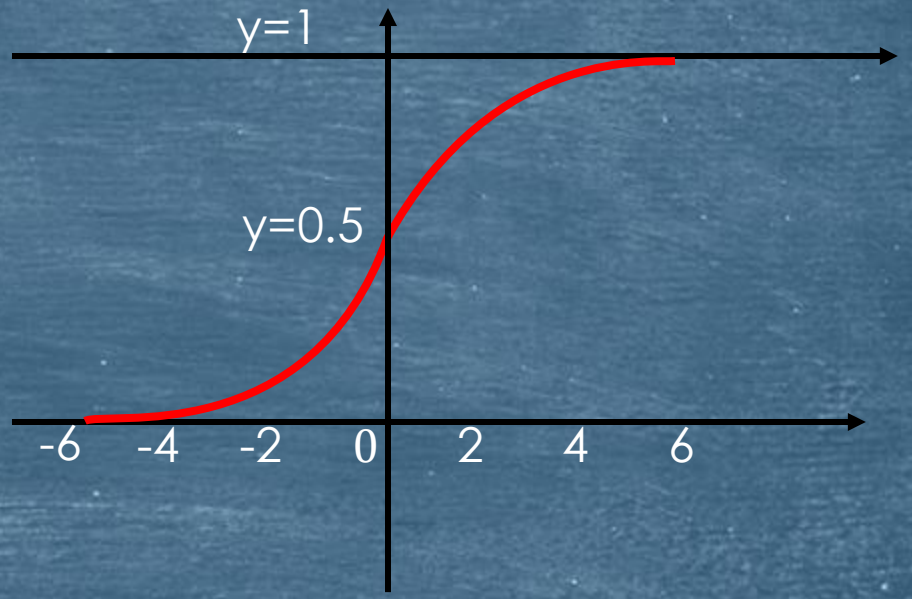
- **Problems with the sigmoid function**

2. They can output 0 when they saturate

For values less than -4, the sigmoid outputs values close to 0.

(gradients can become zero) :

$$\frac{\partial C}{\partial w_{ij}} = \delta_i^l y^{l-1}$$



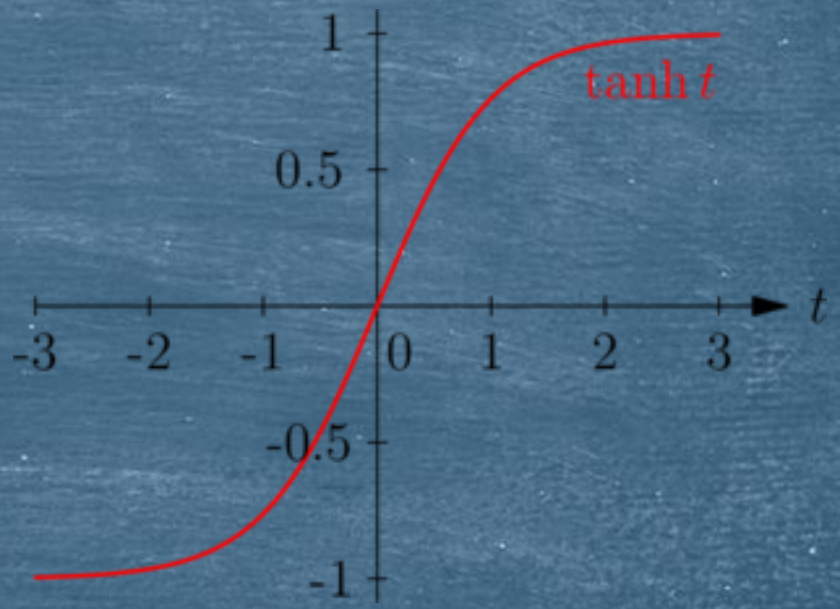
Tanh activation

- **Solution: hyperbolic tangent (tanh)**

$$\tanh(x) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = 2\sigma(2x) - 1$$

$$tanh'(x) = 1 - \tanh^2(x)$$

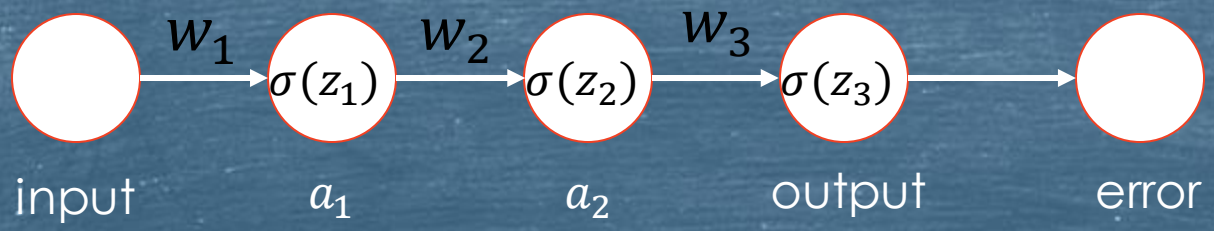
1. It has the mean at zero
2. It still saturates, but saturates on -1 or 1, which means that the weights will still be adjusted
3. The gradients of tanh are higher than for the sigmoid. The error is minimized faster



Tanh activation

- **Another problem of sigmoids: vanishing gradient**

Let's suppose we have the following network:



If we want to find out how to modify w_1 we have to compute the $\frac{\partial E}{\partial w_1}$

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial_{output}} \cdot \frac{\partial output}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} =$$

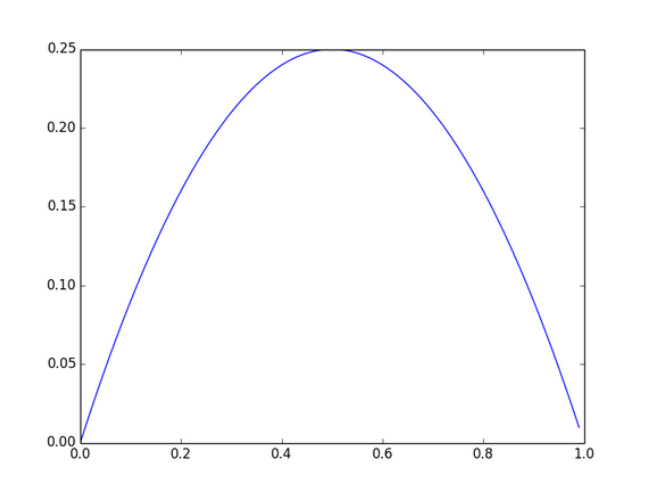
$$= \frac{\partial E}{\partial output} \cdot \sigma'(z_3) \cdot w_3 \cdot \sigma'(z_2) \cdot w_2 \cdot \sigma'(z_1) \cdot input$$

Tanh activation

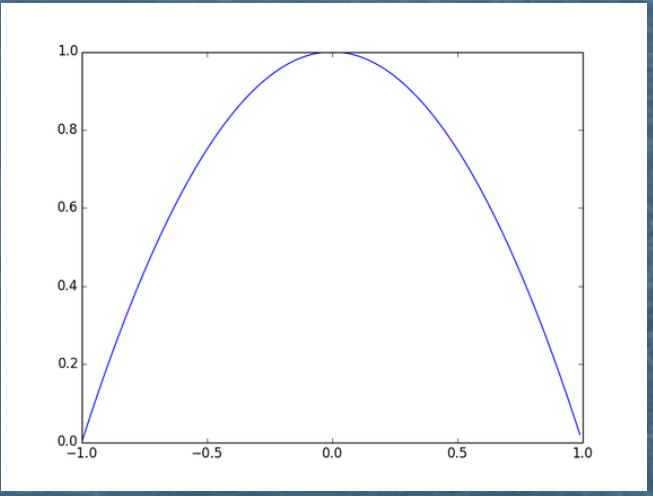
- **Another problem of sigmoids: vanishing gradient**

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial output} \cdot \underbrace{\sigma'(z_3)}_{<1} \cdot \underbrace{w_3}_{<1} \cdot \underbrace{\sigma'(z_2)}_{<1} \cdot \underbrace{w_2}_{<1} \cdot \underbrace{\sigma'(z_1)}_{<1} \cdot input$$

All the gradients are sub unitary. Many times, so are the weights. Think of how they are initialized or if L2 regularization is used



Graph of logistic σ'



Graph of tanh'

Tanh activation

- **Another problem of sigmoids: vanishing gradient**

For a more complex network, with many layers, the error is back propagated by the formula:

$$\delta_i^l = \sigma'(z_i^l) \sum_k \delta_i^{l+1} \cdot w_{ik}^{l+1}$$

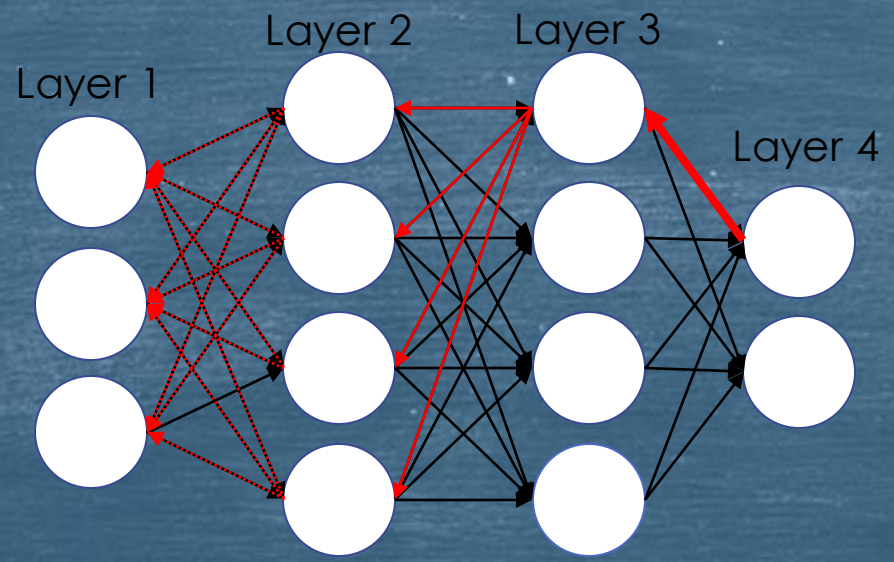
If all these terms are sub unitary (or most of them), then with each added layer, the error becomes smaller and smaller

Tanh activation

- **Another problem of sigmoids: vanishing gradient**

So the last layers will be the first to adapt, while the first layers will be the last.

However, the last layers output values based on the outputs of previous layers, so they adapt based on wrong outputs



Rectified Linear Unit (relu)

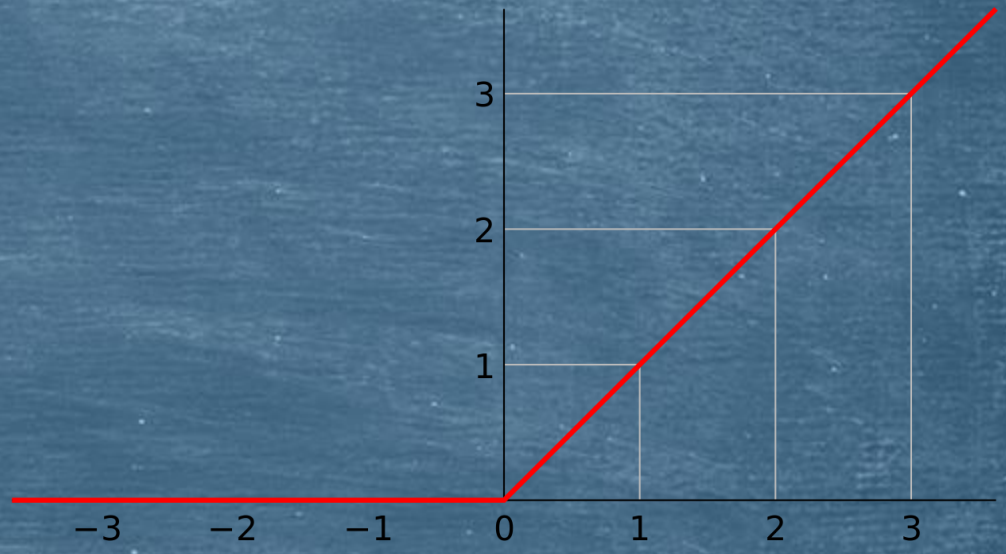
Relu activation

- **Rectified linear unit**

$$relu(x) = \begin{cases} 0, if\ x < 0 \\ x, if\ x \geq 0 \end{cases} = \max(0, x)$$

$$relu'(x) = \begin{cases} 0, if\ x < 0 \\ 1, if\ x \geq 0 \end{cases}$$

The function is not derivable at 0, but we'll just consider 0 for that case

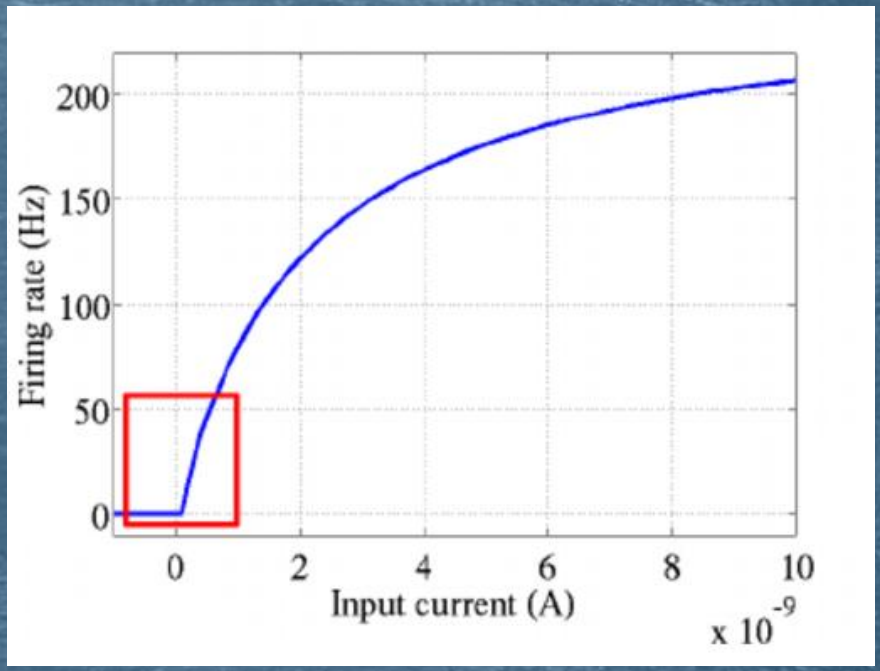


Relu activation

- **Inspired from observation of biologic neurons.**
  - Biological neurons are one-sided

    A biological neuron will fire if it has received enough input, or not fire at all

    It is not symmetric to zero (as tanh activation is) so it does not fire a negative signal



Relation between firing rate and input current in a e leaky integrate-and-fire neuron

Source: Deep Sparse Rectifier Neural Networks , Xavier Glorot , Antoine Bordes, Antoine Bordes

Relu activation

- **Inspired from observation of biologic neurons.**
  - Sparsity

    The model that is based on sigmoid functions ensures that every neuron participates in the output of the network.

    This is not biological possible, since that would mean a lot of energy loss. At any time, there are between 1% and 4% neurons active in the brain.

    Sparsity means to have the size of the feature vector larger than the size of input, but only a fraction of the feature vector is activated for an input

Relu activation

- **Advantages of sparsity**
  - A more simple interpretation of the data.

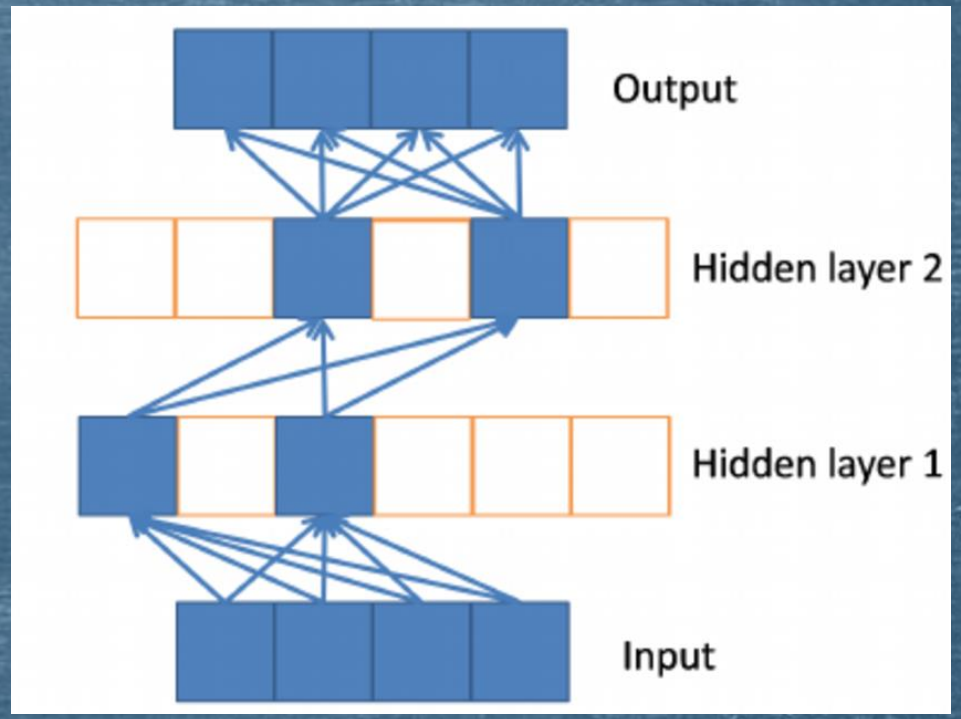
    Data is represented by a smaller number of parts, so it must extract the structure in the data

  - Efficient variable-size representation.

    Different inputs may contain different amounts of information and would be more conveniently represented using a variable size data-structure.

  - More likely to be linear separable,

    The next layer may contain more features than the previous, thus transforming the data into a multidimensional space where it might become separable



Source: Deep Sparse Rectifier Neural Networks , Xavier Glorot , Antoine Bordes, Antoine Bordes

Relu activation

- **Advantages of relu**
  - After the weight initialization, 50% of the neurons in the network have a value of 0. This sparsity can easily increase with sparsity –inducing regularization
  - Regarding the path of neurons that gets activated for a certain input, this is just a linear transformation. The function computed by each neuron or by the network output is linear by parts.
    - This means that the gradients flow well on the active paths of neurons
  - Computation is also cheaper. Just use multiplies and additions.

Radial Basis Function Network

Radial Basis Function Network

- **Structure:**

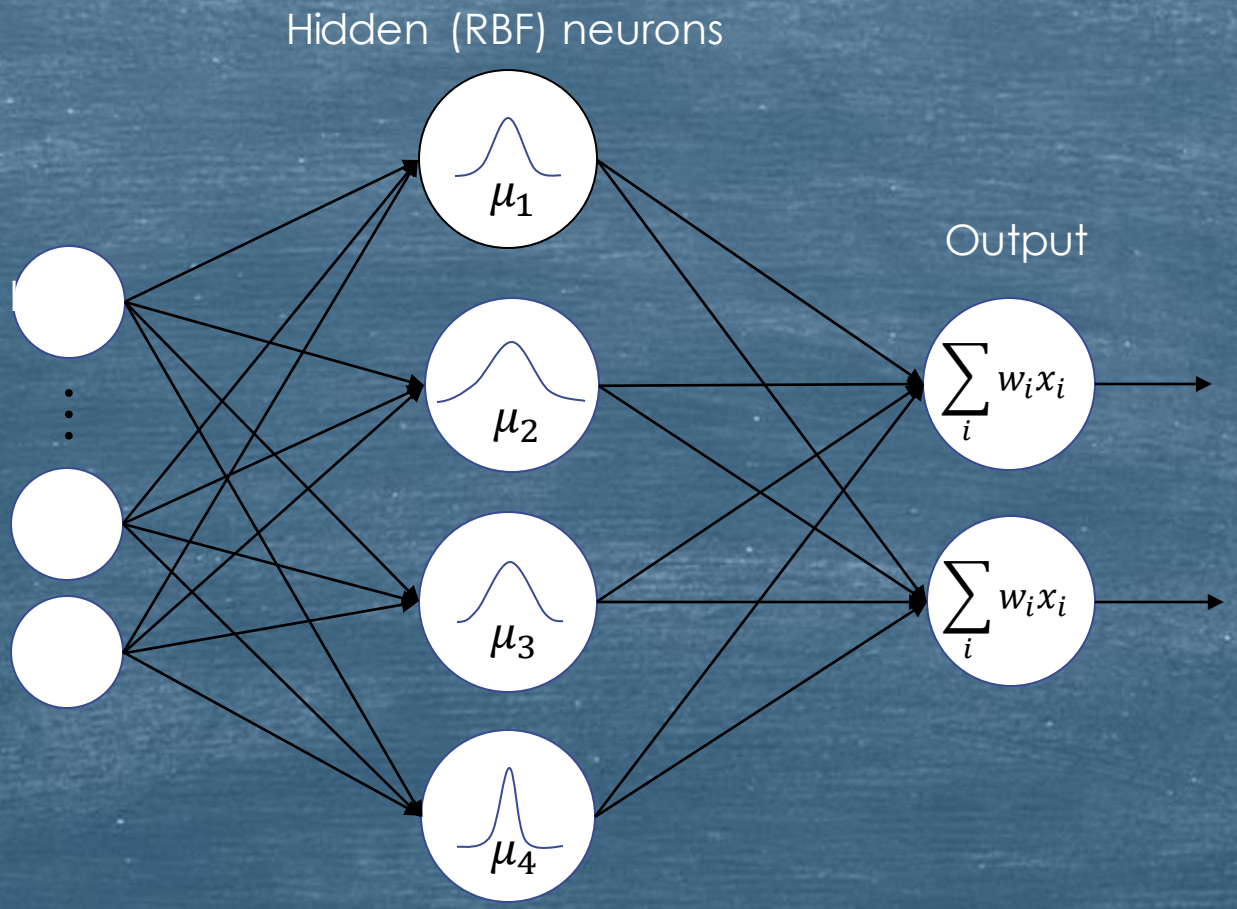
The structure of a RBF network is similar to the one of a MLP network. There are 3 layers:

- **Input**
- **Hidden**
- **Output**

The purpose of the second layer, is the same as for MLP, to find structure in the data while the third layer is used to output the result

Radial Basis Function Network

- **Structure:**



Radial Basis Function Network

- **Structure:**

  Each hidden neuron describes a set of elements from the input space. It is also called prototype. The hidden neuron can even be one of the input samples (the more representative, the better)

  Each input vector will be compared (based on a distance) with each of the (prototypes) hidden neurons.

  Each hidden neuron will output how similar is the training example to the hidden neuron.

  Based on the activation of each hidden neuron, the output neurons will compute a weighted sum and assign a class to the input

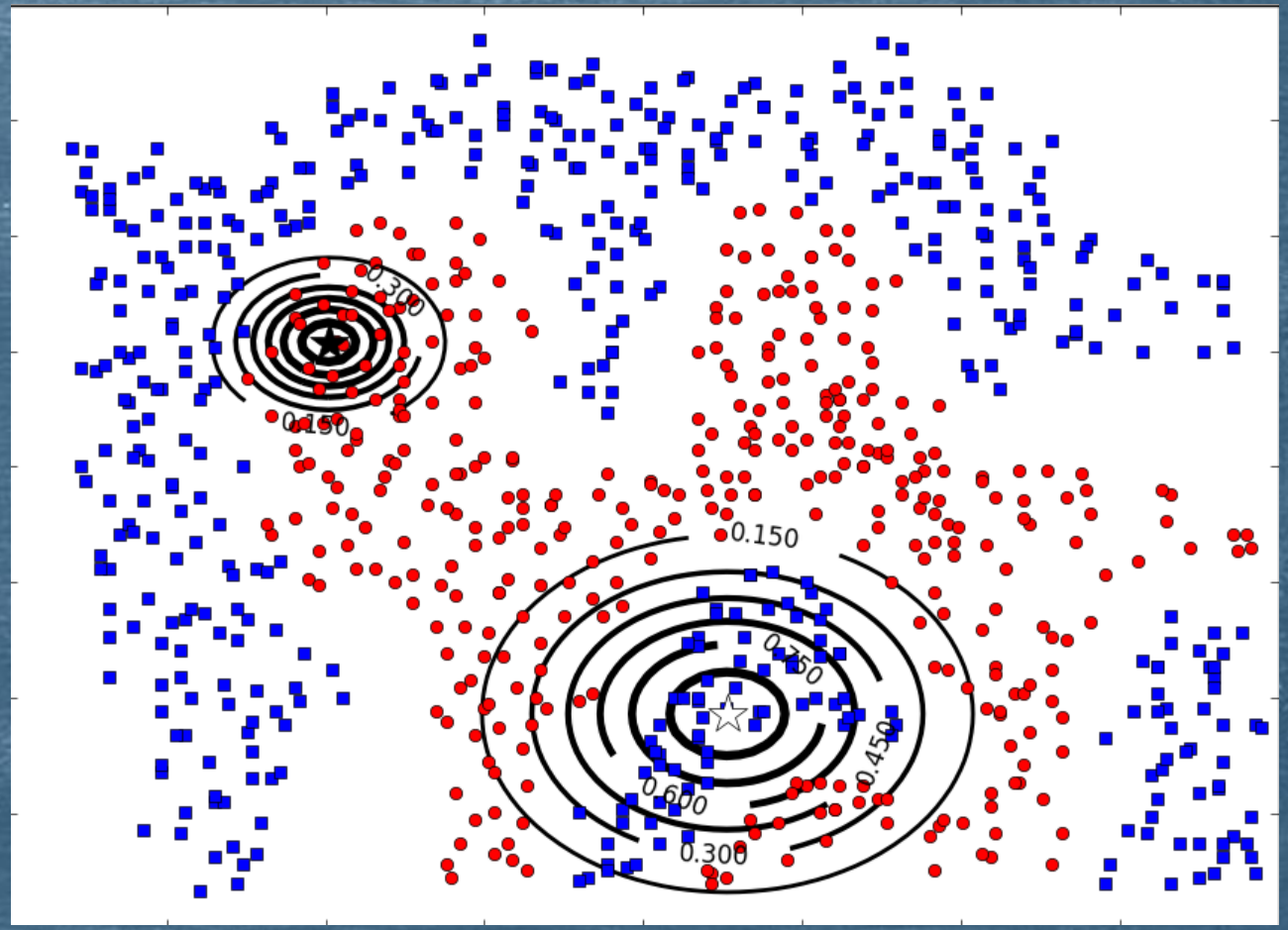
Radial Basis Function Network

- **Structure:**

In the image on the right there are two (prototypes) RBF neurons:

- Black star (represents a section of red circles)
- White star (represents a section of blue squares)

A RBFN will need to weight the scores given by each of these prototypes and take a decision (classify an input)



Radial Basis Function Network

- **Training:**

So, the training is made in two steps:

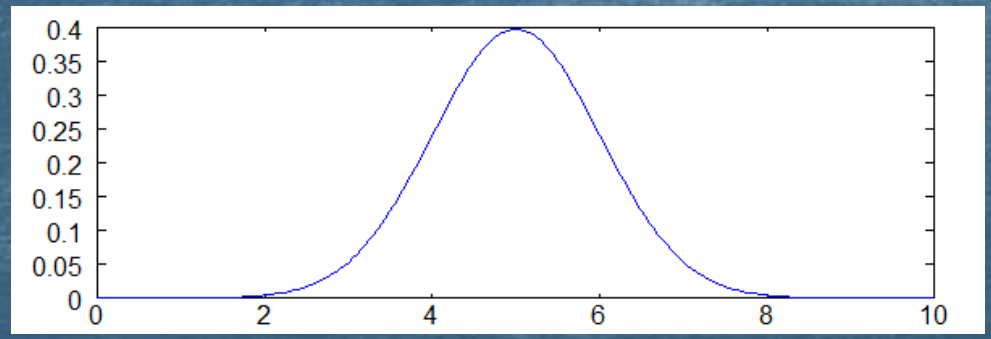
1. Find the right prototypes (unsupervised)
2. Learn how to weight the output given by each prototype (supervised)

It is better if the prototypes are chosen so they are specific to a certain class. In this way, the classifier will learn to assign it a positive weight when training to classify an element from the same class. (or a negative weight if the element is of opposite class)

Radial Basis Function Network

- **Measuring the distance:**
  - For each item, a prototype will output the similarity with it. This is its activation function.
  - We want the activation to be maximum when the element is similar to the prototype and to decrease exponential if its different
  - The most popular is based (It doesn't use it) on the Gaussian:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



Radial Basis Function Network

- **Understanding the activation function:**

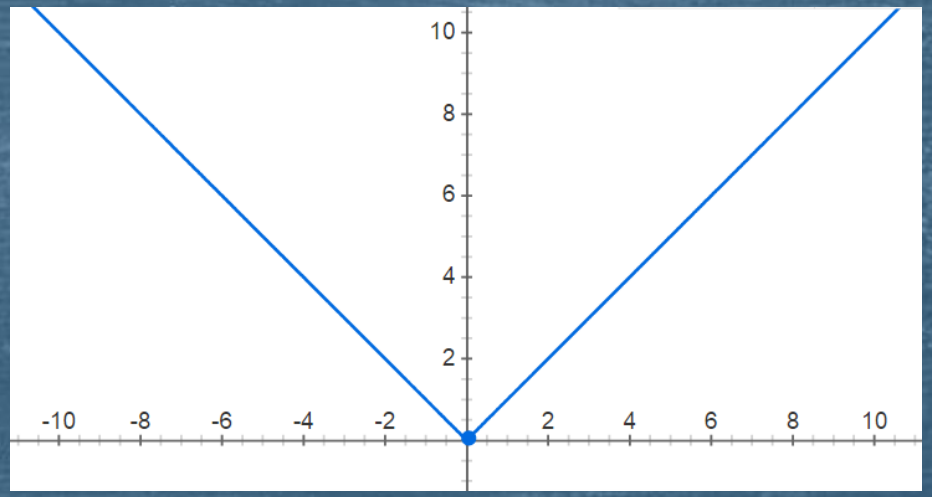
The similarity can be viewed as computing a distance between the prototype (defined by μ) and the item (defined by x)

$$distance(x, \mu) = \left||x - u|\right| = \sqrt{\sum_i^n (x_i - \mu_i)^2}$$

Euclidian Distance between

x and $\mu = 0$

It gives the lowest activation when Items are equal (opposite of what we want. Also, very steep)



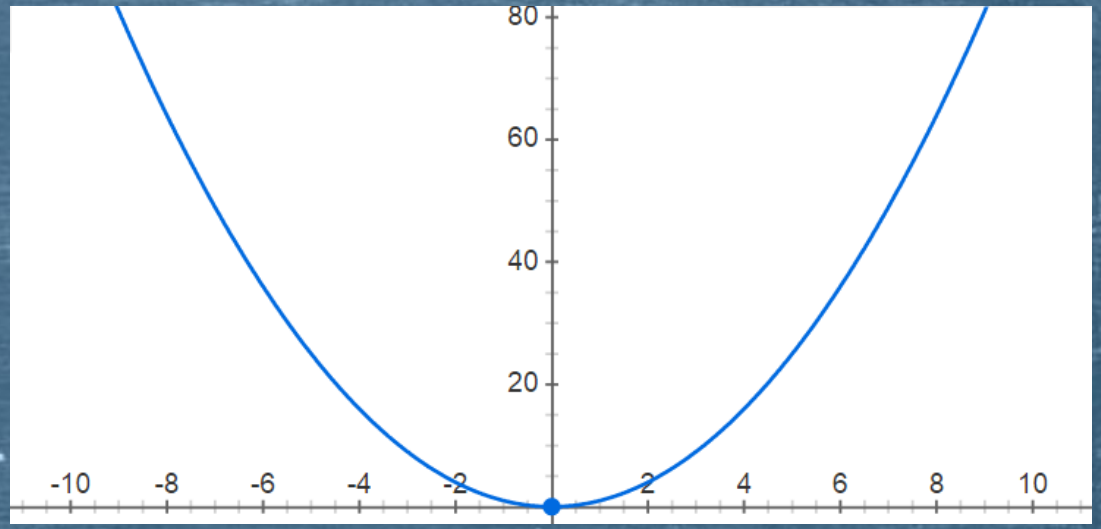
Radial Basis Function Network

The Gaussian function doesn't include the $\sqrt{\ }$ so we'll remove it.

$$distance(x, \mu) = \sum_{i}^{n} (x_i - \mu_i)^2$$

But the function still outputs the lowest value when the value is equal to the target (μ) and largest when it's far away

Also, the function is not bounded. Values close to infinite can be obtained.



$$distance(x, \mu) = (x - \mu)^2$$

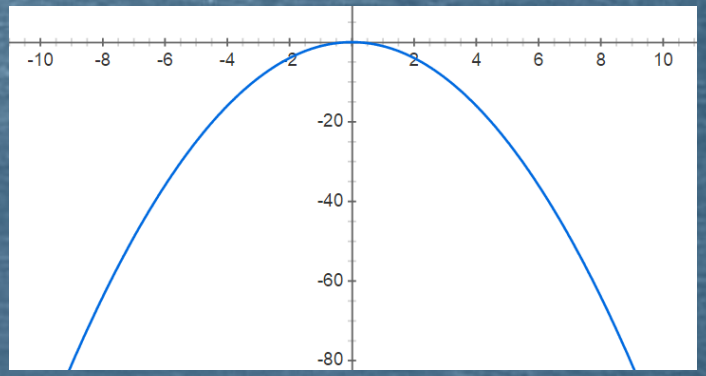
Radial Basis Function Network

So, we'll inverse the sign

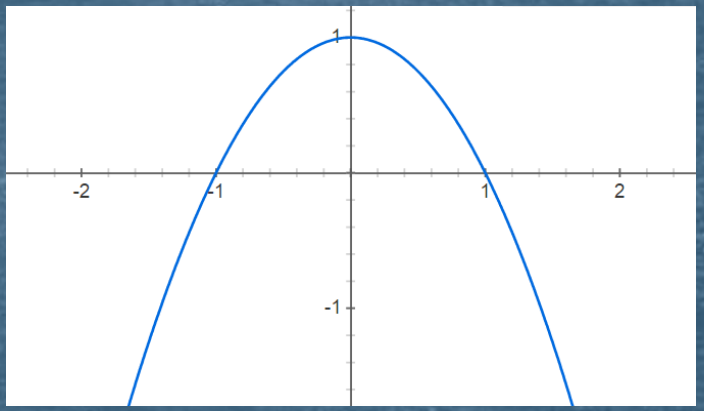
$$distance(x, \mu) = -\sum_{i}^{n} (x_i - \mu_i)^2$$

We've got our function working properly (biggest value when the values are equal), but all the values are negative

A solution would be to add a bias (i.e. for a bias of 1, the graph on the right is obtained), but it doesn't solve the fact that our function is still unbounded and can output negative values



$$distance(x, \mu) = -(x - \mu)^2$$



$$distance(x, \mu) = -(x - \mu)^2 + 1$$

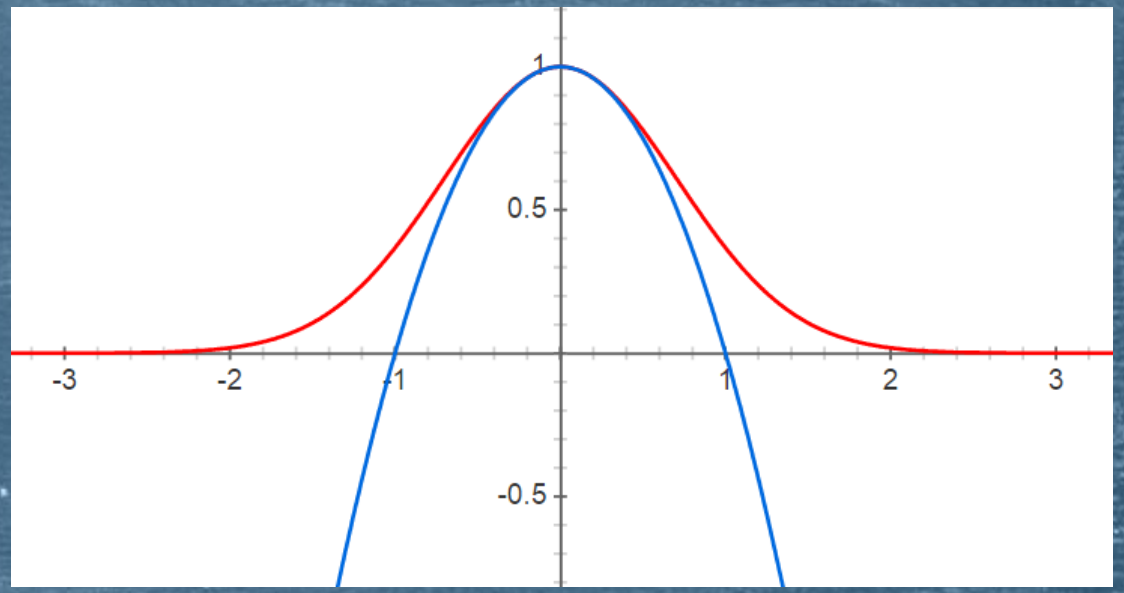
Radial Basis Function Network

A better solution is to use the exponential.

$$distance(x, \mu) = e^{-\sum_i^n (x_i - \mu_i)^2}$$

This way, the function:

- is always positive (so, we can consider it a distance)
- it is bounded
- Another advantage is that the function will output a value very close to zero for all the elements it does not recognize



$$distance(x, \mu) = e^{-(x-\mu)^2}$$

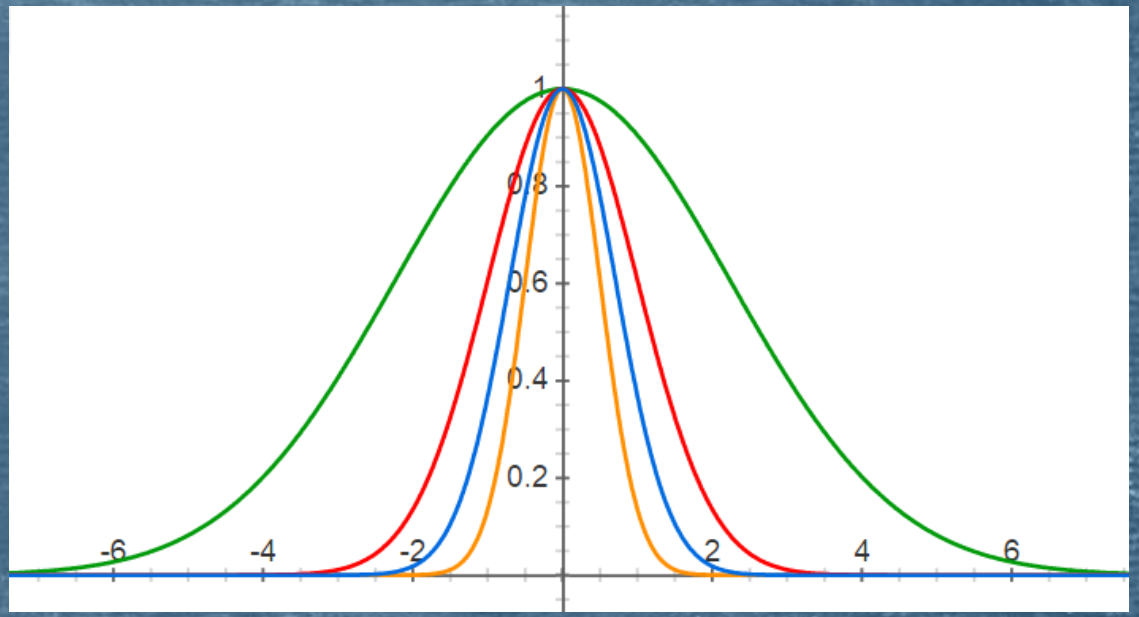
Radial Basis Function Network

The Gaussian equation has another two coefficients.

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

This red one (we'll use β) controls the width of the function graph

We need this coefficient since this dictates the size of the area that the prototype considers to be similar



$$distance(x, \mu) = e^{-\beta(x-\mu)^2}$$

$\beta = 0.1$
$\beta = 0.5$
$\beta = 1$
$\beta = 2$

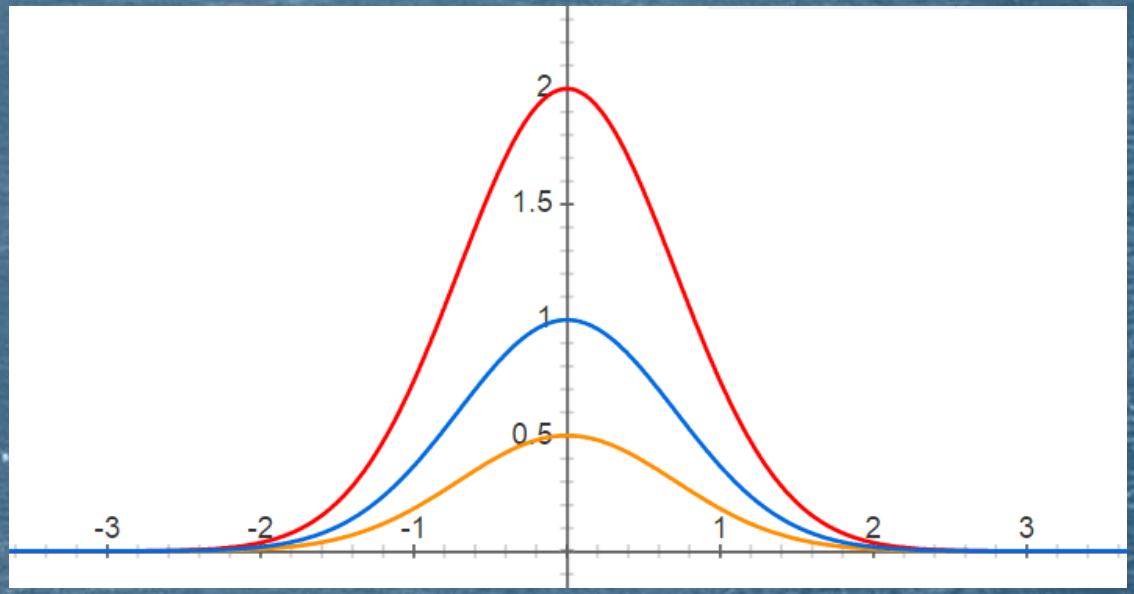
Radial Basis Function Network

The Gaussian equation has another two coefficients.

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

This black one (we'll use w) controls the height of the graph

This will not be outputted by the prototype. The neural network will assign a weight to each prototype.



$$we^{-\beta(x-\mu)^2}$$

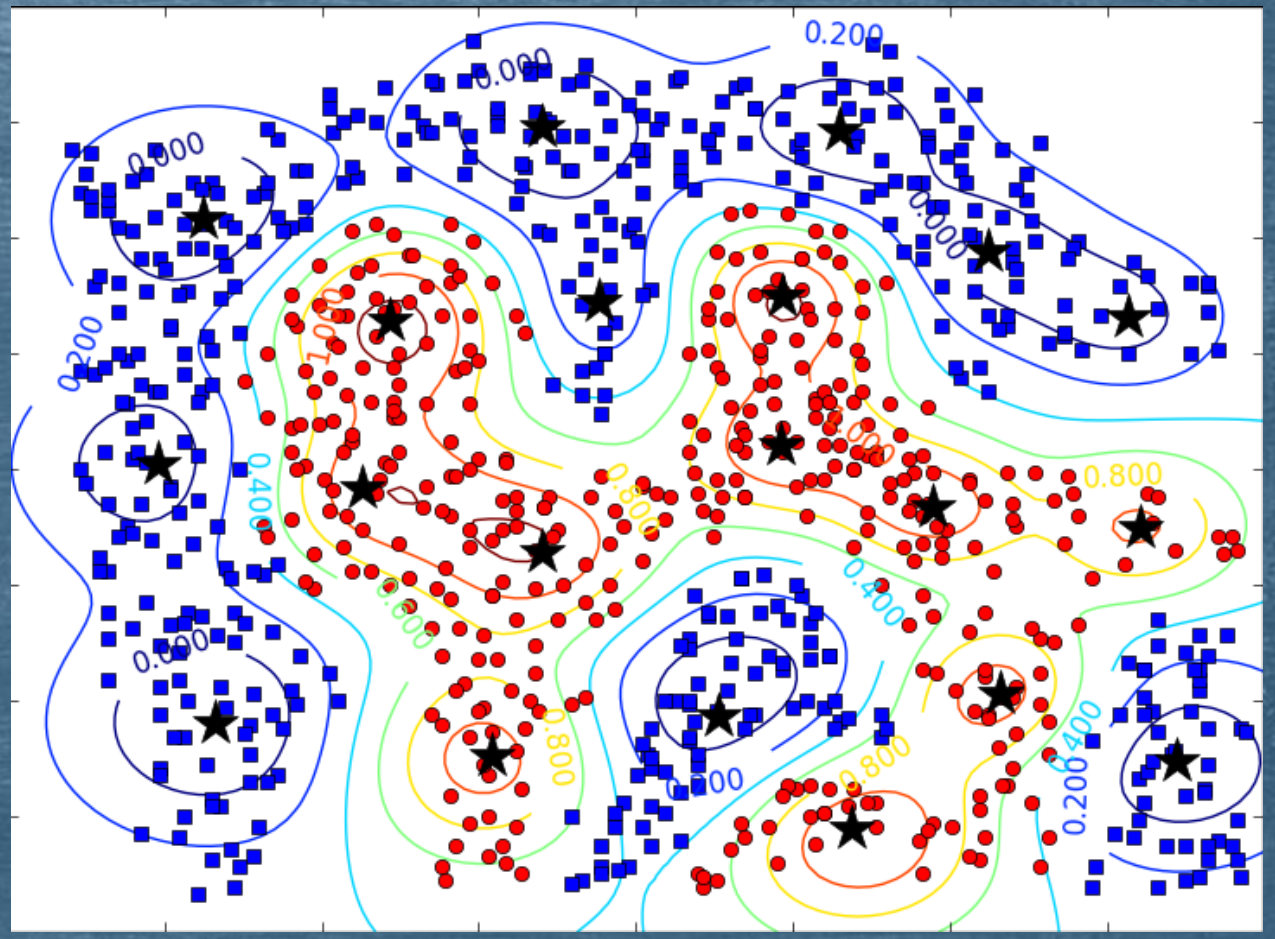
$w = 2$

$w = 1$

$w = 0.5$

Radial Basis Function Network

The neural network will assign big positive weights to the prototypes that are of the same kind with the input and small (even negative weights) to others



Radial Basis Function Network

- **Training:**

The activation function each RBF neuron will use is

$$e^{-\beta(x-\mu)^2}$$

So, during the training of prototypes, two components must be considered:

- The location of each prototype (μ)
- The the area that is represented (that gets affected) by each prototype (β)

Radial Basis Function Network

- **Training: Find location of each prototype**

  Any clustering algorithm can be used here, but we'll use k-means (because it is simple)

- When performing the clustering, it is important to perform a clustering operation on items of each class.

  So, in our case, clustering must be performed once for the red circles and once for the blue squares

- This is because the neural network will weight the space according to the data provided by the clusters (prototypes). This way we help it by showing it how each class is distributed.

Radial Basis Function Network

- **K-means algorithm**

The algorithm starts with some clusters and on each iteration it improves them

So, randomly choose k random cluster centers (in our case, two coordinates for each cluster center)

On each iteration, or until there is no improvement perform two steps:

1. Assignment step
2. Update step

Radial Basis Function Network

- **K-means algorithm**

1. Assignment step:

   For each item in the dataset, compute its distance (square of Euclidian distance) to every cluster center. Assign the item to the nearest cluster

$$Cluster_i = \{x_p : \left|\left|x_p - m_i\right|\right|^2 \leq \left|\left|x_p - m_j\right|\right|^2, \forall j,\ 1 \leq j \leq k\}$$

   where

   m_i is the cluster center of $Cluster_i$

   x_p is an arbitrary item

   $||\cdot||$ is the Euclidian distance

Radial Basis Function Network

- **K-means algorithm**

2. Update step:

Since new elements might have been added to the cluster, its center could have modified. Thus recalculate the cluster center

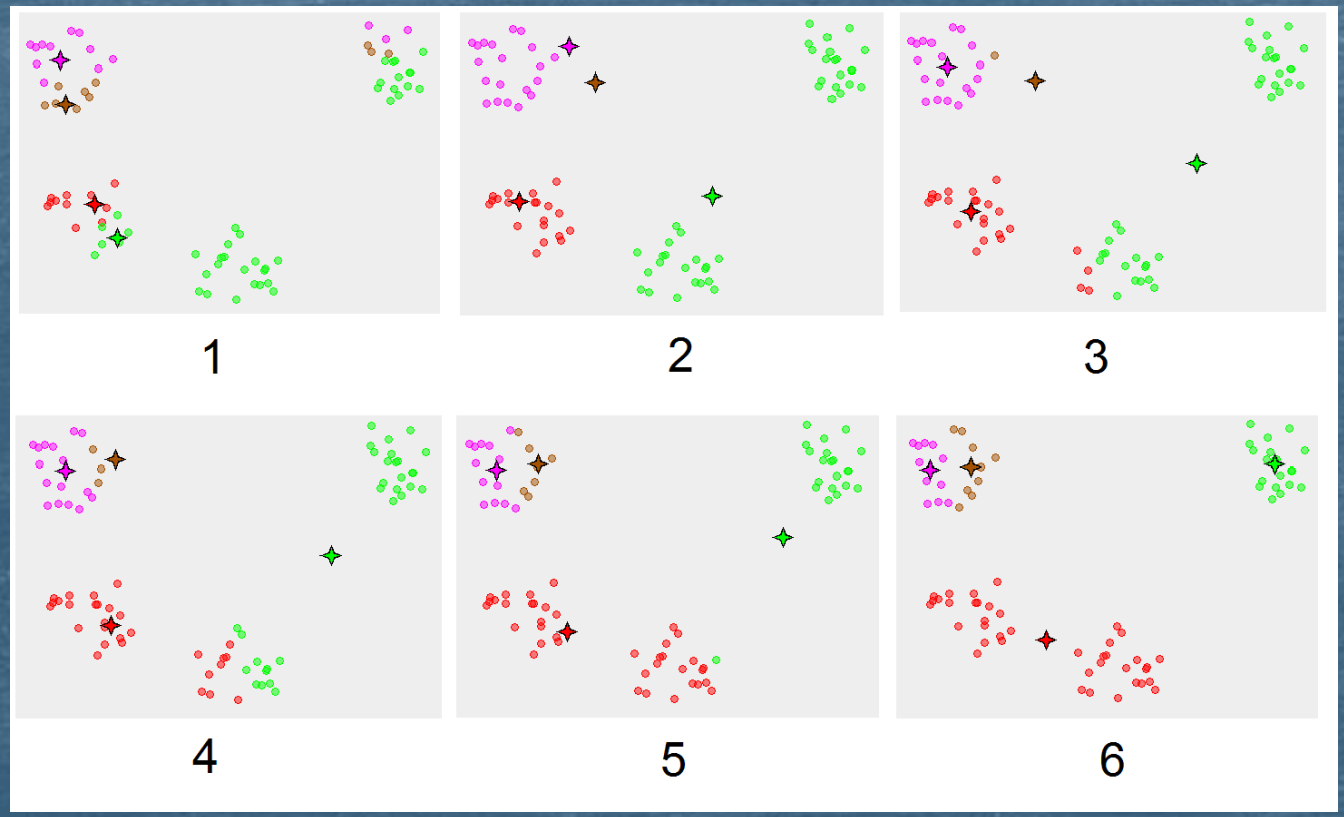
$$m_j = \frac{\sum_{x_i \in Cluster_j} x_i}{|Cluster_j|}$$

where $|Cluster_j|$ represents the number of items in $Cluster_j$

Repeat steps 1 and 2 until there are no modified clusters or until a number of iterations have passed

Radial Basis Function Network

Initialization of cluster centers is very important and can affect the output of the algorithm



Source:wikipedia

Radial Basis Function Network

- **K-means++ algorithm**

A better version of the algorithm modifies the initialization step such that the cluster centers are scattered.

1. Choose an element from the dataset as the first cluster

2. For each of the elements in the dataset, compute the minimal distance $D(x)$ to all the previous centers

3. Choose the next cluster center to be the item x with a probability $D(x)^2$

Radial Basis Function Network

- **Training: choose β values**

β dictates the width of graph.

A simple way (not the best) to set β is to just take its value from the Gaussian equation such that the exponent of e to be equal

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$\beta = \frac{1}{2\sigma^2}$$

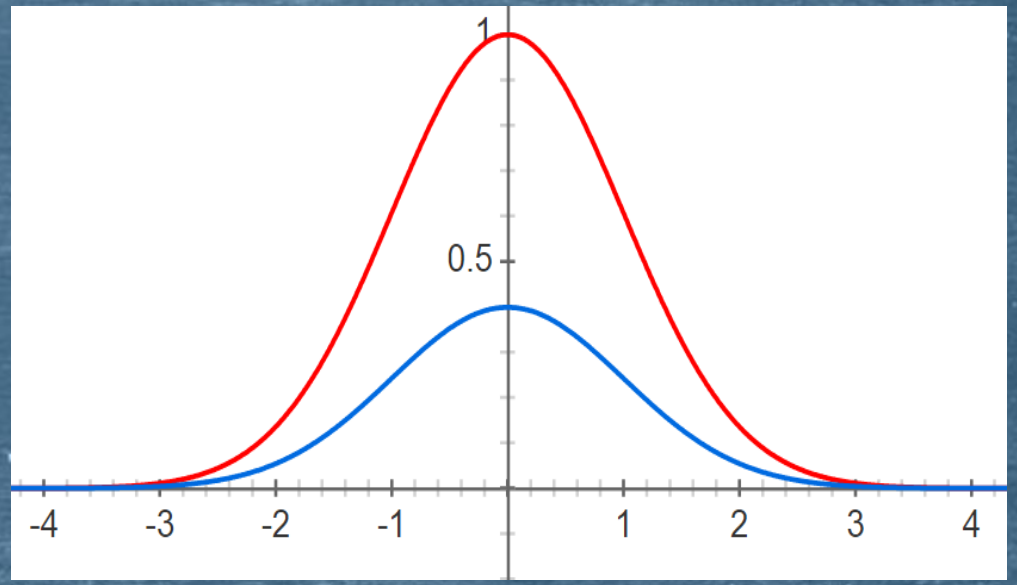
Radial Basis Function Network

- **Training: choose β values**

If we consider the data in a cluster to be normally distributed, then:

- 95% of items will be within 2 standard deviations

(for these items, the function will output a significant value)



Radial Basis Function Network

- **Training: choose β values**

So, to compute β we need the standard deviation within the cluster. These can be computed (as normal):

$$\sigma = \sqrt{\frac{1}{N}\sum_i^N |x_i - \mu||^2}$$

Where

μ is the cluster mean (center)

x_i is an element from the cluster

N is the total number of elements in the cluster

Radial Basis Function Network

- **Training: the output neurons**

The neurons from the output layer need to lean how to adjust the scores provided by every prototype

This can be seen as the if the neuron will curve the space by lifting some of the prototypes (assigning a positive weight) and pressing on others (by assigning a negative weight)

So the neuron will output a weighted sum over the clusters

Radial Basis Function Network

- **Training: the output neurons**

Every output neuron is actually just a linear classifier and trained separately

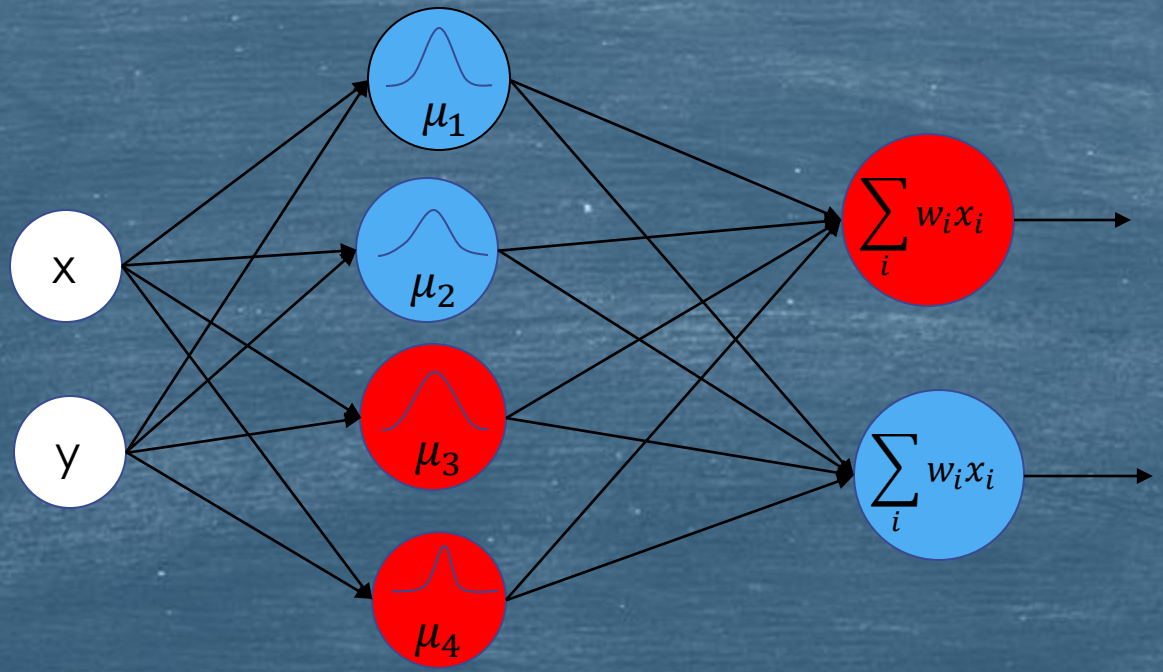
So, in order to train it :

- An item (or a batch of items) will be set as input to the network
- The output of each cluster will be considered part of the input for the perceptron
- The weights are adjusted as described in the class presented on second week

Radial Basis Function Network

- **Training: the output neurons**

  In most of the cases, the neuron specific to a class will learn to give positive weights to all prototypes of the same class and negative to the others



Questions & Discussion

Bibliograpy

- http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf
- http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS2010_GlorotB10.pdf
- http://mccormickml.com/2013/08/15/radial-basis-function-network-rbfn-tutorial/
- http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf
- http://neuralnetworksanddeeplearning.com/
- https://ayearofai.com/rohan-4-the-vanishing-gradient-problem-ec68f76ffb9b#.njhkqb9ed