Intro to Neural Networks

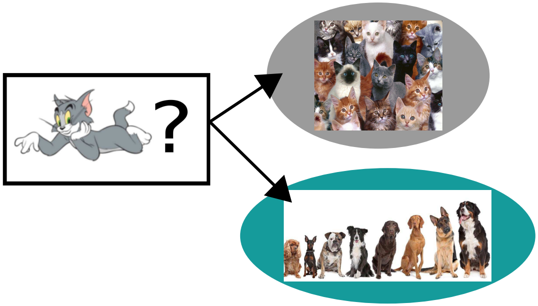
Luis G Sanchez Giraldo
Odelia Schwartz

University of Miami

May 26th, 2017

Object Features and the Classification Problem

Classification Problem:
Given a set of objects, where each object belongs to one of two classes, we want to find a rule or function that can predict to which class each object from the set belongs to.



Object Features and the Classification Problem (cont...)

Features for Object Representation:

Each object is represented by a set of numbers called features. Mathematically, an object can be represented as a *d*-dimensional vector $\mathbf{x} = \{x_1, x_2, \ldots, x_d\}$, where *d* is the number of features.

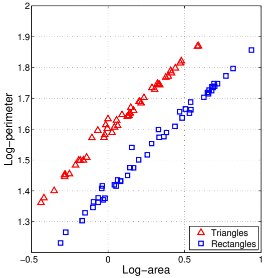
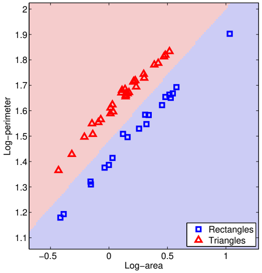


Figure: Representation of rectangles and triangles by their area and perimeter. The *log*-scale is used for mathematical convenience.

Object Features and the Classification Problem (cont...)

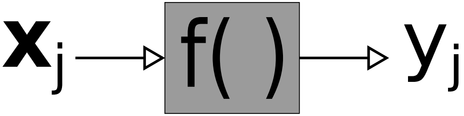
The assumption is that objects that belong together in the same class will have similar values for their features. Therefore, if we want to predict the class of an object, we can hope to do so by looking at the values of its features. However, it is important to notice that features do interact and that these feature interactions (structure) are important in identifying the object's class. For example, a simple rule to separate triangles from rectangles would be to divide the (log-perimeter, log-volume) plane into two regions.



Rosenblatt's Perceptron

Rosenblatt's Perceptron is one of the earliest models for learning with supervision, or what is commonly known as supervised learning.

In supervised learning, we are given a set $\{(\mathbf{x}_j, y_j)\}_{j=1}^{N}$ of feature-class pairs. The goal is to express the relation between $\mathbf{x}_j$ and y_j, for all $j = 1, \ldots, N$, as a function f that receives $\mathbf{x}_i$ as input, and returns y_i as output. Learning consist of finding such an f within a given class of functions.



Rosenblatt's Perceptron (cont...)

In particular, the perceptron implements the following function to solve the classification problem.

$$f(\mathbf{x}_j) = \mathrm{sign}\left(\sum_{i=1}^{d} w_i x_{ij} + b\right). \tag{1}$$

As we can see, changing the values of w_i and b gives us different functions and thus, it defines a class of functions. Learning amounts to finding $\mathbf{w}$ and b that "best capture" the input-output relation.

Perceptron Algorithm for Learning f

Consist of the following steps:

1. Let $j = 1$, and initialize $\{w_i\}_{i=1}^{d}$ and b ($\mathbf{w}$ and b can be initialized with zeros).
2. Compute
$$f(\mathbf{x}_j) = \mathrm{sign}\left(\sum_{i=1}^{d} w_i x_{ij} + b\right). \quad (2)$$
3. If $f(\mathbf{x}_j)$ is NOT equal to y_j, update $\mathbf{w}$ and b as follows:
$$w_i \leftarrow w_i + y_j x_{ij} \quad (3)$$
$$b \leftarrow b + y_j \quad (4)$$
4. Repeat 2 for next value of j, that is, $j = j + 1$. When $j = N$, restart the counter to $j = 1$. If for all $j = 1, \ldots, N$, $f(\mathbf{x}_j)$ is equal to y_j, stop iterating.

Perceptron Algorithm for Learning f (cont...)
Why is this supposed to work? (Basic Idea)

Suppose we have updated $\mathbf{w}$ and b, based on the rule given above, using the jth input-output pair $(\mathbf{x}_j,\ y_j)$. The linear part of the updated function evaluated at $\mathbf{x}_j$ is given by:

$$\sum_{i=1}^{d} w_i x_{ij} + b = \sum_{i=1}^{d} \hat{w}_i x_{ij} + \hat{b} + y_j \left(\sum_{i=1}^{d} x_{ij} x_{ij} + 1 \right), \qquad (5)$$

where $\hat{\mathbf{w}}$ and $\hat{b}$ are the parameters before the update.
As we can see, the update rule applies a positive or negative correction to the current function to enforce agreement with the actual class label y_j.

Going Beyond the Perceptron

The percpetron works at separating objects given a set of features (numerical descriptors). However, in this framework a lot of human expertise is required for crafting such features.

- What if features can be learned from exemplars?
- How can features be learned from exemplars?

Going Beyond the Perceptron (cont...)

Consider the following generalizations of the function in the perceptron

1. $$f(\mathbf{x}_j) = h\left(\sum_{i=1}^{d} w_i x_{ij} + b\right), \tag{6}$$

   where h is a nonlinear function. In this case the values of the outputs of f do not need to be restricted to -1 and $+1$ as with sign function.

2. To compare the output of $f(\mathbf{x}_i)$ with the actual label values y_j, we define a loss function:

   $$\mathcal{L}(f(\mathbf{x}_j), y_j) \tag{7}$$

The loss function can be used to guide the learning since it penalizes errors made by f.

Going Beyond the Perceptron (cont...)

Common examples of loss functions are:

- Zero-One loss

$$\begin{cases} 1 & \text{if}\, \text{sign}[f(\mathbf{x}_j)] \neq y_j, \\ 0 & \text{if}\, \text{sign}[f(\mathbf{x}_j)] = y_j. \end{cases} \tag{8}$$

- Squared loss

$$(y_j - f(\mathbf{x}_j))^2 \,. \tag{9}$$

- Hinge loss

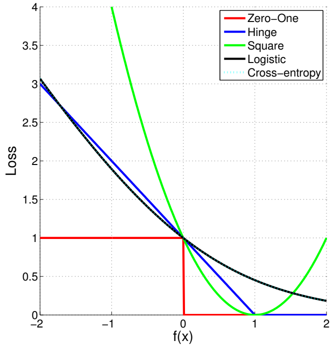
$$\max\{0,\ 1 - y_j f(\mathbf{x}_j)\}\,. \tag{10}$$

- Cross-entropy loss. In this case $f(\mathbf{x}_j)$ is transformed to a function $h(f(\mathbf{x}_j))$ with outputs in the range $[0, 1]$

$$-\frac{y_j + 1}{2}\log\left(f(\mathbf{x}_j)\right) - \frac{1 - y_j}{2}\log\left(1 - f(\mathbf{x}_j)\right). \tag{11}$$

Going Beyond the Perceptron (cont...)

Values of the loss functions for $y_j = 1$ vs $f(\mathbf{x}_j)$



Going Beyond the Perceptron (cont...)

Given a set $\{(\mathbf{x}_j, y_j)\}_{j=1}^{N}$ of feature-class pairs, learning can be accomplished by minimizing the average loss on this set. Having differentiable loss and activation functions provide a mathematically simple framework that allows the use gradient-based minimization techniques for learning. For instance, using the logistic function for h in (6) and the cross-entropy loss (11) yields an iterative procedure.

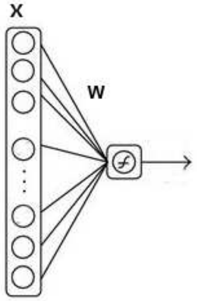
Going Beyond the Perceptron (cont...)

Gradient descent for cross-entropy loss with logistic sigmoid activation.

1. Initialize $\{w_i\}_{i=1}^{d}$ and b ($\mathbf{w}$ and b can be initialized with zeros).
2. Let
$$f(\mathbf{x}_j) = \frac{1}{1+\exp(-z_i)}, \text{ where } z_j = \left(\sum_{i=1}^{d} w_i x_{ij} + b\right), \tag{12}$$
3. and
$$\Delta w_i = \frac{\partial}{\partial w_i}\mathcal{L}(f(\mathbf{x}_j), y_j) = \left(f(\mathbf{x}_j) - \frac{y_j+1}{2}\right) x_{ij}, \tag{13}$$
$$\Delta b = \frac{\partial}{\partial b}\mathcal{L}(f(\mathbf{x}_i), y_i) = \left(f(\mathbf{x}_j) - \frac{y_j+1}{2}\right). \tag{14}$$
4. Update parameters. Similarly to the perceptron
$$w_i \leftarrow w_i - \mu\Delta w_i \text{ and } b \leftarrow b - \mu\Delta b \tag{15}$$

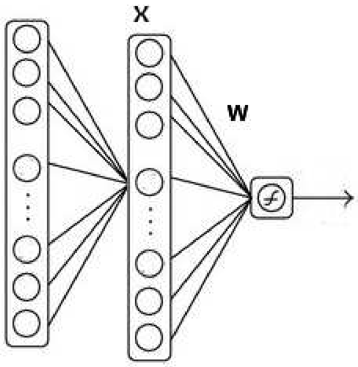
Multilayer Perceptron

The idea of finding a map between features and classes can be extended. Instead of having to predefine the features to be used to classify objects, we can construct them by finding a transformation of raw measurements from our objects.



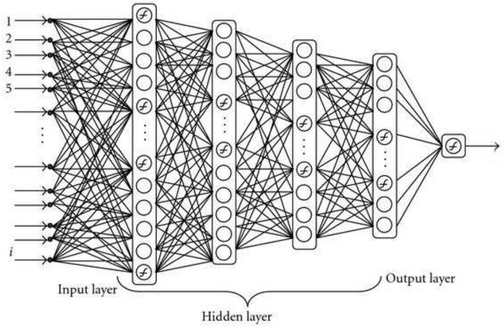
Multilayer Perceptron

The idea of finding a map between features and classes can be extended. Instead of having to predefine the features to be used to classify objects, we can construct them by finding a transformation of raw measurements from our objects.



Multilayer Perceptron

The idea of finding a map between features and classes can be extended. Instead of having to predefine the features to be used to classify objects, we can construct them by finding a transformation of raw measurements from our objects.



Multilayer Perceptron

Let us write these ideas down:

- L layers with ℓ as a our layer index.
- The ℓth layer has n_ℓ units.
- Denote by $x_i^{(\ell)}$ the output of the ith unit in the ℓth layer, given by:

$$\begin{aligned} x_i^{(\ell)} &= f_i^{(\ell)}(\mathbf{x}^{(\ell-1)}) && (16) \\ &= \sigma^{(\ell)}\left(\sum_{j=1}^{n_{\ell-1}} w_{ij}^{(\ell)} x_j^{(\ell-1)}\right). && (17) \end{aligned}$$

Backpropagation

Chain Rule:
Informally, for a composition h of differentiable functions $g : \mathbb{R}^n \mapsto \mathbb{R}$ and $\mathbf{f} : \mathbb{R} \mapsto \mathbb{R}^n$,

$$h'(t) = \sum_{i=1}^{n} f_i'(t) \frac{\partial}{\partial f_i(t)} g(\mathbf{f}(t)). \tag{18}$$

Using the chain rule, we can then express the partial derivatives of the cost with respect to the network parameters.

$$\frac{\partial}{\partial w_{ij}^{\ell}} \mathcal{L} = \frac{\partial}{\partial x_i^{\ell}} \mathcal{L} \frac{\partial}{\partial w_{ij}^{\ell}} x_i^{\ell}. \tag{19}$$

Backpropagation

By noticing that $\frac{\partial}{\partial x_i^\ell}\mathcal{L}$ can be obtained recursively by:

$$\frac{\partial}{\partial x_i^\ell}\mathcal{L} = \sum_{j=1}^{n_{\ell+1}} \left(\frac{\partial}{\partial x_j^{(\ell+1)}}\mathcal{L}\right)\left(\frac{\partial}{\partial x_i^\ell} x_j^{(\ell+1)}\right), \tag{20}$$

we obtain backpropagation.