a quick word on how m and T are obtained in practice -

figures not referenced in text

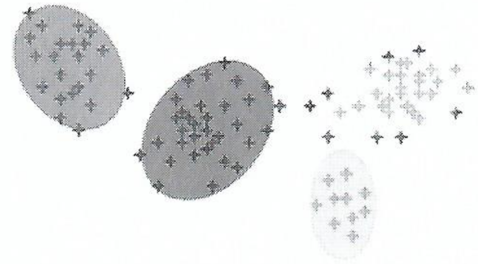

Figure 1. Gaussian Mixture Model with 4 gaussian distributions (GMM)

$$M = m + Tw \tag{2}$$

where

- $m$ is a speaker and channel independent supervector
- $T$ is a rectangular matrix, called *Total-variability matrix*
- $w$ is a an intermediate vector, or *i-vector*

These *i-vectors* provide lighter, and therefore more usable, data for application tasks.

### D. Previous works

Extracting the i-vector is interesting for speaker recognition because it is easier to apply usual classifications method on it as Cosine Distance Scoring (CDS) and Support Vector Machines (SVM) [10, 2]. I-vectors are good tools for speaker recognition because they are channel independant and compact. We will try to keep these two characteristics in the new representation we will build using neuronal networks.

## III. Use of neuronal networks

The recent success of deep learning techniques in various fields such as computer vision ([7]) and natural language processing ([1]) has sparked many explorations of their usefulness in classification tasks on *i-vectors* such as speaker recognition. Many architectures such as Deep Belief Networks ([4],[3]), Deep Neural Networks ([4],[3]), Recurrent Networks ([9]) or even a mix of Deep neural networks and Support Vector machines ([8]) have been tested and yielded better results than previous techniques, though to the best of our knowledge none directly tackled the issue of supervectors' intermediate representation. Instead, all of the aforementionned work relied on pre-existing i-vector extraction techniques. We will explore the possibility of extracting a meaningful intermediate representation of supervectors through the use of deep architectures.

### A. Formal neuron

cheese

*a) Neuron?:* A neuron can be thought of as a function which takes an $n$-dimensional vector $A$ as input and returns a scalar $e$ as output. This function typically has two internal parameters which are a bias $b$ and a weight matrix $W$. The function starts by calculating $WA+b$ before using a non-linear activation function (such as sigmoid or tanh), i.e.,

$$e = f(WA + b) \tag{3}$$

notion undefined at this point

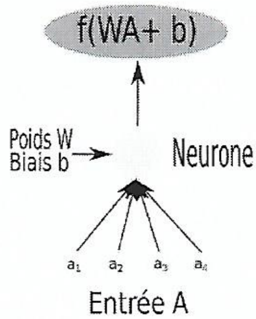


Figure 2. Formal neuron

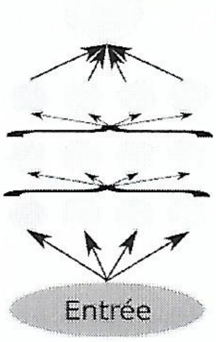


Figure 3. Deep neural network

*b) Adjusting the function:* Our endgoal is to have the neuron, and by extension the neural network, perform a certain task. The formal neuron "learns" by adjusting its function to perform better on this designated task. For simplicity's sake, we will first explain how this process - called "backpropagation" - works with a single neuron.

they are

For instance, suppose we have a bi-dimensional vector given as input and that we want our neuron to return 1 if its two coordinates are the identical and -1 if it is not. A natural way to evaluate how accurate our neuron is by looking at the distance between its output $e$ and the desired result $r$. i.e.

consists in

$$d(e) = |e - r| \tag{4}$$

We want to modify our neuron/function to minimize this distance. That means changing $e$, typically by gradient descent on function $d$ derivative. Here, $\frac{\partial d}{\partial e} = r - e$, which means we want to "move" $e$ in this direction. To this end, we modify our function's two internal parameters $W$ and $b$. $e$, and by extension $d$, can actually be seen as a function of those two parameters: $d(W, b) = |e(W, b) - r|$. Therefore $\frac{\partial d}{\partial W} = \frac{\partial d}{\partial e}\frac{\partial e}{\partial W}$, $\frac{\partial d}{\partial b} = \frac{\partial d}{\partial e}\frac{\partial e}{\partial b}$. We then only need to compute new internal parameters $W'$ and $b'$ with

?

we want to define the weights that

$$W' = W + s\frac{\partial d}{\partial W} = W - s\frac{\partial d}{\partial e}\frac{\partial e}{\partial W} \tag{5}$$

$$b' = b + s\frac{\partial d}{\partial b} = b - s\frac{\partial d}{\partial e}\frac{\partial e}{\partial b} \tag{6}$$

What we just demonstrated was a simple backpropagation algorithm called gradient descent. This method is deeply flawed, but most state of the art backpropagation methods find their origins in this humble algorithm.

*c) Neuronal network?:* Typically, a neural **network** is made up of more than a single neuron. A neural layer refers to multiple neurons working on the same input (or parts of the same input) and producing an output that can be construed as some form of concatenation of their respective outputs. This output can be in turn regarded as an alternate representation