# DEEP SUPPORT VECTOR MACHINES (SVM)

ERNEST YEUNG ERNESTYALUMNI@GMAIL.COM

Contents



Abstract. 1. Executive Summary

## 2. Right $R$-modules

Consider, as a start, the total given (training) input data, consisting of $m \in \mathbb{Z}^+$ (training) examples, each example, say the $i$th example, being represented by a "feature" vector of $d$ features, $X^{(i)} \in \mathbb{K}^d$, where $\mathbb{K}$ is a field or (categorical) classes, i.e. as examples of fields, the real numbers $\mathbb{R}$, or integers $\mathbb{Z}$, so that $\mathbb{K} = \mathbb{R}, \mathbb{Z}$ or $\mathbb{K} = \{0, 1, \dots K-1\}$, where $K$ is the total number of classes that a feature could fall into. Note that for this case, the case of $\mathbb{K} = \{0, 1, \dots K-1\}$, for $K$ classes, though labeled by integers, this set of integer labels is *not* equipped with ordered field properties (it is meaningless to say $0 < 1$, for example), nor the usual field (arithmetic) operations (you cannot add, subtract, multiply, or even take the modulus of these integers). How can we "feed into" our machine such (categorical) class data? Possibly, we should intuitively think of the Kronecker Delta function:

$$\delta_{iJ} = \begin{cases} 0 & \text{if } i \neq J \\ 1 & \text{if } i = J \end{cases}$$

for some (specific) class $J$, represented by an integer. So perhaps our machine can learn kronecker delta, or "signal"-like functions that will be "activated" if the integer value of a piece (feature) of data is exactly equal to $J$ and 0 otherwise.

Onward, supposing $\mathbb{K}$ is a field, consider the total given input data of $m$ examples:

$$\{X^{(i)} \in \mathbb{K}^d\}_{i=1,2,\dots m}^m$$

One can arrange such input data into a $m \times d$ matrix. We want to do this, for one reason, to *take advantage of the parallelism afforded by GPU(s).* Thus we'd want to act upon the entire input data set $\{X^{(i)} \in \mathbb{K}^d\}_{i=1,2,\dots m}^m$.

We'd also want to do *parallel reduce* in order to do a *summation*, $\sum_{i=1}^m$, over all (training) examples, to obtain a cost function(al) $J$.

*Date*: 5 avril 2017.

*Key words and phrases.* Machine Learning, Support Vector Machines, Neural Networks, Deep Neural Networks, Constrained Optimization, Projected Gradient Descent, CUDA C/C++, theano.

For `theano`, parallel `reduce` and `scan` operations can only be done over the first dimension of a `theano` tensor. Thus, we write the total input data as such:

$$\{X^{(i)} \in \mathbb{K}^d\}_{i=1,2,\dots m}^m \mapsto X_j^{(i)} \in \text{Mat}_{\mathbb{K}}(m,d) \tag{1}$$

i.e. $X_j^{(i)}$ is a $m \times d$ matrix of $\mathbb{K}$ values, with each $i$th row corresponding to the $i = 1, 2, \dots m$th example, and $j$th column corresponding to the $j = 1, 2, \dots d$th feature (of the feature vector $X^{(i)} \in \mathbb{K}^d$).

Let's, further, make the following abstraction, in that the input data $\{X^{(i)} \in \mathbb{K}^d\}_{i=1,2,\dots m}^m$ is really an element of a right $R$-module $\mathbf{X}$, in the category of right $R$-modules $\mathbf{Mod}_R$ with ring $R$, $R$ not necessarily being commutative.

A reason for this abstraction is that if we allow the underlying ring $R$ to be a field $\mathbb{K}$, (e.g. $\mathbb{K} = \mathbb{R}, \mathbb{Z}$), then the "usual" scalar multiplication by scalars is recovered. But we also need to equip $X \in \mathbf{Mod}_R$ with a *right action*, where ring $R$ is *noncommutative*, namely

$$R = \text{Mat}_{\mathbb{K}}(d,s) \cong L(\mathbb{K}^d, \mathbb{K}^s)$$

where $\text{Mat}_{\mathbb{K}}(d,s)$ denotes the ring of all matrices over field $\mathbb{K}$ of matrix (size) dimensions $d \times s$ (it has $d$ rows and $s$ columns), $\cong$ is an isomorphism, $L(\mathbb{K}^d, \mathbb{K}^s)$ is the space of all (linear) maps from $\mathbb{K}^d$ to $\mathbb{K}^s$. If $\mathbb{K}$ is a field, this isomorphism exists.

Thus, for

$$\begin{aligned} &X \in \mathbf{X} \in \text{Mod}_R \\ &R = \text{Mat}_{\mathbb{K}}(d,s) \cong L(\mathbb{K}^d, \mathbb{K}^s) \end{aligned} \tag{2}$$

Let $\Theta \in R$. $\Theta$ is also known as the "parameters" or "weights" (and is denoted by $w$ or $W$ by others).

Consider, as a first (pedagogical) step, only a single example ($m = 1$). $X$ is only a single feature vector, $X \in \mathbb{K}^d$. Then for basis $\{e_\mu\}_{\mu=1\dots d}$ of $\mathbb{K}^d$, corresponding dual basis $\{e^\mu\}_{\mu=1\dots d}$ (which is a basis for dual space $(\mathbb{K}^d)^*$), then

$$X\Theta = X^\mu e_\mu (\Theta_\nu{}^j e_j \otimes e^\nu) = \begin{array}{l} \mu, \nu = 1 \dots d \\ j = 1 \dots s \end{array}$$

$$X^\mu \Theta_\nu{}^j e_j \otimes e^\nu(e_\mu) = X^\mu \Theta_\nu{}^j e_j \delta_\mu^\nu = X^\mu \Theta_\mu{}^j e_j$$

In this case where $X$ is simply a vector, one could think of $X$ as a "row matrix" and $\Theta$ is a matrix, acting on the right, in matrix multiplication.

Now suppose, in general, $X \in \mathbf{X} \in \mathbf{Mod}_R$, where $X$ could be a $m \times d$ matrix, or higher-dimensional tensor. For a concrete example, say $\mathbf{X} = \text{Mat}_{\mathbb{K}}(m,d)$. We not only have to equip this right $R$-module with the usual scalar multiplication, setting ring $R = \mathbb{K}$, but also the right action version of matrix multiplication, so that $R = \text{Mat}_{\mathbb{K}}(d,s)$. This $R$ is *non-commutative*, thus, necessitating the abstraction to right $R$-modules.

Indeed, for

$$\Theta \in L(\text{Mat}_{\mathbb{K}}(m,d), \text{Mat}_{\mathbb{K}}(m,s)) \cong (\text{Mat}_{\mathbb{K}}(m,d))^* \otimes \text{Mat}_{\mathbb{K}}(m,s) \cong \text{Mat}_{\mathbb{K}}(d,s), \text{ and so}$$

$$X\Theta \in \text{Mat}_{\mathbb{K}}(m,s)$$

Further

$$X\Theta \in \text{Mat}_{\mathbb{K}}(m,s) \in \mathbf{Mod}_R$$

with ring $R$ in this case being $R = \text{Mat}_{\mathbb{K}}(s, s_2) \cong L(\mathbb{K}^s, \mathbb{K}^{s_2})$.

Since $X\Theta$ is an element in a $R$-module, it is an element in an (additive) abelian group. We can add the "intercept vector" $b$ (in theano, it'd be the usual theano vector, but with its dimensions "broadcasted" for all $m$ examples, i.e. for all $m$ rows).

$$X\Theta + b \in \mathrm{Mat}_{\mathbb{K}}(m, s)$$

Considering these 2 operatons on $X$, the "matrix multiplication on the right" or right action $\Theta$, and addition by $b$ together, through *composition*, $(\Theta, b)$, we essentially have

$$X \in \mathbf{X} \in \mathbf{Mod}_{R_1} \xrightarrow{(\Theta,b)} X\Theta + b \in \mathbf{X_2} \in \mathbf{Mod}_{R_2} \tag{3}$$

where

$$R_1 = \mathrm{Mat}_{\mathbb{K}}(d, s_1) \cong L(\mathbb{K}^d, \mathbb{K}^{s_1})$$
$$R_2 = \mathrm{Mat}_{\mathbb{K}}(s_1, s_2) \cong L(\mathbb{K}^{s_1}, \mathbb{K}^{s_2})$$

## 3. Deep Neural Networks (DNN)

Consider a(n artificial) neural network (NN) of $L+1 \in \mathbb{Z}^+$ "layers" representing $L+1$ neurons, with each layer or neuron represented by a vector $a^{(l)} \in \mathbb{K}^{s_l}$, $s_l \in \mathbb{Z}^+$, $l = 1, 2, \dots L+1$ (or, counting from 0, $l = 0, 1, \dots L$). Again, $\mathbb{K}$ is either a field (e.g. $\mathbb{K} = \mathbb{R}, \mathbb{Z}$), or categorical classes (which is a subset of $\mathbb{Z}^+$, but without any field properties, or field operations).

Nevertheless, for this pedagogical example, currently, let $\mathbb{K} = \mathbb{R}$. Recall the usual (familar) NN, accepting that we do right action multiplications (matrices act on the right, vectors are represented by "row vectors", which, actually, correspond 1-to-1 with `numpy/theano` arrays, exactly). Recall also that the sigmoidal or (general) *activation* function, $\psi^{(l)}$, acts element-wise on a vector. An "axon" between 2 layers, such as layer $l$ and layer $l+1$, is mathematically computed as follows:

$$\begin{aligned} z^{(l+1)} &:= a^{(l)}\Theta^{(l)} + b^{(l)} \\ a^{(l+1)} &:= \psi^{(l)}(z^{(l)}) \end{aligned} \tag{4}$$

where $\Theta^{(l)}, b^{(l)}$ is as above, except there will be a total of $L$ of these tuples ($l = 0, 1, 2, ...L-1$).

With $(\Theta^{(l)}, b^{(l)})$ representing the (right action) linear transformation

$$(\Theta^{(l)}, b^{(l)})(a^{(l)}) = a^{(l)}\Theta^{(l)} + b^{(l)}$$

essentially,

$$\begin{gathered} a^{(l)} \xmapsto{(\Theta^{(l)}, b^{(l)})} z^{(l+1)} \xmapsto{\psi^{(l)}\odot} a^{(l+1)} \\ (\mathbb{R}^{s_l})^m \xrightarrow{(\Theta^{(l)}, b^{(l)})} (\mathbb{R}^{s_{l+1}})^m \xrightarrow{\psi^{(l)}\odot} (\mathbb{R}^{s_{l+1}})^m \\ \mathbf{Mod}_{R^{(l)}} \xrightarrow{(\Theta^{(l)}, b^{(l)})} \mathbf{Mod}_{R^{(l+1)}} \xrightarrow{\psi^{(l)}\odot} \mathbf{Mod}_{\mathbb{R}^{(l+1)}} \end{gathered} \tag{5}$$

Since we need to operate with the activation function $\psi^{(l)}\odot$ elementwise, we (implicitly) equip $\mathbf{Mod}_{R^{(l+1)}}$ with the Hadamard product. In fact, with composition,

we can represent the $l$th axon as

$$a^{(l)} \overset{\psi^{(l)} \odot (\Theta^{(l)}, b^{(l)})}{\longmapsto} a^{(l+1)}$$

$$(\mathbb{R}^{s_l})^m \xrightarrow{\psi^{(l)} \odot (\Theta^{(l)}, b^{(l)})} (\mathbb{R}^{s_{l+1}})^m$$

$$\mathbf{Mod}_{R^{(l)}} \xrightarrow{\psi^{(l)} \odot (\Theta^{(l)}, b^{(l)})} \mathbf{Mod}_{\mathbb{R}^{(l+1)}} \tag{6}$$

The lesson is this: instead of thinking of layers, each separately, think of or focus on the relationship, the relations, between each layers, the axon, as one whole entity.

Suppose we "feed in" input data $X$ into the first or 0th layer of this NN. This means that for $a^{(0)} \in \mathbb{R}^d$,

$$a^{(0)} = X^{(i)}$$

for the $i$th (training) example.

The "output" layer, layer $L$, should output the *predicted* value, given $X$. So

$$a^{(L)} \in \mathbb{R} \text{ or } \{0, 1, \dots K-1\} \text{ or } [0, 1]$$

for regression, or classification (so it takes on discrete values) or the probability likelihood of being in some class $k$, respectively.

The entire NN can mathematically expressed as follows:

$$X^{(i)} \overset{\prod_{l=0}^{L-1} \psi^{(l)} \odot (\Theta^{(l)}, b^{(l)})}{\longmapsto} a^{(L)}$$

$$(\mathbb{R}^d)^m \xrightarrow{\prod_{l=0}^{L-1} \psi^{(l)} \odot (\Theta^{(l)}, b^{(l)})} (\mathbb{R}^{s_L} \text{ or } \mathbb{R} \text{ or } \{0, 1, \dots K-1\} \text{ or } [0, 1])^m \tag{7}$$

## References