Multi Layered Perceptron

Parakram Majumdar

May 14, 2018

1 Introduction

We are given a training set T of t points in $\mathbb{R}^d$, a set L of l labels, and a *classification* $C : T \to L$.

Machine learning is interested in learning classifiers of the form $f : \mathbb{R}^d \to L$ that can closely approximate C.

2 Single Perceptron, Single Point, Single Label

Having truly one label is an uninteresting case, as basically all points would have the same label, and there is nothing to “learn”. Instead we assume two labels, L_1 and L_2. We then train a classifier for L_1, and assign to L_2 whatever is not in L_1.

Given a single point $p \in \mathbb{R}^d$, a single perceptron c with *weight* vector $w_c \in \mathbb{R}^d$, and a *bias* $b_c \in \mathbb{R}$, for classifying a single Label L_1, accepts p with the probability

$$\hat{y}_{p,L_1} = \hat{\mathbb{P}}\left[p \in L_1\right] = \text{sigmoid}\left(w_c \cdot p + b_c\right)$$

where

$$\text{sigmoid}\left(x\right) = \frac{e^x}{e^x + 1}$$

If the actual classification of the training point is given by $y_{p,L_1} = \mathbb{I}\left[p \in L_1\right]$ then training criterion is to minimize the cross entropy:

$$\begin{aligned}
\mathbb{H} &= \mathbb{H}\left[\mathbb{P}, \hat{\mathbb{P}}\right] \\
&= \mathbb{E}\left[\log \frac{1}{\hat{y}}\right] \\
&= \sum_{L_i \in L} \mathbb{H}\left[y_{p,L_i}, \hat{y}_{p,L_i}\right] \\
&= -\sum_{L_i \in L} y_{p,L_i} \log \hat{y}_{p,L_i} \\
&= -y_{p,L_1} \log(\hat{y}_{p,L_1}) - (1 - y_{p,L_1}) \log(1 - \hat{y}_{p,L_1})
\end{aligned}$$

3 Multiple Points

Given a set of t points represented as a 2 dimensional tensor $T \in \mathbb{R}^{t \times d}$, the perceptron c can then be *applied* to all the points in T to get

$$\hat{y}_{T,c} = \operatorname{map}\left(T \cdot w_c + b_c \overrightarrow{u_t}, \operatorname{sigmoid}(\cdot)\right)$$

where

- $\operatorname{map}(\overrightarrow{x}, f)$ is the element-wise application of f on $\overrightarrow{x}$
- $\overrightarrow{u_n} = [1, 1, 1, ...n \text{ times}]$

The i^{th} element of $\hat{y}_{T,c}$ then gives us the probability of the i^{th} point being in L_1.

$$\hat{y}_{T,c}[i] = \hat{\mathbb{P}}\left[T[i] \in L_1\right] = \hat{y}_{T[i],c}$$

And therefore, if the actual classification of all training points is given by

$$y_{T,c} = \operatorname{map}\left(T, \mathbb{I}\left[\cdot \in L_1\right]\right)$$

then the training criterion is to minimize the cross entropy

$$\begin{aligned} \mathbb{H} &= \sum_{i=1}^{t} \mathbb{H}\left[y_{T,c}[i], \hat{y}_{T,c}[i]\right] \\ &= \operatorname{sum}(\operatorname{map}(y_{T,c}, \hat{y}_{T,c}, \mathbb{H}\left[\cdot, \cdot\right])) \end{aligned}$$

where

- $\operatorname{sum}(x)$ gives the sum of all the elements in the tensor x
- $\operatorname{map}(x, y, f(\cdot, \cdot))$ is the element-wise application of the binary function f to the elements of x and y. In other words,

$$\operatorname{map}(x, y, f(\cdot, \cdot))[i] = f(x[i], y[i])$$

4 Multiple Labels

Classifying a point across multiple labels may be achieved using multiple perceptrons. However, if we simply use the multiple perceptrons independently, we end up with unconstrained probabilities for a point belonging to each label. Instead, we would ideally want the probabilities to add up to 1, since a point can only belong to one label.

Suppose we have l labels, and corresponding l perceptrons. Each perceptron produces an output $x_i \in \mathbb{R}^t$, where t is the number of points. Then the probability of point T_j belonging to label L_k is

$$\hat{y}_{T_j, L_k} = \frac{e^{x_k[j]}}{\sum_{m=1}^{l} e^{x_m[j]}}$$

To simplify this notation, we introduce the softmax function:

$$\begin{aligned}
\text{softmax}(\overrightarrow{z}) &= \text{softmax}(z_1, z_2, ...z_n) \\
&= \left[\frac{e^{z_1}}{\sum_{i=1}^{n} e^{z_i}}, \frac{e^{z_2}}{\sum_{i=1}^{n} e^{z_i}}, ..., \frac{e^{z_n}}{\sum_{i=1}^{n} e^{z_i}}\right] \\
\text{softmax}(\overrightarrow{z}; j) &= \frac{e^{z_j}}{\sum_{i=1}^{n} e^{z_i}}
\end{aligned}$$

And concisely represent all the probabilities as:

$$\hat{y}_{T,L} = \text{map}(x_1, x_2, ..., x_l, \text{softmax})$$

Therefore, the training criteria is to minimize the cross-entropy:

$$\mathbb{H}(\hat{y}_{T,L}, y_{T,L}) = \sum_{T_i \in T, L_j \in L} \mathbb{H}(\hat{y}_{T_i,L_j}, y_{T_i,L_j})$$

5 Multiple Layers of Perceptrons

Multiple layers of perceptrons, with non-linear activation functions, are necessary for classifying data that is not linearly separable. An activation function is applied to the output of a neuron before feeding it to the next layer.

6 Sample 2-Layer Perceptron

Let the input data be organized into a 2-dimensional structure $T \in \mathbb{R}^{t \times d}$, where t is the number of points and d is the dimensionality of each point.

We feed these points to the first layer consisting of, say, m perceptrons. Thus, we have m weight vectors:

$$w_{1,1}, w_{1,2}, ..., w_{1,m} \in \mathbb{R}^d$$

and m biases:

$$b_{1,1}, b_{1,2}, ..., b_{1,m} \in \mathbb{R}$$

to give the unactivated output $p_{1,i} \in \mathbb{R}^t$ of the i^{th} perceptron:

$$p_{1,i} = T \cdot w_{1,i} + b_{1,i} \overrightarrow{u}_t$$

These unactivated outputs $p_{1,\cdot}$ are passed through a sigmoid activation function to get the final activated outputs $z_{1,\cdot} \in R^t$ of this layer:

$$z_{1,i} = \text{sigmoid}\,(p_{1,i}) = \frac{e^{p_{1,i}}}{1 + e^{p_{1,i}}}$$

These outputs $z_{1,\cdot}$ then become the inputs for the next and final layer. For convenience, we define $z_1 \in \mathbb{R}^{t \times m}$ as the output vector of the first layer, created by concatenating all the individual outputs.

Since se have l labels for classification, the final layer should have l perceptrons. Thus, we have l weight vectors:

$$w_{2,1}, w_{2,2}, ..., w_{2,l} \in R^m$$

and l biases:

$$b_{2,1}, b_{2,2}, ..., b_{2,l} \in R$$

to give the unactivated outputs $p_{2,\cdot} \in \mathbb{R}^t$:

$$p_{2,i} = z_1 \cdot w_{2,i} + b_{2,i} \overrightarrow{u}_t$$

We leave the outputs unactivated for this final layer, and concatenate the outputs to get $p_2 \in R^{t \times l}$ Thus, the probabilities $\hat{y} \in \mathbb{R}^{t \times l}$ of the points belonging to classes is given by:

$$\hat{y} = \text{map}(p_2, \text{sigmoid}\left(\cdot\right))$$

If the real probabilities are given by $y \in R^{t \times l}$ then the training criterion is to minimize the cross entropy:

$$\mathbb{H} = \mathbb{H}\left[\hat{y}, y\right] = \sum_{i \in T, j \in L} \mathbb{H}\left[\hat{y}_{i,j}, y_{i,j}\right]$$