

CSE 475: Statistical Methods in AI

Monsoon 2019

## SMAI-M-2019 11: Linear Perceptrons

Lecturer: C. V. Jawahar

Date: 9 Sep 2019

## 11.62 GD for Classification

We know the gradient descent equation as

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k - \eta \nabla J \quad (11.16)$$

We start with an initialization  $\mathbf{w}^0$  and update the weights/vector until we get the separating hyperplane (or until it converges).

Let us specially focus on the classification problem today. We wish to use a loss that measures classification accuracy. That is, the percentage of samples misclassified.

Let us now consider the objective as

$$J = \frac{1}{N} \sum_{i=1}^N (1 - y_i \cdot f(\mathbf{w}, \mathbf{x}_i)) \quad (11.17)$$

where  $y_i \in \{-1, +1\}$ , and  $f(\mathbf{x}_i) = \text{sign}(\mathbf{w}^T \mathbf{x}_i)$ . This provides a “unit” loss for all the misclassifications and “zero” loss for all the correct classifications. (see notes elsewhere for details.) We are not using  $\text{sign}()$ . Our ideal goal could have been to minimize this loss.

- Q: Do we have to divide the loss by 2 to get unit loss per sample?
- Q: What makes this problem non-differentiable?

However we have a serious problem. This is not differentiable. Then, what do we do? We can address this in two different ways. This leads to some popular algorithms:

- Perceptron Algorithm
- Logistic Regression
- Support Vector Machines (SVM)

## 11.63 Perceptron Algorithm

Perceptron algorithm makes the assumption that all the samples are linearly separable. In other words  $\exists \mathbf{w}$  such that  $y_i \cdot f(\mathbf{w}, \mathbf{x}_i) = +1 \forall i$ .

- Q: Can every set of (i) Two (ii) Three (iii) Four points be linearly separable in 2D?
- Q: Can any specific set of (i) Two (ii) Three (iii) Four be linearly separable in 2D for all potential labelings of these samples?
- Q: Can every set of (i) Two (ii) Three (iii) Four points be linearly separable in 2D for all potential labelings of these points?
- Note: You may know about the Ex-OR case. You may also want to know about VC-Dimension to appreciate the above questions? Why are they important in ML?

Let us now look at an alternate loss function. We define the loss/objective as

$$J = \sum_{\mathbf{x}_i \in \mathcal{E}} -y_i \cdot \mathbf{w}^T \mathbf{x}_i \quad (11.18)$$

where  $\mathcal{E}$  is the set of misclassified samples. i.e.,

$$\mathcal{E} = \{\mathbf{x}_i | \mathbf{w}^T \mathbf{x}_i < 0\}$$

Note that  $J$  is always non-negative. It can become zero when all samples are correctly classified or  $\mathcal{E}$  is empty.

How is this loss function different?

- The summation is only over the misclassified samples. This does not change anything really. (We had zero loss for all the correctly classified ones anyway).
- We then have a “non-unity” loss for all the misclassified ones. (This is different from the previous loss). It is proportional to how far it is from the line/plane/hyperplane.
- Q: Why not then we use the following objective? (Demonstrate with an example)

$$J = \frac{1}{N} \sum_{i=1}^N -y_i \cdot \mathbf{w}^T \mathbf{x}_i$$

- In our objective, the farther the sample, the more the loss is. (Indeed, not very ideal!!). *Q: Why do you think this is not ideal? Is it really non ideal if our objective sums only over  $\mathcal{E}$*

Pleasantly, this is now differentiable. That is an advantage. Also when all samples are correctly classified (when the problem is linearly separable), the loss becomes zero ( $\mathcal{E}$  becomes empty set.) and our algorithm will converge (loss is zero, derivative is also zero.)

There is also an advantage with this loss. A sample that is far from the line/plane will pull/push/rotate the line/plane more than one that is very near the line. This will help in faster convergence.

Let us now re-write our gradient descent equation as:

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{\mathbf{x}_i \in \mathcal{E}} y_i \mathbf{x}_i \quad (11.19)$$

Q: Verify this.

We start with a single sample version of the perceptron. i.e., the perceptron algorithm is now:

- Initialize  $k = 0$ ,  $\mathbf{w}^0$
- While  $\mathcal{E}$  is not empty
  - Pick an arbitrary element  $\mathbf{x}_j$  from  $\mathcal{E}$
  - $\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + y_j \mathbf{x}_j$
  - $k \leftarrow k + 1$
- Return  $\mathbf{w}_k$

Note that (i) We assumed the learning rate  $\eta = 1$  (ii) We update the  $\mathbf{w}$  for each sample, than for a batch.

For a batch of samples, we can define the weight update as:

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{\mathbf{x}_j \in \mathcal{E}} y_j \mathbf{x}_j$$

Another way we could define the weight update is with the help of desired/target ( $t$ ) and output ( $o$ ). In this situation, the update rule is:

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{i=1}^N (t_i - o_i) \mathbf{x}_i$$

- Q: prove that both the update rules are the same (with a scale change in  $\eta$ ).

Note that when desired and predicted outputs are same,  $t - o$  is zero. Else it is either Positive or negative (when  $y_i$  as well as  $t - o$  will have the same sign when the sample is misclassified.)

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{i=1}^N (t_i - o_i) \mathbf{x}_i \quad (11.20)$$

Note that  $\eta$  is a learning rate and it could absorb any scaling. ( $\eta$  in all these equations need not be identical, see yourself, if they differ by a scale factor.). Here the summation is over all the samples. Note that, this does not change the update rule. The additional terms are zero.

Algorithm now has the following steps:

1. Initialize  $\mathbf{w}$ ,  $k=0$

2. We update the  $\mathbf{w}$  as

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{i=1}^N (t_i - o_i) \mathbf{x}_i$$

or

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{\mathbf{x} \in \mathcal{E}} y_i \mathbf{x}_i$$

3.  $k \leftarrow k + 1$

4. Repeat steps 2-4

- until  $\mathcal{E}$  is empty. (the ideal termination criteria for perceptron algorithm) Or
- until the change in weight is small (say less than  $\theta$ ).

## 11.64 More Details

### 11.64.1 Delta Rule

Let us consider a “regression” problem with  $o_i$  as  $\mathbf{w}^T \mathbf{x}_i$ . Consider a least square objective as

$$J = \sum_{i=1}^N (t_i - o_i)^2$$

The gradient descent update rule is:

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{i=1}^N (t_i - o_i) \mathbf{x}_i$$

- Q: is it really the same as we saw early? Not really!!

### 11.64.2 Back to Perceptrons

Is it really possible to arrive the gradient descent update rule as Equation 11.20 from the objective like:

$$J = \sum_{i=1}^N (t_i - o_i)^2$$

with  $o_i = \text{sign}(\mathbf{w}^T \mathbf{x}_i)$  ? There is a small catch.

With small changes in the  $\mathbf{w}$  or the line, the  $J$  is not changing. This is not very appropriate to look at this as gradient descent objective. Let us modify this as

$$J = \sum_{i=1}^N (t_i - o_i)^2 (-\mathbf{w}^T \mathbf{x}_i)$$

This additional terms pulls (or pushes) *proportionally*. Let us now rewrite this  $J$  as sum of two parts one over  $\mathcal{E}$  and the other on not in  $\mathcal{E}$ .

$$J = J_1 + J_2 = \sum_{\mathbf{x}_i \in \mathcal{E}} (t_i - o_i)^2 (-\mathbf{w}^T \mathbf{x}_i) + \sum_{\mathbf{x}_i \text{ not in } \mathcal{E}} (t_i - o_i)^2 (-\mathbf{w}^T \mathbf{x}_i)$$

$$J = J_1 + J_2 = \sum_{\mathbf{x}_i \in \mathcal{E}} (2 \cdot t_i)^2 (-\mathbf{w}^T \mathbf{x}_i) + \sum_{\mathbf{x}_i \text{ not in } \mathcal{E}} 0 \times (\mathbf{w}^T \mathbf{x}_i)$$

We know that  $J_2$  is zero. When  $\mathbf{x}_i \in \mathcal{E}$ ,  $(t_i - o_i)$  is  $2t_i$  i.e.,  $2y_i$ .

$$\nabla J = 2 \sum_{i=1}^N (t_i - o_i) \mathbf{x}_i$$

Note that:

- Either  $(t_i - o_i)$  is zero
- Or we know that  $(t_i - o_i)$  is  $2t_i$  and  $\frac{\partial -\mathbf{w}^T \mathbf{x}_i}{\partial \mathbf{w}} = -\mathbf{x}_i$ .

This leads to:

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k - \eta \nabla J$$

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{i=1}^N (t_i - o_i) (-\mathbf{x}_i)$$

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{\mathbf{x}_i \in \mathcal{E}} 2 \cdot t_i \mathbf{x}_i$$

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{\mathbf{x}_i \in \mathcal{E}} y_i \mathbf{x}_i$$

(with changes in scale for learning rate  $\eta$ )

## 11.65 Discussions and Examples

Let us consider some simple situations first and see how the perceptron algorithm behaves.

Let us simplify the update rule first. Assume  $\eta = 1$ . Let us assume that there is only one sample. Also let us assume that  $\mathbf{x}$  has only one dimension i.e., scalar. This simplifies to:

$$w^{k+1} \leftarrow w^k + (t_i - o_i) x_i$$

Now let us consider some situations.

- When there is no misclassification. i.e.,  $t_i = o_i$ . In this case,  $w$  does not change.
- Let us consider  $t_i = +1$ ,  $o_i = -1$  and  $x_i$  is positive. Sample is misclassified. Which means  $w^k \cdot x_i$  is negative. ( $x_i$  is fixed and positive.) This means  $w^k$  is negative. We need to increase  $w^k$  to (positive to) minimize/avoid misclassification. If we substitute the values/signs in the above equation, we see that perceptron algorithm does this exactly.
- Now Let us consider  $t_i = +1$ ,  $o_i = -1$  and  $x_i$  is negative. Sample is misclassified. Which means  $w^k \cdot x_i$  is negative. ( $x_i$  is fixed and negative.) Therefore,  $w^k$  is positive. We need to decrease  $w^k$  to minimize/avoid misclassification. If we substitute the values/signs in the above equations, we see that perceptron algorithm does this exactly.
- Q: Convince yourself for all other cases also.

### 11.65.1 Example/Problem in 2D

Q: Consider a set of vectors in 2D.

$$\{[1, 1]^T, [1, 3]^T, [2, 1]^T, [2, 2]^T,$$

$$[-1, -1]^T, [-1, -3]^T, [-2, -1]^T, [-2, -2]^T\}$$

The first four are from class 1 and the rest four are from class 2.

- Plot the samples in a 2D plane with “x” for positive classes and “o” for negative class. Is this set linearly separable?
- Start with a random vector for  $\mathbf{w}$  and show that the perceptron algorithm converges to a separating line/plane for different values of  $\eta$ . (make sure that you start with a vector that has error!!).
- Why is that the final answers are different for different initializations/ $\eta$ ? Then which is the best solution?

## 11.66 Some Theoretical Results

Perceptron algorithm has a number of interesting theoretical properties/results. A summary is below.

- If there exist a set of weights that are consistent with the data, the perceptron algorithm will converge.
- If the training data is not Linearly Separable, the perceptron algorithm will eventually repeat the same set of weights and thereby enter an infinite loop.

- If the training data is linearly separable, algorithm will converge in a maximum of  $M$  steps. (See more below for the bounds).
- Every boolean function can be represented by some network of perceptrons only two levels deep.

## 11.67 Closer Look at the Theoretical Results

**Problem** Our problem is to design a classification algorithm for  $N$  samples  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$  where  $\mathbf{x}_i \in R^d$  and  $y_i \in \{-1, +1\}$

We make the assumption that the data is linearly separable. What does it mean?

This simply means that there exists a solution  $\mathbf{w}^* \in R^d$  such that  $\|\mathbf{w}^*\| = 1$  and a small quantity  $\gamma$  such that

$$y_i(\mathbf{w}^{*T} \mathbf{x}_i) > \gamma$$

for all  $i = 1, \dots, N$ . Note that when the  $\text{sign}(\mathbf{w}^T \mathbf{x})$  is positive and  $y_i$  is  $+1$ , the product is positive. Similarly when the sign  $(\mathbf{w}^T \mathbf{x})$  is negative and  $y_i$  is  $-1$ , the product is still positive. In other-words, when a sample is classified correctly, the product is always positive and the inequality above is true for a small positive quantity  $\gamma$ .

We also make an assumption that samples are bounded. Not a strong assumption. What does it mean?

$$\|\mathbf{x}_i\| \leq R \quad \forall i$$

### 11.67.1 Summary of Perceptron Algorithm Assumptions

We start with a random initialization and update solution based on the misclassified samples. We repeat this until we find that all the samples are correctly classified.

We make two assumptions.

1.  $y_i(\mathbf{w}^{*T} \mathbf{x}_i) > \gamma$  for all  $i$
2.  $\|\mathbf{x}_i\| \leq R$  for all  $i$ .
3.  $\|\mathbf{w}^*\| = 1$

Where  $\mathbf{w}^*$  is the final/optimal vector.  $\gamma$  is a positive quantity that ensure separability and also a margin. The first one says that the samples are separable. The second one says that all the samples are bounded. (not of infinite magnitude.). Note that  $\gamma$  and  $R$  are positive.

## 11.68 Convergence of Perceptron

Now our main result is *The Perceptron Learning Algorithm makes at most  $\frac{R^2}{\gamma^2}$  iterations and converge. It returns a separating hyperplane after that.*

We know the perceptron update equation as

$$\mathbf{w}^{k+1} = \mathbf{w}^k + y_i \mathbf{x}_i$$

This assumes  $\eta = 1$ . Also this is a single sample update rule.

Let us assume that we start with  $\mathbf{w}^0$  as a zero vector  $\mathbf{0}$ .

Let us multiply  $\mathbf{w}^*$  on both sides.

$$(\mathbf{w}^{k+1})^T \mathbf{w}^* = (\mathbf{w}^k)^T \mathbf{w}^* + y_i (\mathbf{x}_i)^T \mathbf{w}^*$$

since the last term is greater than  $\gamma$  (first assumption).

$$(\mathbf{w}^{k+1})^T \mathbf{w}^* > (\mathbf{w}^k)^T \mathbf{w}^* + \gamma$$

From induction

$$(\mathbf{w}^{k+1})^T \mathbf{w}^* > k\gamma$$

We know that  $\mathbf{a}^T \mathbf{b} < \|\mathbf{a}\| \cdot \|\mathbf{b}\|$ . Therefore,

$$(\mathbf{w}^{k+1})^T \mathbf{w}^* < \|\mathbf{w}^*\| \cdot \|\mathbf{w}^{k+1}\| = \|\mathbf{w}^{k+1}\|$$

or finally

$$\|\mathbf{w}^{k+1}\| > k\gamma \quad \text{and} \quad \|\mathbf{w}^{k+1}\|^2 > k^2 \gamma^2 \quad (11.21)$$

We have now a lower bound on  $\|\mathbf{w}^{k+1}\|^2$

Let us also try to get an upper bound for  $\|\mathbf{w}^{k+1}\|^2$

$$\mathbf{w}^{k+1} = \mathbf{w}^k + y_i \mathbf{x}_i$$

Therefore:

$$\|\mathbf{w}^{k+1}\|^2 = \|\mathbf{w}^k + y_i \mathbf{x}_i\|^2$$

$$\|\mathbf{w}^{k+1}\|^2 = \|\mathbf{w}^k\|^2 + \|y_i \mathbf{x}_i\|^2 + 2\|\mathbf{w}^k\| \mathbf{x}_i^T y_i$$

Note that the last term is negative. Since the updates happen only when there is an error in the sample/classification.

$$\|\mathbf{w}^{k+1}\|^2 \leq \|\mathbf{w}^k\|^2 + \|\mathbf{x}_i\|^2$$

$$\|\mathbf{w}^{k+1}\|^2 \leq \|\mathbf{w}^k\|^2 + R^2$$

$$\|\mathbf{w}^{k+1}\|^2 \leq kR^2$$

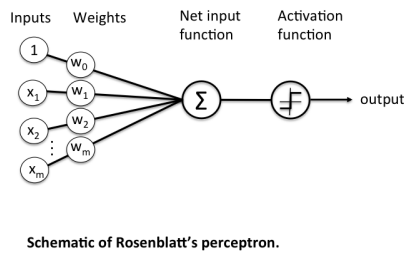


Figure 11.6: A pictorial representation of the neuron. Often a single large circle is shown for a neuron wherein the weighted addition and nonlinearity is combined.

per sample and update for each sample. A sample may be seen multiple times (in a cyclic manner).

- Batch version: Compute the gradient for the batch (full set) and update once.
- Mini-Batch: Instead of taking the full batch, take a smaller batch. This is useful when the data is large.
- Stochastic: Samples in online or mini-batch are selected randomly.

Q: Do write the pseudo code for each of these. Make sure your pseudo code is very close to the programming language that you use.

### 11.68.1 Combining the Bounds

We can now write

$$k^2 \gamma^2 < \|\mathbf{w}^{k+1}\|^2 \leq k R^2$$

Combining the upper and lower bounds.

$$k < \frac{R^2}{\gamma^2}$$

## 11.69 Neural Networks View Point

A popular view point of the perceptron is to appreciate it as a “neuron”. Though the story has origins in our attempts to understand and reverse engineer human brain and perception skills, it is much easier to appreciate it as a powerful mathematical model at this stage.

A neuron accepts multiple inputs. Weigh each one. Add the inputs. Pass through a nonlinearity. In the case of perceptron, the nonlinearity is a step like nonlinearity. See also figure.

We will revisit the neural network view point at a later stage when we discuss “multi layer perceptrons”.

## 11.70 Variations in Gradient Descent

We now know: (i) How to define a loss function (ii) how to optimize it with gradient descent update equations.

There are in fact many variations in the implementation.

- Single sample or online version: which assumes that samples come one by one. This computes gradient