Clarification about Perceptron Rule vs. Gradient Descent vs. Stochastic Gradient Descent implementation

I experimented a little bit with different Perceptron implementations and want to make sure if I understand the "iterations" correctly.

Rosenblatt's original perceptron rule

As far as I understand, in Rosenblatt's classic perceptron algorithm, the weights are simultaneously updated after every training example via

$$\Delta w^{(t+1)} = \Delta w^{(t)} + \eta(target - actual)x_i$$

where eta is the learning rule here. And target and actual are both thresholded (-1 or 1). I implemented it as 1 iteration = 1 pass over the training sample, but the weight vector is updated after each training sample.

And I calculate the "actual" value as

$$sign(\boldsymbol{w}^T \boldsymbol{x}) = sign(w_0 + w_1 x_1 + \ldots + w_d x_d)$$

Stochastic gradient descent

$$\Delta w^{(t+1)} = \Delta w^{(t)} + \eta(target - actual)x_i$$

Same as the perceptron rule, however, `target` and `actual` are not thresholded but real values. Also, I count "iteration" as path over the training sample.

Both, SGD and the classic perceptron rule converge in this linearly separable case, however, I am having troubles with the gradient descent implementation.

Gradient Descent

Here, I go over the training sample and sum up the weight changes for 1 pass over the training sample and updated the weights thereafter, e.g,

for each training sample:

$$\Delta w_{new} \mathrel{+}= \Delta w^{(t)} + \eta(target - actual)x_i$$

...

after 1 pass over the training set:

$$\Delta w \mathrel{+}= \Delta w_{new}$$

I am wondering, if this assumption is correct or if I am missing something. I tried various (up to infinitely small) learning rates but never could get it to show any sign of convergence. So, I am wondering if I misunderstood sth. here.

Thanks, Sebastian

optimization gradient-descent perceptron

edited Feb 15 '15 at 22:00

asked Feb 15 '15 at 21:46

Sebastian
601 3 20

1 Answer

You have a couple mistakes in your updates. I think generally you're confusing the value of the current weights with the *difference* between the current weights and the previous weights. You have Δ symbols scattered around where there shouldn't be any, and += where you should have =.

> Perceptron:

> $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t(y^{(i)} - \hat{y}^{(i)})\boldsymbol{x}^{(i)}$,
>
> where $\hat{y}^{(i)} = sign(\boldsymbol{w}^\top \boldsymbol{x}^{(i)})$ is the model's prediction on the i^{th} training example.

This can be viewed as a stochastic subgradient descent method on the following "perceptron loss" function*:

> Perceptron loss:
>
> $$loss_{\boldsymbol{w}}(y^{(i)}) = \max(0, -y^{(i)}\boldsymbol{w}^\top \boldsymbol{x}^{(i)}).$$
>
> $$\partial loss_{\boldsymbol{w}}(y^{(i)}) = \begin{cases} \{0\}, & \text{if } y^{(i)}\boldsymbol{w}^\top \boldsymbol{x}^{(i)} > 0 \\ \{-y^{(i)}\boldsymbol{x}^{(i)}\}, & \text{if } y^{(i)}\boldsymbol{w}^\top \boldsymbol{x}^{(i)} < 0 \\ [-1, 0] \times y^{(i)}\boldsymbol{x}^{(i)}, & \text{if } \boldsymbol{w}^\top \boldsymbol{x}^{(i)} = 0 \end{cases}.$$

Since perceptron already *is* a form of SGD, I'm not sure why the SGD update should be different than the perceptron update. The way you've written the SGD step, with non-thresholded values, you suffer a loss if you predict an answer *too* correctly. That's bad.

Your batch gradient step is wrong because you're using "+=" when you should be using "=". The current weights are added *for each training instance*. In other words, the way you've written it,

> $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \sum_{i=1}^{n}\{\boldsymbol{w}^{(t)} - \eta_t \partial loss_{\boldsymbol{w}^{(t)}}(y^{(i)})\}$.

What it should be is:

> $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \sum_{i=1}^{n} \partial loss_{\boldsymbol{w}^{(t)}}(y^{(i)})$.

Also, in order for the algorithm to converge on every and any data set, you should decrease your learning rate on a schedule, like $\eta_t = \frac{\eta_0}{\sqrt{t}}$.

* The perceptron algorithm is not *exactly* the same as SSGD on the perceptron loss. Usually in SSGD, in the case of a tie ($\boldsymbol{w}^\top \boldsymbol{x}^{(i)} = 0$), $\partial loss = [-1, 0] \times y^{(i)}\boldsymbol{x}^{(i)}$, so $\mathbf{0} \in \partial loss$, so you would be allowed to not take a step. Accordingly, perceptron loss can be minimized at $\boldsymbol{w} = \mathbf{0}$, which is useless. But in the perceptron *algorithm*, you are required to break ties, and use the subgradient direction $-y^{(i)}\boldsymbol{x}^{(i)} \in \partial loss$ if you choose the wrong answer.

So they're not exactly the same, but if you work from the assumption that the perceptron algorithm is SGD for some loss function, and reverse engineer the loss function, perceptron loss is what you end up with.

edited Feb 16 '15 at 13:27

answered Feb 16 '15 at 1:24

Sam Thomson
46 2

> Thank you Sam, and I do apologize for my messy question. I don't know where the deltas come from, but the "+=" was the the thing that went wrong. I completely overlooked that part. Thanks for the thorough answer!
> – Sebastian Feb 16 '15 at 3:13