The perceptron learning algorithm

Marco Kuhlmann
Department of Computer and Information Science

LINKÖPING UNIVERSITY

The perceptron

feature vector

$$\hat{k} = \arg\max_k \boldsymbol{x}\boldsymbol{w}_k + b_k$$

bias

weight vector

Intuition for the perceptron learning algorithm

- The weight vector $\boldsymbol{w}_k$ for a given class k can be interpreted as a prototypical example from that class.

- The dot product between $\boldsymbol{w}_k$ and the feature vector $\boldsymbol{x}$ can be interpreted as a measure of similarity between the two.

  We predict the class whose weight vector is closest to $\boldsymbol{x}$.

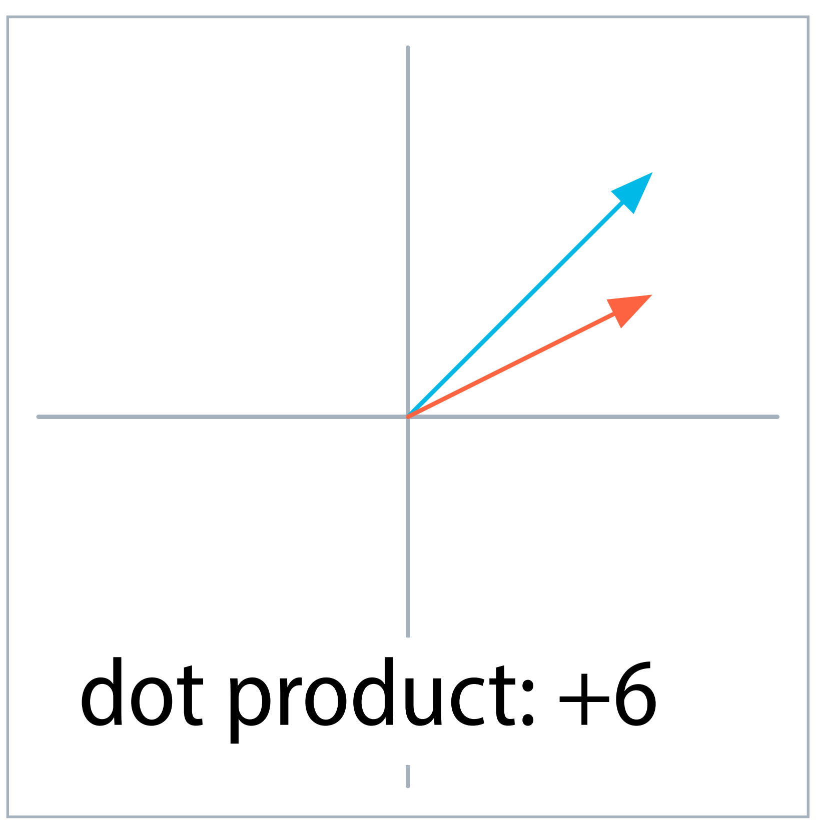
- During training, we want to push the weight vector $\boldsymbol{w}_k$ closer to those $\boldsymbol{x}$ that are instances of k, and away from the other ones.

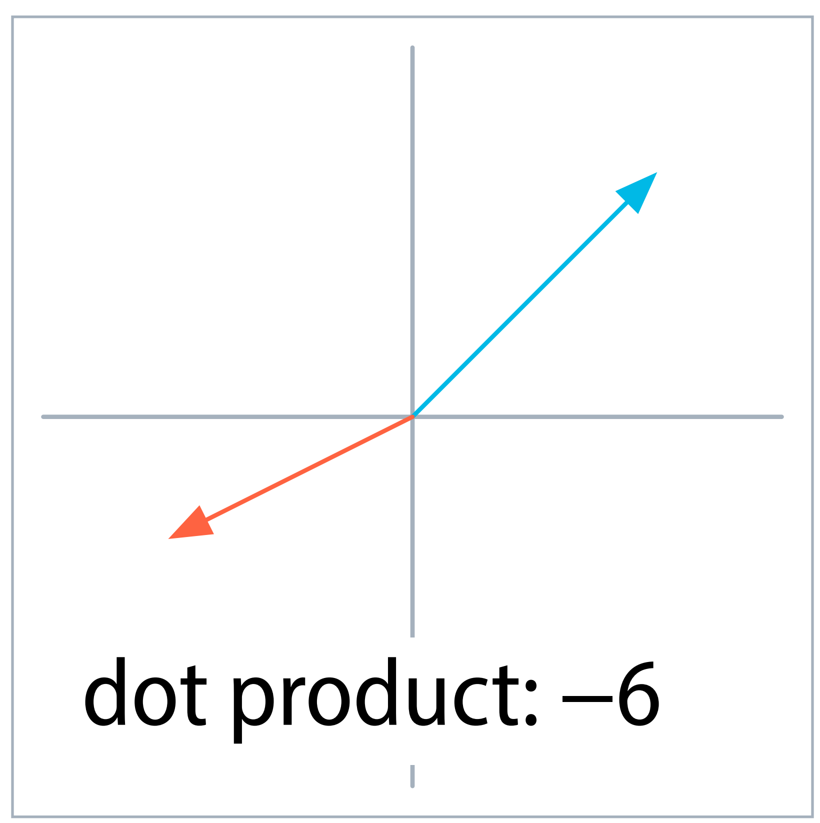
Geometric interpretation of the dot product

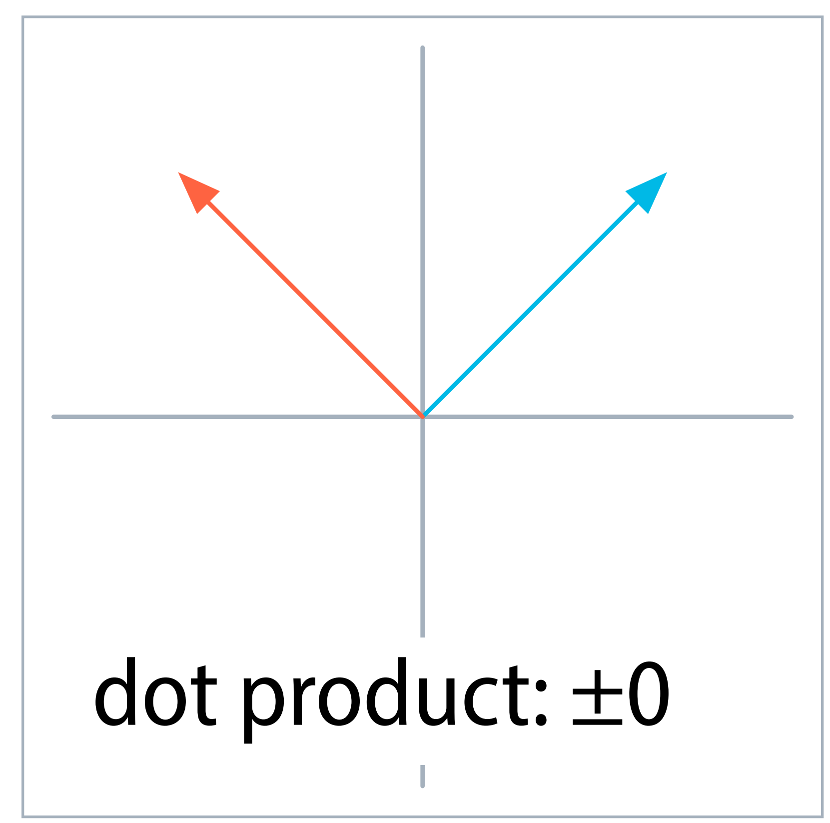
x_1	x_2	w_1	w_2
+2	+1	+2	+2

x_1	x_2	w_1	w_2
−2	−1	+2	+2

x_1	x_2	w_1	w_2
−2	+2	+2	+2







The perceptron learning algorithm

for each class k **do**

$\boldsymbol{w}_k \leftarrow \mathbf{0}$; $b_k \leftarrow 0$ // initialise weight vector and bias

for each epoch e **do**

for each training example $(\boldsymbol{x}, y)$ **do**

$p \leftarrow$ that class k for which the activation $\boldsymbol{x}\boldsymbol{w}_k + b_k$ is maximal

if $p \neq y$ **do**

$\boldsymbol{w}_p \leftarrow \boldsymbol{w}_p - \boldsymbol{x}$; $b_p \leftarrow b_p - 1$

$\boldsymbol{w}_y \leftarrow \boldsymbol{w}_y + \boldsymbol{x}$; $b_y \leftarrow b_y + 1$

Eisenstein § 2.3.1

Properties of the learning algorithm

- **Online learning**

  Instead of processing the entire data set as one large batch, process one example at a time.

- **Error-driven learning**

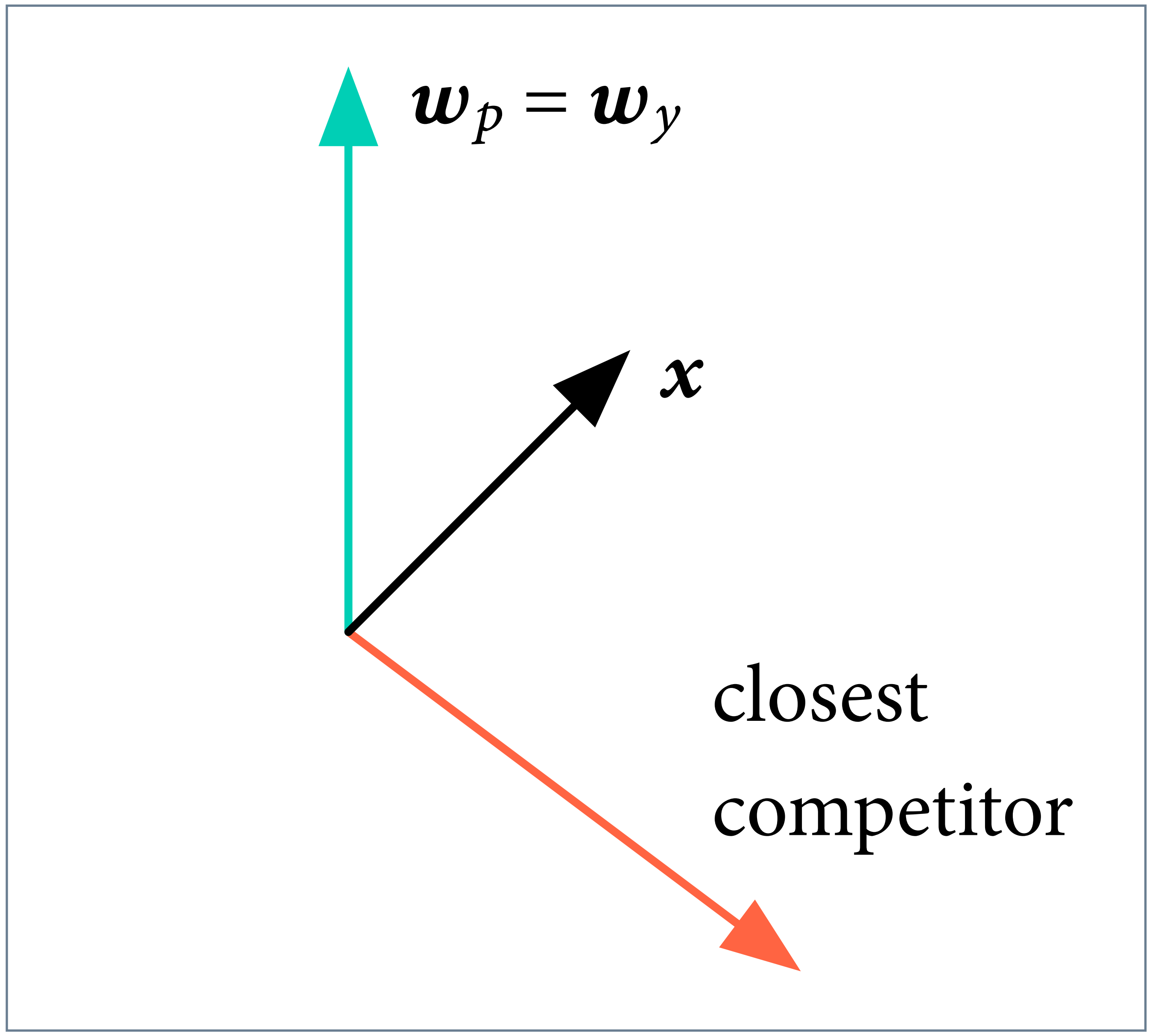
  Do not update the weight vectors unless there are classification errors (misclassified training examples).

No need to update

The predicted class p
is the same as
the gold-standard class y.

That means that the vector
that is closest to $\boldsymbol{x}$
is identical to $\boldsymbol{w}_y$.

All weight vectors
remain unchanged.

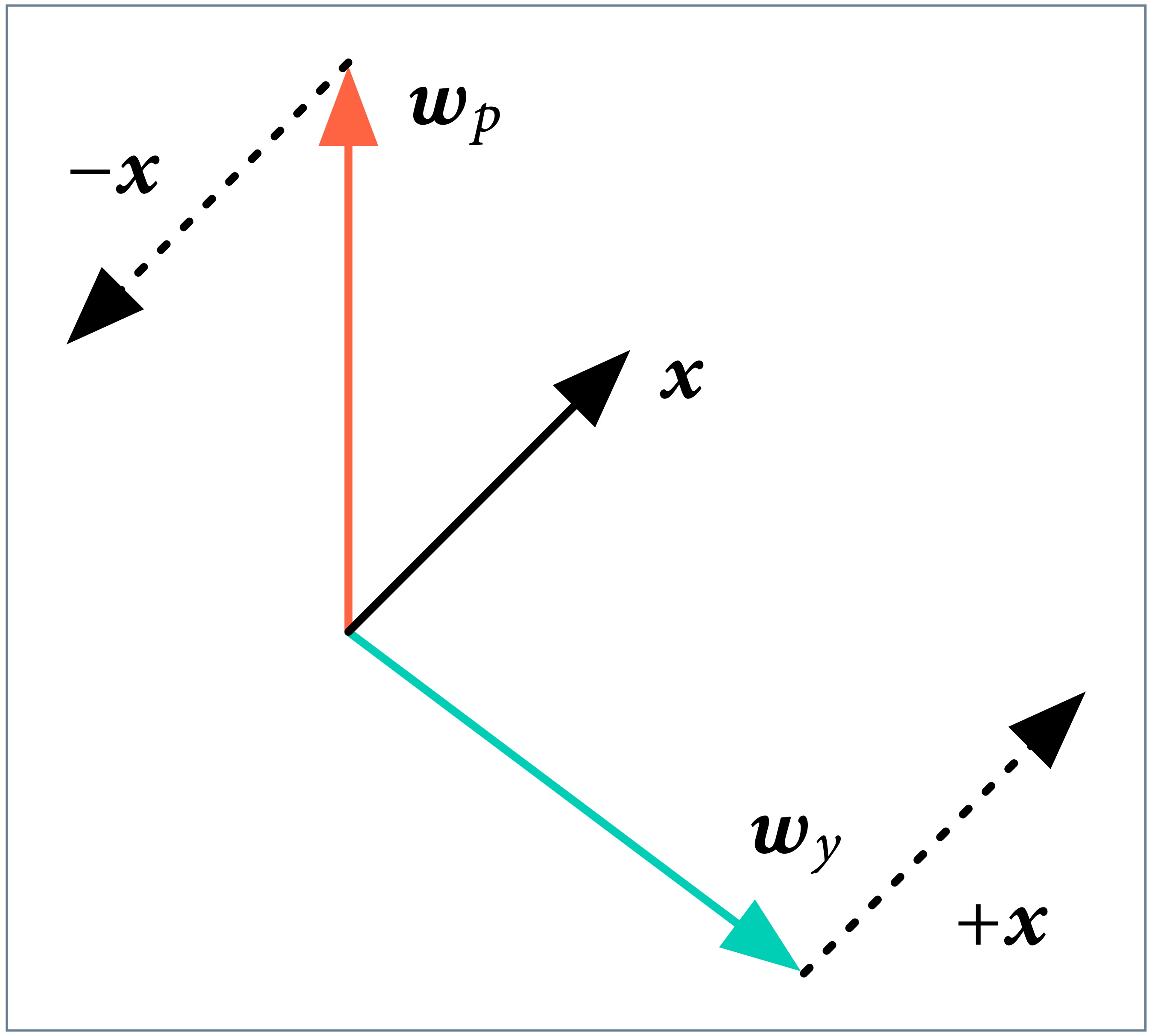


Update on error

The predicted class p
is different from
the gold-standard class y.

That means that the vector
that is closest to $\boldsymbol{x}$
is different from $\boldsymbol{w}_y$.

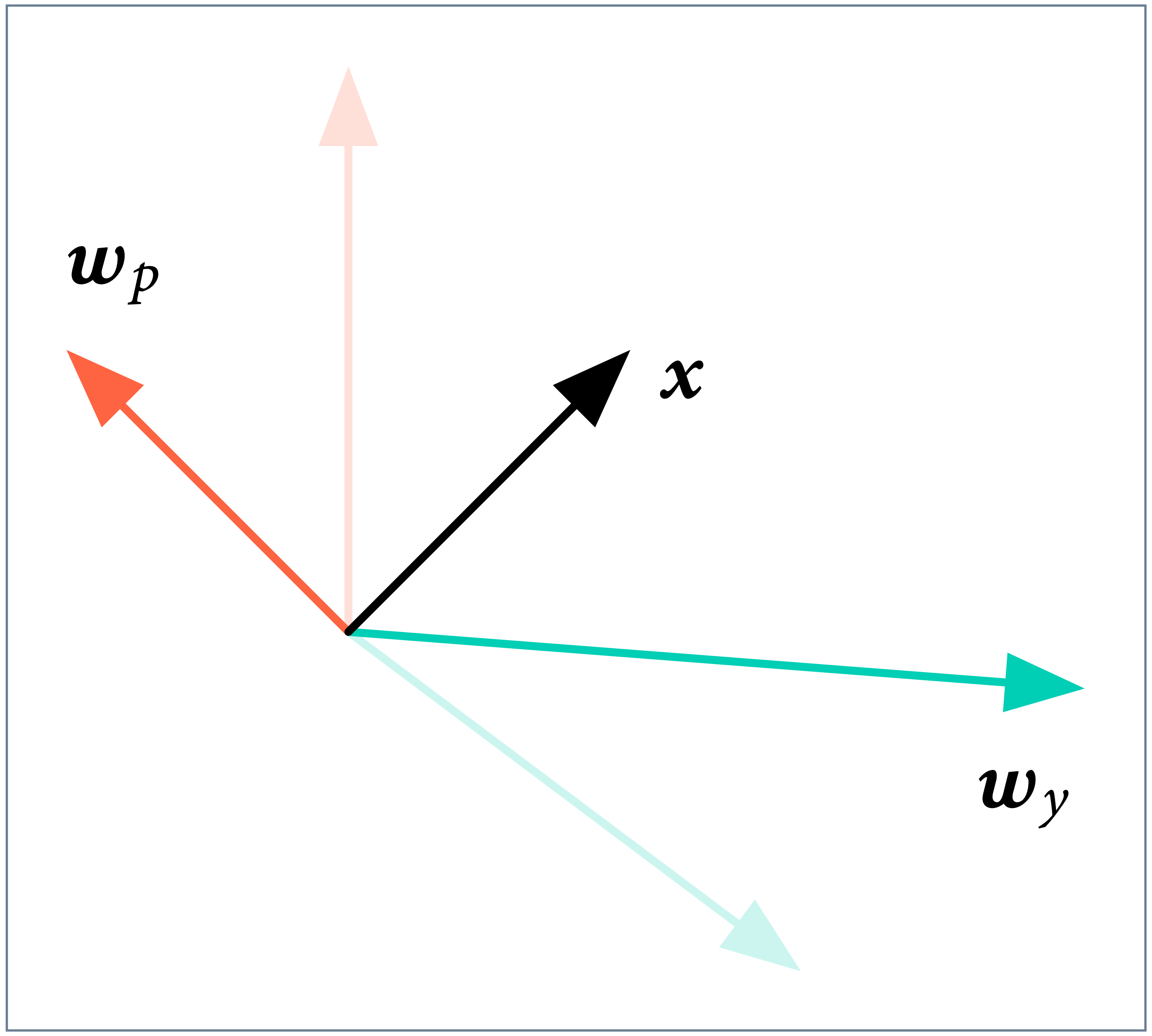
$$\boldsymbol{w}_p \leftarrow \boldsymbol{w}_p - \boldsymbol{x}$$
$$\boldsymbol{w}_y \leftarrow \boldsymbol{w}_y + \boldsymbol{x}$$



Update on error

The update pushes
$\boldsymbol{w}_p$ away from $\boldsymbol{x}$, and
$\boldsymbol{w}_y$ closer towards $\boldsymbol{x}$.

The next time we see $\boldsymbol{x}$,
we are more likely to predict
the gold-standard class y.



Averaged perceptron

Eisenstein § 2.3.2

- Consider a data set with 10,000 examples.
- Suppose that after the first 100 examples, the weight vector is so good that no updates happen for the next 9,899 examples.
- Suppose now that the perceptron does wrong on #10,000.

The perceptron is too sensitive to late examples. A simple idea for how to remedy this is to *average* the weight vectors after training.

Example attributed to Hal Daumé

The averaging trick

- A naive implementation of the averaged perceptron would collect all weight vectors into a list to compute the average.
- A slightly less naive implementation would keep track of an averaged weight vector during learning.
- This implementation can be quite inefficient because it would need to update the averaged vector at every example.

  In contrast, remember that the weight vector is only updated on errors.

The averaged perceptron learning algorithm

for each class k **do**

  $\boldsymbol{w}_k \leftarrow \mathbf{0}$; $b_k \leftarrow 0$; $\boldsymbol{w}_k \leftarrow \mathbf{0}$; $b_k \leftarrow 0$

count $\leftarrow 1$

for each epoch e **do**

  for each training example $(\boldsymbol{x}, y)$ **do**

    $p \leftarrow$ that class k for which the activation $\boldsymbol{x}\boldsymbol{w}_k + b_k$ is maximal

    if $p \neq y$ **do**

      $\boldsymbol{w}_p \leftarrow \boldsymbol{w}_p - \boldsymbol{x}$; $b_p \leftarrow b_p - 1$; $\boldsymbol{w}_p \leftarrow \boldsymbol{w}_p - \text{count} \cdot \boldsymbol{x}$; $b_p \leftarrow b_p - \text{count}$

      $\boldsymbol{w}_y \leftarrow \boldsymbol{w}_y + \boldsymbol{x}$; $b_y \leftarrow b_y + 1$; $\boldsymbol{w}_y \leftarrow \boldsymbol{w}_y + \text{count} \cdot \boldsymbol{x}$; $b_y \leftarrow b_y + \text{count}$

    count $\leftarrow$ count + 1

for each class k **do**

  $\boldsymbol{w}_k \leftarrow \boldsymbol{w}_k - \boldsymbol{w}_k$ / count ; $b_k \leftarrow b_k - b_k$ / count

The new weight vectors and biases in red are used to implement the averaging trick.

Eisenstein § 2.3.2