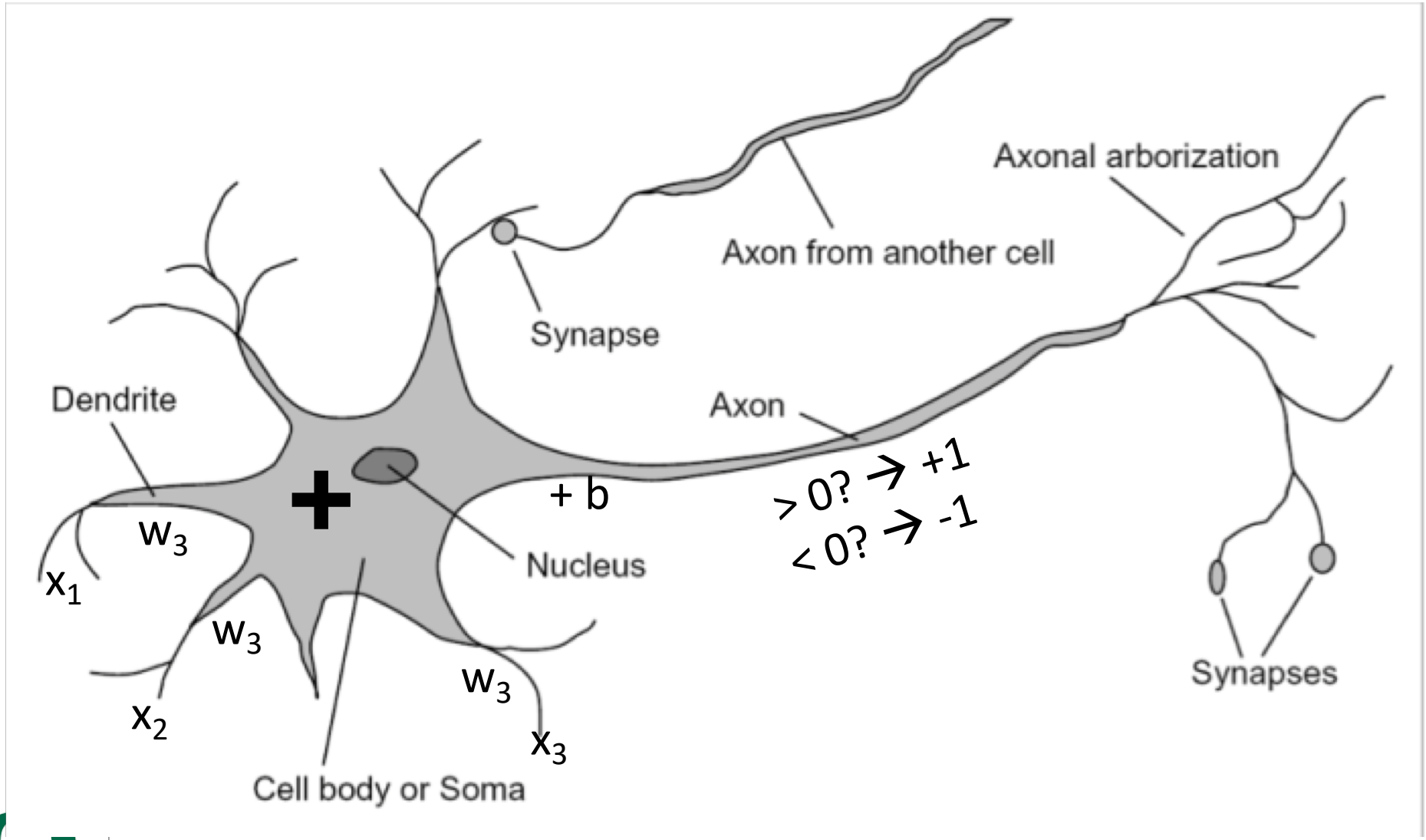
Perceptrons

Based on slides by Daniel Lowd, Vibhav Gogate, Pedro Domingos, Tom Mitchell, Carlos Guestrin, Luke Zettlemoyer and Dan Weld.

Neurons


Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
+ b
> 0? → +1
< 0? → -1
w_3
x_1
Nucleus
w_3
w_3
x_2
x_3
Synapses
Cell body or Soma

Perception / Linear Models

Example: $x = (x_1, x_2, \dots, x_D), y \in \{-1, 1\}$

Perceptron finds a weight for each features and a bias

$w = (w_1, w_2, \dots, w_D), b$ ⟵ Model Parameters

Activation: $a = \left[\sum_{d=1}^{D} w_d x_d\right] + b$

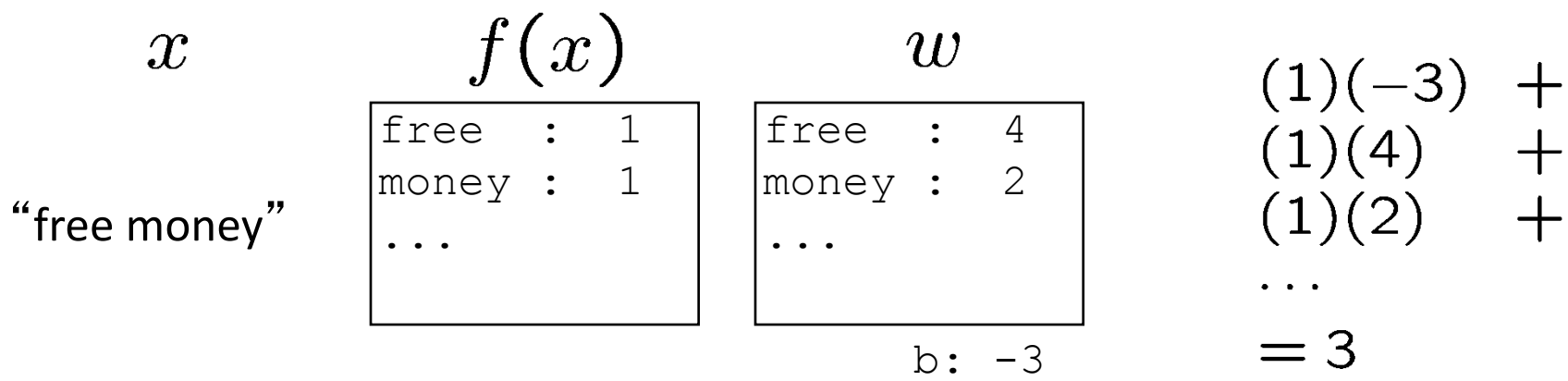
Prediction: $sign(a) = \begin{cases} +1 \; if \; a > 0 \\ -1 \; otherwise \end{cases}$

Example: Spam

- Imagine 2 features (spam is "positive" class):
  - free (number of occurrences of "free")
  - money (occurrences of "money")

$$w \cdot f(x)$$

$$\sum_i w_i \cdot f_i(x)$$



w.f(x) > 0 ➔ SPAM!!!

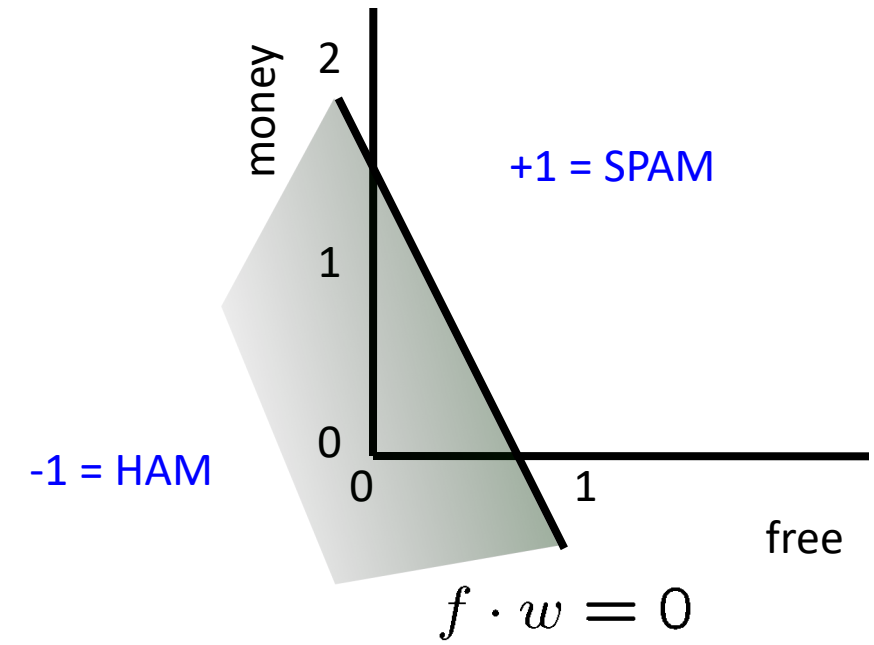
Binary Decision Rule

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to Y=+1
  - Other corresponds to Y=-1

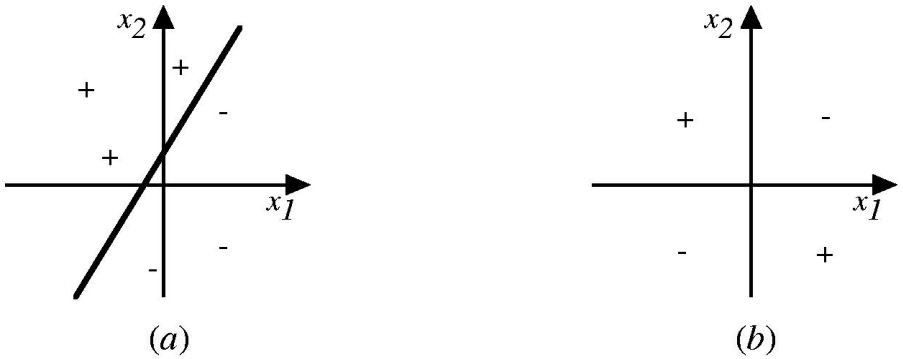
w

```
free   :   4
money  :   2
...
```

b: -3



Decision Surface of a Perceptron



Represents some useful functions

- What weights represent $g(x_1, x_2) = AND(x_1, x_2)$?

But some functions not representable

- All not linearly separable
- Therefore, we'll want networks of these...

How to find the parameters?

Algorithm 5 PERCEPTRONTRAIN($\mathbf{D}$, $MaxIter$)

```
1:  w_d ← 0, for all d = 1 ... D                    // initialize weights
2:  b ← 0                                           // initialize bias
3:  for iter = 1 ... MaxIter do                     ← epoch
4:      for all (x,y) ∈ D do
5:          a ← Σ_{d=1}^{D} w_d x_d + b             // compute activation for this example
6:          if ya ≤ 0 then
7:              w_d ← w_d + y x_d, for all d = 1 ... D   // update weights
8:              b ← b + y                           // update bias
9:          end if
10:     end for
11: end for
12: return w_0, w_1, ..., w_D, b
```

Algorithm 6 PERCEPTRONTEST(w_0, w_1, ..., w_D, b, $\hat{\mathbf{x}}$)

```
1:  a ← Σ_{d=1}^{D} w_d x̂_d + b                     // compute activation for the test example
2:  return SIGN(a)
```

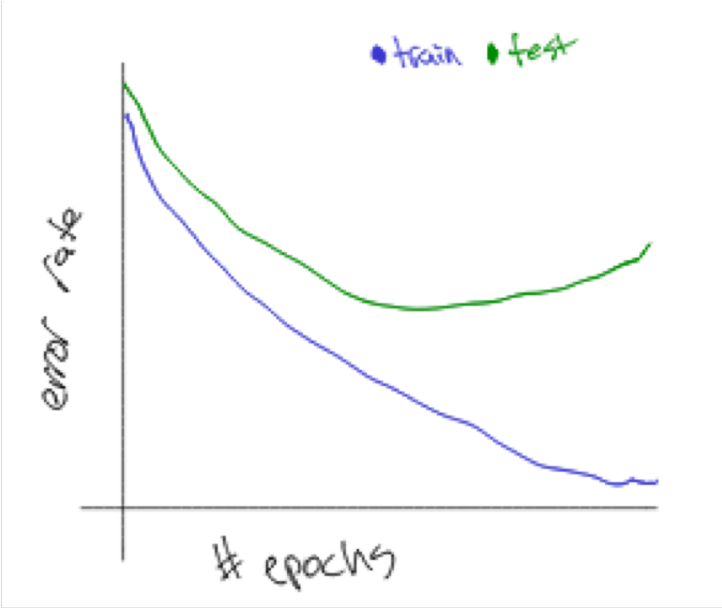
What the updates do?

- Moves the decision boundary in the direction of the training examples
- Intuition: adjust parameters so that they are better for the current example (i.e., if we see an example two times, the parameters should do a better job on the second example than on the first one).
- Example:
  - (x, +1): the current example, $(w_1, w_2, \ldots, w_D), b$: the current parameters, the parameters assign incorrect class in this case (i.e., $a < 0$)
  - New parameters: $(w_1', w_2', \ldots, w_D'), b'$, new activation: a'

$$
\begin{aligned}
a' &= \sum_{d=1}^{D} w_d' x_d + b' \\
&= \sum_{d=1}^{D} (w_d + x_d) x_d + (b + 1) \\
&= \sum_{d=1}^{D} w_d x_d + b + \sum_{d=1}^{D} x_d x_d + 1 \\
&= a + \sum_{d=1}^{D} x_d^2 + 1 \quad > \quad a
\end{aligned}
$$

How to iterate over data?

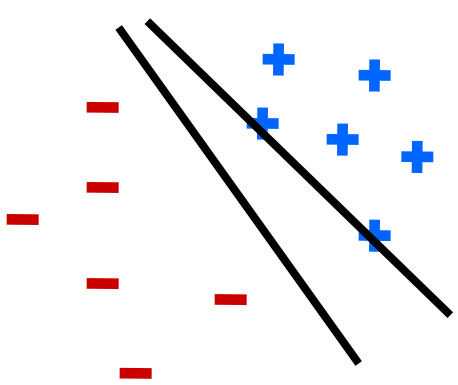
- IMPORTANT: permute the training dataset $\boldsymbol{D}$ at the beginning of each epoch
  - Why? Image having a fixed order of data in the iterations (i.e., positive example come first, followed by negative examples)
- How many epochs should we run (i.e., what's the good value for $MaxIter$)?
  - Many epochs tend to overfit (overtraining) while little epochs would underfit



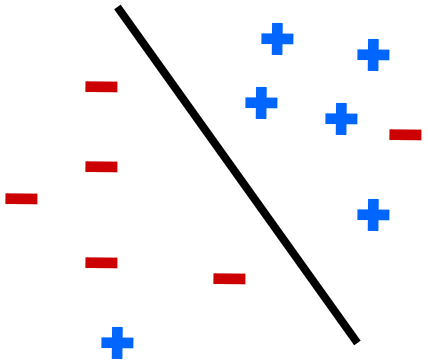
Does the perceptron converge? How long does it take?

- Convergence in perceptron: a classifier is considered as converged if it can correctly classify every training example
- Training data is linearly separable if we can find some hyperplane that puts positive examples on one side and negative examples on the other side (we say the hyperplane/parameter separates the data)

Separable



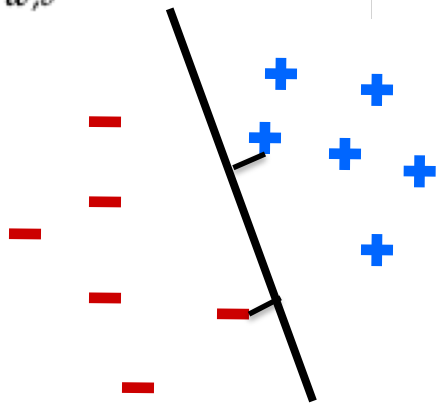
Non-Separable



Margins

$$margin(\mathbf{D}, \boldsymbol{w}, b) = \begin{cases} \min_{(\boldsymbol{x},y)\in\mathbf{D}} y(\boldsymbol{w}\cdot\boldsymbol{x}+b) & \text{if } \boldsymbol{w} \text{ separates } \mathbf{D} \\ -\infty & \text{otherwise} \end{cases}$$

$$margin(\mathbf{D}) = \sup_{\boldsymbol{w},b} margin(\mathbf{D}, \boldsymbol{w}, b)$$

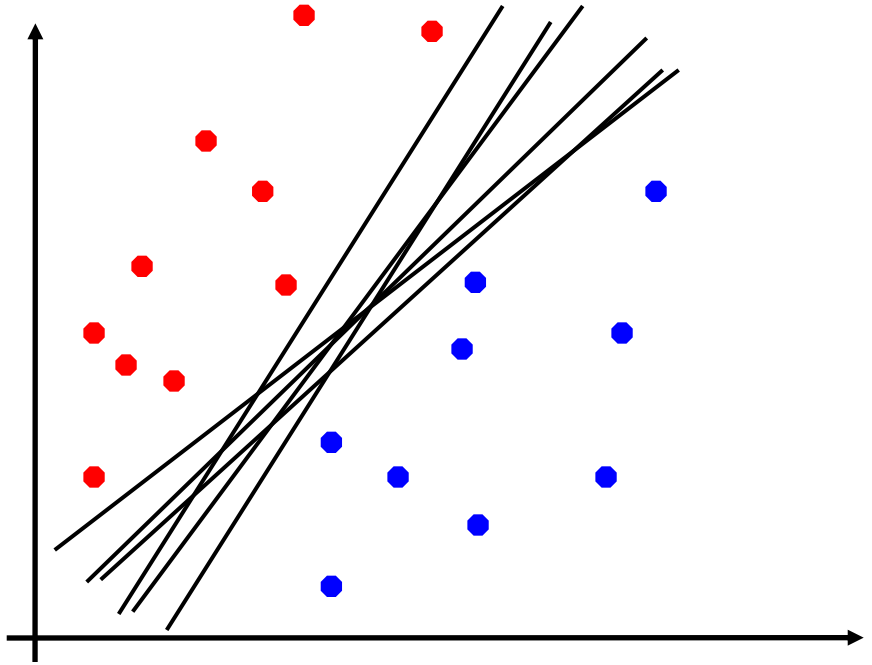


What does the margin tell you about your problem?

Convergence in perceptron

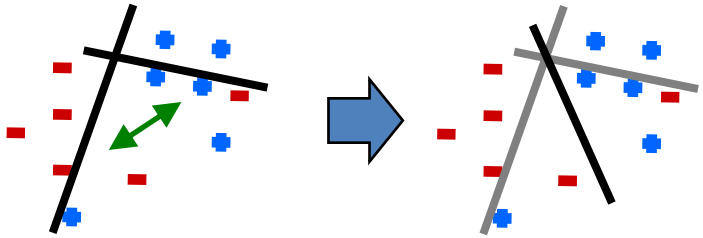
Theorem 2 (Perceptron Convergence Theorem). *Suppose the perceptron algorithm is run on a linearly separable data set* $\mathbf{D}$ *with margin* $\gamma > 0$. *Assume that* $||\boldsymbol{x}|| \leq 1$ *for all* $\boldsymbol{x} \in \mathbf{D}$. *Then the algorithm will converge after at most* $\frac{1}{\gamma^2}$ *updates.*

- What does it tell you?

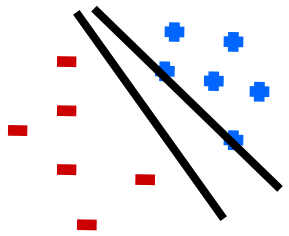


Problems with the Perceptron

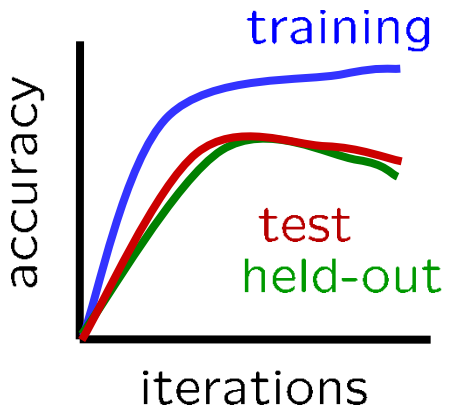
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



- Mediocre generalization: finds a "barely" separating solution



- Overtraining: test / validation accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



Improved Generalization

- Problem: The vanilla perceptron counts later points more than it counts earlier points (i.e., the last examples might have significant effect on the parameters)
- Fix: ensure that all the weight vectors encountered during training contribute to the prediction, the weight vectors that survive/appear for longer time should contribute more than those with shorter time.
- Voted perceptron:

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(k)} \text{sign}\left(\boldsymbol{w}^{(k)} \cdot \hat{\boldsymbol{x}} + b^{(k)}\right)\right)$$

  - But: need storage to store many weight vectors, prediction time is also slower
- Averaged perceptron:

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(k)} \left(\boldsymbol{w}^{(k)} \cdot \hat{\boldsymbol{x}} + b^{(k)}\right)\right)$$

$$\hat{y} = \text{sign}\left(\left(\sum_{k=1}^{K} c^{(k)} \boldsymbol{w}^{(k)}\right) \cdot \hat{\boldsymbol{x}} + \sum_{k=1}^{K} c^{(k)} b^{(k)}\right)$$

  - so, same prediction time as vanilla perceptron, good practical performance

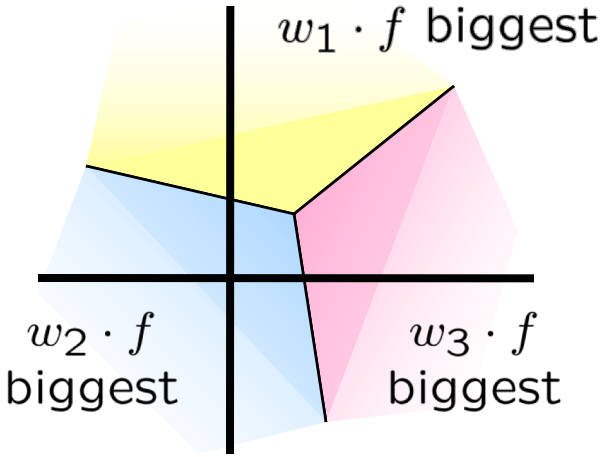
Algorithm 7 AVERAGEDPERCEPTRONTRAIN(**D**, *MaxIter*)

1: $\boldsymbol{w} \leftarrow \langle 0, 0, \ldots 0 \rangle$, $b \leftarrow 0$ // initialize weights and bias
2: $\boldsymbol{u} \leftarrow \langle 0, 0, \ldots 0 \rangle$, $\beta \leftarrow 0$ // initialize cached weights and bias
3: $c \leftarrow 1$ // initialize example counter to one
4: **for** $iter = 1 \ldots MaxIter$ **do**
5: **for all** $(\boldsymbol{x}, y) \in \mathbf{D}$ **do**
6: **if** $y(\boldsymbol{w} \cdot \boldsymbol{x} + b) \leq 0$ **then**
7: $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\,\boldsymbol{x}$ // update weights
8: $b \leftarrow b + y$ // update bias
9: $\boldsymbol{u} \leftarrow \boldsymbol{u} + y\,c\,\boldsymbol{x}$ // update cached weights
10: $\beta \leftarrow \beta + y\,c$ // update cached bias
11: **end if**
12: $c \leftarrow c + 1$ // increment counter regardless of update
13: **end for**
14: **end for**
15: **return** $\boldsymbol{w} - \frac{1}{c}\,\boldsymbol{u}, b - \frac{1}{c}\,\beta$ // return averaged weights and bias

Why?

Multiclass Decision Rule

- If we have more than two classes:
  - Have a weight vector for each class: w_y
  - Calculate an activation for each class



$$\text{activation}_w(x, y) = w_y \cdot f(x)$$

  - Highest activation wins

$$y = \arg\max_y \ (\text{activation}_w(x, y))$$

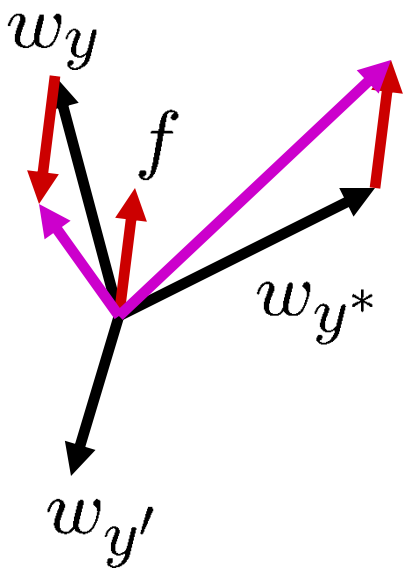
The Multi-class Perceptron Alg.

- Start with zero weights
- Iterate training examples
  - Classify with current weights

$$y = \arg\max_y \; w_y \cdot f(x)$$
$$= \arg\max_y \; \sum_i w_{y,i} \cdot f_i(x)$$

  - If correct, no change!
  - If wrong: lower score of wrong answer, raise score of right answer

$$w_y = w_y - f(x)$$
$$w_{y^*} = w_{y^*} + f(x)$$



Example

"win the vote"

"win the election"

"win the game"

w_{SPORTS}

```
BIAS  :
win   :
game  :
vote  :
the   :
...
```

$w_{POLITICS}$

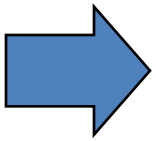
```
BIAS  :
win   :
game  :
vote  :
the   :
...
```

w_{TECH}

```
BIAS  :
win   :
game  :
vote  :
the   :
...
```

Example

"win the vote"



```
BIAS   :  1
win    :  1
game   :  0
vote   :  1
the    :  1
...
```

w_{SPORTS}

```
BIAS   : -2
win    :  4
game   :  4
vote   :  0
the    :  0
...
```

$w_{POLITICS}$

```
BIAS   :  1
win    :  2
game   :  0
vote   :  4
the    :  0
...
```

w_{TECH}

```
BIAS   :  2
win    :  0
game   :  2
vote   :  0
the    :  0
...
```

Which type wins?