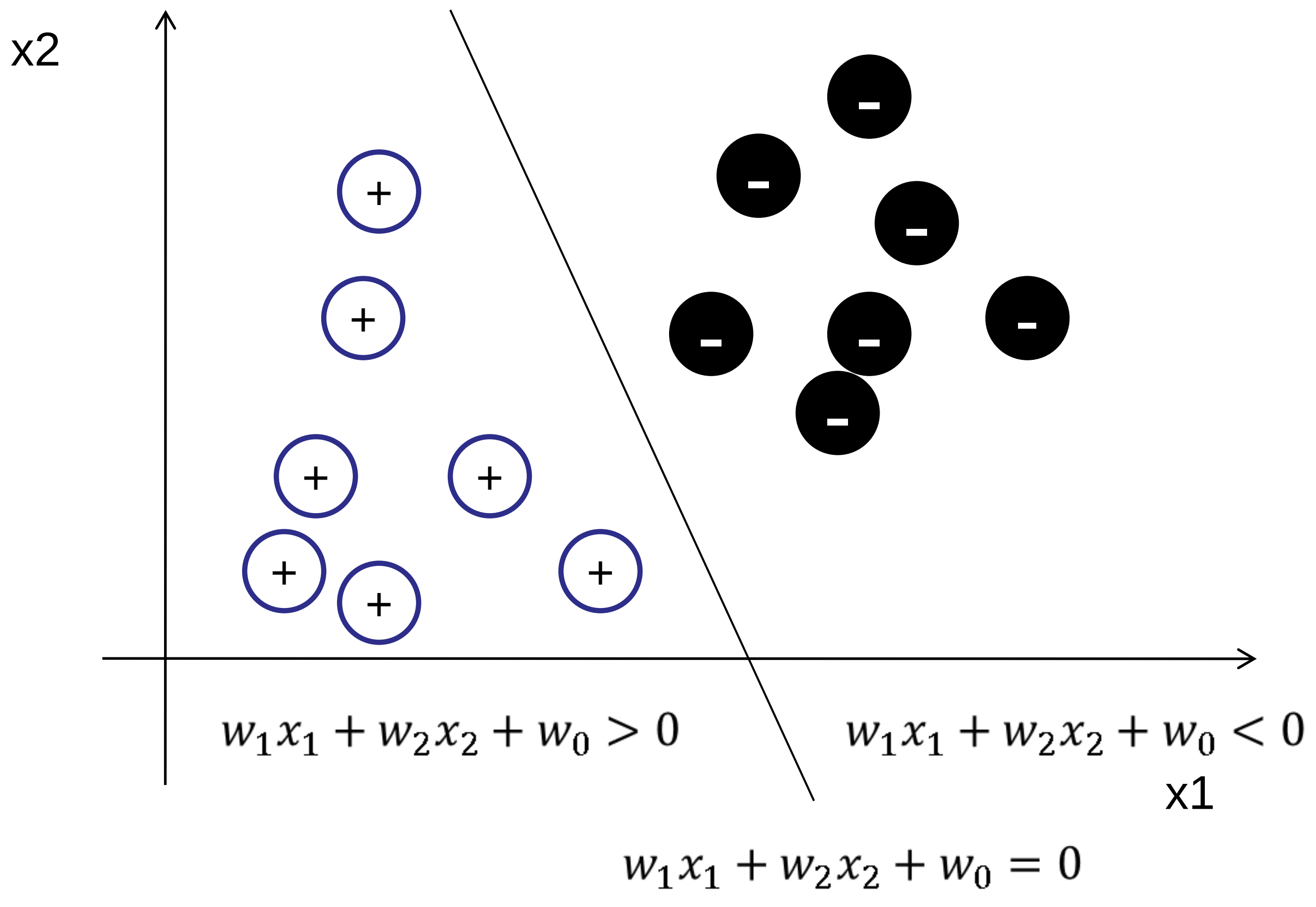
Linear classification models: Perceptron

Basic concepts:

The Perceptron algorithm
Perceptron loss/ hinge loss
Subgradient descent
Convergence proof of Perceptron
Concept of Margin
Voted and average Perceptrons

Linear Classifier


x2
+
-
$w_1x_1 + w_2x_2 + w_0 > 0$
$w_1x_1 + w_2x_2 + w_0 < 0$
x1
$w_1x_1 + w_2x_2 + w_0 = 0$

- We have discussed Logistic Regression
- LR learns $p(y|\mathbf{x})$:

$$P(y=1|\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x})}$$

which yields a linear decision boundary

$$\mathbf{w}^T\mathbf{x} = 0$$

- We will now look at a different paradigm for learning a linear decision boundary directly
  - Without a probabilistic model
  - Based on a loss function

Binary classification: General Setup

- Given a set of training examples $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$, where each $\mathbf{x}_i \in R^d$, $y_i \in \{-1,1\}$
- Learn a linear function

$$g(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_d x_d$$

  Given an example $\mathbf{x} = [x_1, \dots, x_d]^T$:

  - predict $y(\mathbf{x}) = 1$ if $g(\mathbf{x}, \mathbf{w}) \geq 0$
  - predict $y(\mathbf{x}) = -1$ otherwise
- Compactly the classifier can be represented as:
  - $y(\mathbf{x}) = \text{sign}(w_0 + w_1 x_1 + \cdots + w_d x_d) = \text{sign}(\mathbf{w}^T \mathbf{x})$

  where $\mathbf{w} = [w_0, w_1, \dots, w_d]^T$, and $\mathbf{x} = [1, x_1, \dots, x_d]^T$
- Goal: find a good $\mathbf{w}$ that minimizes some loss function $J(\mathbf{w})$

Loss function

$$J(\mathbf{w}) = \frac{1}{n}\sum_{m=1}^{n} L(g(\mathbf{w}, \mathbf{x}_m), y_m)$$

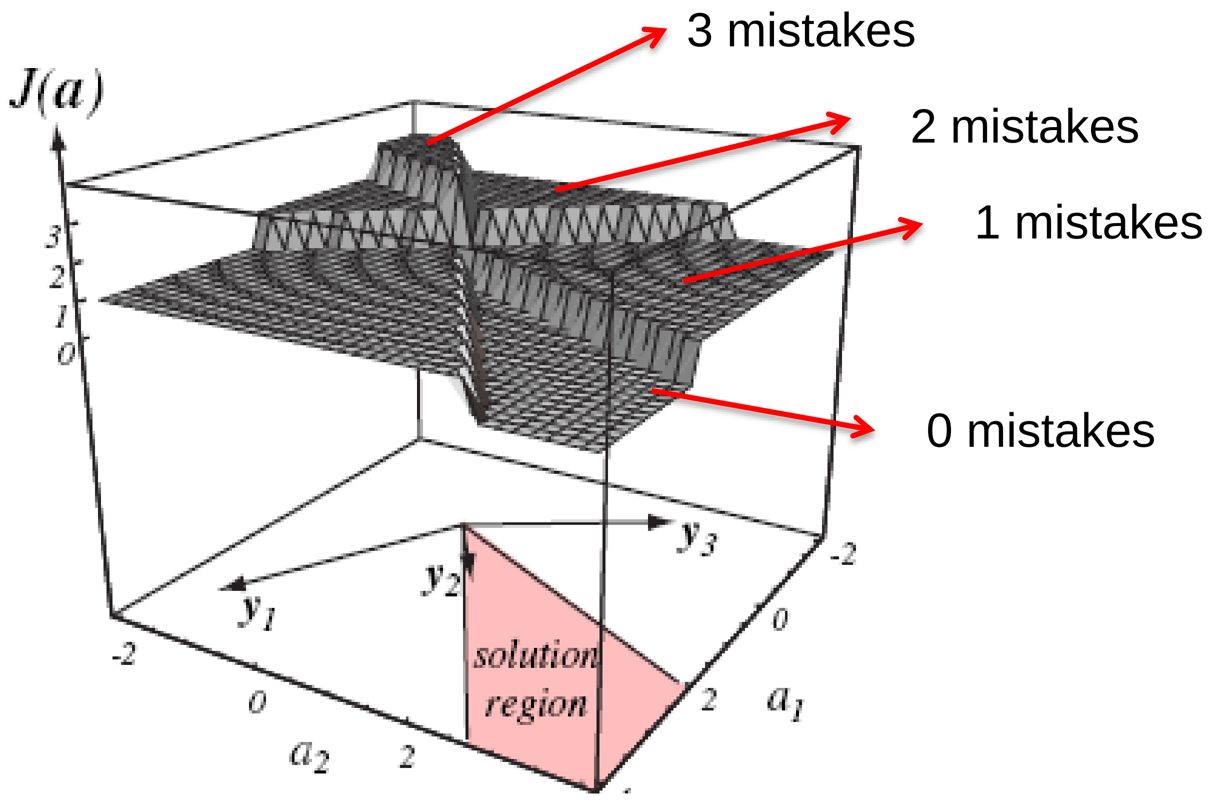
Where $L(g(\mathbf{w}, \mathbf{x}), y)$ is the loss of $g(\mathbf{w}, \mathbf{x})$ given its true label is y

0/1-loss:

$$L_{0/1}(g(\mathbf{w}, \mathbf{x}), y) = \begin{cases} 1 & \text{if } g(w, x) \text{ predicts wrong} \\ 0 & \text{otherwise} \end{cases}$$

This loss is conceptually aligned with our goal of maximizing accuracy, and minimizing error.

0/1 Loss counts the # of mistakes



Issue:

- Non convex – in general can be NP-hard to optimize
- Non-smooth – does not produce useful gradient since the surface of $J_{0/1}$ is **piece-wise flat**

Perceptron Loss

$$L_p(g(\mathbf{w}, \mathbf{x}), y) = \max(0, -y\mathbf{w}^T\mathbf{x})$$

- If prediction is correct, $-y\mathbf{w}^T\mathbf{x} < 0$,

$$L_p = \max(0, -y\mathbf{w}^T\mathbf{x}) = 0$$

- If prediction is incorrect, $-y\mathbf{w}^T\mathbf{x} > 0$,

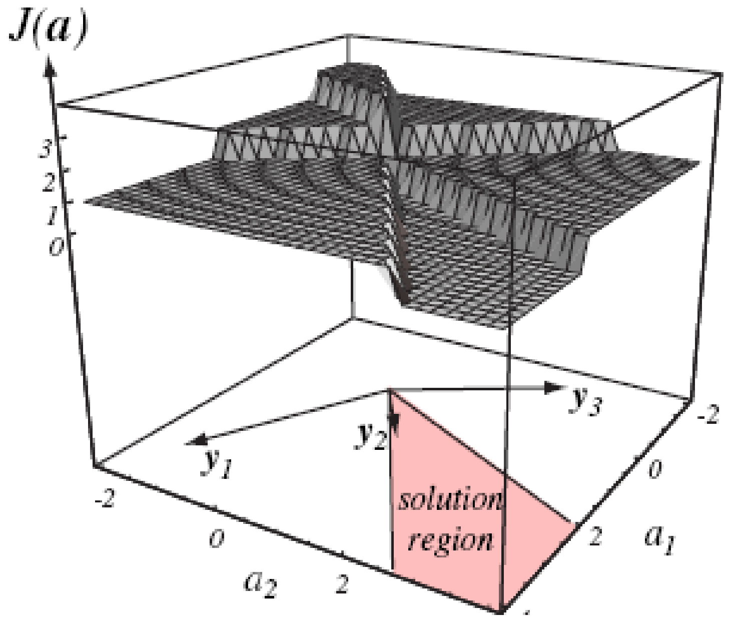
$$L_p = \max(0, -y\mathbf{w}^T\mathbf{x}) = -y\mathbf{w}^T\mathbf{x} > 0$$

- Loss is a linear function of the weights

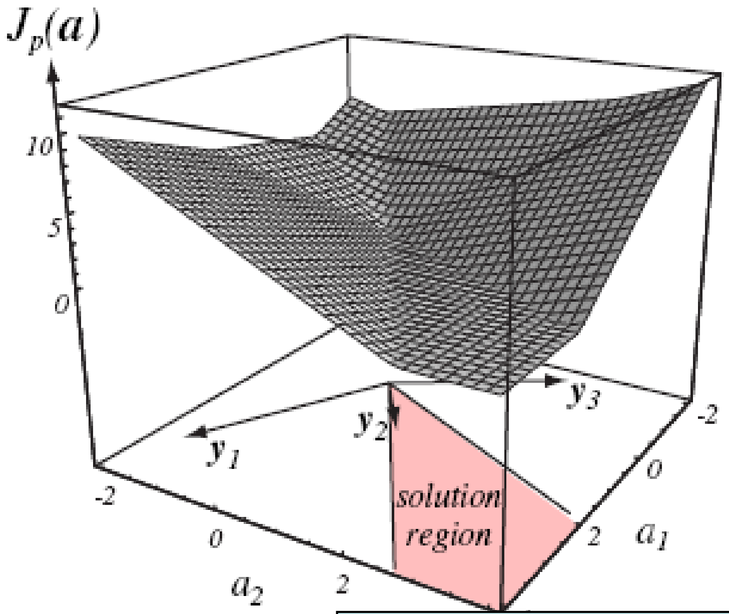
Perceptron Loss

$$J_p(w) = \frac{1}{n}\sum_{m=1}^{n} \max(0, -y_m \mathbf{w}^T \mathbf{x}_m)$$

- J_p is still non-smooth but piecewise linear
- Imagine if we drop a ball on this surface, it will with subgradient has a nice gradient leading to the solution region



0/1 loss: piecewise constant

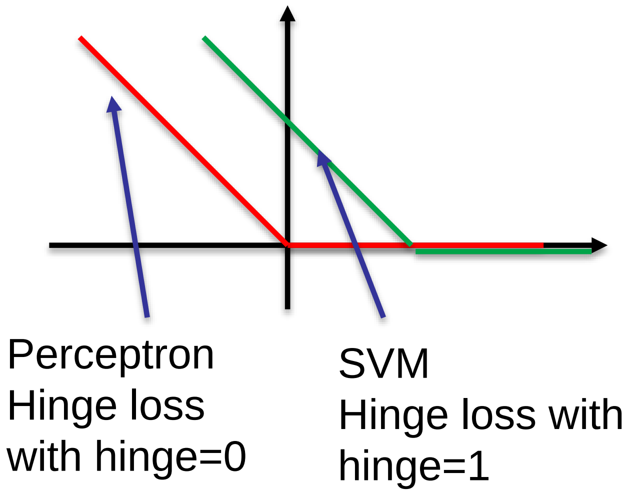


Perceptron criterion: piecewise linear

Stochastic Gradient Descent

- The objective function consists of a sum over data points---**Stochastic Gradient Descent** updates the parameter after observing each example

$$hinge(t) = \max(0, h - t)$$



$$J(\mathbf{w}) = \frac{1}{n}\sum_{m=1}^{n} \max(0, -y_m \mathbf{w}^T \mathbf{x}_m)$$

$$J_m(\mathbf{w}) = \max(0, -y_m \mathbf{w}^T \mathbf{x}_m)$$

Subgradient:

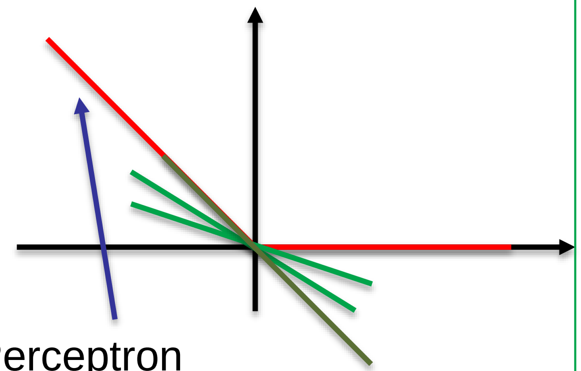
$$\nabla J_m = \begin{cases} 0 & \text{if } \; y_m \mathbf{w} \cdot \mathbf{x}_m > 0 \\ -y_m \mathbf{x}_m & \text{otherwise} \end{cases}$$

Perceptron Update Rule

After observing $(\mathbf{x}_m, y_m)$, if it is a mistake $\mathbf{w} \leftarrow \mathbf{w} + y_m \mathbf{x}_m$

Subgradient (subderivative)

$hinge(t) = \max(0, -t)$



Perceptron Hinge loss with hinge=0

Generalization of gradient to non-differentiable functions:

- Gradient of a function f can be viewed as defining a tangent line that touches f and lies below f

- Subgraident can be viewed as the set of lines that touches f and lies below f
  - At a differentiable point of the function it will be unique and the same as gradient
  - At non-smooth point, this will be a real set just pick one to work with
- Subgradient descent behaves similarly to gradient descent in terms of convergence

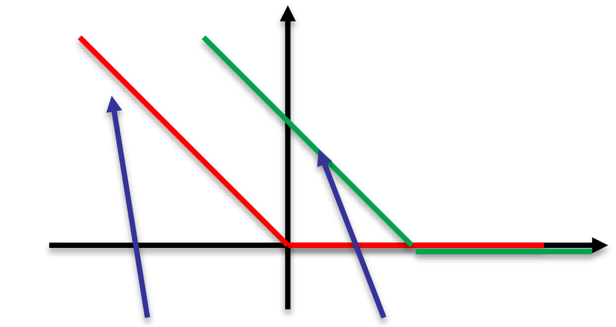
Stochastic Gradient Descent

- The objective function consists of a sum over data points--- **Stochastic Gradient Descent** updates the parameter after observing each example

$$hinge(t) = \max(0, h - t)$$

$$J(\mathbf{w}) = \frac{1}{n}\sum_{m=1}^{n} \max(0, -y_m \mathbf{w}^T \mathbf{x}_m)$$

$$J_m(\mathbf{w}) = \max(0, -y_m \mathbf{w}^T \mathbf{x}_m)$$



$hinge(t)$ is not smooth, we compute its subgradient

$$\nabla J_m = \begin{cases} 0 & \text{if } \ y_m \mathbf{w}\cdot\mathbf{x}_m > 0 \\ -y_m \mathbf{x}_m & \text{otherwise} \end{cases}$$

Perceptron Update Rule

After observing $(\mathbf{x}_m, y_m)$, if it is a mistake $\mathbf{w} \leftarrow \mathbf{w} + y_m \mathbf{x}_m$

The Perceptron Algorithm (online) (Stochastic gradient descent)

```
Let $\mathbf{w} \leftarrow (0,0,0,\ldots,0)$
Repeat until convergence
        for every training example $m = 1,\ldots,n$:
            $u_m \leftarrow \mathbf{w}^T \mathbf{x}_m$
            if $y_m \cdot u_m \leq 0$   $\mathbf{w} \leftarrow \mathbf{w} + y_m \mathbf{x}_m$
```

Online

- Look at one example at a time, update the model as soon as we make an error – as opposed to batch algorithms that update parameters after seeing the entire training set.

Error-driven

- We only update parameters/model if we make an error

Effect of the perceptron update

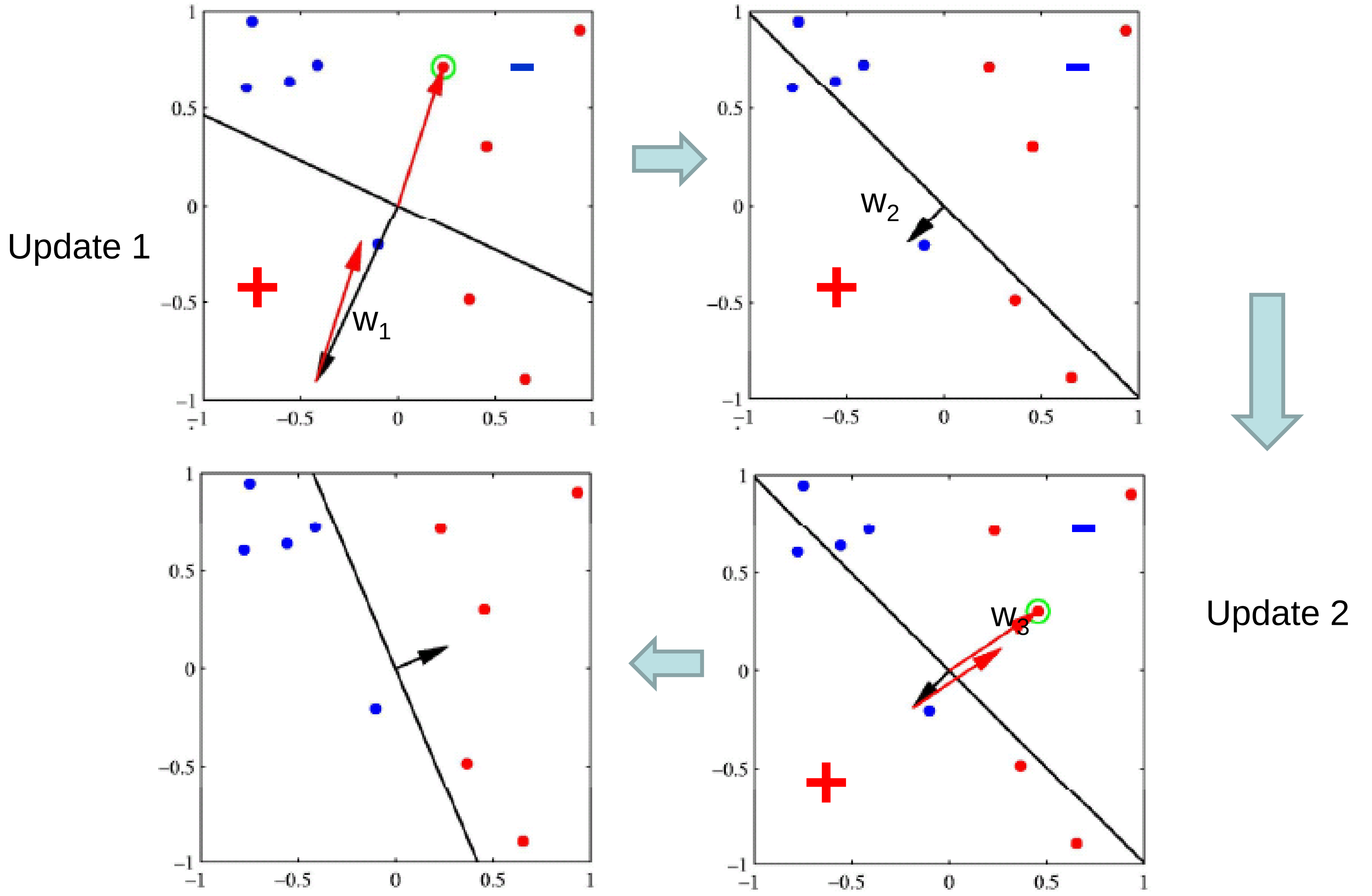
- Our current weight is w_t and it makes a mistaken on (x_i, y_i), i.e., $\mathrm{y_i} w_t^T x_i \leq 0$
- Perform update

$$w_{t+1} = w_t + y_i x_i$$

- What can we say about w_{t+1}'s performance on (x_i, y_i)?

See concept warehouse

When an error is made, moves the weight in a direction that corrects the error



Convergence Theorem

(Block, 1962, Novikoff, 1962)

Given training example sequence $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots (\mathbf{x}_N, y_N)$.

If $\forall i,\ |\mathbf{x}_i| \le D$, and $\exists\, \mathbf{u},\ |\mathbf{u}| = 1$ and $y_i \mathbf{u}^T \mathbf{x}_i \ge \gamma > 0$ for all i,

then the number of mistakes that the perceptron algorithm

makes is at most $(D/\gamma)^2$.

Note that | · | denotes the Euclidean norm of a vector.

Proof

Let $\mathbf{u}$ be a unit vector that achieves γ margin, i.e., $|\mathbf{u}| = 1$ and $\forall i, y_i \mathbf{u}^T \mathbf{x}_i \geq \gamma$

Let $\mathbf{x}_k$ be the kth mistake, we have $\mathbf{w}(k) = \mathbf{w}(k-1) + y_k \mathbf{x}_k$

Main idea: we want to show that the direction of $\mathbf{w}_k$ converges to $\mathbf{u}$**,** i.e., $\frac{\mathbf{u}^\mathrm{T}\mathbf{w}_k}{|\mathbf{u}||\mathbf{w}_k|}$ converges to 1. To show this, we work out two parts:

1. $\mathbf{u}^T\mathbf{w}_k$ grows bigger as k increases
2. $|\mathbf{w}_k|$ does not grow as fast

Part 1:

$$\mathbf{u}^T\mathbf{w}_k = \mathbf{u}^T(\mathbf{w}_{k-1} + y_k \boldsymbol{x}_k) = \mathbf{u}^T\mathbf{w}_{k-1} + y_k \mathbf{u}^T \mathbf{x}_k \geq \mathbf{u}^T\mathbf{w}_{k-1} + \gamma \geq k\gamma$$

Part 2:

$$\mathbf{w}_k^T\mathbf{w}_k = (\mathbf{w}_{k-1} + y_k \boldsymbol{x}_k)^T(\mathbf{w}_{k-1} + y_k \boldsymbol{x}_k)$$
$$= \mathbf{w}_{k-1}^T\mathbf{w}_{k-1} + 2y_k \mathbf{w}_{\boldsymbol{k-1}}^T \mathbf{x}_k + \mathbf{x}_k^T\mathbf{x}_k \leq \mathbf{w}_{k-1}^T\mathbf{w}_{k-1} + D^2 \leq kD^2$$

Putting it together: $\frac{\mathbf{u}^\mathrm{T}\mathbf{w}_k}{|\mathbf{u}||\mathbf{w}_k|} \geq \frac{k\gamma}{\sqrt{k}D}$, but this cannot exceed 1. so we mus

have $\frac{k\gamma}{\sqrt{k}D} \leq 1 \Rightarrow$

$$k \leq \left(\frac{D}{\gamma}\right)^2$$

Margin

- γ is referred to as the **margin**
  - Min distance from data points to the decision boundary
  - Bigger margin -> easier classification problems

- This concept will be utilized later by support vector machines

Implications of the convergence theorem

- See concept warehouse

Practical considerations for online perceptron

- The order of training examples matters!
  - Random is better
- Given large amounts of training data, early stopping can be used to avoid overfitting
- Simple modification can significantly improve performance
  - Voted perceptron and average perceptron

Voted Perceptron

- Keep intermediate hypotheses and have them vote [Freund and Schapire 1998]

```
Let $\mathbf{w} \leftarrow (0,0,0,...,0)$
$c_0 = 0,\ n = 0$
Repeat for T times
    for each training example $i$:
        $u_i \leftarrow \mathbf{w}^T \mathbf{x}_i$
        if $y_i u_i \le 0$
            $\mathbf{w}_{n+1} \leftarrow \mathbf{w}_n + y_i \mathbf{x}_i$
            $n = n + 1$
            $c_n = 0$
        else $c_n = c_n + 1$
```

The output will be a collection of linear separators $\mathbf{w}_0\ \mathbf{w}_1, \ldots, \mathbf{w}_M$ along with their survival time $c_0, c_1, \ldots, c_M$

The c's can be viewed as measures of the reliability of the $\mathbf{w}$'s

For classification, take a weighted vote among all separators:

$$\hat{y} = \text{sign}\{\sum_{n=0}^{N} c_n \text{sign}(\mathbf{w}_n^T \mathbf{x})\}$$

Average Perceptron

- Voted perceptron requires storing all intermittent weights
  - Large memory consumption
  - Slow prediction time
- Average perceptron

$$\hat{y} = \text{sign}\{(\sum_{n=0}^{N} c_n \mathbf{w}_n^T)\mathbf{x}\}$$

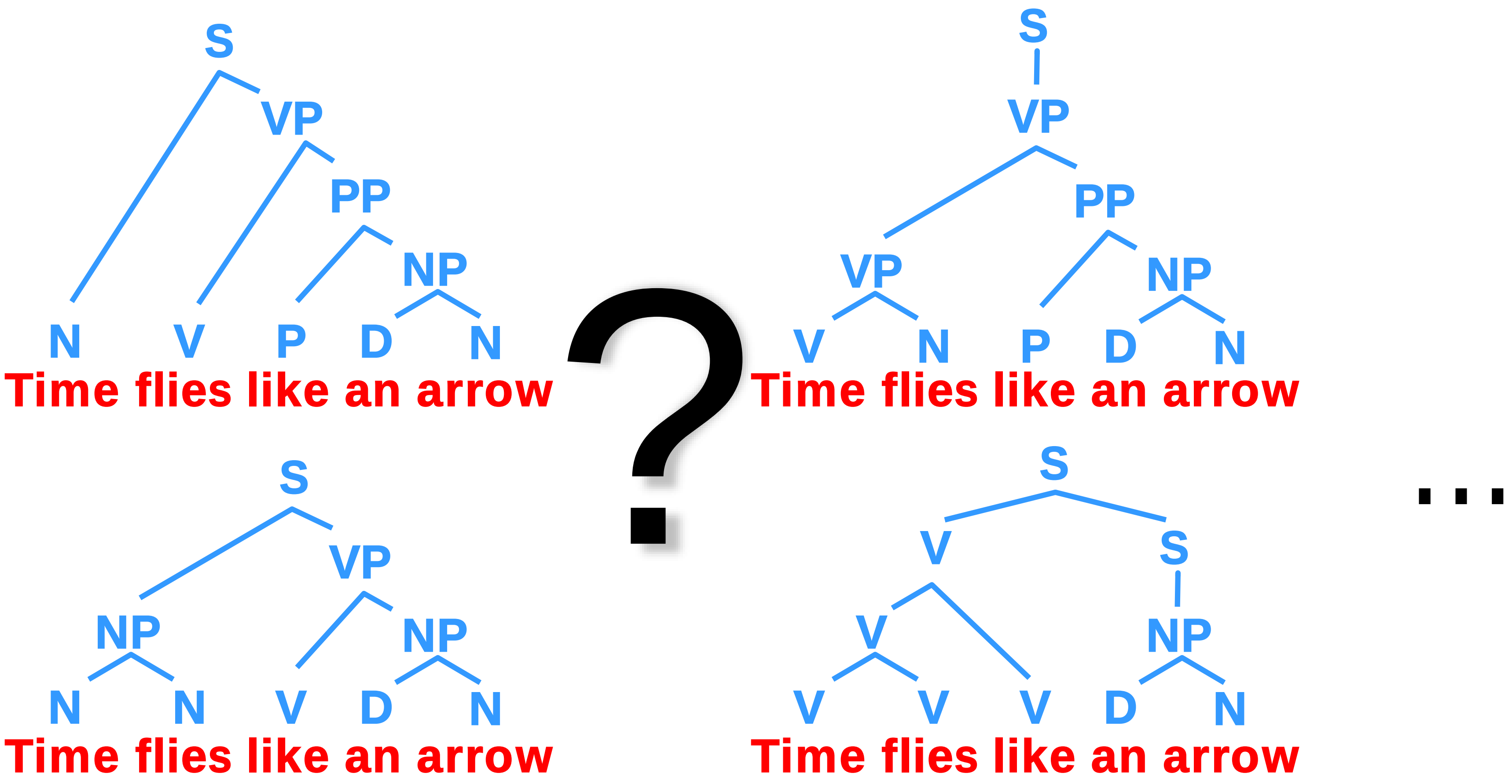
  - Take the weighted average of all the intermittent weights
  - Can be implemented by maintaining an running average, no need to store all weights
  - Fast prediction time

Final Discussion on Perceptron

- Learns $\hat{y} = f(\mathbf{x})$ directly – a **discriminative** method
- Performs stochastic (sub-)gradient descent to optimize the perceptron loss (also called hinge loss with hinge=0, will see more hinge loss later in SVM)
- Guaranteed to converge in finite steps if the data is linearly separable
  - # of updates is inversely proportional to the **margin** of the optimal decision boundary
  - Guarantees convergence but not necessarily to the maximum margin separator – again we will address this later in SVM
- Voted and average perceptrons provide significant performance improvement in practice

Beyond the Basic Perceptron

Structured Prediction with Perceptrons



Based on Jason Eisner's notes

A general problem

- Given some input x
  - An email, a sentence …
- Consider a set of candidate outputs y
  - Classifications for x (small number: often just 2)
  - Taggings of x (exponentially many)
  - Parses of x (exponentially many)
  - Translations of x (exponentially many)
  - …

Structured prediction

- Want to find the "best" y, given x

Based on Jason Eisner's notes

Scoring by Linear Models

- Given some input x
- Consider a set of candidate outputs y
- Define a scoring function score(x,y)

Linear function: A sum of feature weights (you pick the features!)

Weight of feature k
(learned or set by hand)

$$\text{score}(x, y) = \sum_{\boxed{k}} \boxed{\theta_k} \boxed{f_k(x, y)}$$

Ranges over all features,
e.g., k=5 (numbered features)
or k="see Det Noun" (named features)

Whether (x,y) has feature k(0 or 1)
Or **how many times** it fires (≥ 0)
Or **how strongly** it fires (real #)

- Choose y that maximizes score(x,y)

Based on Jason Eisner's notes

Scoring by Linear Models

- Given some input x
- Consider a set of candidate outputs y
- Define a scoring function score(x,y)

Linear function: A sum of feature weights (you pick the features!)

(learned or set by hand)

$$\text{score}(x, y) = \boxed{\vec{\theta}} \cdot \vec{f}(x, y)$$

This linear decision rule is sometimes called a "perceptron."
It's a "structured perceptron" if it does structured prediction
(number of y candidates is unbounded, e.g., grows with |x|).

- Choose y that maximizes score(x,y)

Based on Jason Eisner's notes

Perceptron Training Algorithm

- initialize θ (usually to the zero vector)
- repeat:
  - Pick a training example (x,y)
  - Model predicts y^* that maximizes score(x,y^*)
  - Update weights by a step of size $\varepsilon > 0$:

$$\theta = \theta + \varepsilon \cdot (f(x,y) - f(x,y^*))$$

If model prediction was wrong ($y \neq y^*$), then we must have score$(x,y) \leq$ score(x,y^*) instead of > as we want.

Equivalently, $\theta \cdot f(x,y) \leq \theta \cdot f(x,y^*)$

Equivalently, $\theta \cdot (f(x,y) - f(x,y^*)) \leq 0$ but we want it positive.

Our update <u>increases</u> it (by $\varepsilon \cdot \| f(x,y) - f(x,y^*) \|^2 \geq 0$)

Based on Jason Eisner's notes

Perceptron for Structured Prediction

- What we see here is the same as the regular perceptron
- Similar convergence guarantee
- The challenge is the inference part
  - Finding the y that maximizes the score for given x
  - Cannot resort to brute-force enumeration
  - Much research goes into
    - How to devise proper features and efficient algorithms for inference
    - How to perform approximate inference
    - How to learn when inference is approximate