Statistical Techniques in Robotics (16-831, S22) Lecture #06 (Monday, February 7)

Online Convex Optimization (Convexity, FTL)

Lecturer: Kris Kitani *Scribes: Jinkun Liu, Thomas Xu*

1 Review

In the last lecture, on the topic of Online Linear Classification, we formalized the linear classification problem and analyzed two approaches to learning an online linear classifier: the Perceptron algorithm and the Winnow algorithm. As a quick recap, the Perceptron algorithm uses additive updates whereas the Winnow algorithm uses multiplicative updates to the model weights each timestep. After the review section, we cover online convex optimization, which is a generalization of online learning problems like online classification and online experts, and introduce the Follow the Leader algorithm which is a general method for online optimization problems.

1.1 Online Linear Classification

Online linear classification is a form of supervised learning; the learner trains on samples that are received online and iteratively learns to classify them correctly. There are many real-world applications of these types of problems, such as a quality assurance system to identify faulty products in an assembly line, or an agricultural robot learning to differentiate different kinds of fruit.

The inputs to online linear classification problems are typically denoted by $\boldsymbol{x}^{(t)}$, where the superscript indicates the timestep. These inputs are vectors, whose elements or "features" may be either continuous (e.g. weight, size) or binary (e.g. stamped, purple). The outputs of the learner at each timestep are its classification predictions, which are typically denoted by $\hat{y}^{(t)}$. Similarly, the true label or correct classification is denoted by $y^{(t)}$ or y^t.

Linear classification of the samples is achieved using hyperplanes, which are defined such that "positive" samples satisfy the following inequality:

$$\boldsymbol{w}^{(t)} \cdot \boldsymbol{x}^{(t)} \geq N \qquad \boldsymbol{w}^{(t)} \in \mathbb{R}^N, \boldsymbol{x}^{(t)} \in \mathbb{R}_+^N \tag{1}$$

Here, $w^{(t)}$ indicates the parameters of the hyperplane. N is a threshold value that can be set arbitrarily. If a sample does not satisfy the above inequality, then it will be classified as "negative." In other words, the prediction rule of the hyperplane is given by:

$$\hat{y}^{(t)} = \text{sign}(\boldsymbol{w}^{(t)} \cdot \boldsymbol{x}^{(t)} - N) \tag{2}$$

1.2 Perceptron Algorithm

The Perceptron algorithm is an online linear classification learning algorithm that makes predictions using the sign of a dot product between its learned weights and the received feature vector. It learns the weights after making mistakes using additive updates based on the true labels received.

Algorithm 1 Perceptron algorithm

```
1: $\mathbf{w}^{(1)} \leftarrow 0$                                          ▷ Initialize weights
2: for $t = 1, \cdots, T$ do
3:     RECEIVE $(\mathbf{x}^{(t)} \in \mathbb{R}^N)$                      ▷ Receive feature vector
4:     $\hat{y}^{(t)} = \text{sign}\Big(\langle \mathbf{w}^{(t)}, \mathbf{x}^{(t)} \rangle\Big)$          ▷ Predict label
5:     RECEIVE $(y^{(t)} \in \{-1, 1\})$                                ▷ Receive true label
6:     $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y^{(t)} \cdot \mathbf{x}^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$    ▷ Update weights
7: end for
```

One benefit of the Perceptron algorithm is speed, as its prediction and weight update entails only a dot product and an addition respectively. However, it is not a large margin classifier as it only uses sign to make decisions, and does not use any notion of associated distance or margin. Furthermore, the Perceptron algorithm will not converge and stop making mistakes if the data isn't linearly separable, since its decision boundary is a hyperplane. For linearly separable data, the Perceptron algorithm's mistake bound [2] is

$$M \leq \frac{R^2}{\gamma^2} \tag{3}$$

Here, $R = \max_t \|x^{(t)}\|$ is the infinity norm of data examples and $\gamma = \min_t y_t \langle \mathbf{w}^\star, \mathbf{x}^{(t)} \rangle$ is the data's margin of separability.

1.3 Winnow Algorithm (Introduction)

The Winnow algorithm is an online linear classification algorithm that operates on binary features as inputs. The name "winnow" originally came from the term "winnowing," which is the act of shaking or blowing air through a bundle of grain to remove unnecessary parts ("chaff" or "husks"). The key assumption of the Winnow algorithm is that only a certain subset of the features are relevant for classification. Specifically, the algorithm assumes that a perfect hypothesis $h^\star$ is a disjunctive Boolean function [1] of the form

$$h^\star(\boldsymbol{x}) = x_{n,1} \ \vee \cdots \vee x_{n,K} \tag{4}$$

where $\boldsymbol{x}$ is a vector of binary features and K is the number of relevant features. The Winnow algorithm uses the following equation to predict labels:

$$\hat{y}^{(t)} = \mathbf{1}\left[\langle \boldsymbol{w}^{(t)}, \boldsymbol{x}^{(t)} \rangle > \Theta\right] \tag{5}$$

Here, Θ is the threshold that is used to define the decision boundary (hyperplane). The details of Winnow algorithm will be discussed in Section 2.1.

2 Summary

2.1 Winnow Algorithm (Weight Update and Mistake Bound)

As mentioned in the review section, the Winnow algorithm is an approach that can be used for online linear classification problems. It is similar to the Perceptron algorithm, and also classifies examples using a hyperplane, but has some notable differences. For instance, the Winnow algorithm uses multiplicative instead additive weight updates, and does not always use a threshold of zero.

The input to the Winnow algorithm is denoted by $\boldsymbol{x}$, which is a vector of binary features:

$$\boldsymbol{x} = \{x_1, \ldots, x_N\}, \quad x_n \in \{0, 1\} \tag{6}$$

As discussed in the review section, the Winnow algorithm assumes that the perfect hypothesis (h^*) is a disjunctive Boolean function. Note that the weights (w_n) in the Winnow algorithm are always non-negative, and that the weights are initialized to 1 (instead of 0, as in the Perceptron algorithm).

Algorithm 2 Winnow algorithm

```
1: $\mathbf{w}^{(1)} \leftarrow \{1, ..., 1\}$                                  ▷ Initialize weights
2: for $t = 1, \cdots, T$ do
3:     RECEIVE ($\mathbf{x}^{(t)} \in \{0,1\}^N$)                      ▷ Receive feature vector
4:     $\hat{y}^{(t)} = \mathbf{1}[\langle \mathbf{w}^{(t)}, \mathbf{x}^{(t)} \rangle > \Theta]$                   ▷ Predict label
5:     RECEIVE ($y^{(t)} \in \{0,1\}$)                           ▷ Receive true label
6:     $w_n^{(t+1)} \leftarrow w_n^{(t)}(1+\beta)^{(y^{(t)}-\hat{y}^{(t)}) \cdot x_n^{(t)}}$           ▷ Update weights
7: end for
```

As seen in the above algorithm (line 6), the Winnow weight update rule is given by:

$$w_n^{(t+1)} \leftarrow w_n^{(t)}(1+\beta)^{(y^{(t)}-\hat{y}^{(t)}) \cdot x_n^{(t)}} \tag{7}$$

Here, β is a constant parameter that affects the magnitude of weight updates. A typical value [1] is $\beta = 1$. Note that the $(1+\beta)$ term may be given in different forms. For example, other papers may replace that term with e^η or α [1].

From the expression in the exponential in the update rule, we can observe that weight updates are only performed when mistakes are made ($y^{(t)} \neq \hat{y}^{(t)}$), and only for weights corresponding to the positive input features ($x_n = 1$). Additionally, the direction of weight updates depend on the type of mistakes made. This is summarized in Table 1 below.

Table 1: Winnow updates weights depending on the type of mistake.

Label ($y^{(t)}$)	Prediction ($\hat{y}^{(t)}$)	Gain ($y^{(t)} - \hat{y}^{(t)}$)	Type of Mistake	Weight Update	Update Name [1]
0	1	-1	Mistake on Negative	Decrease by $\frac{1}{(1+\beta)}$	Demotion Step
1	0	1	Mistake on Positive	Increase by $(1+\beta)$	Promotion Step

Mistake Bound Let M be the total number of mistakes made by the learner using the Winnow algorithm. Assuming $\beta = 1$, then M is upper-bounded as:

$$M < 2 + 3k(\log_2 N + 1) \tag{8}$$

where N is the total number of input features and k is the number of relevant features. The derivation of this mistake bound can be found in the Appendix.

2.2 Online Convex Optimization

Online convex optimization serves as a generalized framework of many online tasks, including the online learning problem covered in previous lectures. It was first formalized in 2003 by Zinkevich [4]. In summary, an online convex optimizer interacts with nature by producing predictions using its parameters $\boldsymbol{w}^{(t)} \in \mathcal{S}$, and receiving loss function $l^{(t)}$ from nature that it will use to update its parameters. Here, $\mathcal{S}$ must be a convex set and $l^{(t)}$ must be a convex function; we will cover these concepts later in this section. The diagram in Figure 1 demonstrates this framework:

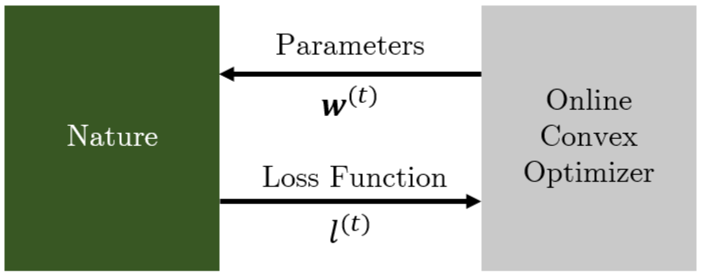

Figure 1: Online convex optimizer's interaction with nature.

There are two ways to interpret how online convex optimization is a generalization of online learning:

1. An online learner with convex loss can be viewed as part of the nature that the online convex optimizer interacts with.
2. The online convex optimizer can be viewed as the update method for an online learner with convex loss.

Figure 2 below illustrates both interpretations.

Convex Set A set $\mathcal{S}$ is considered a convex set if for all $\boldsymbol{w}, \boldsymbol{v} \in \mathcal{S}$:

$$\alpha \boldsymbol{w} + (1 - \alpha)\boldsymbol{v} \in \mathcal{S} \qquad \forall \alpha \in [0, 1] \tag{9}$$

In other words, the points that lie linearly between $\boldsymbol{w}$ and $\boldsymbol{v}$ must all also be inside $\mathcal{S}$. The α term indicates how far along a given point is between $\boldsymbol{w}$ and $\boldsymbol{v}$ (e.g. $\alpha = 1$ means the point is at $\boldsymbol{w}$, and $\alpha = 0.5$ means the point is halfway between $\boldsymbol{w}$ and $\boldsymbol{v}$). Figure 3 provides numerical and geometrical examples of convex and non-convex sets.

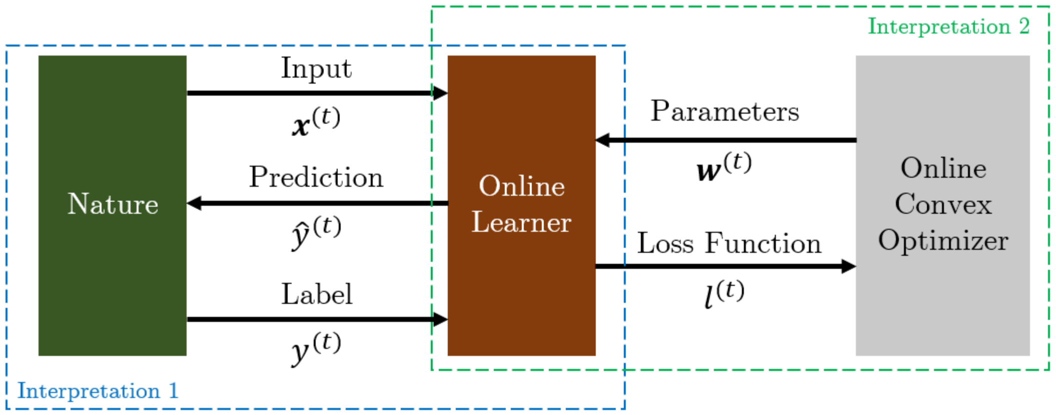

Figure 2: Two ways to interpret online convex optimization as a generalization of online learning.

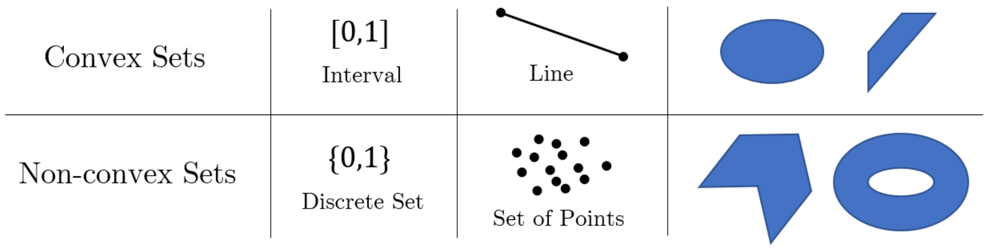

Figure 3: Examples of convex and non-convex sets.

Convex Function A function $f : \mathcal{S} \to \mathbb{R}$ is considered a convex function if for all $\boldsymbol{w}, \boldsymbol{v} \in \mathcal{S}$:

$$f(\alpha \boldsymbol{w} + (1-\alpha)\boldsymbol{v}) \leq \alpha f(\boldsymbol{w}) + (1-\alpha) f(\boldsymbol{v}) \qquad \forall \alpha \in [0,1] \tag{10}$$

The interpretation of this definition is as follows: the right-hand side of the inequality forms a straight line between two points $f(\boldsymbol{w})$ and $f(\boldsymbol{v})$. All points on this line must be above or equal to the function itself across the same interval, for the function to be convex. Figure 4 illustrates this interpretation. Additionally, a function is **strictly convex** if the inequality in Equation 10 holds strictly (i.e., $<$ rather than $\leq$) for all $\alpha \in (0,1)$ and $\boldsymbol{w} \neq \boldsymbol{v}$.

Lipschitz Continuity A function $f(\cdot)$ is called “L-Lipschitz” over a set $\mathcal{S}$ with respect to a norm $||\cdot||$ if:

$$|f(\boldsymbol{u}) - f(\boldsymbol{w})| \leq L||\boldsymbol{u} - \boldsymbol{w}|| \qquad \forall \boldsymbol{u}, \boldsymbol{w} \in \mathcal{S} \tag{11}$$

Here, L is the “Lipschitz constant.” Any standard norm or distance function can be used here, such as the L-2 norm. This expression essentially sets a bound on the function’s rate of change: the left side is the function value difference or “rise” between two points ($\boldsymbol{u}$ and $\boldsymbol{v}$), and the right side (without the Lipschitz constant) is the input value difference or “run.” Also, note that this expression can also be used to show that compositions of Lipschitz functions are also Lipschitz functions [3]: if $f(x) = g(h(x))$, where $g(x)$ is L_1-Lipschitz and $h(x)$ is L_2-Lipschitz, then $f(x)$ is $L_1 L_2$-Lipschitz.

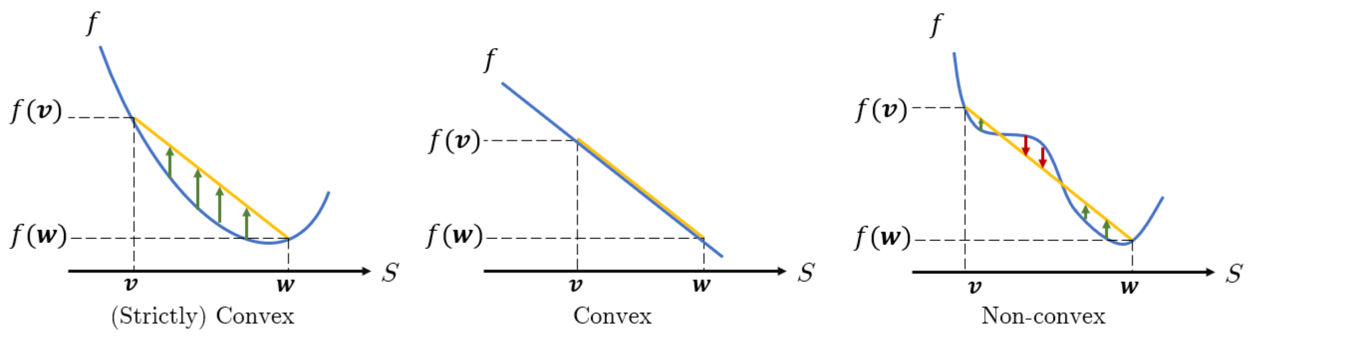

Figure 4: Examples of convex and non-convex functions. The function is convex if the yellow line connecting $f(\boldsymbol{w})$ and $f(\boldsymbol{v})$ is entirely above or on the function itself, for all $\boldsymbol{w}, \boldsymbol{v} \in \mathcal{S}$.

As discussed earlier, performing convex optimization requires both a convex function and a convex solution space. For learning problems, this typically means that the hypothesis class must be a convex set with elements with bounded magnitudes and the loss function must be a convex function. Note that in order for the optimization to actually be solved, the function must also be L-Lipschitz. Together, these conditions are sufficient to form a "Convex-Lipschitz-Bounded" learning problem.

Another similar class of learning problems that can be solved with convex optimization is "Convex-Smooth-Bounded" learning problems, where the function is "convex-smooth" instead (see Appendix).

Convexification Certain methods can be employed to convert non-convex problems into convex ones, typically by converting a non-convex loss function into a convex one.

One method is convexification by **randomization**. By adding randomness to an algorithm, its loss function may be changed from a non-convex one to a convex one. For example, the Weighted Majority Algorithm (WMA) has a loss function given by:

$$l^{(t)} = \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}] \tag{12}$$

This loss function forms the shape of a step function when graphed, and is not convex. However, if we add randomization to WMA and make it the Randomized Weighted Majority Algorithm (RWMA), the random sampling changes the zero-one loss function into an expectation:

$$l^{(t)} = E_{\boldsymbol{p}}[\mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]] = \sum_{p_n} p_n \cdot \mathbf{1}[y^{(t)} \neq \hat{y}_n^{(t)}] \tag{13}$$

The probability term allows this new loss function to take on any values between 0 and 1, making it a linear and convex function.

Another method is convexification by **surrogate loss**. This method replaces the original non-convex loss function with a surrogate loss function that:

- Upper-bounds the original loss function
- Is a convex function

This method is viable because minimizing the surrogate loss is equivalent to minimizing an upper bound on the original loss function. An example surrogate loss for WMA can be derived as below:

$$\begin{aligned}
l^{(t)} &= \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}] && (14)\\
&= \mathbf{1}[-y^{(t)}\hat{y}^{(t)} > 0] && (15)\\
&= \mathbf{1}[1 - y^{(t)}\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)}\rangle > 1] && (16)\\
&\leq \max\big[0, 1 - y^{(t)}\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)}\rangle > 1\big] && (17)\\
\tilde{l}^{(t)} &= \max\big[0, 1 - y^{(t)}\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)}\rangle > 1\big] && (18)
\end{aligned}$$

The resulting surrogate loss $\tilde{l}^{(t)}$ is known as the hinge loss, and as seen in Figure 5 below it both upper-bounds the original zero-one WMA loss and is convex. This surrogate loss function thus has transformed this problem into a convex problem.

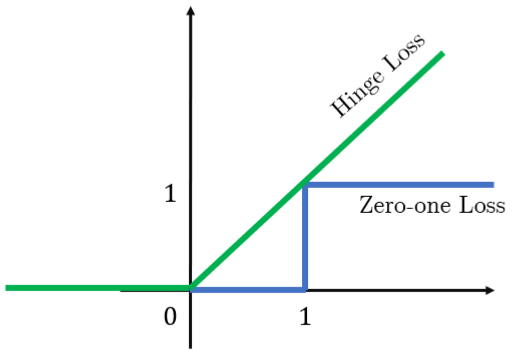

Figure 5: The non-convex zero-one loss for WMA and the convex hinge loss.

2.3 Follow the Leader

Follow the Leader (FTL) is a generic algorithm for online (not necessarily convex) optimization problems. Its main idea is that the learner should go by the best choice in the game so far. The general pseudocode for FTL is as seen below:

Algorithm 3 Follow the Leader

```
1: for $t = 1, 2, \cdots, T$ do
2:     $\boldsymbol{w}^{(t)} = \arg\min_{\boldsymbol{w} \in W} \sum_{i=1}^{t-1} f^{(i)}(\boldsymbol{w})$        ▷ Update parameter
3:     RECEIVE($f^{(T)} : W \to \mathbb{R}$)        ▷ Receive loss
4: end for
```

Assuming a **linear** loss of the form $f^{(t)}(w) = wz^{(t)}$ and solution space $w \in S = [-1, 1]$, the FTL algorithm becomes:

Algorithm 4 Follow the Leader (Linear Loss)

```
1: for $t = 1, 2, \cdots, T$ do
2:     $w^{(t)} = \text{argmin}_{w \in S}\Big(\Sigma_{i=1}^{t-1} z^{(i)}\Big) \cdot w$        ▷ Update parameter
3:     Receive $(f^{(t)}(w) = wz^{(t)})$        ▷ Receive loss
4: end for
```

Note that in some scenarios, FTL with linear loss can have very poor performance. For example, consider the following sequence of observations in an adversarial environment:

$$z^{(1)}, z^{(2)}, z^{(3)}, z^{(4)}, z^{(5)}, \cdots = -0.5, 1.0, -1.0, 1.0, -1.0, \ldots \tag{19}$$

Consequently, the sequence of sum of observations is:

$$\Sigma_{i=1}^{0} z^{(i)}, \Sigma_{i=1}^{1} z^{(i)}, \Sigma_{i=1}^{2} z^{(i)}, \Sigma_{i=1}^{3} z^{(i)}, \Sigma_{i=1}^{4} z^{(i)}, \cdots = 0, -0.5, 0.5, -0.5, 0.5, \ldots \tag{20}$$

In this case, the sequence of weights that would be chosen by the FTL algorithm in order to optimize linear loss are:

$$w^{(1)}, w^{(2)}, w^{(3)}, w^{(4)}, w^{(5)}, \cdots = 0.0, 1.0, -1.0, 1.0, -1.0, \ldots \tag{21}$$

Then, the loss that would be incurred at each timestep would be:

$$f^{(1)}, f^{(2)}, f^{(3)}, f^{(4)}, f^{(5)}, \cdots = 0.0, 1.0, 1.0, 1.0, 1.0, \ldots \tag{22}$$

As a result, the total loss of the FTL learner is

$$L = \Sigma_{t=2}^{T} 1 = T - 1 \tag{23}$$

which increases linearly with time. Additionally, the learner may be compared to an expert that simply chooses $w = 0$ at each timestep; this strategy incurs zero loss over time. Therefore, the regret of the FTL learner in this case is equal to its loss and also grows linearly over time ($O(T)$), which is considered poor performance.

A more preferable loss function for FTL is **quadratic** loss of the form $f^{(t)}(w) = \frac{1}{2}\|w - z^{(t)}\|_2^2$. The parameter w that minimizes this sum of squared differences is simply the average value of z^t. The corresponding algorithm is given below.

Algorithm 5 Follow the Leader (Quadratic Loss)

```
1: $f^{(0)} \leftarrow 0$        ▷ Initial loss
2: for $t = 1, 2, \cdots, T$ do
3:     $w^{(t)} = \frac{1}{t-1}\Sigma_{i=1}^{t-1} z^{(i)}$        ▷ Update parameter
4:     Receive $(f^{(t)}(w) = \frac{1}{2}\|w - z^{(t)}\|_2^2)$        ▷ Receive loss
5: end for
```

If it is assumed that points z are bounded such that $||z||_2^2 \leq L$, then the regret bound of FTL with quadratic loss can be shown to be:

$$\text{Regret} \leq 4L^2\Big(\log(T) + 1\Big) \tag{24}$$

Note that this grows sub-linearly in T, making FTL with quadratic loss a no-regret algorithm. The derivation of this regret bound will be discussed in the next lecture.

References

[1] N. Littlestone. Learning quickly when irrelevant attributes abound. *Machine Learning*, (2):253–318, 1988.

[2] A. B. Novikoff. On convergence proofs for perceptrons. Technical report, Stanford Research Inst Menlo Park CA, 1963.

[3] X. Wu. Lecture 9: Convex learning problems. *ELEG/CISC 867: Advanced Machine Learning*, 2019.

[4] M. A. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, 2003.

3 Appendix

3.1 Winnow Algorithm Mistake Bound Derivation

First, choose the potential function for the Winnow algorithm (at time t) as the sum of all weights:

$$\Phi^t = \sum_n w_n^{(t)} \tag{25}$$

The total mistakes made by the learner is given by:

$$M = m^- + m^+ \tag{26}$$

Here, m^+ indicates a "mistake on positive" and m^- indicates a "mistake on negative" (as defined in Summary above). Throughout this derivation, we will also assume that $\beta = 1$.

For mistakes on positives, the weights can only increase. Thus, the following holds:

$$\Phi^t = \Phi^{t-1} + \boldsymbol{w}^{(t-1)} \cdot \boldsymbol{x}^{(t-1)} \tag{27}$$

Additionally, because $\boldsymbol{w}^{(t-1)} \cdot \boldsymbol{x}^{(t-1)} \leq \Theta$ for mistakes on positives, this simplifies to:

$$\Phi^t \leq \Phi^{t-1} + \Theta \tag{28}$$

By induction, this can be restated as:

$$\Phi^t \leq \Theta + m^+\Theta \tag{29}$$

Next, we analyze the mistakes on negatives. For these mistakes, the weights can only decrease:

$$\Phi^t = \Phi^{t-1} - \sum_{n:x_n^{(t)}=1} \frac{1}{2} w_n^{t-1} \tag{30}$$

$$\implies \Phi^t = \Phi^{t-1} - \frac{1}{2} w^{(t-1)} \cdot x^{(t-1)} \tag{31}$$

For mistakes on negatives, $w^{(t-1)} \cdot x^{(t-1)} > \Theta$ holds. Thus:

$$\Phi^t < \Theta - m^- \frac{1}{2}\Theta \tag{32}$$

Combining these two bounds leads to the upper bound of the potential function:

$$\Phi^t \leq \Theta + m^+\Theta - m^- \frac{1}{2}\Theta \tag{33}$$

The lower bound of the potential is zero, because the Winnow weights are initialized to 1 and multiplicative updates are used. Combining this with the upper bound gives:

$$0 \leq \Theta + m^+\Theta - m^- \frac{1}{2}\Theta \tag{34}$$

Rearranging this yields:

$$m^- < 2 + 2m^+ \tag{35}$$

To convert this to a bound on total mistakes, we make use of two observations. First, $w_n^{(t)} = 2^{m^+}$ after m^+ mistakes on positives. Second, $w_n^{(t)} < \Theta$ when $x_n^{(t)} = 1$, otherwise it would be impossible to make a mistake on positive. Using these observations leads to:

$$w_n^{(t)} = 2^{m^+-1} < \Theta \tag{36}$$
$$\implies m^+ < \log_2\Theta + 1 \tag{37}$$

Considering k relevant features, and knowing that only one weight will be updated after each mistake in the worst case, yields:

$$m^+ < k(\log_2\Theta + 1) \tag{38}$$

Substituting this into our previous inequality yields an upper bound for m^-:

$$m^- < 2 + 2k(\log_2\Theta + 1) \tag{39}$$

Recalling that the total mistakes $M = m^+ + m^-$, we obtain the final mistake bound for the Winnow algorithm:

$$M < 2 + 3k(\log_2\Theta + 1) \tag{40}$$

3.2 Convex-Smooth-Bounded Learning Problems

These learning problems are very similar to "Convex-Lipschitz-Bounded" learning problems, which were discussed in lecture. Both types of problems define certain properties of the solution space and corresponding function(s) such that a convex optimization problem can be solved successfully.

Smooth Functions A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is L-smooth if its gradient is L-Lipschitz [3]:

$$||\nabla f(\boldsymbol{v}) - \nabla f(\boldsymbol{w})|| \leq L||\boldsymbol{v} - \boldsymbol{w}|| \qquad \forall \boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^d \tag{41}$$

Note that it is also common for smooth functions to be defined using β instead of L.

A learning problem is called a Convex-Smooth-Bounded learning problem when the following conditions [3] are satisfied:

1. The hypothesis class H is a convex set.
2. $||\boldsymbol{w}|| \leq B$ for all $\boldsymbol{w} \in H$, where B is a constant parameter.
3. The loss function l is convex and non-negative.
4. The loss function l is L-smooth.