1 Perceptron

Goal

Understand the celebrated perceptron algorithm for online binary classification.

Alert 1.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Definition 1.2: Binary classification

Given a set of n known example pairs $\{(\mathbf{x}_i, y_i) : i = 1, 2, \dots, n\}$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{\pm 1\}$, we want to learn a (binary) "classification rule" $h : \mathbb{R}^d \to \{\pm 1\}$, so that

$$h(\mathbf{x}) = y$$

on *most* unseen (future) examples $(\mathbf{x}, y)$. Throughout we will call $\mathbf{x}_i$ the feature vector of the i-th example, and y_i the (binary) label of the i-th example. Together, the known example pairs $\{(\mathbf{x}_i, y_i) : i = 1, 2, \dots, n\}$ are called the training set, with n being its size and d being its dimension. The unseen future example $(\mathbf{x}, y)$ will be called the test example. If we have a set of test examples, together they will be called a test set.

Alert 1.3: Notations

We use boldface letters, e.g. $\mathbf{x}$, for a vector of appropriate size. Subscripts are used for two purposes: (the bold) $\mathbf{x}_i$ denotes a vector that may have nothing to do with $\mathbf{x}$, while (the non-bold) x_i denotes the i-th coordinate of $\mathbf{x}$. The j-th coordinate of $\mathbf{x}_i$ will be denoted as x_{ji}. We use $\mathbf{1}$ and $\mathbf{0}$ to denote a vector of all 1s and 0s of appropriate size (which should be clear from context), respectively.

By default, all vectors are column vectors and we use $\mathbf{x}^\top$ to denote the transpose (i.e. a row vector) of a column vector $\mathbf{x}$.

Definition 1.4: Functions and sets are equivalent

A binary classification rule $h : \mathbb{R}^d \to \{\pm 1\}$ can be identified with a set $P \subseteq \mathbb{R}^d$ and its complement $N = \mathbb{R}^d \setminus P$, where $h(\mathbf{x}) = 1 \iff \mathbf{x} \in P$.

Exercise 1.5: Multiclass rules

Let $h : \mathbb{R}^d \to \{1, 2, \dots, c\}$, where $c \geq 2$ is the number of classes. How do we identify the function h with sets?

Remark 1.6: Memorization does NOT work... Or does it?

The challenge of binary classification lies in two aspects:

- on a test example $(\mathbf{x}, y)$, we actually only have access to $\mathbf{x}$ but not the label y. It is our job to predict y, hopefully correctly most of the time.
- the test example $\mathbf{x}$ can be (very) different from any of the training examples $\{\mathbf{x}_i : i = 1, \dots, n\}$. So we can not expect *naive* memorization to work.

Essentially, we need a (principled?) way to interpolate from the training set (where labels are known) and hopefully generalize to the test set (where labels need to be predicted). For this to be possible, we need

- the training set to be "indicative" of what the test set look like, and/or
- a proper baseline (competitor) to compare against.

Definition 1.7: Statistical learning

We assume the training examples $(\mathbf{x}_i, y_i)$ and the test example $(\mathbf{x}, y)$ are drawn independently and identically (i.i.d.) from an unknown distribution $\mathbb{P}$:

$$(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n), (\mathbf{x}, y) \overset{i.i.d.}{\sim} \mathbb{P},$$

in which case we usually use capital letters $(\mathbf{X}_i, Y_i)$ and $(\mathbf{X}, Y)$ to emphasize the random nature of these quantities. Our goal is then to find a classification rule $h : \mathbb{R}^d \to \{\pm 1\}$ so that the classification error

$$\mathbb{P}(h(\mathbf{X}) \neq Y)$$

is as small as possible. Put in optimization terms, we are interested in solving the following (abstract) optimization problem:

$$\min_{h:\mathbb{R}^d \to \{\pm 1\}} \mathbb{P}(h(\mathbf{X}) \neq Y). \tag{1.1}$$

We will shortly see that if $\mathbb{P}$ is known, then the classification problem (1.1) admits a closed-form solution known as the Bayes classifier. In the more realistic case where $\mathbb{P}$ is not known, our hope is that the training set $\{(\mathbf{X}_i, Y_i) : i = 1, \dots, n\}$ may provide enough information about $\mathbb{P}$, as least when n is sufficiently large, which is basically the familiar law of large numbers in (serious) disguise.

Remark 1.8: i.i.d., seriously?

Immediate objections to the i.i.d. assumption in statistical learning include (but not limit to):

- the training examples are hardly i.i.d..
- the test example may follow a different distribution than the training set, known as domain shift.

Reasons to support the i.i.d. assumption include (but not limit to):

- it is a simple, clean mathematical abstraction that allows us to take a first step in understanding and solving the binary classification problem.
- for many real problems, the i.i.d. assumption is not terribly off. In fact, it is a reasonably successful approximation.
- there exist more complicated ways to alleviate the i.i.d. assumption, usually obtained by refining results under the i.i.d. assumption.

We will take a more pragmatic viewpoint: we have to start from somewhere and the i.i.d. assumption seems to be a good balance between what we can analyze and what we want to achieve.

Definition 1.9: Online learning

A different strategy, as opposed to statistical learning, is not to put any assumption whatsoever on the data, but on what we want to compare against: Given a collection of existing classification rules $\mathcal{G} = \{g_k : k \in K\}$, we want to construct a classification rule h that is competitive against the "best" $g^* \in \mathcal{G}$, in terms of the number of mistakes:

$$\mathfrak{M}(h) := \sum_{i=1}^{n} [\![h(\mathbf{x}_i) \neq y_i]\!].$$

The "subtlety" is that even the best $g^* \in \mathcal{G}$ may not perform well on the data $\mathcal{D} = \{(\mathbf{x}_i, y_i) : i = 1, \dots, n\}$, so being competitive against the best g^* in $\mathcal{G}$ may or may not be as significant as you would have liked.

When the examples $(\mathbf{x}_i, y_i)$ come one at a time, i.e. in the online (streaming) fashion, we can give ourselves even more flexibility: we construct a sequence of classification rules $\{h_i : i = 1, 2, \dots\}$, and the evaluation proceeds as follows. Start with $i = 1$ and choose h_1:

(I). receive $\mathbf{x}_i$ and predict $\hat{y}_i = h_i(\mathbf{x}_i)$

(II). receive true label y_i and possibly suffer a mistake if $\hat{y}_i \neq y_i$

(III). adjust h_i to h_{i+1} and increment i by 1.

(We could also allow h_i to depend on $\mathbf{x}_i$, i.e. delay the adjustment of h_i until receiving $\mathbf{x}_i$.) Note that while we are allowed to adaptively adjust our classification rules $\{h_i\}$, the competitor is more restricted: it has to stick to some fixed rule $g_k \in \mathcal{G}$ chosen well before seeing any example.

Definition 1.10: Proper vs. improper learning

When we require $h \in \mathcal{G}$, we are in the proper learning regime, where we (who chooses h) must act in the same space as the competitor (who is constrained to choose from $\mathcal{G}$). However, sometimes learning is easier if we abandon $h \in \mathcal{G}$ and operate in the improper learning regime. Life is never fair anyways ☺.

Of course, this distinction is relative to the class $\mathcal{G}$. In particular, if $\mathcal{G}$ consists of all possible functions, then any learning algorithm is proper but we are competing against a very strong competitor.

Alert 1.11: Notation

The Iverson notation $[\![A]\!]$ or sometimes also $\mathbf{1}(A)$ (or even $\mathbf{1}_A$) denotes the indicator function of the event $A \subseteq \mathbb{R}^d$, i.e., $[\![A]\!]$ is 1 if the event A holds and 0 otherwise.

We use $|A|$ to denote the size (i.e. the number of elements) of a set A.

Definition 1.12: Thresholding

Often it is more convenient to learn a real-valued function $f : \mathbb{R}^d \to \mathbb{R}$ and then use thresholding to get a binary-valued classification rule: $h = \operatorname{sign}(f)$, where say we define $\operatorname{sign}(0) = -1$ (or $\operatorname{sign}(0) = 1$, the actual choice is usually immaterial).

Definition 1.13: Linear and affine functions

Perhaps the simplest multivariate function is the class of linear/affine functions. Recall that a function $f : \mathbb{R}^d \to \mathbb{R}$ is linear if for all $\mathbf{w}, \mathbf{z} \in \mathbb{R}^d$ and $\alpha, \beta \in \mathbb{R}$:

$$f(\alpha\mathbf{w} + \beta\mathbf{z}) = \alpha f(\mathbf{w}) + \beta f(\mathbf{z}).$$

From the definition it follows that $f(\mathbf{0}) = 0$ for any linear function f.

Similarly, a function $f : \mathbb{R}^d \to \mathbb{R}$ is called affine if for all $\mathbf{w}, \mathbf{z} \in \mathbb{R}^d$ and $\alpha \in \mathbb{R}$:

$$f(\alpha\mathbf{w} + (1-\alpha)\mathbf{z}) = \alpha f(\mathbf{w}) + (1-\alpha)f(\mathbf{z}).$$

Compared to the definition of linear functions, the restriction $\alpha + \beta = 1$ is enforced.

Exercise 1.14: Representation of linear and affine functions

Prove:

- If $f : \mathbb{R}^d \to \mathbb{R}$ is linear, then for any $n \in \mathbb{N}$, any $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$, any $\alpha_1, \dots, \alpha_n \in \mathbb{R}$, we have

$$f\left(\sum_{i=1}^n \alpha_i \mathbf{x}_i\right) = \sum_{i=1}^n \alpha_i f(\mathbf{x}_i).$$

  What is the counterpart for affine functions?

- A function $f : \mathbb{R}^d \to \mathbb{R}$ is linear iff there exists some $\mathbf{w} \in \mathbb{R}^d$ so that $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$.
- A function $f : \mathbb{R}^d \to \mathbb{R}$ is affine iff there exists some $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ so that $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$, i.e. an affine function is a linear function translated by some constant.

Definition 1.15: Inner product

Recall the inner product (i.e. dot product) between two vectors $\mathbf{w}, \mathbf{x} \in \mathbb{R}^d$ is defined as

$$\mathbf{w}^\top \mathbf{x} = \sum_{j=1}^d w_j x_j = \mathbf{x}^\top \mathbf{w}.$$

The notation $\langle \mathbf{w}, \mathbf{x} \rangle$ is also used (especially when we want to abstract away coordinates/basis).

Algorithm 1.16: Perceptron

Combining thresholding (see Definition 1.12) and affine functions (see Definition 1.13), the celebrated perceptron algorithm of Rosenblatt (1958) tries to learn a classification rule

$$h(\mathbf{x}) = \operatorname{sign}(f(\mathbf{x})), \qquad f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b,$$

parameterized by the weight vector $\mathbf{w} \in \mathbb{R}^d$ and bias (threshold) $b \in \mathbb{R}$ so that

$$\forall i, \quad y_i = h(\mathbf{x}_i) \iff y_i \hat{y}_i > 0, \quad \hat{y}_i = f(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i + b, \tag{1.2}$$

where we have used the fact that $y_i \in \{\pm 1\}$ (and ignored the possibility of $\hat{y}_i = 0$ for the direction $\Rightarrow$).

In the following perceptron algorithm, the training examples come in the online fashion (see Definition 1.9), and the algorithm updates only when it makes a "mistake" (line 3).

Algorithm: Perceptron (Rosenblatt 1958)

Input: Dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{\pm 1\} : i = 1, \dots, n\}$, initialization $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$, threshold $\delta \geq 0$

Output: approximate solution $\mathbf{w}$ and b

1 **for** $t = 1, 2, \dots$ **do**
2 receive training example index $I_t \in \{1, \dots, n\}$ `// the index` I_t `can be random`
3 **if** $y_{I_t}(\mathbf{w}^\top \mathbf{x}_{I_t} + b) \leq \delta$ **then**
4 $\mathbf{w} \leftarrow \mathbf{w} + y_{I_t}\mathbf{x}_{I_t}$ `// update only after making a "mistake"`
5 $b \leftarrow b + y_{I_t}$

We can break the for-loop if a maximum number of iterations has been reached, or if all training examples are correctly classified in a full cycle (in which case the algorithm will no longer update itself).

Rosenblatt, F. (1958). "The perceptron: A probabilistic model for information storage and organization in the brain". *Psychological Review*, vol. 65, no. 6, pp. 386–408.

Remark 1.17: Padding

If we define $\mathbf{a}_i = y_i \begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix}$, $\mathsf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_n] \in \mathbb{R}^{p \times n}$ (where $p = d + 1$ stands for the number of predictors), and $\mathbf{w} = \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}$, then clearly

$$\mathbf{a}_i^\top \mathbf{w} = y_i(\mathbf{w}^\top \mathbf{x}_i + b).$$

Thus, the perceptron problem (1.2) can be concisely reduced to the following (slightly more general) system of linear inequalities:

$$\text{Given } \mathsf{A} \in \mathbb{R}^{p \times n} \quad \text{find } \mathbf{w} \in \mathbb{R}^p \text{ so that } \mathsf{A}^\top \mathbf{w} > \mathbf{0}, \tag{1.3}$$

where the (strict) inequality is meant elementwise, i.e. $\mathbf{x} > \mathbf{w} \iff \forall j,\ x_j > w_j$. In the sequel we will identify the perceptron problem (1.2) with the above system of linear equalities in (1.3).

The trick to pad the constant 1 to $\mathbf{x}_i$ and the bias b to $\mathbf{w}$ so that we can deal with the pair $(\mathbf{w}, b)$ more concisely is used ubiquitously in machine learning. The trick to multiply the *binary* label y_i to $\mathbf{x}_i$ is also often used in binary classification problems.

Alert 1.18: Notation

We use $\mathbf{x}$ and $\mathbf{w}$ for the original vectors and $\mathbf{x}$ and $\mathbf{w}$ for the padded versions (with constant 1 and bias b respectively). Similar, we use X and W for the original matrices and X and $\mathbf{w}$ for the padded versions.

We use $\hat{\mathrm{y}} \in \mathbb{R}$ for a real-valued prediction and $\hat{y} \in \{\pm 1\}$ for a binary prediction, keeping in mind that usually $\hat{y} = \operatorname{sign}(\hat{\mathrm{y}})$.

Remark 1.19: "If it ain't broke, don't fix it"

The perceptron algorithm is a perfect illustration of the good old wisdom: "If it ain't broke, don't fix it." Indeed, it maintains the same weight vector $(\mathbf{w}, b)$ until when a "mistake" happens, i.e. line 3 in Algorithm 1.16. This principle is often used in designing machine learning algorithms, or in life ☺.

On the other hand, had we always performed the updates in line 4 and 5 (even when we predicted correctly), then it is easy to construct an infinite sequence $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots$, that is strictly linearly separable (see Definition 1.24 below), and yet the modified (aggressive) perceptron will make infinitely many mistakes.

Indeed, let $\delta = 0$ and $y_i \equiv 1$. This dataset (with any $\mathbf{x}_i$) is clearly linearly separable (simply take $\mathbf{w} = \mathbf{0}, b = 1$). The aggressive perceptron maintains the weight (assuming w.l.o.g that $\mathbf{w}_0 = \mathbf{0}$)

$$\mathbf{w}_t = \sum_{i=1}^{t} \begin{pmatrix} \mathbf{x}_i \\ 1 \end{pmatrix}.$$

Thus, to fool it we need

$$\langle \mathbf{w}_t, \mathbf{a}_{t+1} \rangle = \left\langle \sum_{i=1}^{t} \mathbf{x}_i, \mathbf{x}_{t+1} \right\rangle + t < 0, \quad i.e., \quad \langle \bar{\mathbf{x}}_t, \mathbf{x}_{t+1} \rangle < -1,$$

where $\bar{\mathbf{x}}_t := \frac{1}{t} \sum_{i=1}^{t} \mathbf{x}_i$. Now, let

$$\mathbf{x}_t = \begin{cases} -2\mathbf{x}, & \text{if } \log_2 t \in \mathbb{N} \\ \mathbf{x}, & \text{otherwise} \end{cases},$$

for any fixed $\mathbf{x}$ with unit length, i.e. $\|\mathbf{x}\|_2 = 1$. Then, we verify $\bar{\mathbf{x}}_t \to \mathbf{x}$ while for sufficiently large t such that $\log_2(t+1) \in \mathbb{N}$, we have $\langle \bar{\mathbf{x}}_t, \mathbf{x}_{t+1} \rangle \to \langle \mathbf{x}, -2\mathbf{x} \rangle = -2 < -1$. Obviously, such t's form an infinite subsequence, on which the aggressive perceptron errs.

Exercise 1.20: Mistakes all the time

Construct a linearly separable sequence $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots$, so that the aggressive perceptron makes mistakes on every example! [Hint: you may let $\mathbf{x}_i$ go unbounded. Can you make it bounded?]

Remark 1.21: History

Historically, perceptron was the first algorithm that kicked off the entire field of artificial intelligence. Its design, analysis, and application have had lasting impact on the machine learning field to this day. Ironically, the failure of perceptron on nonlinear problems (to be discussed in later lectures) almost killed the entire artificial intelligence field as well...

Exercise 1.22: Perceptron for solving (homogeneous) linear inequalities

Modify Algorithm 1.16 to solve the system of (homogeneous) linear inequalities (1.3).

Alert 1.23: Existence and uniqueness of solution

For any problem you are interested in solving, the first question you should ask is:

Does there exist a solution?

- If the answer is "no," then the second question you should ask is:

  If there is no solution at all, can we still "solve" the problem in certain meaningful ways?

- If the answer is "yes," then the second question you should ask is:

  If there is at least one solution, then is the solution unique?

  – If the answer is "no," then the third question you should ask is:

Is there any reason to prefer a certain solution?

Too often in ML, we hasten to design algorithms and run experiments, without fully understanding or articulating what we are trying to solve, or deciding if the problem is even solvable in any well-defined sense. There is certain value in this philosophy but also great danger.

Definition 1.24: (Strictly) linear separable

We say that the perceptron problem (1.3) is (strictly) linearly separable if for some hence all $s>0$, there exists some $\mathbf{w}$ such that $\forall i, \mathbf{a}_i^\top \mathbf{w} \geq s > 0$ (or in matrix notation $\mathsf{A}^\top \mathbf{w} \geq s\mathbf{1}$). Otherwise, we say the perceptron problem is linearly inseparable.

This is the reason why the threshold parameter δ in Algorithm 1.16 is immaterial, at least in terms of convergence when the problem is indeed linearly separable.

Definition 1.25: Norms and Cauchy-Schwarz inequality

For any vector $\mathbf{w} \in \mathbb{R}^d$, its Euclidean ($\ell_2$) norm (i.e., length) is defined as:

$$\|\mathbf{w}\|_2 := \sqrt{\mathbf{w}^\top \mathbf{w}} = \sqrt{\sum_{j=1}^d |w_j|^2}.$$

More generally, for any $p \geq 1$, we define the ℓ_p norm

$$\|\mathbf{w}\|_p := \big(\sum_{j=1}^d |w_j|^p\big)^{1/p}$$

while for $p = \infty$ we define the max norm

$$\|\mathbf{w}\|_\infty := \max_{j=1,\dots,d} |w_j|.$$

Even more generally, a norm is any function $\|\cdot\| : \mathbb{R}^d \to \mathbb{R}_+$ that satisfies:

- (definite) $\|\mathbf{w}\| = 0 \iff \mathbf{w} = \mathbf{0}$
- (homogeneous) for all $\lambda \in \mathbb{R}$ and $\mathbf{w} \in \mathbb{R}^d$, $\|\lambda \mathbf{w}\| = |\lambda| \cdot \|\mathbf{w}\|$
- (triangle inequality) for all $\mathbf{w}$ and $\mathbf{z} \in \mathbb{R}^d$:

$$\|\mathbf{w} + \mathbf{z}\| \leq \|\mathbf{w}\| + \|\mathbf{z}\|.$$

The norm function is a convenient way to convert a vector quantity to a real number, for instance, to facilitate numerical comparison. Part of the business in machine learning is to understand the effect of different norms on certain learning problems, even though all norms are "formally equivalent:" for any two norms $\|\cdot\|$ and $|||\cdot|||$, there exist constants $c_d, C_d \in \mathbb{R}$ so that $\forall \mathbf{w} \in \mathbb{R}^d$,

$$c_d \|\mathbf{w}\| \leq |||\mathbf{w}||| \leq C_d \|\mathbf{w}\|.$$

The subtlety lies on the dependence of the constants c_d, C_d on the dimension d: could be exponential and could affect a learning algorithm a lot.

The dual (norm) $\|\cdot\|_\circ$ of the norm $\|\cdot\|$ is defined as:

$$\|\mathbf{z}\|_\circ := \max_{\|\mathbf{w}\|=1} \mathbf{w}^\top \mathbf{z} = \max_{\mathbf{w}\neq 0} \frac{\mathbf{w}^\top \mathbf{z}}{\|\mathbf{w}\|} = \max_{\|\mathbf{w}\|=1} |\mathbf{w}^\top \mathbf{z}| = \max_{\mathbf{w}\neq 0} \frac{|\mathbf{w}^\top \mathbf{z}|}{\|\mathbf{w}\|}.$$

From the definition it follows the important inequality:

$$\mathbf{w}^\top \mathbf{z} \le |\mathbf{w}^\top \mathbf{z}| \le \|\mathbf{w}\| \cdot \|\mathbf{z}\|_\circ.$$

The Cauchy-Schwarz inequality, which will be repeatedly used throughout the course, is essentially a self-duality property of the ℓ_2 norm:

$$\mathbf{w}^\top \mathbf{z} \le |\mathbf{w}^\top \mathbf{z}| \le \|\mathbf{w}\|_2 \cdot \|\mathbf{z}\|_2,$$

i.e., the dual norm of the ℓ_2 norm is itself. The dual norm of the ℓ_p norm is the ℓ_q norm, where

$$\infty \ge p, q \ge 1 \text{ and } \frac{1}{p} + \frac{1}{q} = 1.$$

Exercise 1.26: Norms

Prove the following:

- for any $\mathbf{w} \in \mathbb{R}^d$: $\|\mathbf{w}\|_p \to \|\mathbf{w}\|_\infty$ as $p \to \infty$
- for any $\infty \ge p \ge 1$, the ℓ_p norm is indeed a norm. What about $p < 1$?
- the dual of any norm $\|\cdot\|$ is indeed again a norm
- for any $\infty \ge p \ge q \ge 1$, $\|\mathbf{w}\|_p \le \|\mathbf{w}\|_q \le d^{\frac{1}{q}-\frac{1}{p}} \|\mathbf{w}\|_p$
- for any $\mathbf{w}, \mathbf{z} \in \mathbb{R}^d$: $\|\mathbf{w}+\mathbf{z}\|_2^2 + \|\mathbf{w}-\mathbf{z}\|_2^2 = 2(\|\mathbf{w}\|_2^2 + \|\mathbf{z}\|_2^2)$.

Remark 1.27: Key insight

The key insight for the success of the perceptron Algorithm 1.16 is the following simple inequality:

$$\langle \mathbf{a}, \mathbf{w}_{t+1} \rangle = \langle \mathbf{a}, \mathbf{w}_t + \mathbf{a} \rangle = \langle \mathbf{a}, \mathbf{w}_t \rangle + \|\mathbf{a}\|_2^2 > \langle \mathbf{a}, \mathbf{w}_t \rangle .$$

(Why can we assume w.l.o.g. that $\|\mathbf{a}\|_2^2 > 0$?) Therefore, if the condition $\langle \mathbf{a}, \mathbf{w}_t \rangle > \delta$ is violated, then we perform an update which brings us strictly closer to satisfy that constraint. [The magic is that by doing so we do not ruin the possibility of satisfying all other constraints, as we shall see.]

This particular update rule of perceptron can be justified as performing stochastic gradient descent on an appropriate objective function, as we are going to see in a later lecture.

Theorem 1.28: Perceptron convergence theorem (Block 1962; Novikoff 1962)

Assuming the data A *(see Remark 1.17) is (strictly) linearly separable and denoting* $\mathbf{w}_t$ *the iterate after the* t*-th update in the perceptron algorithm. Then,* $\mathbf{w}_t \to$ *some* $\mathbf{w}^*$ *in finite time. If each column of* A *is selected infinitely often, then* $\mathsf{A}^\top \mathbf{w}^* > \delta \mathbf{1}$.

Proof. Under the linearly separable assumption there exists some solution $\mathbf{w}^\star$, i.e., $\mathsf{A}^\top \mathbf{w}^\star \ge s\mathbf{1}$ for some $s > 0$. Then, upon making an update from $\mathbf{w}_t$ to $\mathbf{w}_{t+1}$ (using the data example denoted as $\mathbf{a}$):

$$\langle \mathbf{w}_{t+1}, \mathbf{w}^\star \rangle = \langle \mathbf{w}_t, \mathbf{w}^\star \rangle + \langle \mathbf{a}, \mathbf{w}^\star \rangle \ge \langle \mathbf{w}_t, \mathbf{w}^\star \rangle + s.$$

Hence, by telescoping we have $\langle \mathbf{w}_t, \mathbf{w}^\star \rangle \ge \langle \mathbf{w}_0, \mathbf{w}^\star \rangle + ts$, which then, using the Cauchy-Schwarz inequality,

implies $\|\mathbf{w}_t\|_2 \geq \frac{\langle \mathbf{w}_0, \mathbf{w}^\star \rangle + ts}{\|\mathbf{w}^\star\|_2}$. [Are we certain that $\|\mathbf{w}^\star\|_2 \neq 0$?]

On the other hand, using the fact that we make an update only when $\langle \mathbf{a}, \mathbf{w}_t \rangle \leq \delta$:

$$\|\mathbf{w}_{t+1}\|_2^2 = \|\mathbf{w}_t + \mathbf{a}\|_2^2 = \|\mathbf{w}_t\|_2^2 + 2\langle \mathbf{w}_t, \mathbf{a} \rangle + \|\mathbf{a}\|_2^2 \leq \|\mathbf{w}_t\|_2^2 + 2\delta + \|\mathbf{a}\|_2^2.$$

Hence, telescoping again we have $\|\mathbf{w}_t\|_2^2 \leq \|\mathbf{w}_0\|_2^2 + (2\delta + \|\mathsf{A}\|_{2,\infty}^2)t$, where we use the notation $\|\mathsf{A}\|_{2,\infty} := \max_i \|\mathbf{a}_i\|_2$.

Combine the above two (blue) inequalities: $\frac{\langle \mathbf{w}_0, \mathbf{w}^\star \rangle + ts}{\|\mathbf{w}^\star\|_2} \leq \sqrt{\|\mathbf{w}_0\|_2^2 + (2\delta + \|\mathsf{A}\|_{2,\infty}^2)t}$, solving which gives:

$$t \leq \frac{(2\delta + \|\mathsf{A}\|_{2,\infty}^2)\|\mathbf{w}^\star\|_2^2 + 2s\|\mathbf{w}^\star\|_2\|\mathbf{w}_0\|_2}{s^2}. \tag{1.4}$$

Thus, the perceptron algorithm performs at most a finite number of updates, meaning that $\mathbf{w}_t$ remains unchanged thereafter. □

Typically, we start with $\mathbf{w}_0 = \mathbf{0}$ and we choose $\delta = 0$, then the perceptron algorithm converges after at most $\frac{\|\mathsf{A}\|_{2,\infty}^2 \|\mathbf{w}^\star\|_2^2}{s^2}$ updates.

Block, H. D. (1962). "The perceptron: A model for brain functioning". *Reviews of Modern Physics*, vol. 34, no. 1, pp. 123–135.

Novikoff, A. (1962). "On Convergence proofs for perceptrons". In: *Symposium on Mathematical Theory of Automata*, pp. 615–622.

Exercise 1.29: Data normalization

Suppose the data $\mathcal{D} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{\pm 1\} : i = 1, \ldots, n\}$ is linearly separable, i.e., there exists some $s > 0$ and $\mathbf{w} = (\mathbf{w}; b)$ such that $\mathsf{A}^\top \mathbf{w} \geq s\mathbf{1}$, where recall that $\mathbf{a}_i = y_i \binom{\mathbf{x}_i}{1}$.

- If we scale each instance $\mathbf{x}_i$ to $\lambda \mathbf{x}_i$ for some $\lambda > 0$, is the resulting data still linearly separable? Does perceptron converge faster or slower after scaling? How does the bound (1.4) change?
- If we translate each instance $\mathbf{x}_i$ to $\mathbf{x}_i + \bar{\mathbf{x}}$ for some $\bar{\mathbf{x}} \in \mathbb{R}^d$, is the resulting data still linearly separable? Does perceptron converge faster or slower after translation? How does the bound (1.4) change? [Hint: you can initialize $\mathbf{w}_0$ differently after the scaling.]

Remark 1.30: Optimizing the bound

As we mentioned above, (for linearly separable data) the perceptron algorithm converges after at most $\frac{\|\mathsf{A}\|_{2,\infty}^2 \|\mathbf{w}^\star\|_2^2}{s^2}$ steps, if we start with $\mathbf{w}_0 = \mathbf{0}$ (and choose $\delta = 0$). Note, however, that the "solution" $\mathbf{w}^\star$ is introduced merely for the analysis of the perceptron algorithm; the algorithm in fact does not "see" it at all. In other words, $\mathbf{w}^\star$ is "fictional," hence we can tune it to optimize our bound as follows:

$$\min_{(\mathbf{w}^\star, s): \mathsf{A}^\top \mathbf{w}^\star \geq s\mathbf{1}} \frac{\|\mathbf{w}^\star\|_2^2}{s^2} = \min_{(\mathbf{w}, s): \|\mathbf{w}\|_2 \leq 1,\ \mathsf{A}^\top \mathbf{w} \geq s\mathbf{1}} \frac{1}{s^2} = \left[\frac{1}{\max_{(\mathbf{w}, s): \|\mathbf{w}\|_2 \leq 1,\ \mathsf{A}^\top \mathbf{w} \geq s\mathbf{1}} s}\right]^2 = \underbrace{\left[\frac{1}{\max_{\mathbf{w}: \|\mathbf{w}\|_2 \leq 1} \min_i \langle \mathbf{a}_i, \mathbf{w} \rangle}\right]^2}_{\text{the margin } \gamma_2},$$

where we implicitly assumed the denominator is positive (i.e. A is linearly separable). Therefore, the perceptron algorithm (with $\mathbf{w}_0 = \mathbf{0}, \delta = 0$) converges after at most

$$T = T(\mathsf{A}) := \frac{\max_i \|\mathbf{a}_i\|_2^2}{\left(\max_{\|\mathbf{w}\|_2 \leq 1} \min_i \langle \mathbf{a}_i, \mathbf{w} \rangle\right)^2}$$

steps. If we scale the data so that $\|\mathsf{A}\|_{2,\infty} := \max_i \|\mathbf{a}_i\|_2 = 1$, then we have the appealing bound:

$$T = T(\mathsf{A}) = \frac{1}{\gamma_2^2}, \quad \text{where} \quad \gamma_2 = \gamma_2(\mathsf{A}) = \max_{\|\mathbf{w}\|_2 \le 1} \min_i \langle \mathbf{a}_i, \mathbf{w} \rangle \le \min_i \max_{\|\mathbf{w}\|_2 \le 1} \langle \mathbf{a}_i, \mathbf{w} \rangle = \min_i \|\mathbf{a}_i\|_2 \le \|\mathsf{A}\|_{2,\infty} = 1. \tag{1.5}$$

Intuitively, the margin parameter γ_2 characterizes how "linearly separable" a dataset A is, and the perceptron algorithm converges faster if the data is "more" linearly separable!

Remark 1.31: Uniqueness

The perceptron algorithm outputs a solution $\mathbf{w}$ such that $\mathsf{A}\mathbf{w} > \delta \mathbf{1}$, but it does not seem to care which solution to output if there are multiple ones. The iteration bound in (1.5) actually suggests a different algorithm, famously known as the support vector machines (SVM). The idea is simply to find the weight vector $\mathbf{w}$ that attains the margin in (1.5):

$$\max_{\|\mathbf{w}\|_2 \le 1} \min_i \langle \mathbf{a}_i, \mathbf{w} \rangle \iff \min_{\mathbf{w}: \mathsf{A}\mathbf{w} \ge \mathbf{1}} \|\mathbf{w}\|_2^2,$$

where the right-hand side is the usual formula for hard-margin support vector machines (SVM), to be discussed in a later lecture! (Strictly speaking, we need to replace $\|\mathbf{w}\|_2^2$ with $\|\mathbf{w}\|_2^2$, i.e., unregularizing the bias b in SVM.)

Alert 1.32: Notation

For two real numbers $u, v \in \mathbb{R}$, the following standard notations will be used throughout the course:

- $u \vee v := \max\{u, v\}$: maximum of the two
- $u \wedge v := \min\{u, v\}$: minimum of the two
- $u^+ = u \vee 0 = \max\{u, 0\}$: positive part
- $u^- = \max\{-u, 0\}$: negative part

These operations extend straightforwardly in the elementwise manner to two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$.

Exercise 1.33: Decomposition

Prove the following claims (note that the negative part u^- is a positive number by definition):

- $u^+ = (-u)^-$
- $u = u^+ - u^-$
- $|u| = u^+ + u^-$

Theorem 1.34: Optimality of perceptron

Let $n = 1/\gamma^2 \wedge d$. For any deterministic algorithm $\mathcal{A}$, there exists a dataset $\{(\mathbf{e}_i, y_i)\}_{i=1}^n$ with margin at least γ such that $\mathcal{A}$ makes at least n mistakes on it.

Proof. For any deterministic algorithm $\mathcal{A}$, set $y_i = -\mathcal{A}(\mathbf{e}_1, y_1, \ldots, \mathbf{e}_{i-1}, y_{i-1}, \mathbf{e}_i)$. Clearly, $\mathcal{A}$ makes n mistakes on the dataset $\{(\mathbf{e}_i, y_i)\}_{i=1}^n$ (due to the hostility of nature in constructing y_i).

We need only verify the margin claim. Let $w_i^\star = y_i\gamma$ (and $b = 0$), then $y_i \langle \mathbf{e}_i, \mathbf{w}^\star \rangle = \gamma$. Thus, the dataset $\{(\mathbf{e}_i, y_i)\}_{i=1}^n$ has margin at least γ. □

Therefore, for high dimensional problems (i.e. large d), the perceptron algorithm achieves the optimal worst-case mistake bound.

Theorem 1.35: Perceptron boundedness theorem (Amaldi and Hauser 2005)

Let $\mathsf{A} \in \mathbb{R}^{p\times n}$ be a matrix with nonzero columns, $\mathbf{w}_0 \in \mathbb{R}^p$ arbitrary, $\eta_t \in [0, \bar\eta]$, and define

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta_t \mathbf{a}_{I_t},$$

where $\mathbf{a}_{I_t}$ is some column of A chosen such that $\langle \mathbf{w}_t, \mathbf{a}_{I_t} \rangle \le 0$. Then, for all t,

$$\|\mathbf{w}_t\|_2 \le 2\max\left[\|\mathbf{w}_0\|_2,\ \ \bar\eta \max_i \|\mathbf{a}_i\|_2 \times \left((1 \wedge \min_i \|\mathbf{a}_i\|_2)^{\operatorname{rank}(\mathsf{A})} \times \kappa(\mathsf{A}) 2^{3p/2} + 1\right)\right], \tag{1.6}$$

where the condition number

$$\kappa^{-2}(\mathsf{A}) := \min\{\det(B^\top B) : B = [\mathbf{a}_{i_1}, \mathbf{a}_{i_2}, \dots, \mathbf{a}_{i_{\operatorname{rank}(\mathsf{A})}}] \text{ is a submatrix of } \mathsf{A} \text{ with full column rank}\}.$$

Proof. We omit the somewhat lengthy proof. □

The preceptron algorithm corresponds to $\eta_t \equiv 1$, in which case the boundedness claim (without the quantitative bound (1.6)) was first established in Minsky and Papert (1988, originally published in 1969) and Block and Levin (1970).

Amaldi, Edoardo and Raphael Hauser (2005). "Boundedness Theorems for the Relaxation Method". *Mathematics of Operations Research*, vol. 30, no. 4, pp. 939–955.

Minsky, Marvin L. and Seymour A. Papert (1988, originally published in 1969). *Perceptron*. second expanded. MIT press.

Block, H. D. and S. A. Levin (1970). "On the boundedness of an iterative procedure for solving a system of linear inequalities". *Proceedings of the American Mathematical Society*, vol. 26, pp. 229–235.

Remark 1.36: Reducing multiclass to binary

We can easily adapt the perceptron algorithm to datasets with $c > 2$ classes, using either of the following general reduction schemes:

- one-vs-all: For each class k, use its examples as positive and examples from all other classes as negative, allowing us to train a perceptron with weight $\mathbf{w}_k = [\mathbf{w}_k; b_k]$. Upon receiving a new example $\mathbf{x}$, we predict according to the "winner" of the c perceptrons (break ties arbitrarily):

  $$\hat y = \operatorname*{argmax}_{k=1,\dots,c} \ \mathbf{w}_k^\top \mathbf{x} + b_k.$$

  The downside of this scheme is that when we train the k-th perceptron, the dataset is imbalanced, i.e. we have much more negatives than positives. The upside is that we only need to train c (or $c-1$ if we set one class as the default) perceptrons.

- one-vs-one: For each pair (k, k') of classes, we train a perceptron $\mathbf{w}_{k,k'}$ where we use examples from class k as positive and examples from class k' as negative. In total we train $\binom{c}{2}$ perceptrons. Upon receiving a new example $\mathbf{x}$, we count how many times each class k is the (binary) prediction:

  $$|\{k : \mathbf{x}^\top \mathbf{w}_{k,k'} + b_{k,k'} > 0 \text{ or } \mathbf{x}^\top \mathbf{w}_{k',k} + b_{k',k} \le 0\}|.$$

  Of course, we take again the "winner" as our predicted class. The downside here is we have to train $O(c^2)$ perceptrons while the upside is that each time the training set is more balanced.

Algorithm 1.37: General linear inequalities (Agmon 1954; Motzkin and Schoenberg 1954)

More generally, to solve any (non-homogeneous) linear inequality system

$$\mathsf{A}^\top \mathbf{w} \le \mathbf{c}, \quad i.e., \quad \mathbf{a}_i^\top \mathbf{w} \le c_i, \ i = 1, \dots, n,$$

we can extend the idea of perceptron to the following projection algorithm:

Algorithm: Projection Algorithm for Linear Inequalities

Input: $\mathsf{A} \in \mathbb{R}^{p \times n}, \mathbf{c} \in \mathbb{R}^n$, initialization $\mathbf{w} \in \mathbb{R}^p$, relaxation parameter $\eta \in (0, 2]$
Output: approximate solution $\mathbf{w}$

1 **for** $t = 1, 2, \dots$ **do**
2 select an index $I_t \in \{1, \dots, n\}$ `// the index` I_t `can be random`
3 $\mathbf{w} \leftarrow (1 - \eta)\mathbf{w} + \eta \left[\mathbf{w} - \frac{(\mathbf{a}_{I_t}^\top \mathbf{w} - c_{I_t})^+}{\langle \mathbf{a}_{I_t}, \mathbf{a}_{I_t} \rangle} \mathbf{a}_{I_t} \right]$

The term within the square bracket is exactly the projection of $\mathbf{w}$ onto the halfspace $\mathbf{a}_{I_t}^\top \mathbf{w} \le c_{I_t}$. If we choose $\eta \equiv 1$ then we just repeatedly project $\mathbf{w}$ onto each of the halfspaces. With $\eta \equiv 2$ we actually perform reflections, which, as argued by Motzkin and Schoenberg (1954), can accelerate convergence a lot in certain settings.

Agmon, Shmuel (1954). "The Relaxation Method for Linear Inequalities". *Canadian Journal of Mathematics*, vol. 6, pp. 382–392.
Motzkin, T. S. and I. J. Schoenberg (1954). "The Relaxation Method for Linear Inequalities". *Canadian Journal of Mathematics*, vol. 6, pp. 393–404.

Remark 1.38: Choosing the index

There are a couple of ways to choose the index I_t, i.e., which example we are going to deal with at iteration t:

- cyclic: $I_t = (I_{t-1} + 1) \mod n$.
- chaotic: $\exists \tau \ge n$ so that for any $t \in \mathbb{N}$, $\{1, 2, \dots, n\} \subseteq \{I_t, I_{t+1}, \dots, I_{t+\tau-1}\}$.
- randomized: $I_t = i$ with probability p_i. A typical choice is $p_i = \|\mathbf{a}_i\|_2^2 / \sum_i \|\mathbf{a}_i\|_2^2$.
- permuted: in each epoch randomly permute $\{1, 2, \dots, n\}$ and then follow cyclic.
- maximal distance: $\frac{(\mathbf{a}_{I_t}^\top \mathbf{w} - c_{I_t})^+}{\|\mathbf{a}_{I_t}\|_2} = \max_{i=1,\dots,n} \frac{(\mathbf{a}_i^\top \mathbf{w} - c_i)^+}{\|\mathbf{a}_i\|_2}$ (break ties arbitrarily).
- maximal residual: $(\mathbf{a}_{I_t}^\top \mathbf{w} - c_{I_t})^+ = \max_{i=1,\dots,n} (\mathbf{a}_i^\top \mathbf{w} - c_i)^+$ (break ties arbitrarily).

Remark 1.39: Understanding perceptron mathematically

Let us define a polyhedral cone $\mathrm{cone}(\mathsf{A}) := \{\mathsf{A}\boldsymbol{\lambda} : \boldsymbol{\lambda} \ge \mathbf{0}\}$ whose dual is $[\mathrm{cone}(\mathsf{A})]^* = \{\mathbf{w} : \mathsf{A}^\top \mathbf{w} \ge \mathbf{0}\}$. The linear separability assumption in Definition 1.24 can be written concisely as $\mathrm{int}([\mathrm{cone}(\mathsf{A})]^*) \ne \emptyset$, but it is known in convex analysis that the dual cone $[\mathrm{cone}(\mathsf{A})]^*$ has nonempty interior iff $\mathrm{int}([\mathrm{cone}(\mathsf{A})]^*) \cap \mathrm{cone}(\mathsf{A}) \ne \emptyset$, i.e., iff there exists some $\boldsymbol{\lambda} \ge \mathbf{0}$ so that $\mathbf{w} = \mathsf{A}\boldsymbol{\lambda}$ satisfies $\mathsf{A}^\top \mathbf{w} > \mathbf{0}$. Slightly perturb $\boldsymbol{\lambda}$ we may assume w.l.o.g. $\boldsymbol{\lambda}$ is rational. Perform scaling if necessary we may even assume $\boldsymbol{\lambda}$ is integral. The perceptron algorithm gives a constructive way to find such an *integral* $\boldsymbol{\lambda}$ (hence also $\mathbf{w}$).

Remark 1.40: Variants

We mention some interesting variants of the perceptron algorithm: (Cesa-Bianchi et al. 2005; Dekel et al. 2008; Soheili and Peña 2012; Soheili and Peña 2013).

Cesa-Bianchi, Nicolò, Alex Conconi, and Claudio Gentile (2005). "A Second-Order Perceptron Algorithm". *SIAM Journal on Computing*, vol. 34, no. 3, pp. 640–668.

Dekel, Ofer, Shai Shalev-Shwartz, and Yoram Singer (2008). "The Forgetron: A Kernel-based Perceptron on A Budget". *SIAM Journal on Computing*, vol. 37, no. 5, pp. 1342–1372.

Soheili, Negar and Javier Peña (2012). "A Smooth Perceptron Algorithm". *SIAM Journal on Optimization*, vol. 221, no. 2, pp. 728–737.

— (2013). "A Primal–Dual Smooth Perceptron–von Neumann Algorithm". In: *Discrete Geometry and Optimization*, pp. 303–320.

Remark 1.41: More refined results

A primal-dual version is given in (Spingarn 1985; Spingarn 1987), which solves the general system of linear inequalities in finite time (provided a solution exists in the interior).

For a more refined analysis of the perceptron algorithm and related, see (Goffin 1980; Goffin 1982; Ramdas and Peña 2016; Peña et al. 2021).

Spingarn, Jonathan E. (1985). "A primal-dual projection method for solving systems of linear inequalities". *Linear Algebra and its Applications*, vol. 65, pp. 45–62.

— (1987). "A projection method for least-squares solutions to overdetermined systems of linear inequalities". *Linear Algebra and its Applications*, vol. 86, pp. 211–236.

Goffin, J. L. (1980). "The relaxation method for solving systems of linear inequalities". *Mathematis of Operations Research*, vol. 5, no. 3, pp. 388–414.

— (1982). "On the non-polynomiality of the relaxation method for systems of linear inequalities". *Mathematical Programming*, vol. 22, pp. 93–103.

Ramdas, Aaditya and Javier Peña (2016). "Towards a deeper geometric, analytic and algorithmic understanding of margins". *Optimization Methods and Software*, vol. 31, no. 2, pp. 377–391.

Peña, Javier F., Juan C. Vera, and Luis F. Zuluaga (2021). "New characterizations of Hoffman constants for systems of linear constraints". *Mathematical Programming*, vol. 187, pp. 79–109.

Remark 1.42: Solving conic linear system

The perceptron algorithm can be used to solve linear programs (whose KKT conditions form a system of linear inequalities) and more generally conic linear programs, see (Dunagan and Vempala 2008; Belloni et al. 2009; Peña and Soheili 2016; Peña and Soheili 2017).

Dunagan, John and Santosh Vempala (2008). "A simple polynomial-time rescaling algorithm for solving linear programs". *Mathematical Programming*, vol. 114, no. 1, pp. 101–114.

Belloni, Alexandre, Robert M. Freund, and Santosh Vempala (2009). "An Efficient Rescaled Perceptron Algorithm for Conic Systems". *Mathematics of Operations Research*, vol. 34, no. 3, pp. 621–641.

Peña, Javier and Negar Soheili (2016). "A deterministic rescaled perceptron algorithm". *Mathematical Programming*, vol. 155, pp. 497–510.

— (2017). "Solving Conic Systems via Projection and Rescaling". *Mathematical Programming*, vol. 166, no. 1-2, pp. 87–111.

Remark 1.43: Herding

Some interesting applications of the perceptron algorithm and its boundedness can be found in (Gelfand et al. 2010; Harvey and Samadi 2014), (Briol et al. 2015; Briol et al. 2019; Chen et al. 2016), and (Phillips and Tai 2020; Dwivedi and Mackey 2021; Turner et al. 2021).

Gelfand, Andrew, Yutian Chen, Laurens Maaten, and Max Welling (2010). "On Herding and the Perceptron Cycling Theorem". In: *Advances in Neural Information Processing Systems*.

Harvey, Nick and Samira Samadi (2014). "Near-Optimal Herding". In: *Proceedings of The 27th Conference on Learning Theory*, pp. 1165–1182.

Briol, François-Xavier, Chris J. Oates, Mark Girolami, Michael A. Osborne, and Dino Sejdinovic (2015). "Frank-Wolfe Bayesian Quadrature: Probabilistic Integration with Theoretical Guarantees". In: *Advances in Neural Information Processing Systems*.

— (2019). "Probabilistic Integration: A Role in Statistical Computation?" *Statistical Science*, vol. 34, no. 1, pp. 1–22.

Chen, Yutian, Luke Bornn, Nando de Freitas, Mareija Eskelin, Jing Fang, and Max Welling (2016). "Herded Gibbs Sampling". *Journal of Machine Learning Research*, vol. 17, no. 10, pp. 1–29.

Phillips, Jeff M. and Wai Ming Tai (2020). "Near-Optimal Coresets of Kernel Density Estimates". *Discrete & Computational Geometry*, vol. 63, pp. 867–887.

Dwivedi, Raaz and Lester Mackey (2021). "Kernel Thinning". In: *Proceedings of Thirty Fourth Conference on Learning Theory*, pp. 1753–1753.

Turner, Paxton, Jingbo Liu, and Philippe Rigollet (2021). "A Statistical Perspective on Coreset Density Estimation". In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pp. 2512–2520.