Deep Learning

Cristian Perez Jensen

October 26, 2024

Note that these are not the official lecture notes of the course, but only notes written by a student of the course. As such, there might be mistakes. The source code can be found at `github.com/cristianpjensen/eth-cs-notes`. If you find a mistake, please create an issue or open a pull request.

Contents

List of symbols

$\doteq$	Equality by definition
$\overset{!}{=}$	Conditional equality
$\approx$	Approximate equality
$\propto$	Proportional to
$\mathbb{N}$	Set of natural numbers
$\mathbb{R}$	Set of real numbers
$i : j$	Set of natural numbers between i and j. *I.e.*, $\{i, i+1, \ldots, j\}$
$f : A \to B$	Function f that maps elements of set A to elements of set B
$\mathbb{1}\{\text{predicate}\}$	Indicator function (1 if predicate is true, otherwise 0)

$\boldsymbol{v} \in \mathbb{R}^n$	n-dimensional vector
$\boldsymbol{M} \in \mathbb{R}^{m \times n}$	$m \times n$ matrix
$\boldsymbol{M}^\top$	Transpose of matrix $\boldsymbol{M}$
$\boldsymbol{M}^{-1}$	Inverse of matrix $\boldsymbol{M}$
$\det(\boldsymbol{M})$	Determinant of $\boldsymbol{M}$

$\frac{\mathrm{d}}{\mathrm{d}x} f(x)$	Ordinary derivative of $f(x)$ w.r.t. x at point $x \in \mathbb{R}$
$\frac{\partial}{\partial x} f(\boldsymbol{x})$	Partial derivative of $f(\boldsymbol{x})$ w.r.t. x at point $\boldsymbol{x} \in \mathbb{R}^n$
$\boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x}) \in \mathbb{R}^n$	Gradient of $f : \mathbb{R}^n \to \mathbb{R}$ at point $\boldsymbol{x} \in \mathbb{R}^n$
$\boldsymbol{\nabla}^2_{\boldsymbol{x}} f(\boldsymbol{x}) \in \mathbb{R}^{n \times n}$	Hessian of $f : \mathbb{R}^n \to \mathbb{R}$ at point $\boldsymbol{x} \in \mathbb{R}^n$

1 Connectionism

1.1 McCulloch-Pitts neuron

One of the first approaches to modeling functions of nervous functions with an abstract mathematical model is the McCulloch-Pitts neuron [McCulloch and Pitts, 1943]. It treats neurons as linear threshold elements, which receive and integrate a large number of inputs and produce a Boolean output. More specifically, it receives $\boldsymbol{x} \in \{0,1\}^n$ as input and has $\boldsymbol{\sigma} \in \{-1,1\}^n, \theta \in \mathbb{R}$ as parameters. Its transfer function is formalized as

$$f[\boldsymbol{\sigma}, \theta](\boldsymbol{x}) = \mathbb{1}\{\boldsymbol{\sigma}^\top \boldsymbol{x} \geq \theta\}.$$

The synapses $\boldsymbol{\sigma}$ are inhibitory if -1 and excitatory if $+1$. However, the problem with this model is that it does not specify how to set or adjust its parameters.

1.2 Perceptron

The perceptron [Rosenblatt, 1958] is the first model to perform supervised learning, where patterns are represented as feature vectors $\boldsymbol{x} \in \mathbb{R}^d$ and have binary class memberships $y \in \{-1,+1\}$. Rosenblatt [1958] proposed to use a linear threshold unit with synaptic weights $\boldsymbol{w} \in \mathbb{R}^d$ and threshold $b \in \mathbb{R}$,

$$f[\boldsymbol{w}, b](\boldsymbol{x}) = \operatorname{sgn}\Big(\boldsymbol{w}^\top \boldsymbol{x} + b\Big),$$

where

$$\operatorname{sgn}(z) \doteq \begin{cases} +1 & z > 0 \\ 0 & z = 0 \\ -1 & z < 0. \end{cases}$$

This model implicitly induces a decision boundary, where

$$\boldsymbol{w}^\top \boldsymbol{x} + b \stackrel{!}{=} 0 \iff \frac{\boldsymbol{w}^\top \boldsymbol{x}}{\|\boldsymbol{w}\|} + \frac{b}{\|\boldsymbol{w}\|} \stackrel{!}{=} 0.$$

The perceptron thus models the decision boundary as a hyperplane in $\mathbb{R}^n$ with normal vector $w/\|w\|$ and $-b/\|w\|$ is the signed distance of the hyperplane to the origin.[1] Furthermore, we can formalize how bad/good the model is for a data point by the signed distance function,

$$\gamma[\boldsymbol{w}, b](\boldsymbol{x}, y) = \frac{y\big(\boldsymbol{w}^\top \boldsymbol{x} + b\big)}{\|\boldsymbol{w}\|}.$$

The sign of $\gamma(\cdot,\cdot)$ encodes the correctness of the classification. The following is a short proof of this fact,

$$\begin{aligned} f[\boldsymbol{w}, b](\boldsymbol{x}) = y &\iff \operatorname{sgn}\Big(\boldsymbol{w}^\top \boldsymbol{x} + b\Big) = y \\ &\iff \operatorname{sgn}\Big(y\Big(\boldsymbol{w}^\top \boldsymbol{x} + b\Big)\Big) = 1 \\ &\iff \operatorname{sgn}(\gamma[\boldsymbol{w}, b](\boldsymbol{x}, y)) = 1 \\ &\iff \gamma[\boldsymbol{w}, b](\boldsymbol{x}, y) > 0. \end{aligned}$$

[1] In Hesse normal form, a hyperplane is formulated by

$$\boldsymbol{n}^\top \boldsymbol{x} - d = 0,$$

where $\boldsymbol{n}$ is a unit vector and d is the shortest distance of the hyperplane to the origin.

We define the margin of a classifier on training data $\mathcal{S}$ as the minimum signed distance,

$$\gamma[\boldsymbol{w}, b](\mathcal{S}) = \min_{(\boldsymbol{x},y)\in\mathcal{S}} \gamma[\boldsymbol{w}, b](\boldsymbol{x}, y).$$

If $\gamma[\boldsymbol{w}, b](\mathcal{S}) > 0$, then the dataset has been linearly separated by a hyperplane, formed by the parameters, *i.e.*, all classifications are correct.

The version space—see Figure 1.2—is defined as the set of all model parametrizations that correctly classify the data,

$$\mathcal{V}(\mathcal{S}) \doteq \{(\boldsymbol{w}, b) \mid \gamma[\boldsymbol{w}, b](\mathcal{S}) > 0\} \subseteq \mathbb{R}^{n+1}.$$

Hence, $\mathcal{S}$ is linearly separable if and only if $\mathcal{V}(\mathcal{S}) \neq \emptyset$. Adding data points to the dataset can only shrink the version space.

The perceptron algorithm. The groundbreaking aspect of [Rosenblatt, 1958] is that it specified how to iteratively adjust the weights to provably find a solution for a linearly separable dataset.[2] Given a dataset $\mathcal{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{s}$, the perceptron algorithms aims to find some solution $(\boldsymbol{w}, b) \in \mathcal{V}(\mathcal{S})$. Note that this means that it does not aim to find classifiers with small error if $\mathcal{V}(\mathcal{S}) = \emptyset$.

The perceptron algorithm is a mistake-driven algorithm, meaning that it will only consider data points that are misclassified by the current parameters. Given a misclassified data point $(\boldsymbol{x}, y) \in \mathcal{S}$, it has the following update rule,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$$
$$b \leftarrow b + y.$$

We keep going through the dataset until every data point is correctly classified—see Algorithm 1. Note that this algorithm will never converge if $\mathcal{S}$ is not linearly separable.

Proof of convergence. In order to prove convergence of the perceptron algorithm for linearly separable data, we will assume that there is no bias. We denote the weights after t updates of the perceptron algorithm (ignoring correctly classified samples) as $\boldsymbol{w}_t$.

We will first need the following two lemmas,

Lemma 1.1. Let $\boldsymbol{w} \in \mathbb{R}^n$ with $\|\boldsymbol{w}\| = 1$ and $\gamma \doteq \gamma[\boldsymbol{w}](\mathcal{S}) > 0$. (*I.e.*, $\mathcal{S}$ is γ-separable.) Then,

$$\boldsymbol{w}^\top \boldsymbol{w}_t \geq t\gamma.$$

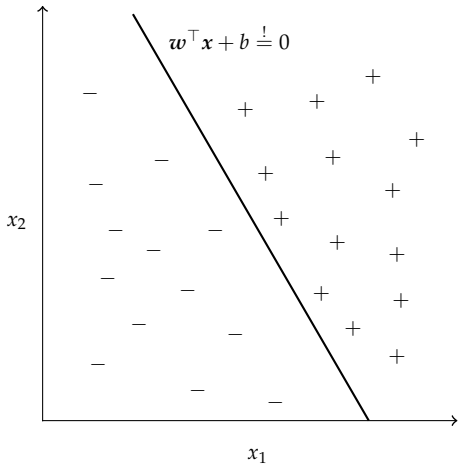

Figure 1.1. Linear separability of negative and positive data points.

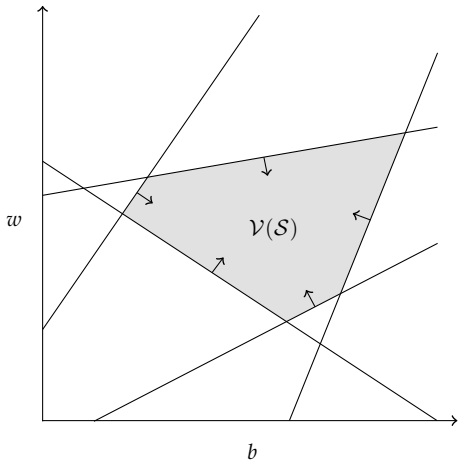

Figure 1.2. In this version space, every line represents a data point's halfspace in which it is correctly classified. As can be seen, adding data points can only shrink the version space.

[2] A solution is defined as any parameters that correctly classify all data points.

$\boldsymbol{w} \leftarrow \mathbf{0}$
$b \leftarrow 0$
mistake $\leftarrow$ `true`
while mistake $=$ `true` **do**
  mistake $\leftarrow$ `false`
  for $(\boldsymbol{x}, y) \in \mathcal{S}$ **do**
    if $f[\boldsymbol{w}, b](\boldsymbol{x}) \neq y$ **then**
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$
      $b \leftarrow b + y$
      mistake $\leftarrow$ `true`
    end if
  end for
end while
return $(\boldsymbol{w}, b)$

Algorithm 1. The perceptron algorithm.

Proof. This can easily be shown by a recursion,

$$\begin{aligned}
\boldsymbol{w}^\top \boldsymbol{w}_{t+1} &= \boldsymbol{w}^\top (\boldsymbol{w}_t + y\boldsymbol{x}) && \text{Perceptron update.} \\
&= \boldsymbol{w}^\top \boldsymbol{w}_t + y\boldsymbol{w}^\top \boldsymbol{x} && \text{Linearity.} \\
&= \boldsymbol{w}^\top \boldsymbol{w}_t + \gamma[\boldsymbol{w}](\boldsymbol{x}) && \|\boldsymbol{w}\| = 1. \\
&\geq \boldsymbol{w}^\top \boldsymbol{w}_t + \gamma. && \gamma = \min_{\boldsymbol{x},y} \gamma[\boldsymbol{w}](\boldsymbol{x}, y) \leq \gamma[\boldsymbol{w}](\boldsymbol{x}, y), \forall \boldsymbol{x}, y.
\end{aligned}$$

Now, it is easy to show the result by induction, starting from $\boldsymbol{w}_0 = \mathbf{0}$. ■

Lemma 1.2. Let $R \doteq \max_{\boldsymbol{x} \in \mathcal{S}} \|\boldsymbol{x}\|$, then

$$\|\boldsymbol{w}_t\| \leq R\sqrt{t}.$$

Proof. This can easily be shown by a recursion,

$$\begin{aligned}
\|\boldsymbol{w}_{t+1}\|^2 &= \|\boldsymbol{w}_t + y\boldsymbol{x}\|^2 && \text{Perceptron update.} \\
&= \|\boldsymbol{w}_t\|^2 + \|y\boldsymbol{x}\|^2 + 2y\boldsymbol{w}_t^\top \boldsymbol{x} && \text{Cosine theorem.} \\
&\leq \|\boldsymbol{w}_t\|^2 + \|\boldsymbol{x}\|^2 && \text{The perceptron update condition is } \gamma[\boldsymbol{w}](\boldsymbol{x}, y) \leq 0. \\
&\leq \|\boldsymbol{w}_t\|^2 + R^2.
\end{aligned}$$

The claim follows by induction, starting from $\boldsymbol{w}_0 = \mathbf{0}$, and taking the square root. ■

Theorem 1.3 ([Novikoff, 1962])**.** Let $\mathcal{S}$ be γ-separable and $R \doteq \max_{\boldsymbol{x} \in \mathcal{S}} \|\boldsymbol{x}\|$, then the perceptron algorithm converges in less than $\lfloor R^2/\gamma^2 \rfloor$ steps.

Proof. By Lemmas 1.1 and 1.2, we have the following inequality,

$$1 \geq \cos \angle(\boldsymbol{w}, \boldsymbol{w}_t) = \frac{\boldsymbol{w}^\top \boldsymbol{w}_t}{\|\boldsymbol{w}_t\|} \geq \frac{t\gamma}{R\sqrt{t}} = \sqrt{t}\frac{\gamma}{R},$$

where $\boldsymbol{w} \in \mathcal{V}(\mathcal{S})$. Hence,

$$t \leq \frac{R^2}{\gamma^2}.$$

Thus, the number of updates is upper bounded. When there are no more updates, there are no more mistakes—we only make updates when we find a mistake. Hence, $\boldsymbol{w}_t$ will have converged. Since t is integer, this bound is $\lfloor R^2/\gamma^2 \rfloor$. ■

This theorem does not only guarantee convergence of the perceptron algorithm, but also relates the separation margin γ to the number of steps necessary for convergence. If γ is large, it should be easier to find parameters that classify all data points correctly than if γ is small, because then you have to be very precise; see Figure 1.1.

However, the problem with this approach is that it requires linear separability of the data, which is not fulfilled for simple problems like the XOR,

$$\mathcal{S} = \left\{ \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, 1 \right), \left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, 1 \right), \left(\begin{bmatrix} 0 \\ 1 \end{bmatrix}, -1 \right), \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, -1 \right) \right\}.$$

Number of unique linear classifications. Assume that we are given a dataset $\mathcal{S} \subset \mathbb{R}^n$ of s points, then we define the set of possible linear classifications of this dataset as,

$$\mathcal{C}(\mathcal{S}, n) \doteq \left| \left\{ y \in \{-1, +1\}^s \;\middle|\; \exists_{\boldsymbol{w} \in \mathbb{R}^n} \forall_{i \in [s]} \left[y_i \left(\boldsymbol{w}^\top \boldsymbol{x}_i \right) > 0 \right] \right\} \right|.$$

We assume points to be in general position, which means that any subset $\Xi \subseteq \mathcal{S}$ with $|\Xi| \leq n$ is linearly independent.[3]

[3] This is a very weak condition.

Theorem 1.4 ([Cover, 1965])**.** Given s n-dimensional points in general position,

$$\mathcal{C}(s+1, n) = 2 \sum_{i=0}^{n-1} \binom{s}{i}.$$

Proof. It is easy to show that the initial values are

$$\mathcal{C}(1, n) = 2, \quad \mathcal{C}(s, 1) = 2.$$

Consider a realizable classification of s points. *I.e.*, any classification of all $\boldsymbol{x} \in \mathcal{S}$ that is linearly separable. This classification has a non-empty version space $\mathcal{V}$. Let $\boldsymbol{x}_{s+1}$ be a pattern that we add to $\mathcal{S}$. This gives us two new version spaces,

$$\mathcal{V}^+ \doteq \mathcal{V} \cap \left\{ \boldsymbol{w} \;\middle|\; \boldsymbol{w}^\top \boldsymbol{x}_{s+1} > 0 \right\}, \quad \mathcal{V}^- \doteq \mathcal{V} \cap \left\{ \boldsymbol{w} \;\middle|\; -\boldsymbol{w}^\top \boldsymbol{x}_{s+1} > 0 \right\},$$

There are two situations,

1. $\mathcal{V}^+$ and $\mathcal{V}^-$ are non-empty. Hence, $\boldsymbol{x}_{s+1}$ can be classified as either $+1$ or -1. This is the case if and only if there is a $\boldsymbol{w} \in \mathcal{V}$ such that $\boldsymbol{w}^\top \boldsymbol{x}_{s+1} = 0$.[4] Recall that we want to know the number of classifications of this new dataset $\mathcal{S} \cup \{\boldsymbol{x}_{s+1}\}$. For any classification of $\mathcal{S}$ that is in this situation, we can make two new classifications; one where $\boldsymbol{x}_{s+1}$ is classified $+1$ or -1. There are $\mathcal{C}(s, n-1)$ such that classifications, because the constraint on $\boldsymbol{w}$ makes the problem effectively $(n-1)$-dimensional with s data points. Hence, we gain $2\mathcal{C}(s, n-1)$ classifications;
2. $\mathcal{V}^+$ is non-empty and $\mathcal{V}^-$ is empty or $\mathcal{V}^+$ is empty and $\mathcal{V}^-$ is non-empty. In this case, we would only be able to create one new classification for each existing classification, and there are $\mathcal{C}(s, n) - \mathcal{C}(s, n-1)$ such original classifications. Hence, we gain $\mathcal{C}(s, n) - \mathcal{C}(s, n-1)$ classifications.

[4] Because then we would be able to shift the hyperplane, formed by $\boldsymbol{w}$, infinitesimally to allow arbitrary classification of $\boldsymbol{x}_{s+1}$ while keeping all other classifications the same.

In conclusion, in total we can create

$$\begin{aligned}\mathcal{C}(s+1, n) &= \mathcal{C}(s, n) - \mathcal{C}(s, n-1) + 2 \cdot \mathcal{C}(s, n-1) \\ &= \mathcal{C}(s, n) + \mathcal{C}(s, n-1)\end{aligned}$$

classifications of $s+1$ data points. The claim follows by induction using Pascal's identity. ■

It turns out that after $s = 2n$, there is a steep decrease in number of linear classifications, quickly moving toward 0.

1.3 *Parallel distributed processing*

The philosophy behind modern machine learning comes from parallel distributed processing (PDP) [Rumelhart et al., 1986]. The elements of PDP are the following,

1. A set of processing units with states of activation, which are the basic building blocks that models consist of;
2. Output functions for each unit, which define how the output of the units is computed;
3. A pattern of connectivity between units, which defines how the units interact with each other;
4. Propagation rules for propagating patterns of activity, which makes the dependence of the units explicit;
5. Activation functions for units, which make the model more expressive;
6. A learning rule to modify connectivity based on experience, which the training data is used for;
7. An environment within which the system must operate, which is formalized by a loss function.

All of these elements are design choices that can be changed and experimented with. The fact that we still use this wording says much about the impact of PDP.

1.4 Hopfield networks

The Hopfield model [Hopfield, 1982] defines a parameterized energy function via second-order interactions between n binary neurons,

$$H(\boldsymbol{X}) \doteq -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_i j X_i X_j + \sum_{i=1}^{n} b_i X_i, \quad \boldsymbol{X} \in \{-1,+1\}^n.$$

The couplings w_{ij} quantify the interaction strength between neurons and the biases b_i act as thresholds. We constrain the weights such that

$$w_{ii} = 0, w_{ij} = w_{ji}, \quad \forall i,j \in [n].$$

Hopfield networks follow a simple dynamic,

$$X_i \leftarrow \begin{cases} +1 & H([\ldots, X_{i-1}, +1, X_{i+1}, \ldots]) \leq H([\ldots, X_{i-1}, -1, X_{i+1}, \ldots]) \\ -1 & \text{otherwise.} \end{cases}$$

Hence, X_i becomes the value that minimizes the energy function, given the rest of the state. In practice, we do not need to evaluate the full energy function for the update—we only need the effective field per neuron,

$$H_i \doteq \sum_{j=1}^{n} w_{ij} X_j - b_i.$$

Then, updates can equivalently be expressed as

$$X_i \leftarrow \operatorname{sgn}(H_i), \quad \operatorname{sgn}(z) = \begin{cases} +1 & z \geq 0 \\ -1 & z < 0. \end{cases}$$

The goal of Hopfield networks is to use the update dynamics to evolve noisy stimulus toward a target pattern. For example, we might want noisy greyscale images to converge to images of numbers 0–9. Given a set of patterns that we wish to memorize,

$$\mathcal{S} \subseteq \{-1,+1\}^n,$$

Hebbian learning involves setting the weights as outer products,

$$w_{ij} = \frac{1}{n}\sum_{t=1}^{s} x_{t,i} x_{t,j} \implies \boldsymbol{W} = \frac{1}{n}\sum_{t=1}^{s} \boldsymbol{x}_t \boldsymbol{x}_t^\top.$$

Intuitively, neurons that are frequently in the same state reinforce each other (positive coupling), whereas neurons that are frequently in opposite states repel each other (negative coupling).

The minimal requirement of considering a pattern as memorized is that it is meta-stable, *i.e.*, when in the state of a pattern, the update rule will not make any updates,

$$x_{t,i} \stackrel{!}{=} \operatorname{sgn}\left(\sum_{j=1}^{n} w_{ij} x_{t,j}\right).$$

Expanding this with the couplings from Hebbian learning, we get

$$\begin{aligned} x_{t,i} &\stackrel{!}{=} \operatorname{sgn}\left(\frac{1}{n}\sum_{j=1}^{n}\sum_{r=1}^{s} x_{r,i} x_{r,j} x_{t,j}\right) \\ &= \operatorname{sgn}\left(x_{t,i} + \underbrace{\frac{1}{n}\sum_{j=1}^{n}\sum_{r\neq t}^{s} x_{r,i} x_{r,j} x_{t,j}}_{\doteq C_{t,i}}\right). \end{aligned}$$

$C_{t,i}$ is the cross-talk between patterns, and ideally $|C_{t,i}| < 1$, for all patterns $t \in [s]$ and indices $i \in [n]$, because then the minimal requirement is fulfilled.

If we assume that the patterns have i.i.d. random signs and we look at the limit $n \to \infty$, then we have

$$C_{t,i} \sim \mathcal{N}\left(0, \frac{s}{n}\right).$$

The probability of a single sign being flipped is then

$$P[-x_{t,i} C_{t,i} \geq 1] \approx \int_{1}^{\infty} \exp\left(-\frac{nz^2}{2s}\right) \mathrm{d}z = \frac{1}{2}\left(1 - \operatorname{erf}\left(\sqrt{n/2s}\right)\right).$$

Hence, the ratio s/n controls the asymptotic error rate. At $s/n \approx 0.138$, a phase transition occurs, beyond which an avalanche of errors occur. Requiring a pattern to be retrieved with high probability, one gets a sublinear capacity bound of

$$s \leq \frac{n}{2 \log_2 n}.$$

Recently, research has been done on increasing the capacity of Hopfield networks by making use of higher-order energy functions [Krotov and Hopfield, 2016, Demircigil et al., 2017]. The increased capacity is the consequence of increased number of local minima in complex cost functions. Furthermore, Ramsauer et al. [2020] have investigated a connection between Hopfield networks and transformers.

2 *Feedforward networks*

TODO

References

Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, (3):326–334, 1965.

Mete Demircigil, Judith Heusel, Matthias Löwe, Sven Upgang, and Franck Vermet. On a model of associative memory with huge storage capacity. *Journal of Statistical Physics*, 168:288–299, 2017.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

Dmitry Krotov and John J Hopfield. Dense associative memory for pattern recognition. *Advances in neural information processing systems*, 29, 2016.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5: 115–133, 1943.

Albert BJ Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622. New York, NY, 1962.

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, et al. Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*, 2020.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

David E Rumelhart, Geoffrey E Hinton, James L McClelland, et al. A general framework for parallel distributed processing. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1(45-76): 26, 1986.