Training One Neuron

Deep Learning (ENGG*6600*01)

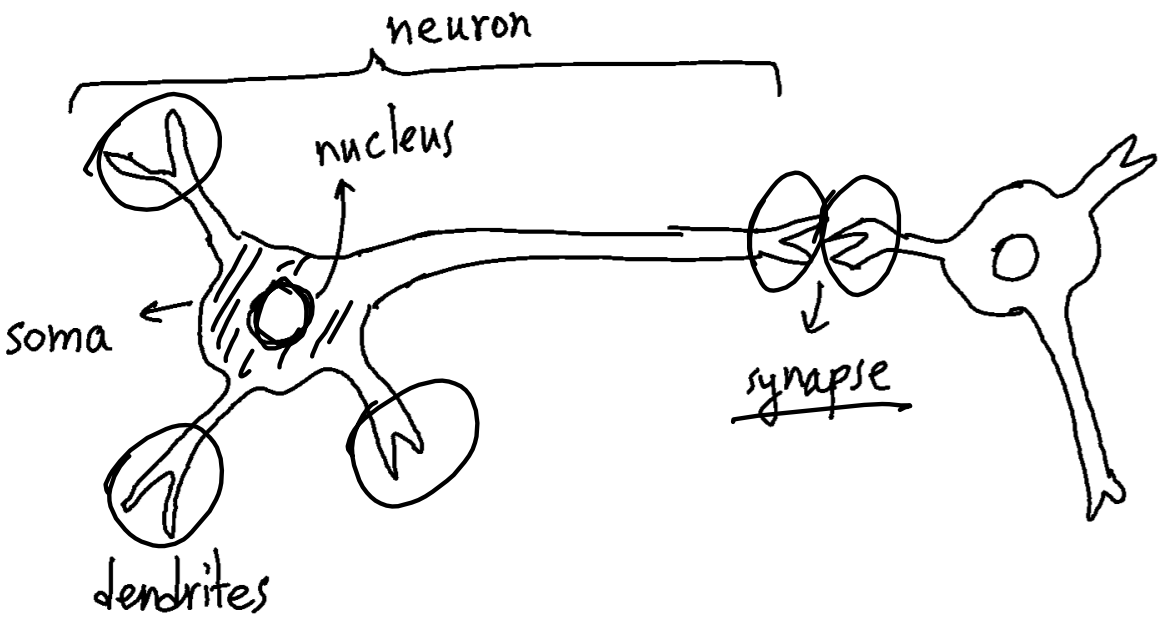
School of Engineering,
University of Guelph, ON, Canada

Course Instructor: Benyamin Ghojogh
Summer 2023

Neuron and The McCulloch-Pitts Model

Biological Neuron

- **Nucleus**: The nucleus in the neuron cell
- **Soma**: Soma is the cell body of neuron containing the nucleus
- **Dendrite**: the branched protoplasmic extensions of a nerve cell that propagate the electrochemical stimulation received from other neural cells to the cell body, or soma, of the neuron
- **Synapse**: the small gap between the dendrites of connecting neurons.

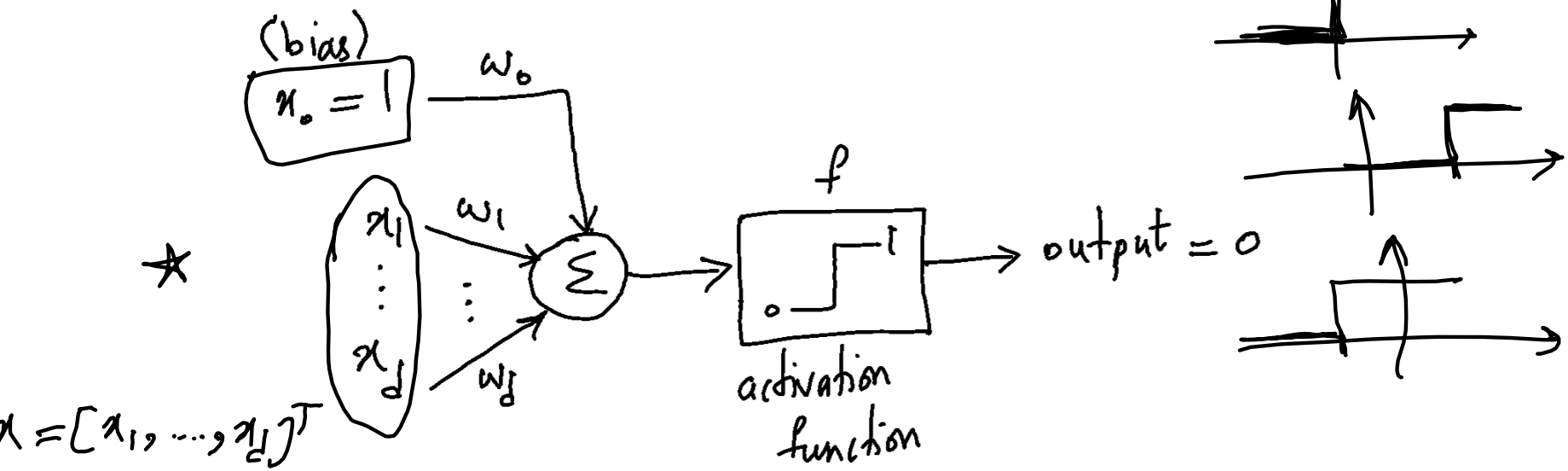


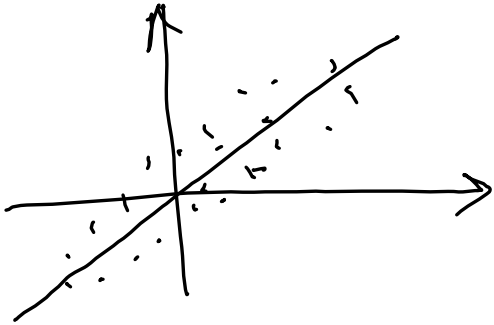
McCulloch-Pitts Model

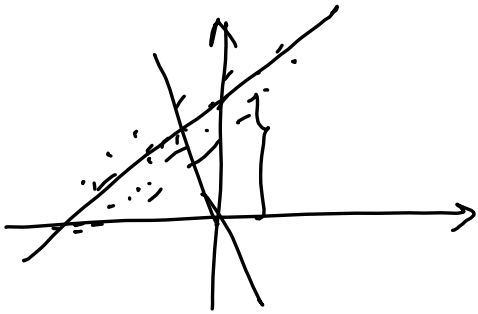
- in 1943: First model of neuron was invented by McCulloch (physiologist) and Pitts (logician) [1]. It was later named the **McCulloch-Pitts model**.
- The model had two inputs and a single output.
- A neuron would not activate if only one of the inputs was active.
- Weights for each input were equal and fixed and the output was binary.
- Until inputs summed up to a certain threshold level, output would remain zero.
- This model is known nowadays as a **logical gate**, such as AND, OR, NOT, etc.
- Formula:

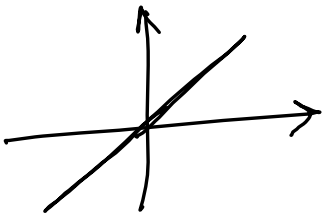
$$o = f\Big(\sum_{j=1}^{d} w_j x_j + w_0\Big) = f\Big(\sum_{j=0}^{d} w_j x_j\Big), \tag{1}$$

where $x_0 = 1$ is for bias (intercept).



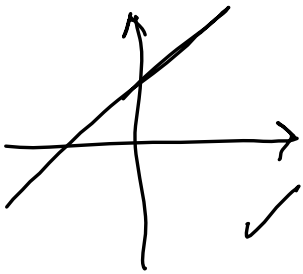






linear

$a^T x$



affine

$a^T x + b$

Perceptron

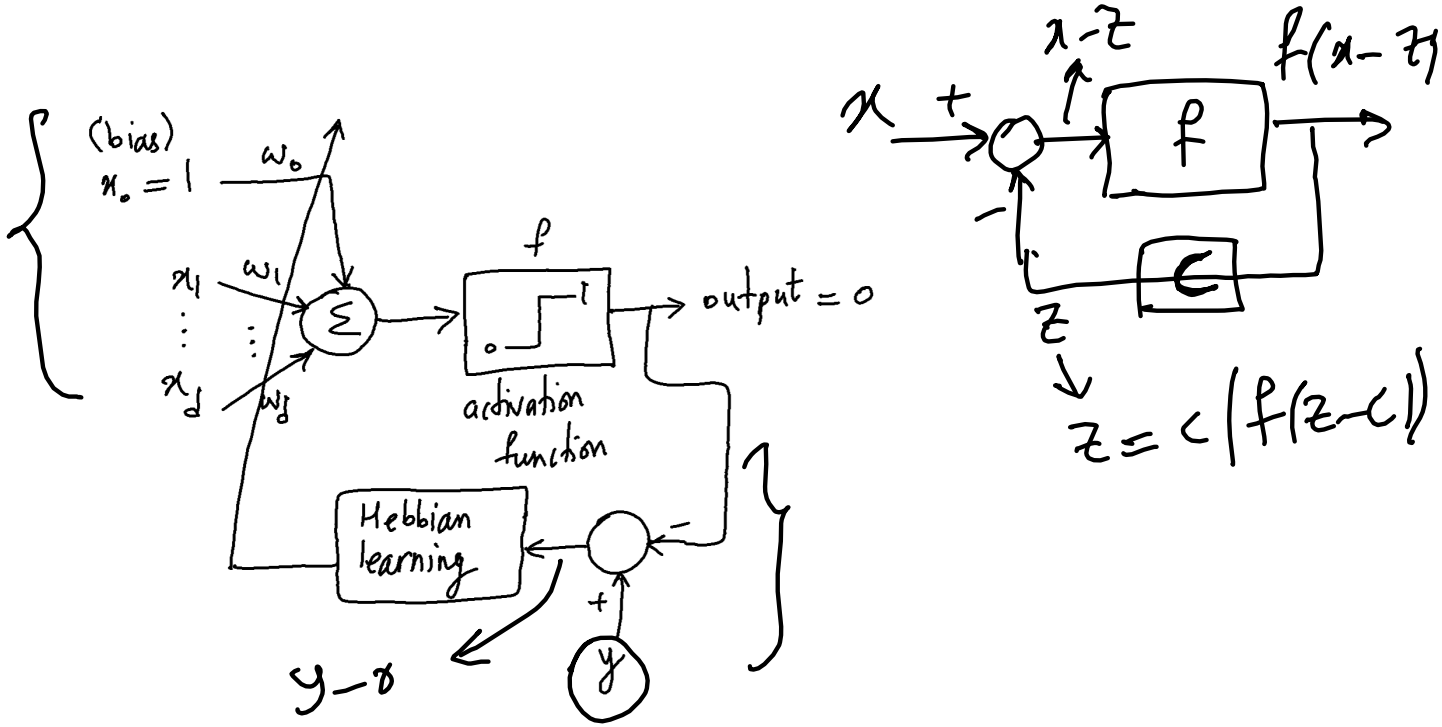
Hebbian learning

- **Hebbian theory** is a neuropsychological theory claiming that an increase in synaptic efficacy arises from a presynaptic cell's repeated and persistent stimulation of a postsynaptic cell.
- In simple words: the more two adjacent neurons are activated (fired) together, the stronger the connection becomes between them.
- **Hebbian learning** - proposed by Donald Hebb in his 1949 book "The Organization of Behavior" [2]:

Perceptron

- **Perceptron** tried to learn by neuron, as done in brain.
- Perceptron is the building block of nowadays' neural networks.
- Perceptron was implemented by Rosenblatt (physiologist) in 1958, at Cornell Aeronautical Laboratory [3].
- It was for **binary classification**. It was useful because:
  - Binary logic could do computer operations.
  - Even when we have multiple classes, we can consider pairs of classes.
- Rosenblatt randomly connected Perceptrons and changed the weights in order to achieve "learning". In later attempts, Hebbian learning [2] was used for learning in Perceptron.
- In a 1958 press conference organized by the US Navy, Rosenblatt gave a speech about the perceptron. Afterwards, New York Times reported the Perceptron to be "The embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself, and be conscious of its existence." [4]
- They thought they have solved AI!

Perceptron



- **Hebbian learning** [2]:

$$\begin{cases} w_j := w_j + \eta(y_i - o_i)x_{ij}, & \forall j \in \{1, \ldots, d\}, \forall i \in \{1, \ldots, n\}, \\ w_0 := w_0 + \eta(y_i - o_i), & \forall i \in \{1, \ldots, n\}, \end{cases} \tag{2}$$

where $\eta > 0$ is the learning rate, $\boldsymbol{x}_i = [x_{i1}, \ldots, x_{id}]^\top \in \mathbb{R}^d$ is the i-th input data, y_i is its target label and o_i is the corresponding output of Perceptron for that input data.

- We can merge these two into one formula:

$$w_j := w_j + \eta(y_i - o_i)x_{ij}, \quad \forall j \in \{0, \ldots, d\}, \forall i \in \{1, \ldots, n\}, \tag{3}$$

where $x_{i0} := 1$.

Perceptron

$\lfloor \frac{n}{b} \rfloor$ batches

- In batch learning of Hebbian learning:

$$\begin{cases} w_j := w_j + \eta \underbrace{\sum_{i=1}^{b} (y_i - o_i) x_{ij}}_{b}, & \forall j \in \{1, \ldots, d\}, \\ w_0 := w_0 + \eta \sum_{i=1}^{b} (y_i - o_i), \end{cases} \tag{4}$$

  where b is the batch size.

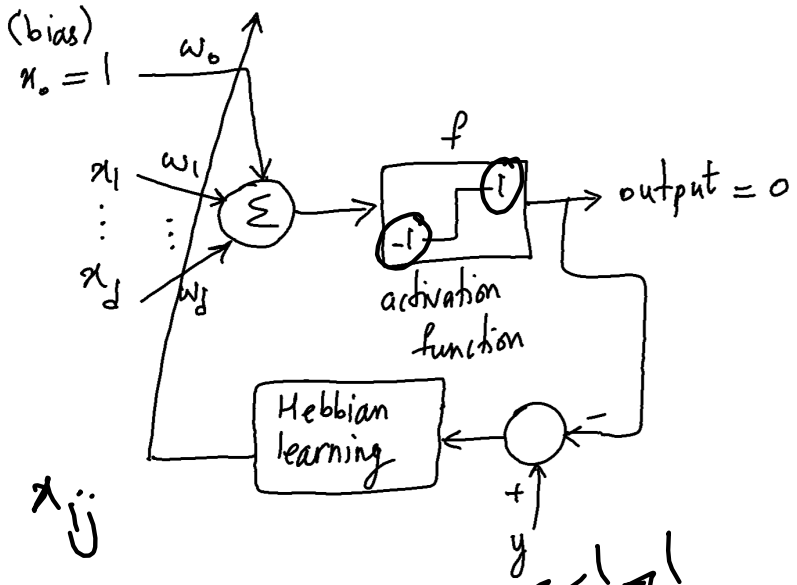
- Interpretation:

$$w_j := w_j + \eta (y_i - o_i) x_{ij}, \quad \forall j \in \{1, \ldots, d\}, \forall i \in \{1, \ldots, n\}.$$

  When the output o_i is the same as the target y_i, then we should not have any update:

$$o_i = y_i \implies \eta (y_i - o_i) x_{ij} = 0.$$

Perceptron with Signum Activation Function



$$w_j = w_j + \eta(y_i - o_i)\, x_{ij}$$

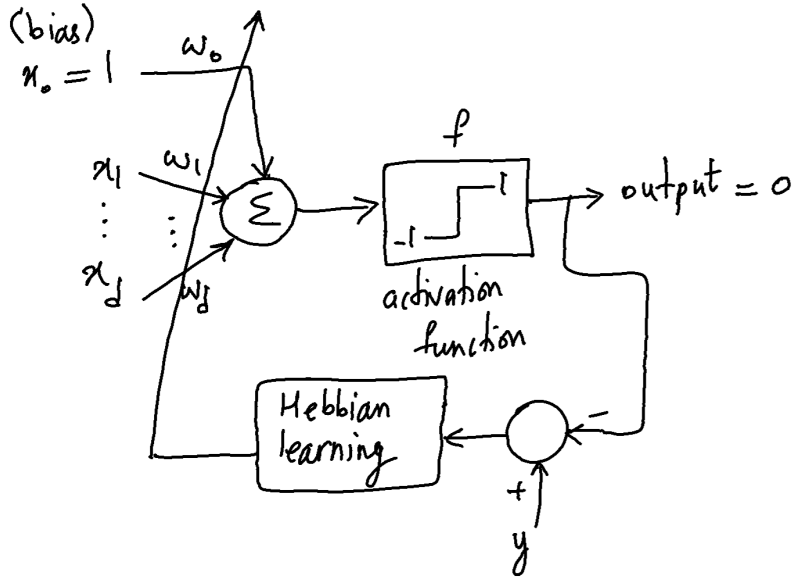
- If the activation function is the sign (signum) function, then we have:

$$w_j := \begin{cases} w_j + 2\eta y_i x_{ij} & \text{if } o_i \neq y_i \\ w_j & \text{if } o_i = y_i, \end{cases} \tag{5}$$

$\forall j \in \{1, \ldots, d\}, \forall i \in \{1, \ldots, n\}.$

$$w_0 := \begin{cases} w_0 + 2\eta y_i & \text{if } o_i \neq y_i \\ w_0 & \text{if } o_i = y_i, \end{cases} \tag{6}$$

Perceptron with Signum Activation Function



- If the activation function is the sign (signum) function, in batch learning of Hebbian learning:

$$w_j := w_j + 2\eta \sum_{i=1}^{b} y_i \, x_{ij} \, \mathbb{I}(o_i \neq y_i), \tag{7}$$

$\forall j \in \{1, \ldots, d\}$.

$$w_0 := w_0 + \sum_{i=1}^{b} y_i \, \mathbb{I}(o_i \neq y_i), \tag{8}$$

Perceptron with Signum Activation Function

- Proof 1:

$$w_j := w_j + \eta(y_i - o_i)x_{ij}, \quad \forall j \in \{1, \ldots, d\}, \forall i \in \{1, \ldots, n\}.$$

When the output o_i and the target label y_i have levels ± 1:

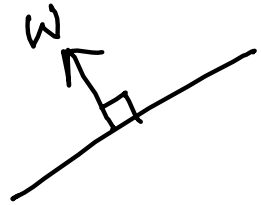
$$\begin{aligned}
&y_i, o_i \in \{-1, 1\}, \\
&y_i = 1, o_i = 1 \implies \eta(y_i - o_i)x_{ij} = 0, \\
&y_i = 1, o_i = -1 \implies \eta(y_i - o_i)x_{ij} = \eta(1 - (-1))x_{ij} = 2\eta x_{ij} = 2\eta y_i x_{ij}, \\
&y_i = -1, o_i = 1 \implies \eta(y_i - o_i)x_{ij} = \eta(-1 - 1)x_{ij} = -2\eta x_{ij} = 2\eta y_i x_{ij}, \\
&y_i = -1, o_i = -1 \implies \eta(y_i - o_i)x_{ij} = \eta(-1 - (-1))x_{ij} = 0,
\end{aligned}$$

Perceptron with Signum Activation Function

Proof 2 (using gradient descent):

- We want to find a linear decision boundary where one class falls in one side and the other class falls on the other side.
- The equation of a line for linear decision boundary:

$$\boldsymbol{w}^\top \boldsymbol{x} + w_0 = 0, \tag{9}$$



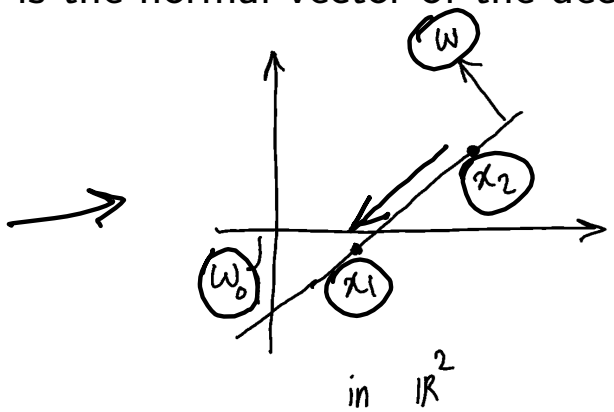
where $\boldsymbol{x} \in \mathbb{R}^d$ is the data point, $\boldsymbol{w} \in \mathbb{R}^d$ is the normal vector of the linear line, and w_0 is the bias (intercept) of the line.

- Consider any two points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ on the decision boundary. As the line passes through each of them, they both satisfy Eq. (9):

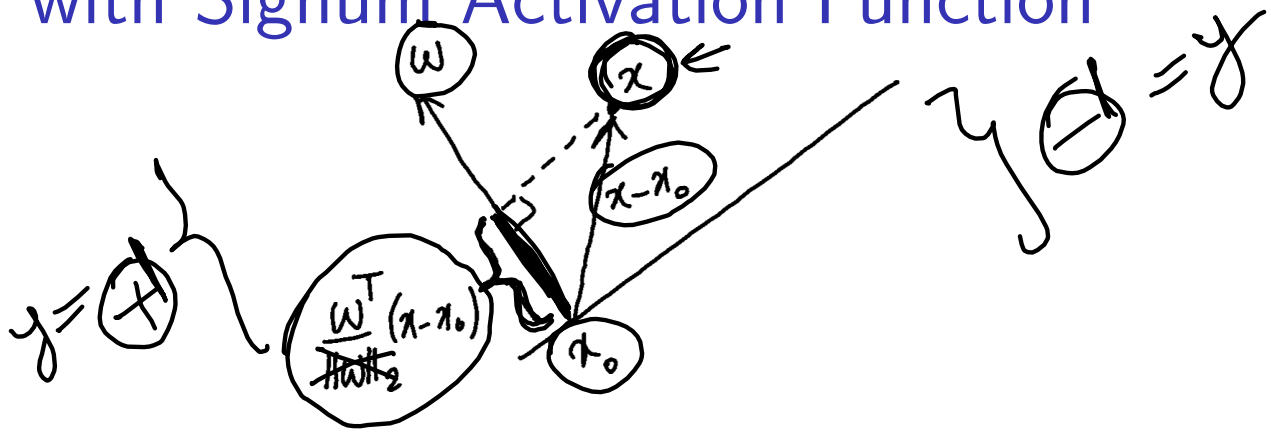
$$\boldsymbol{w}^\top \boldsymbol{x}_1 + w_0 = 0, \quad \boldsymbol{w}^\top \boldsymbol{x}_2 + w_0 = 0 \implies \boldsymbol{w}^\top \boldsymbol{x}_1 + w_0 = \boldsymbol{w}^\top \boldsymbol{x}_2 + w_0,$$

$$\implies \boldsymbol{w}^\top (\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0 \implies \boldsymbol{w} \perp (\boldsymbol{x}_1 - \boldsymbol{x}_2),$$

which verifies that $\boldsymbol{w}$ is the normal vector of the decision boundary.



Perceptron with Signum Activation Function



- Consider a point $\boldsymbol{x}_0$ on the decision boundary and a point $\boldsymbol{x}$ on one of the sides of the decision boundary. Therefore:

$$\boldsymbol{w}^\top \boldsymbol{x}_0 + w_0 = 0 \implies w_0 = -\boldsymbol{w}^\top \boldsymbol{x}_0. \tag{10}$$

- Assume the normal vector $\boldsymbol{w}$ is normalized, i.e., it has unit length. The distance of point $\boldsymbol{x}$ from the decision boundary is:

$$d = \boldsymbol{w}^\top (\boldsymbol{x} - \boldsymbol{x}_0) = \boldsymbol{w}^\top \boldsymbol{x} - \boldsymbol{w}^\top \boldsymbol{x}_0 \overset{(10)}{=} \boldsymbol{w}^\top \boldsymbol{x} + w_0. \tag{11}$$

- The distance should be non-negative so we have:

$$d = |\boldsymbol{w}^\top \boldsymbol{x} + w_0|.$$

However, the absolute value is non-smooth and non-differentiable. Therefore, we can multiply the distance with the target label to make it always non-negative:

$$y = +1 \implies \boldsymbol{w}^\top \boldsymbol{x} + w_0 > 0, \qquad y = -1 \implies \boldsymbol{w}^\top \boldsymbol{x} + w_0 < 0, \tag{12}$$

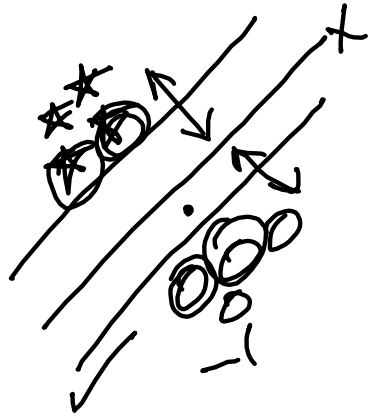
$$\implies d := y(\boldsymbol{w}^\top \boldsymbol{x} + w_0). \tag{13}$$

Perceptron with Signum Activation Function

- A possible cost function: number of misclassified data points to be minimized. But it is discrete and hard to optimize. So, let's optimize the distance as the cost function.
- Approach 1: We want to maximize the distance of points from the decision boundary, so we use gradient ascent for maximizing the distances of points from the decision boundary[1]:

$$\text{distance} = \sum_{i=1}^{b} y_i (\boldsymbol{w}^\top \boldsymbol{x}_i + w_0),$$

$$\frac{\partial \text{distance}}{\partial \boldsymbol{w}} = \sum_{i=1}^{b} y_i \boldsymbol{x}_i, \quad \frac{\partial \text{distance}}{\partial w_0} = \sum_{i=1}^{b} y_i.$$



We should use gradient ascent to maximize the distances:

$$\begin{aligned} \boldsymbol{w} &:= \boldsymbol{w} + \eta \sum_{i=1}^{b} y_i x_{ij}, \quad \forall j \in \{1, \dots, d\}, \\ w_0 &= w_0 + \eta \sum_{i=1}^{b} y_i, \end{aligned} \tag{14}$$

where we can update only for $o_i \neq y_i$ cases to have Eqs. (7) and (8).

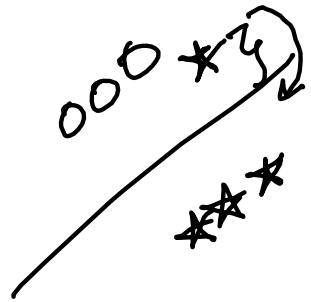
[1] Note that it is not like support vector machine which maximizes the distance of only support vectors, and not all points, from the decision boundary.

Perceptron with Signum Activation Function

- Approach 2: We take the distances of misclassified data points as the cost function. According to Eq. (12), the distance in Eq. (13) is for correctly classified data points. So, we should multiply distance by -1 to have the distances of misclassified points:

$$\text{error} = \sum_{i=1}^{b} -y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + w_0) = -\sum_{i=1}^{b} y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + w_0),$$

$$\frac{\partial \text{error}}{\partial \boldsymbol{w}} = -\sum_{i=1}^{b} y_i \boldsymbol{x}_i, \quad \frac{\partial \text{error}}{\partial w_0} = -\sum_{i=1}^{b} y_i.$$

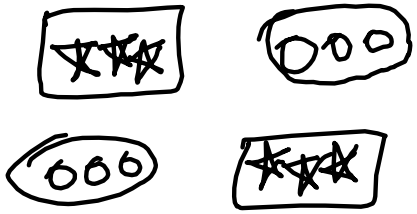


We should use gradient descent to minimize the error:

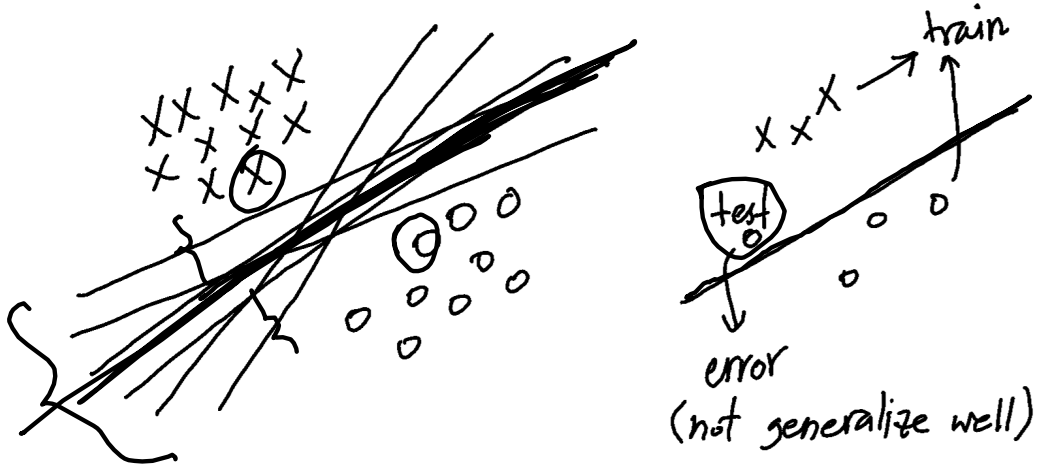
$$\begin{aligned} \boldsymbol{w} &:= \boldsymbol{w} + \eta \sum_{i=1}^{b} y_i x_{ij}, \quad \forall j \in \{1, \dots, d\}, \\ w_0 &= w_0 + \eta \sum_{i=1}^{b} y_i, \end{aligned} \tag{15}$$

where we can update only for $o_i \neq y_i$ cases to have Eqs. (7) and (8).

Problems with Perceptron



- Perceptron is for **binary** classification.
- In 1969, Minsky and Papert published a book titled "Perceptrons" [5] and showed that Perceptron can only solve linearly separable problems. For example, they showed that it cannot classify XOR classes, which is a nonlinear classification problem.
- Therefore, researchers lost interest in Perceptron and artificial neural networks!
- Researchers guessed that they should have multilayer Perceptrons but they did not know how to train multilayer Perceptrons.
- Also, Perceptron cannot generalize well enough
  because, for linearly separable classes, it finds one of the many possible decision boundaries.



- Because of this problem, two things were developed:
  - ADALINE which generalizes better.
  - Support Vector Machines (SVM) which found the best decision boundary by optimization of teh distances of the support vectors from the decision boundary.

ADALINE

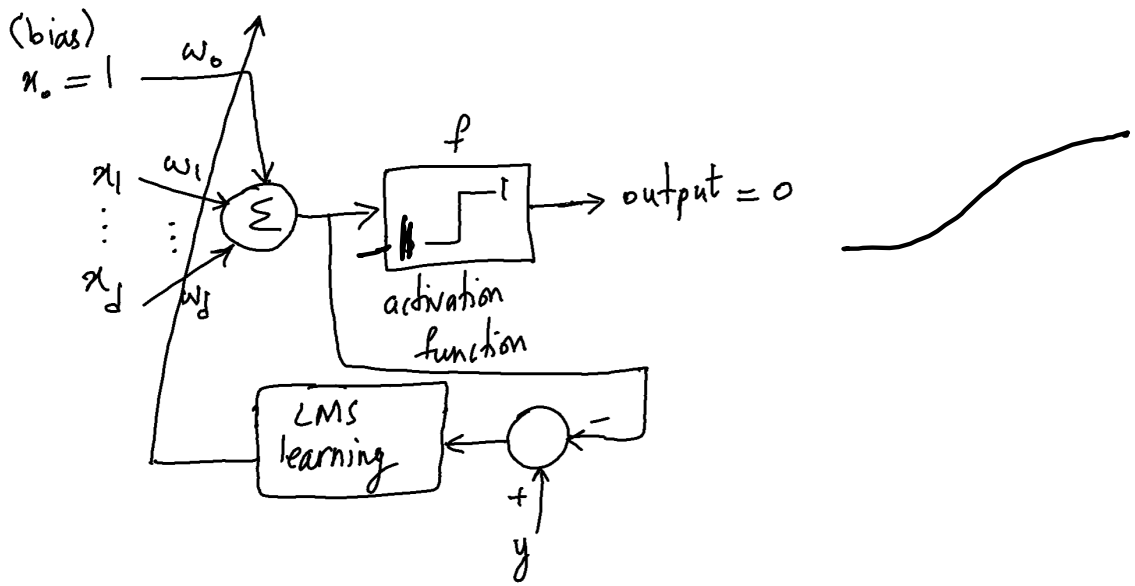
ADALINE

- In 1960: Widrow and his student Hoff, at Stanford University, proposed a method, named **ADALINE**, for ~~adjusting~~ adjusting weights [6, 7].
- In 1960: articles claimed that robots can think!
- It has more generalization compared to Perceptron.
- ADALINE minimizes (least) mean squares (LMS) error using gradient descent. This training method is referred to as **LMS algorithm** or **Widrow-Hoff learning rule**.

$$\min \frac{1}{b} \sum_{j=1}^{b} (y_j - o_j)^2$$

mean error

ADALINE



- In ADALINE, the target label is compared with the output before activation function. This is while Perceptron compares the target label with the output after activation function.
- LMS error:

$$e = \frac{1}{2} \sum_{i=1}^{b} \left(y_i - \left(\sum_{j=1}^{d} w_j x_{ij} + w_0 \right) \right)^2, \tag{16}$$

where b is the batch size, $\boldsymbol{x}_i = [x_{i1}, \ldots, x_{id}]^\top \in \mathbb{R}^d$ is the i-th input data, and y_i is its target label.

ADALINE

- We had the LMS error:

$$e = \frac{1}{2}\sum_{i=1}^{b}\left(y_i - \left(\sum_{j=1}^{d} w_j x_{ij} + w_0\right)\right)^2. \tag{17}$$

- The gradients:

$$\begin{cases} \dfrac{\partial e}{\partial w_j} = -\sum_{i=1}^{b}\left(y_i - \left(\sum_{j=1}^{d} w_j x_{ij} + w_0\right)\right) x_{ij} & \forall j \in \{1, \ldots, d\}, \\ \dfrac{\partial e}{\partial w_0} = -\sum_{i=1}^{b}\left(y_i - \left(\sum_{j=1}^{d} w_j x_{ij} + w_0\right)\right) \end{cases} \tag{18}$$

- The gradient descent updates:

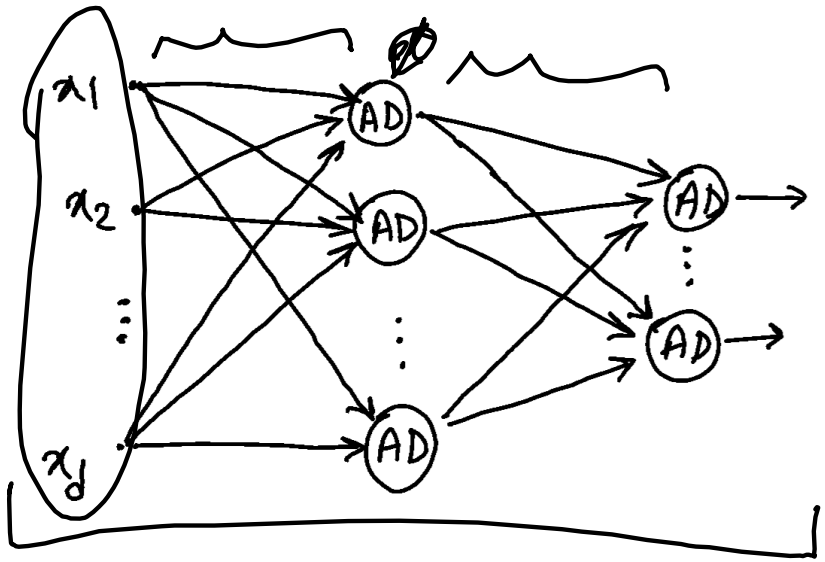
$$\begin{cases} w_j := w_j - \eta \dfrac{\partial e}{\partial w_j}, & \forall j \in \{1, \ldots, d\}, \\ w_0 := w_0 - \eta \dfrac{\partial e}{\partial w_0}, \end{cases} \tag{19}$$

where $\eta > 0$ is the leanring rate.

MADALINE

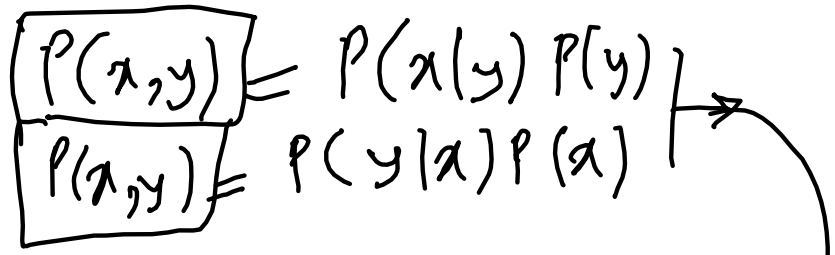
MADALINE

- At Stanford university, Widrow and his students stacked several ADALINE neurons to be able to have nonlinear classification. They proposed **MADALINE** (Many ADALINEs).
- **MADALINE Rule 1 (MRI)**: proposed in 1962 [8] and could not adapt the weights of the hidden-output layer.
- **MADALINE Rule 2 (MRII)**: proposed in 1988 [9] and improved MRI to be able to also train the weights of the hidden-output layer.
- **MADALINE Rule 3 (MRIII)**: proposed in 1990 [10] and changed signum activation function to sigmoid function for having float outputs rather than merely binary outputs.

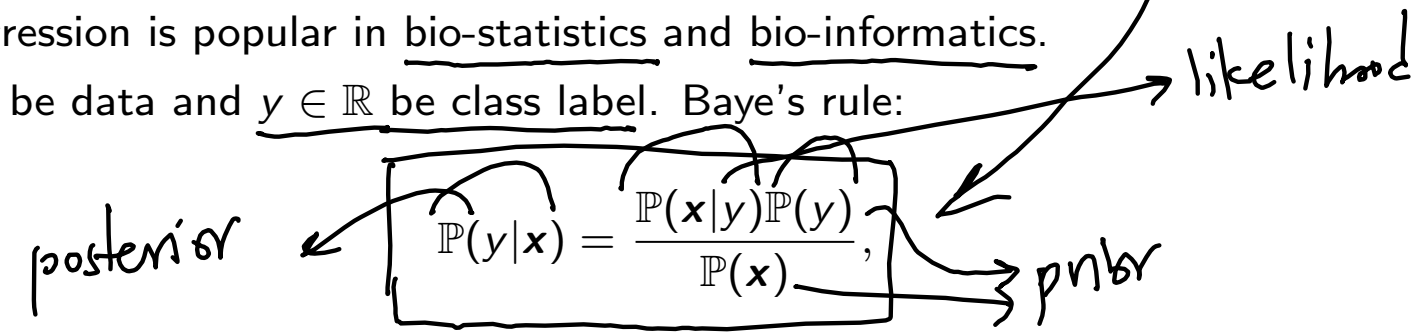


Logistic Regression

Logistic Regression



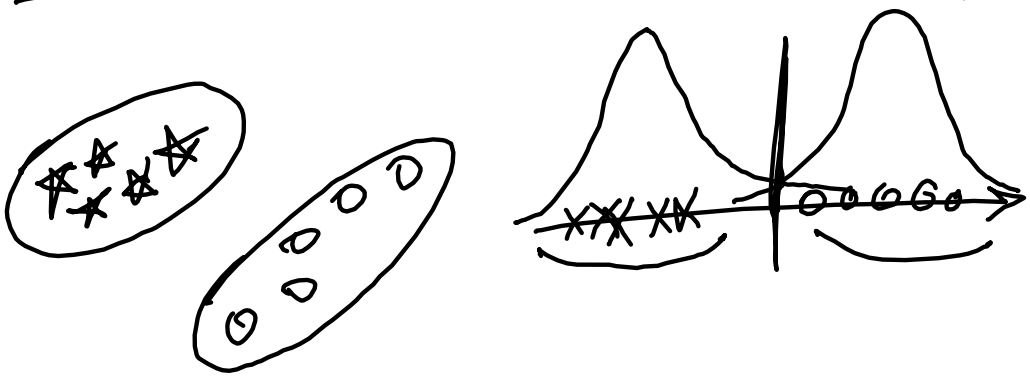
- Logistic regression is popular in bio-statistics and bio-informatics.
- Let $\boldsymbol{x} \in \mathbb{R}^d$ be data and $y \in \mathbb{R}$ be class label. Baye's rule:



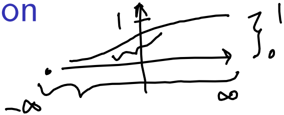
$$\mathbb{P}(y|\boldsymbol{x}) = \frac{\mathbb{P}(\boldsymbol{x}|y)\mathbb{P}(y)}{\mathbb{P}(\boldsymbol{x})}, \tag{20}$$

where $\mathbb{P}(y|\boldsymbol{x})$ and $\mathbb{P}(\boldsymbol{x}|y)$ are the posterior and likelihood, respectively, and $\mathbb{P}(\boldsymbol{x})$ and $\mathbb{P}(y)$ are the priors.

- In contrast to Linear Discriminant Analysis (LDA), logistic regression works on the posterior $\mathbb{P}(y|\boldsymbol{x})$ directly rather than working on likelihood $\mathbb{P}(\boldsymbol{x}|y)$ and prior $\mathbb{P}(y)$.



Logistic Regression



$\frac{e^{x_i}}{\sum_i e^{x_i}}$ $\frac{x_i}{\sum x_i}$

- Logistic regression is a binary classifier where it assigns probability between zero and one for belonging to one of the classes.
- The logistic function, used in logistic regression, was initially proposed in 1845 for modeling the population growth [11]. It was further improved in the 20th century [12]. See [13] for the history of logistic regression.
- It considers the classification problem as a regression problem where it regresses (predicts) the probability of belonging to a class. It first considers a linear regression $\boldsymbol{\beta}^\top \boldsymbol{x} + \beta_0$. However, in order to not have the bias, it assumes that $\boldsymbol{x}$ is $d+1$ dimensional with an additional element of 1 for bias, i.e., $\boldsymbol{x} = [x_1, \dots, x_d, 1]^\top$. The $\boldsymbol{\beta} \in \mathbb{R}^{d+1}$ is the learnable parameter of the logistic regression model. As a result, the linear regression becomes $\boldsymbol{\beta}^\top \boldsymbol{x}$.
- However, there is no bound on this regression while logistic regression desires the output to be in the range $[0, 1]$ to behave like a probability. Therefore, Logistic regression models the posterior using a **logistic function**, also called the **sigmoid function**, to make this regression between zero and one.

$\beta^\top x$

$\frac{1}{Z} e^{-\frac{\|x_i - \mu\|_2}{\sigma^2}}$

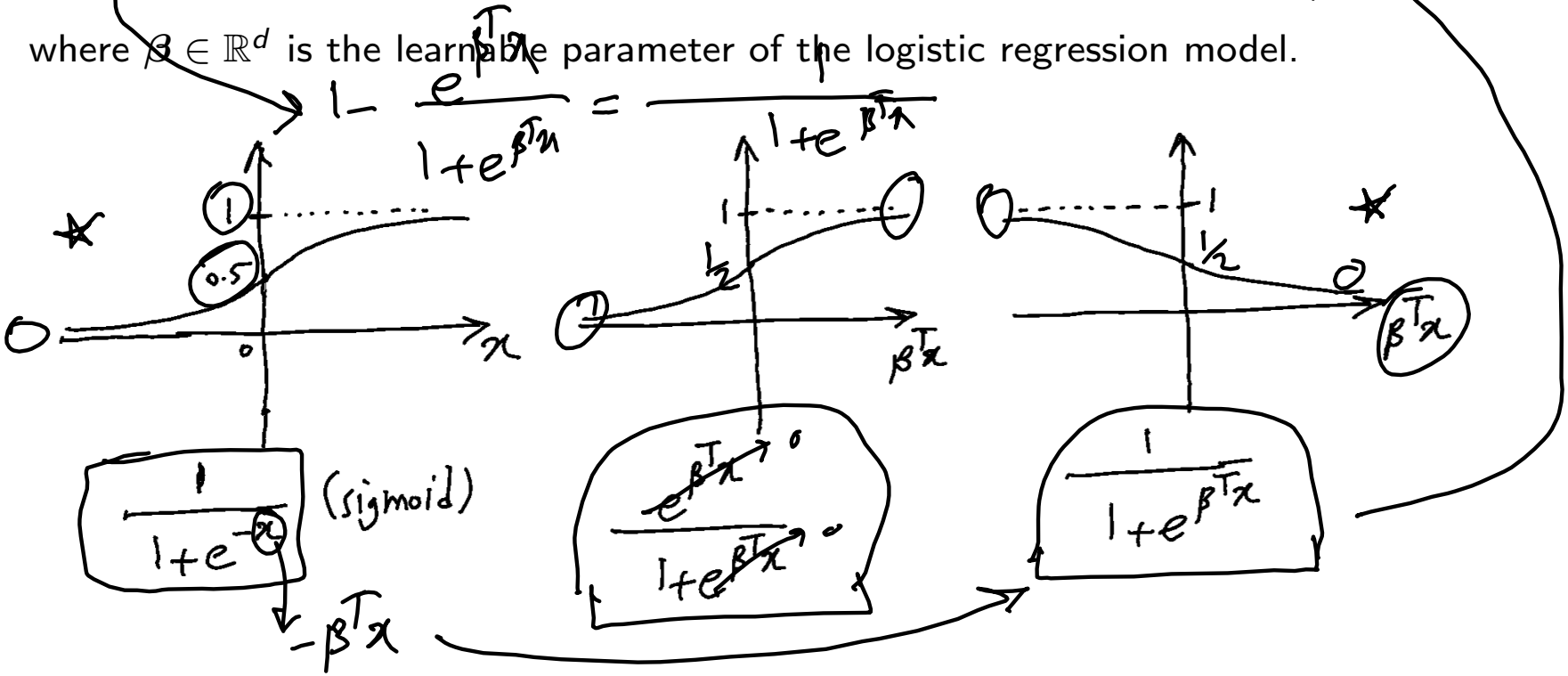
Logistic Regression

- Assume we have two classes $y \in \{0, 1\}$.
- Logistic regression models the posterior using a **logistic function**, also called the **sigmoid function**:

$$\mathbb{P}(y = 1|X = x) = \frac{e^{\beta^\top x}}{1 + e^{\beta^\top x}}, \tag{21}$$

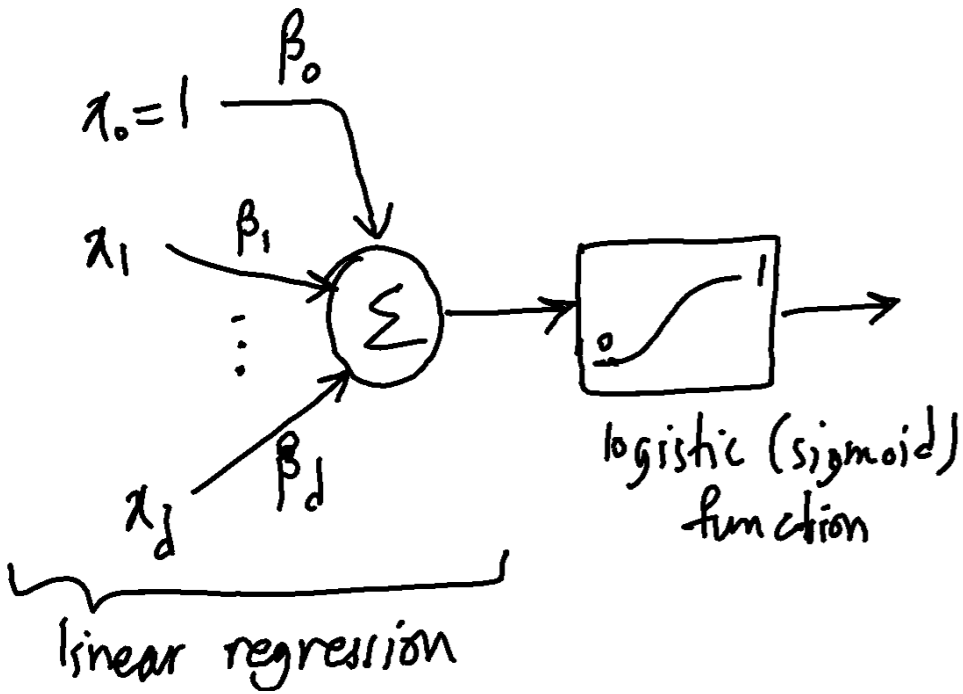
$$\mathbb{P}(y = 0|X = x) = 1 - \mathbb{P}(y = 1|X = x) = \frac{1}{1 + e^{\beta^\top x}}, \tag{22}$$

where $\beta \in \mathbb{R}^d$ is the learnable parameter of the logistic regression model.



Logistic Regression as a Neural Network

- Logistic regression can be seen as a neural network with one neuron where the activation function is the nonlinear sigmoid (logistic) function.



Logistic Regression

- Consider n data points $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ in the dataset. Assuming that they are independent and identically distributed (i.i.d), the posterior over all data points is:

$$\mathbb{P}(y|X) = \prod_{i=1}^{n} \Big(\mathbb{P}(y_i = 1|X = x_i)\mathbb{I}(y_i = 1) + \mathbb{P}(y_i = 0|X = x_i)\mathbb{I}(y_i = 0) \Big), \tag{23}$$

  where $\mathbb{I}(.)$ is the indicator function which is one if its condition is satisfied and is zero otherwise.

- As the labels are either zero or one, i.e., $y_i \in \{0, 1\}$, this equation can be restated as:

$$\mathbb{P}(y|X) = \prod_{i=1}^{n} \big(\mathbb{P}(y_i = 1|X = x_i)\big)^{y_i} \big(\mathbb{P}(y_i = 0|X = x_i)\big)^{1-y_i}. \tag{24}$$

- Substituting Eqs. (21) and (22) in this equation gives:

$$\mathbb{P}(y|X) = \prod_{i=1}^{n} \Big(\frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}{1 + e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}\Big)^{y_i} \Big(\frac{1}{1 + e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}\Big)^{1-y_i}. \tag{25}$$

Logistic Regression

$\log(\text{Ben} \times \text{Jack}) = \log(\text{Ben}) + \log(\text{Jack})$

- The log posterior is:

$$
\begin{aligned}
\ell(\boldsymbol{\beta}) := \mathbb{P}(y|X=x) &= \log \prod_{i=1}^{n} \left(\frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}{1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}\right)^{y_i} \left(\frac{1}{1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}\right)^{1-y_i} \\
&= \sum_{i=1}^{n} \left(\log\left(\frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}{1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}\right)^{y_i} + \log\left(\frac{1}{1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}\right)^{1-y_i} \right) \\
&= \sum_{i=1}^{n} \left(y_i \log(e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}) - y_i \log(1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}) - (1-y_i)\log(1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}) \right) \\
&= \sum_{i=1}^{n} \left(y_i \boldsymbol{\beta}^\top \boldsymbol{x}_i - y_i \log(1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}) - \log(1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}) + y_i \log(1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}) \right) \\
&= \sum_{i=1}^{n} \left(y_i \boldsymbol{\beta}^\top \boldsymbol{x}_i - \log(1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}) \right).
\end{aligned}
$$

Logistic Regression

- The log ~~likelihood of the~~ posterior is:

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left(y_i \boldsymbol{\beta}^\top \boldsymbol{x}_i - \log(1 + e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}) \right).$$

- Newton's method can be used to find the optimum $\boldsymbol{\beta}$. The first derivative, or the gradient, it:

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{n} \left(y_i \boldsymbol{x}_i - \frac{1}{1 + e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}} e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i} \boldsymbol{x}_i \right) = \sum_{i=1}^{n} \left(y_i - \frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}{1 + e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}} \right) \boldsymbol{x}_i. \tag{26}$$

  Its transpose is:

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}^\top} = \sum_{i=1}^{n} \left(y_i - \frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}{1 + e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}} \right) \boldsymbol{x}_i^\top.$$

Logistic Regression

- The second derivative is:

$$\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^\top} = \frac{\partial}{\partial \boldsymbol{\beta}} \left(\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}^\top} \right) = \frac{\partial}{\partial \boldsymbol{\beta}} \left(\sum_{i=1}^{n} \left(y_i - \frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}{1 + e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}} \right) \boldsymbol{x}_i^\top \right)$$

$$= \sum_{i=1}^{n} \left(- \frac{\partial}{\partial \boldsymbol{\beta}} \left(\frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}{1 + e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}} \right) \right) \boldsymbol{x}_i^\top .$$

- We define:

$$\mathbb{P}(\boldsymbol{x}_i | \boldsymbol{\beta}) := \frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}{1 + e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}. \tag{27}$$

Therefore:

$$\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^\top} = - \sum_{i=1}^{n} \left(\frac{\partial}{\partial \boldsymbol{\beta}} \left(\mathbb{P}(\boldsymbol{x}_i | \boldsymbol{\beta}) \right) \right) \boldsymbol{x}_i^\top . \tag{28}$$

Logistic Regression

- We have:

$$\frac{\partial}{\partial \beta}\left(\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{\beta})\right) = \frac{\partial}{\partial \beta}\left(\frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}{1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}\right)$$

$$= \frac{1}{(1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i})^2}\left(e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}\boldsymbol{x}_i(1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}) - e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}(e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}\boldsymbol{x}_i)\right)$$

$$= \frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}{(1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i})^2}\left(1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i} - e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}\right)\boldsymbol{x}_i = \frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}{(1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i})^2}\boldsymbol{x}_i$$

$$= \frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i}}{(1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i})}\frac{1}{(1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_i})}\boldsymbol{x}_i \overset{(27)}{=} \mathbb{P}(\boldsymbol{x}_i|\boldsymbol{\beta})\left(1-\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{\beta})\right)\boldsymbol{x}_i$$

- Substituting it in Eq. (28) gives the second derivative, i.e., the Hessian matrix:

$$\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}\partial \boldsymbol{\beta}^\top} = -\sum_{i=1}^{n}\left(\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{\beta})\left(1-\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{\beta})\right)\boldsymbol{x}_i\right)\boldsymbol{x}_i^\top. \tag{29}$$

Logistic Regression

- It is possible to write the Newton's method in matrix form. We define:

$$\mathbb{R}^{(d+1)\times n} \ni \boldsymbol{X} := \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ 1 & 1 & \cdots & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \end{bmatrix}$$

$$\mathbb{R}^{n\times n} \ni \boldsymbol{W} := \mathbf{diag}\Big(\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{\beta})\big(1-\mathbb{P}(\boldsymbol{x}_i|\boldsymbol{\beta})\big)\Big),$$

$$\mathbb{R}^{n} \ni \boldsymbol{y} := [y_1, \ldots, y_n]^\top,$$

$$\mathbb{R}^{n} \ni \boldsymbol{p} := \Big[\frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}_1}}{1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_1}}, \ldots, \frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}_n}}{1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}_n}}\Big]^\top.$$

- The Eqs. (26) and (29) can be restated as:

$$\frac{\partial \ell}{\partial \boldsymbol{\beta}} = \sum (y_i - \frac{e^{\boldsymbol{\beta}^T x_i}}{1+e^{\boldsymbol{\beta}^T x_i}}) x_i$$

$$\mathbb{R}^{(d+1)} \ni \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \boldsymbol{X}(\boldsymbol{y}-\boldsymbol{p}), \tag{30}$$

$$\mathbb{R}^{(d+1)\times(d+1)} \ni \frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}\partial \boldsymbol{\beta}^\top} = -\boldsymbol{X}\boldsymbol{W}\boldsymbol{X}^\top. \tag{31}$$

- Using Newton's method for maximization of the log posterior is:

$$\boldsymbol{\beta}^{(\tau+1)} := \boldsymbol{\beta}^{(\tau)} - \Big(\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}\partial \boldsymbol{\beta}^\top}\Big)^{-1} \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \Longrightarrow$$

$$\boldsymbol{\beta}^{(\tau+1)} := \boldsymbol{\beta}^{(\tau)} - (\boldsymbol{X}\boldsymbol{W}\boldsymbol{X}^\top)^{-1}\boldsymbol{X}(\boldsymbol{y}-\boldsymbol{p}), \tag{32}$$

where τ is the iteration index. It is repeated until convergence of $\boldsymbol{\beta}$.

Logistic Regression

- In the test phase, the class of a point $\boldsymbol{x}$ is determined as:

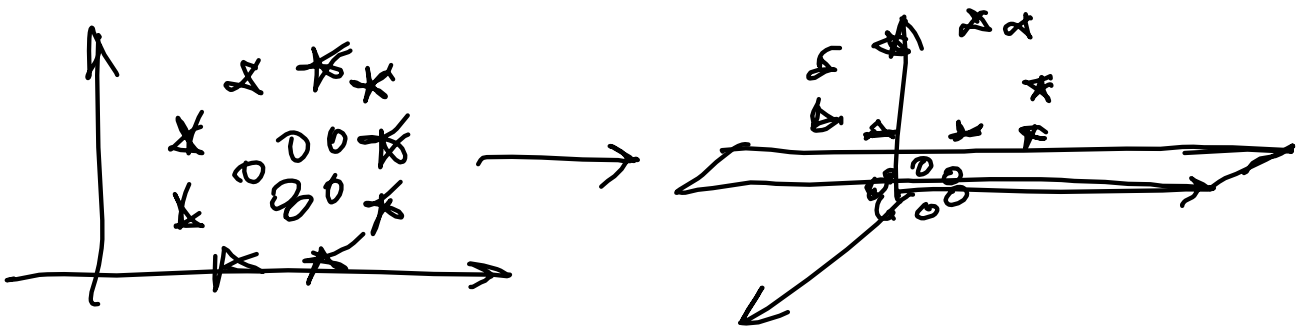
$$y = \begin{cases} 1 & \text{if } \frac{e^{\boldsymbol{\beta}^\top \boldsymbol{x}}}{1+e^{\boldsymbol{\beta}^\top \boldsymbol{x}}} \geq 0.5, \\ 0 & \text{Otherwise.} \end{cases} \tag{33}$$

- Comparison to LDA:
  - Logistic regression estimates $(d+1)$ parameters in $\boldsymbol{\beta}$, but LDA estimates many more parameters:
    - ★ prior of each class: 1. We have two classes: $2 \times 1 = 2$.
    - ★ mean of each class: d. We have two classes: $2 \times d = 2d$.
    - ★ covariance matrix of each class: $d(d+1)/2$. We have two classes: $2 \times (d(d+1)/2) = d(d+1)$.
    - ★ so, in total: $2 + 2d + d(d+1) = d^2 + 2d + 2$.
  - LDA assumes the distribution of each class is Gaussian which may not be true. However, logistic regression does not assume anything about the distribution of data.

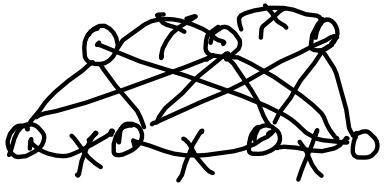
Other History

Other History

- in 1969: Arthur E. Bryson and Yu-Chi Ho described backpropagation as a multi-stage dynamic system optimization method [14, 15].
- Starting 1969, people started inventing and re-inventing backpropagation algorithm for training multilayer Perceptron.
- in 1972: Stephen Grossberg proposed networks capable of learning XOR function.
- in 1986: the main and succesful backpropagation was proposed by David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams [16].
- in 1980's: successful era of neural networks.
- Kernel support vector machines [17] resulted in the winter of neural networks in the last years of previous century until around 2006.



Other History



- Hinton et. al. had proposed Boltzmann Machine (BM) and Restricted Boltzmann Machine (RBM) in 1983 and 1985 [18, 19].
- During the winter of neural networks, Hinton tried to save neural networks from being forgotten in the history of machine learning. So, he returned to his previously proposed RBM and proposed a learning method for RBM with the help of some other researchers including Max Welling [20, 21].
- They proposed training the weights of BM and RBM using maximum likelihood estimation. BM and RBM can be seen as generative models where new values for neurons can be generated using Gibbs sampling [22].
- Hinton noticed RBM because he knew that the set of weights between every two layers of a neural network is an RBM. It was in the year 2006 [23, 24] that he thought it is possible to train a network in a greedy way[2] [25] where the weights of every layer of network is trained using RBM training.
- This stack of RBM models with a greedy algorithm for training was named Deep Belief Network (DBN) [24, 26]. DBN allowed the networks to become deep by preparing a good initialization of weights (using RBM training) for backpropagation. This good starting point for backpropagation optimization did not face the problem of vanishing gradients anymore.

[2] A greedy algorithm makes every decision based on the most benefit at the current step and does not care about the final outcome at the final step. This greedy approach hopes that the final step will obtain a good result by small best steps based on their current benefits.

Other History

- Since the breakthrough in 2006 [23], the winter of neural networks started to end gradually because the networks could get deep to become more nonlinear and handle more nonlinear data.
- DBN was used in different applications including speech recognition [27, 28, 29] and action recognition [30].
- Hinton was very excited about the success of RBM and was thinking that the future of neural networks belongs to DBN.
- However, two important techniques were proposed, which were the ReLU activation function (2011) [31] and the dropout technique (2014) [32]. These two regularization methods prevented overfitting [33] and resolved vanishing gradients even without RBM pre-training.
- Hence, backpropagation could be used alone if the new regularization methods were utilized. The success of neural networks was found out more [34] by its various applications, for example in image recognition [35].

Acknowledgment

- For more information on early history of artificial neural networks, see the book [36]: "Fundamentals of neural networks: architectures, algorithms and applications"
- Some slides of this slide deck were inspired by teachings of Prof. Ali Ghodsi (at University of Waterloo, Department of Statistics), Prof. Fakhri Karray (at University of Waterloo, Department of Electrical and Computer Engineering), and Prof. Saeed Bagheri Shouraki (at Sharif University of Technology, Department of Electrical Engineering).

References

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[2] D. O. Hebb, *The Organization of Behavior*.
New York: Wiley & Sons, 1949.

[3] F. Rosenblatt, "The Perceptron – a perceiving and recognizing automaton project para," tech. rep., Report 85-460-1, Cornell Aeronautical Laboratory., 1957.

[4] M. Olazaran, "A sociological study of the official history of the perceptrons controversy," *Social Studies of Science*, vol. 26, no. 3, pp. 611–659, 1996.

[5] M. Minsky and S. A. Papert, *Perceptrons*.
MIT press, 1969.

[6] B. Widrow and M. E. Hoff, "Adaptive switching circuits," tech. rep., Stanford University, California, Stanford Electronics Labs, 1960.

[7] B. Widrow, "An adaptive "adaline" neuron using chemical "memistors"," tech. rep., No. 1553-2, Stanford Electronics Laboratories, 1960.

[8] B. Widrow, "Generalization and information storage in networks of adaline neurons," *Self-organizing systems*, pp. 435–461, 1962.

[9] C. R. Winter and B. Widrow, "Madaline rule ii: A training algorithm for neural networks," in *Secondf Annual International Conference on Neural Networks*, pp. 1–401, 1988.

References (cont.)

[10] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, 1990.

[11] P. F. Verhulst, "Resherches mathematiques sur la loi d'accroissement de la population," *Nouveaux memoires de l'academie royale des sciences*, vol. 18, pp. 1–41, 1845.

[12] S. H. Walker and D. B. Duncan, "Estimation of the probability of an event as a function of several independent variables," *Biometrika*, vol. 54, no. 1-2, pp. 167–179, 1967.

[13] J. S. Cramer, "The origins of logistic regression," 2002.

[14] A. E. Bryson Jr, W. F. Denham, and S. E. Dreyfus, "Optimal programming problems with inequality constraints," *AIAA journal*, vol. 1, no. 11, pp. 2544–2550, 1963.

[15] A. E. Bryson and H. Yu-Chi, *Applied optimal control: optimization, estimation and control*.
CRC Press, 1969.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[17] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152, 1992.

References (cont.)

[18] G. E. Hinton and T. J. Sejnowski, "Optimal perceptual inference," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, vol. 448, IEEE, 1983.

[19] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for Boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[20] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[21] M. Welling, M. Rosen-Zvi, and G. E. Hinton, "Exponential family harmoniums with an application to information retrieval.," in *Advances in neural information processing systems*, vol. 4, pp. 1481–1488, 2004.

[22] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Transactions on pattern analysis and machine intelligence*, vol. PAMI-6, no. 6, pp. 721–741, 1984.

[23] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[24] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[25] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in neural information processing systems*, pp. 153–160, 2007.

References (cont.)

[26] G. E. Hinton, "Deep belief networks," *Scholarpedia*, vol. 4, no. 5, p. 5947, 2009.

[27] A.-r. Mohamed, G. Dahl, G. Hinton, *et al.*, "Deep belief networks for phone recognition," in *Nips workshop on deep learning for speech recognition and related applications*, vol. 1, p. 39, Vancouver, Canada, 2009.

[28] A.-r. Mohamed and G. Hinton, "Phone recognition using restricted Boltzmann machines," in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 4354–4357, IEEE, 2010.

[29] A.-r. Mohamed, G. E. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE transactions on audio, speech, and language processing*, vol. 20, no. 1, pp. 14–22, 2011.

[30] G. W. Taylor, G. E. Hinton, and S. T. Roweis, "Modeling human motion using binary latent variables," in *Advances in neural information processing systems*, pp. 1345–1352, 2007.

[31] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, JMLR Workshop and Conference Proceedings, 2011.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

References (cont.)

[33] B. Ghojogh and M. Crowley, "The theory behind overfitting, cross validation, regularization, bagging, and boosting: tutorial," *arXiv preprint arXiv:1905.12787*, 2019.

[34] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[35] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3128–3137, 2015.

[36] L. V. Fausett, *Fundamentals of neural networks: architectures, algorithms and applications*.
Pearson Education India, 2006.