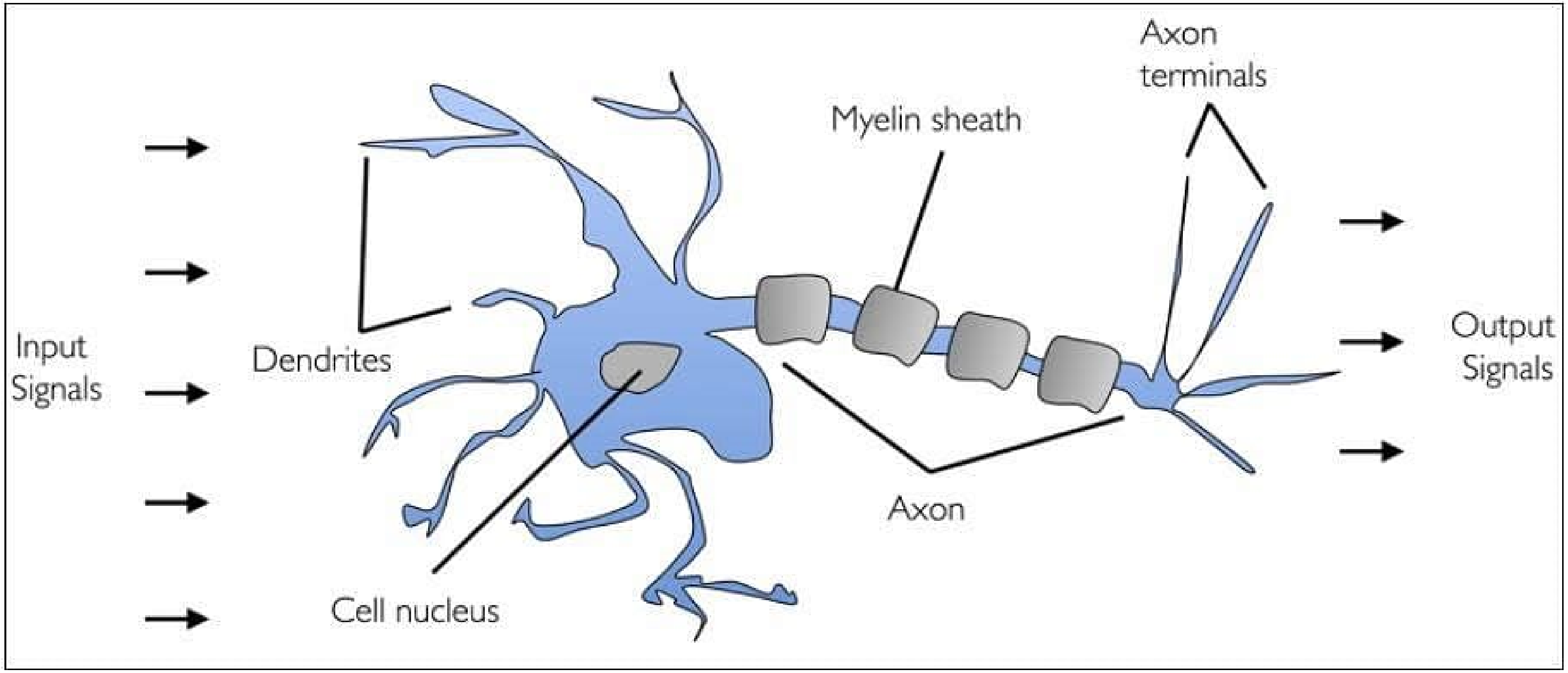
Perceptron

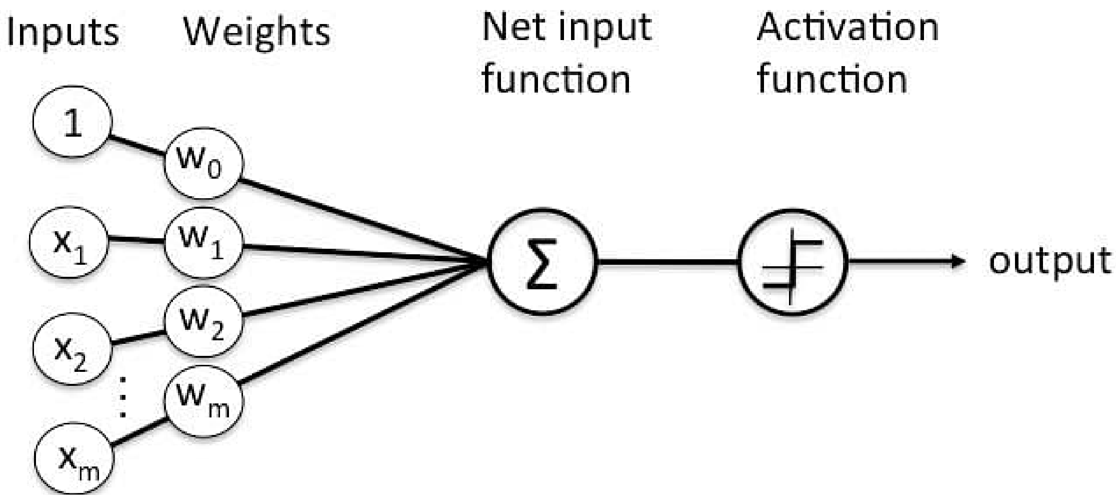
Fatemeh Mansoori

A Biological Neuron



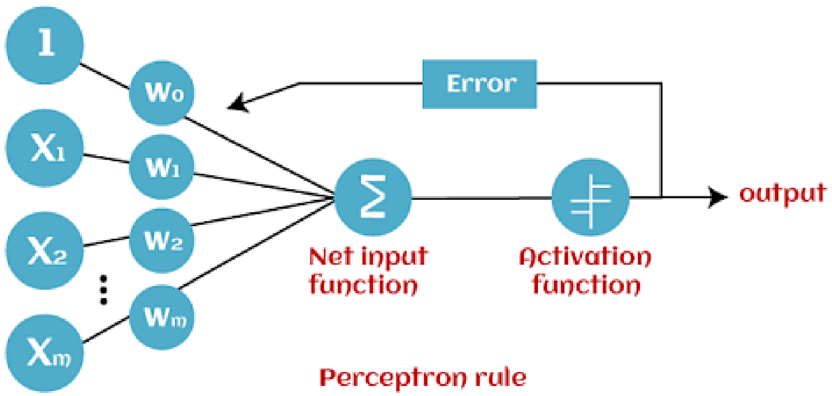
Perceptron

- Perceptron was introduced by Frank Rosenblatt in 1957
- He proposed a Perceptron learning rule
- A Perceptron is an algorithm for supervised learning of binary classifiers
  - Has limited capacity to learn complex pattern
  - inability to handle non-linearly separable data

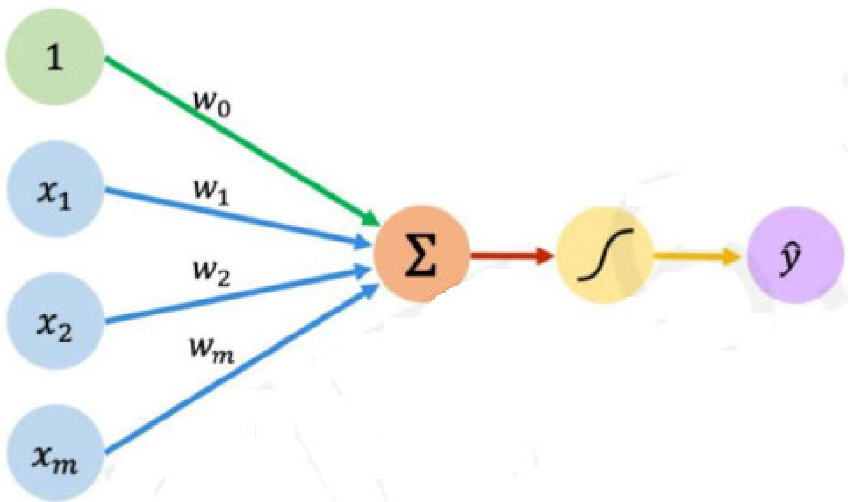


How Does Perceptron Work?

- multiplying all input values and their weights
- adds these values to create the weighted sum
- weighted sum is applied to the activation function ‘f’ to obtain the desired output
- activation function is also known as the step function and is represented by ‘f.’
- single-layer perceptron model analyze the linearly separable objects with binary outcomes



A simple mathematical model of a neuron



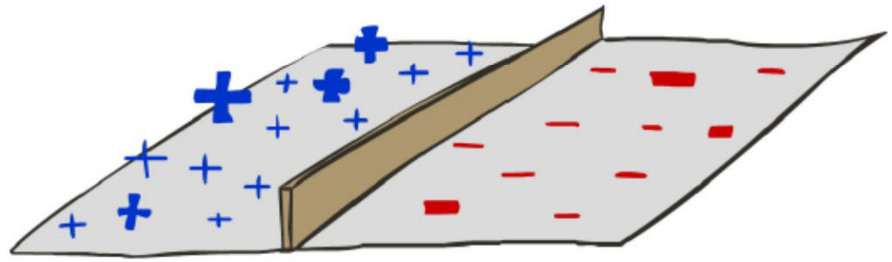
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Binary Decision Rule

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to Y=+1
  - Other corresponds to Y=-1



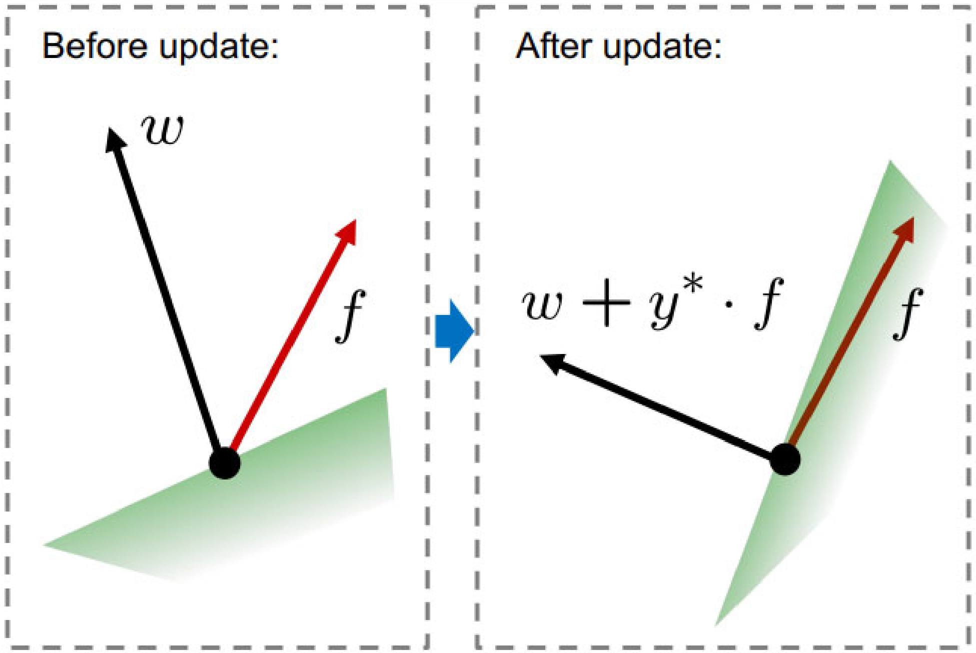
Learning: Binary Perceptron

- Start with weights w = 0
- For each training instance f(x), y*:
  - Classify with current weights

$$y = \begin{cases} +1 & \text{if } \ w \cdot f(x) \geq 0 \\ -1 & \text{if } \ w \cdot f(x) < 0 \end{cases}$$

  - **If correct:** (i.e., y=y*), no change!
  - **If wrong:** adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

$$w = w + y^* \cdot f$$



$w \cdot f$

$(w + y^* \cdot f) \cdot f$
$= w \cdot f + y^* \cdot f \cdot f$

Learning : Binary Perceptron

- Start with weights w = 0
- For each training instance f(x), y*:
  - Classify with current weights

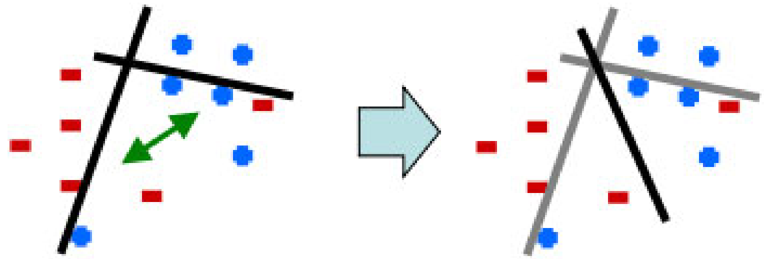
$$y = \begin{cases} +1 & \text{if } \ w \cdot f(x) \geq 0 \\ -1 & \text{if } \ w \cdot f(x) < 0 \end{cases}$$

  - If correct (i.e., y=y*), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

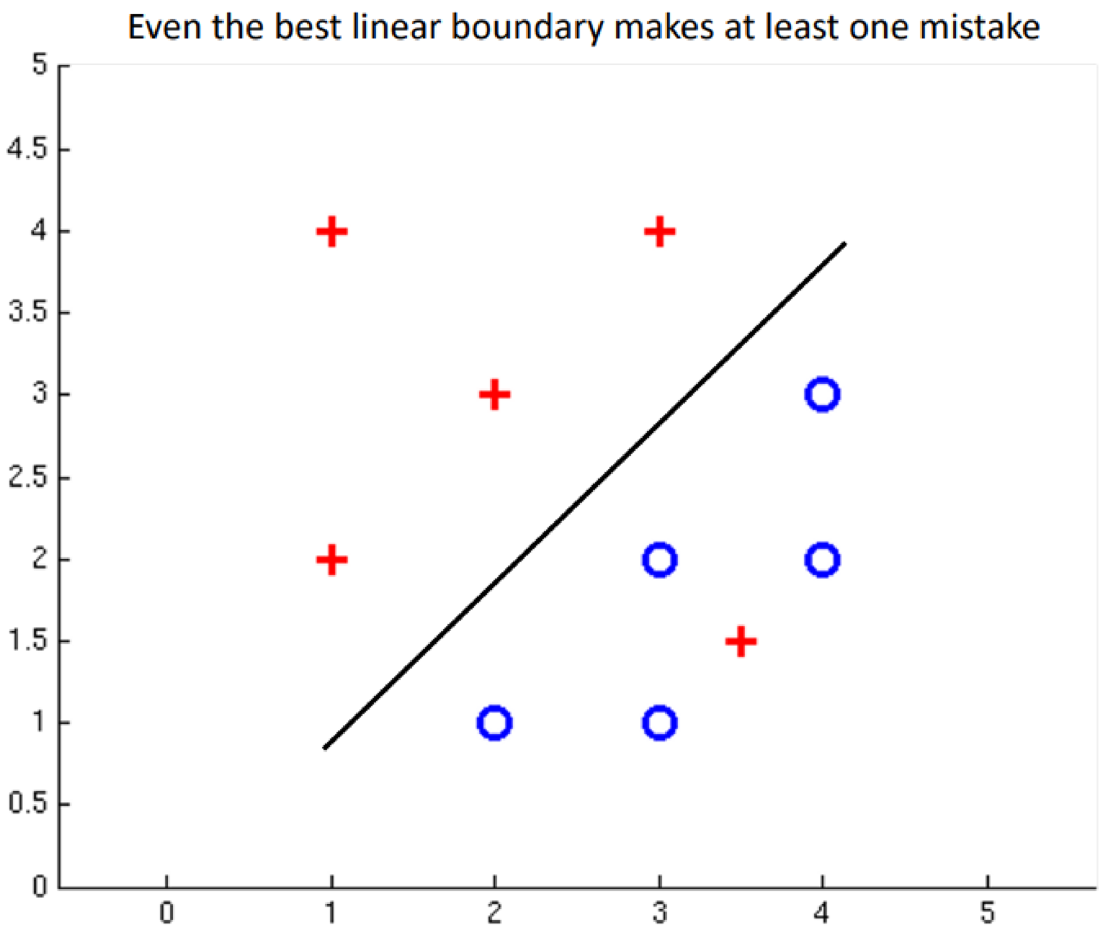
$$w = w + y^* \cdot f$$

Problems with the Perceptron

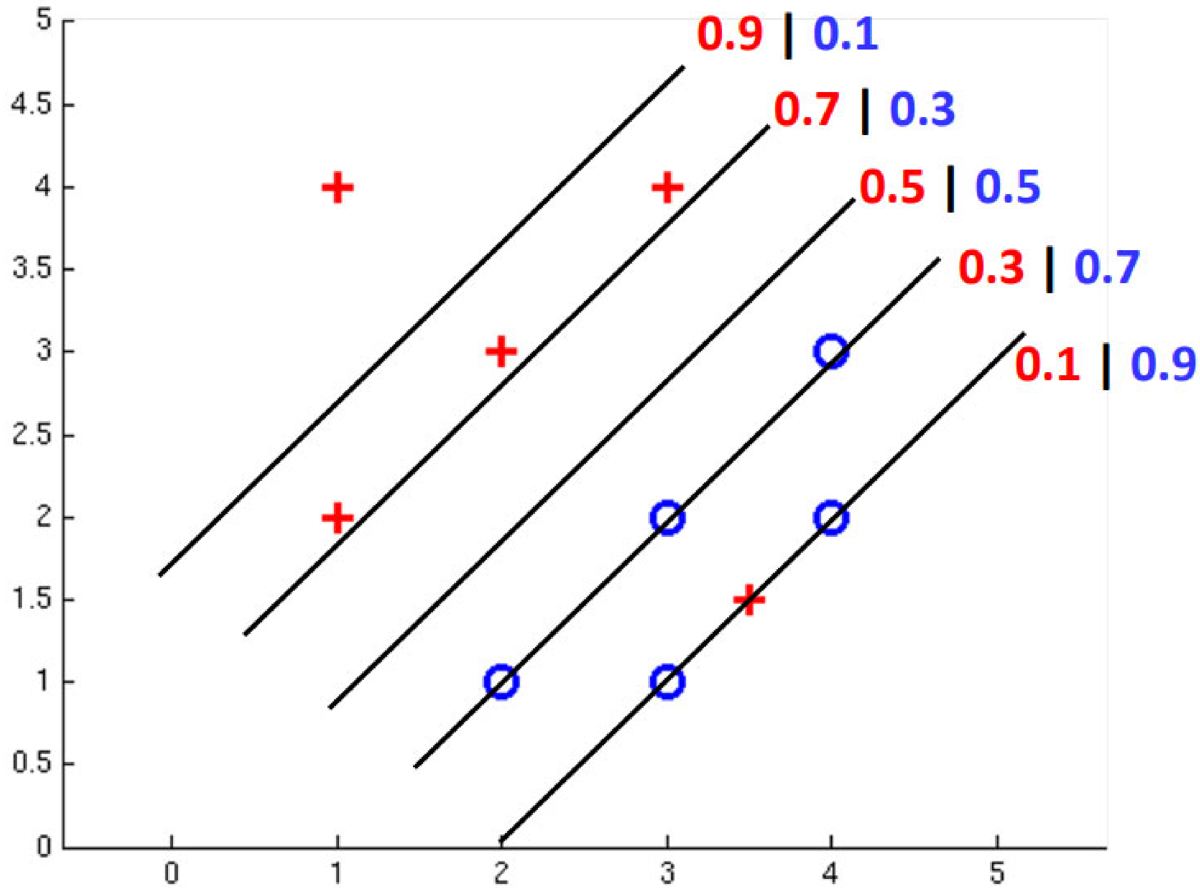
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



Non-Separable Case



Non-Separable Case: Probabilistic Decision



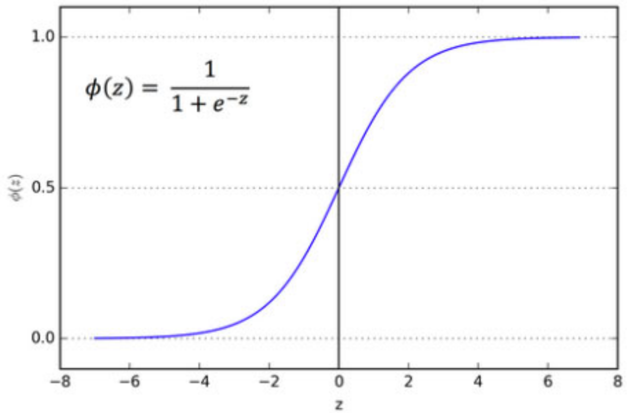
How to get probabilistic decisions?

- Perceptron scoring: $z = w \cdot f(x)$
- If $z = w \cdot f(x)$ very positive → want probability of + going to 1
- If $z = w \cdot f(x)$ very negative → want probability of + going to 0

- Sigmoid function

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

$$= \frac{e^z}{e^z + 1}$$



How to get probabilistic decisions?

- Perceptron scoring: $z = w \cdot f(x)$
- If $z = w \cdot f(x)$ very positive → want probability of + going to 1
- If $z = w \cdot f(x)$ very negative → want probability of + going to 0

- Sigmoid function

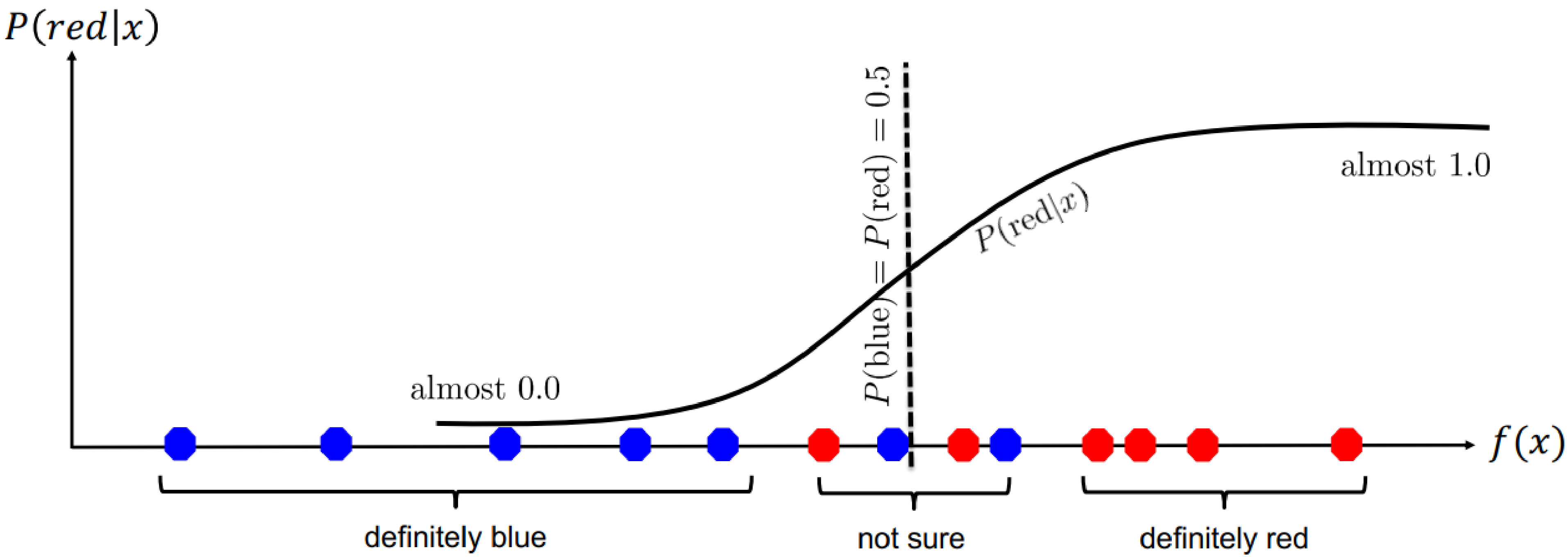
$$\phi(z) = \frac{1}{1 + e^{-z}}$$

$$P(y = +1 \mid x ; w) = \frac{1}{1+e^{-w \cdot f(x)}}$$

$$P(y = -1 \mid x ; w) = 1 - \frac{1}{1+e^{-w \cdot f(x)}}$$

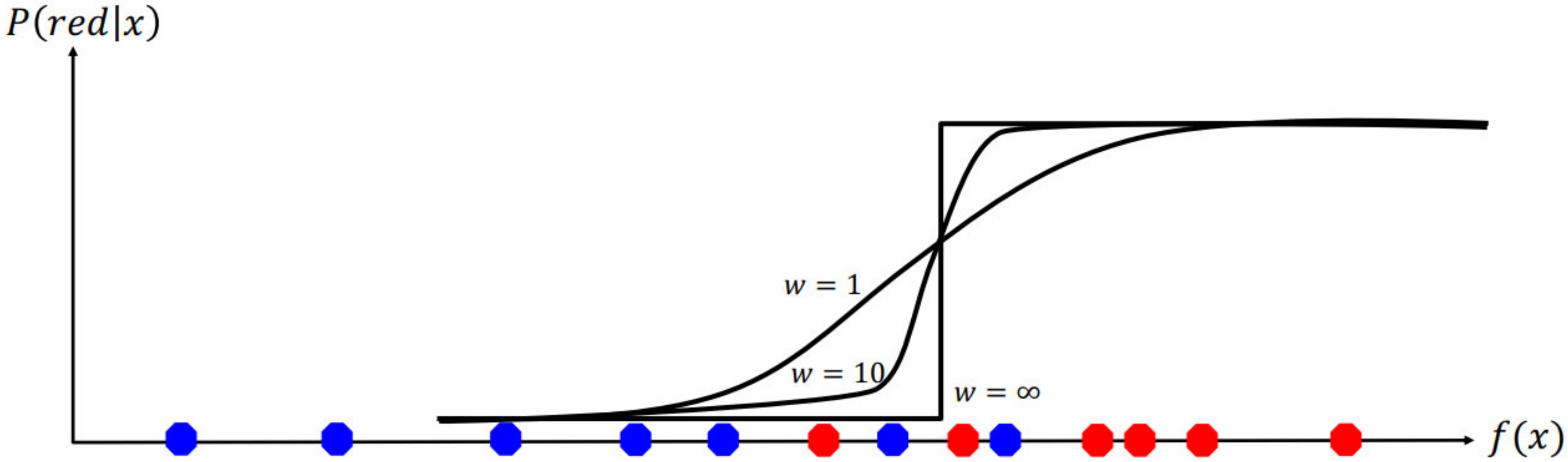
= Logistic Regression

A 1D Example



$$P(red|x\,; w) = \phi(w \cdot f(x)) = \frac{1}{1 + e^{-w \cdot f(x)}}$$

A 1D Example: varying w



$$P(red|x\,;w) = \phi(w \cdot f(x)) = \frac{1}{1 + e^{-w \cdot f(x)}}$$

Best w?

$$
\begin{aligned}
\text{Likelihood} &= P(\text{training data}|w) \\
&= \prod_i P(\text{training datapoint } i \mid w) \\
&= \prod_i P(\text{point } x^{(i)} \text{ has label } y^{(i)}|w) \\
&= \prod_i P(y^{(i)}|x^{(i)}; w) \\
\text{Log Likelihood} &= \sum_i \log P(y^{(i)}|x^{(i)}; w)
\end{aligned}
$$

Best w?

$$\max_{w} \quad ll(w) = \max_{w} \quad \sum_{i} \log P(y^{(i)} | x^{(i)}; w)$$

Logistic regression cost function

If $y = 1: \quad p(y|x) = \hat{y}$

If $y = 0: \quad p(y|x) = 1 - \hat{y}$

$$\boxed{p(y|x) = \hat{y}^{y} (1-\hat{y})^{(1-y)}}$$

If $y=1$: $p(y|x) = \hat{y} \quad \underbrace{(1-\hat{y})^{0}}_{=1}$

If $y=0$: $p(y|x) = \underbrace{\hat{y}^{0}}_{=1} \quad (1-\hat{y})^{(1-y)} = 1 \times (1-\hat{y}) = 1-\hat{y}$

$$\uparrow \log p(y|x) = \log \hat{y}^{y} (1-\hat{y})^{(1-y)} = \boxed{y \log \hat{y} + (1-y) \log(1-\hat{y})}$$

$$= -\mathcal{L}(\hat{y}, y) \downarrow$$

Cost on m examples

$$\underline{\log p(\text{labels in training set})} = \log \prod_{i=1}^{m} p(y^{(i)} | x^{(i)})$$

$$\underline{\log p(\ldots\ldots)} = \sum_{i=1}^{m} \underbrace{\log p(y^{(i)} | x^{(i)})}_{-\mathcal{L}(\hat{y}^{(i)}, y^{(i)})}$$

$$= -\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Maximum likelihood estimation

Cost: (minimize) $\underline{J(w,b)} = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

Loss Optimization

We want to find the network weights that ***achieve the lowest loss***

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

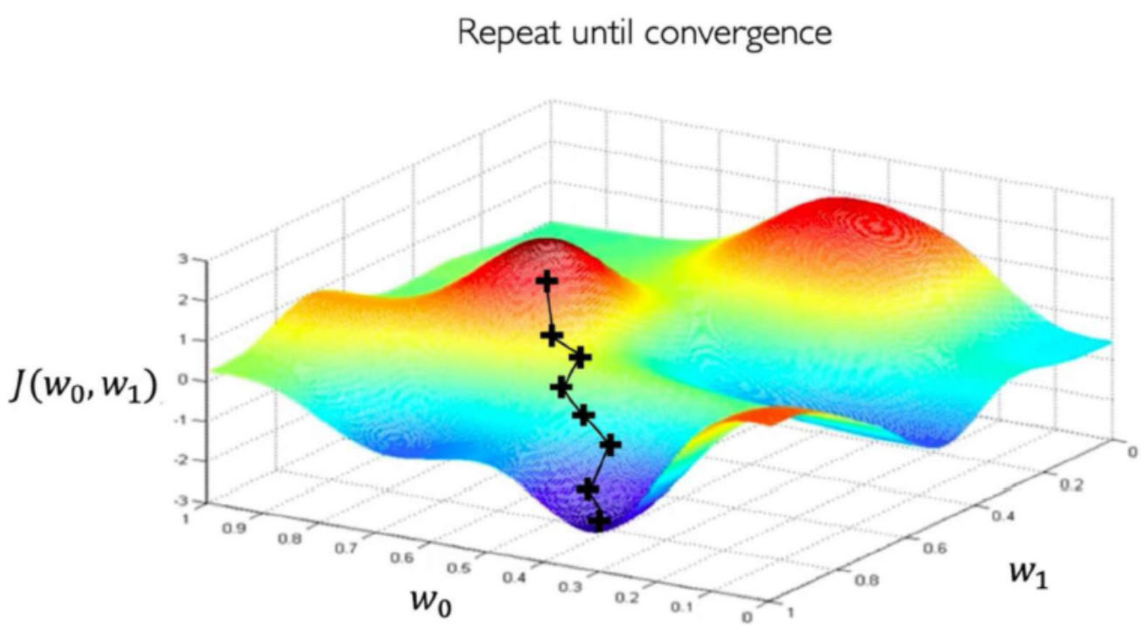
Cross Entropy Loss Optimization

Recap: $\hat{y} = \sigma(w^T x + b),\ \sigma(z) = \frac{1}{1+e^{-z}}$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

Want to find w, b that minimize $J(w,b)$

Loss Optimization



Gradient Descent

"Walking downhill and always taking a step in the direction that goes down the most."

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Logistic regression derivatives

$$\left.\begin{matrix} x_1 \\ w_1 \\ x_2 \\ w_2 \\ b \end{matrix}\right\} \rightarrow \boxed{z = w_1 x_1 + w_2 x_2 + b} \rightarrow \boxed{a = \sigma(z)} \rightarrow \boxed{\mathcal{L}(\mathrm{a}, y)}$$

$$dz = \frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}(a,y)}{dz}$$

$$= a - y$$

$$= \frac{d\mathcal{L}}{da} \cdot \underbrace{\boxed{\frac{da}{dz}}}_{a(1-a)}$$

$$\text{"}da\text{"} = \frac{d\mathcal{L}(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \text{"}dw_1\text{"} = x_1 \cdot dz. \qquad dw_2 = x_2 \cdot dz. \qquad db = dz.$$

$$w_1 := w_1 - \alpha\, dw_1$$
$$w_2 := w_2 - \alpha\, dw_2$$
$$b := b - \alpha\, db.$$

Logistic regression on m examples

$$\underline{J(w,b)} = \underbrace{\frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y^{(i)})}$$

$$\rightarrow a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

$$(x^{(i)}, y^{(i)})$$

$$\underline{dw_1^{(i)}},\ \underline{dw_2^{(i)}},\ \underline{db^{(i)}}$$

$$\underline{\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m}} \sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_1} \mathcal{L}(a^{(i)}, y^{(i)})}_{\underline{dw_1^{(i)}}\ -\ (x^{(i)}, y^{(i)})}$$

What is Vectorization ?

$$z = \underbrace{w^T x} + b$$

$$w = \begin{bmatrix} \vdots \\ \\ \vdots \end{bmatrix} \qquad x = \begin{bmatrix} \vdots \\ \\ \vdots \end{bmatrix} \qquad \begin{matrix} w \in \mathbb{R}^{n_x} \\ x \in \mathbb{R}^{n_x} \end{matrix}$$

Non-vectorized:

```
z = 0
for i in range(n-x):
    z += w[i] * x[i]
z += b
```

Vectorized

$$z = \underbrace{np.\underline{dot}(w, x)}_{w^T x} + b$$

→ GPU
→ CPU } SIMD - single instruction multiple data.