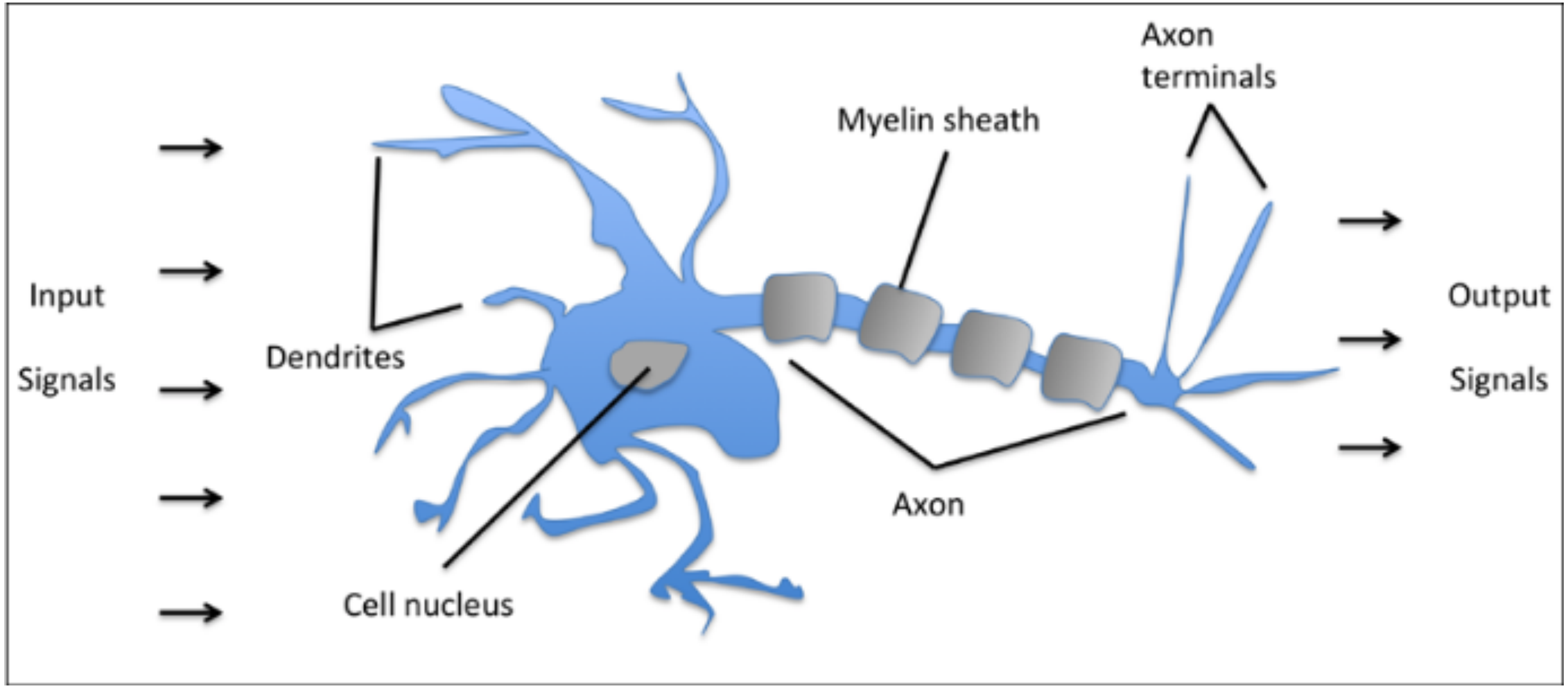
Perceptron

Suleyman Demirel University

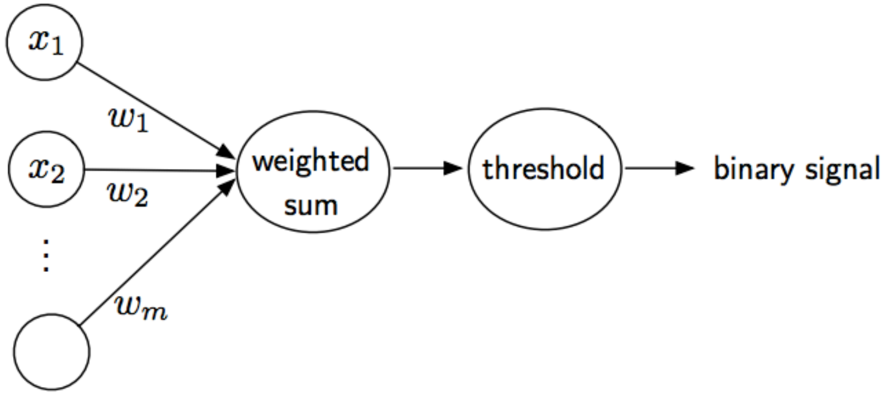
CSS634: Deep Learning

PhD Abay Nussipbekov

Inspired by Biological Brains and Neurons



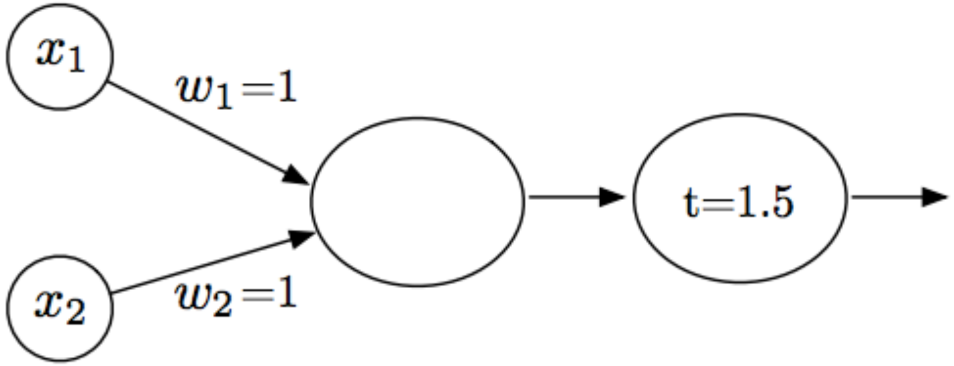
Logic Gate



- Simple logic gate with binary outputs
- Signals arrive at dendrites
- Integrated into cell body
- If signal exceeds threshold, generate output, and pass to axon

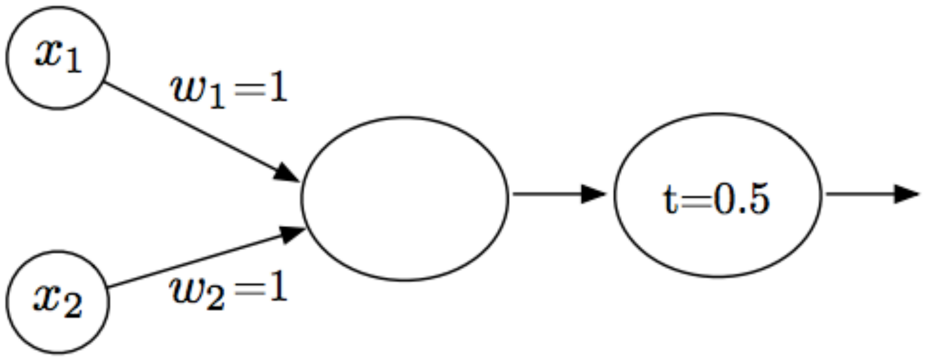
Logic AND Gate

x_1	x_2	Out
0	0	0
0	1	0
1	0	0
1	1	1



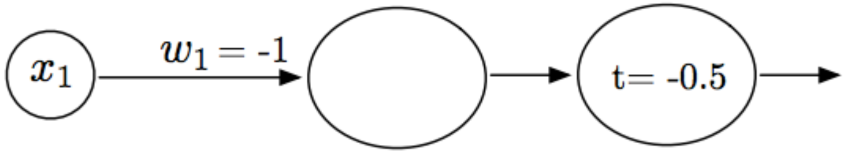
Logic OR Gate

x_1	x_2	Out
0	0	0
0	1	1
1	0	1
1	1	1



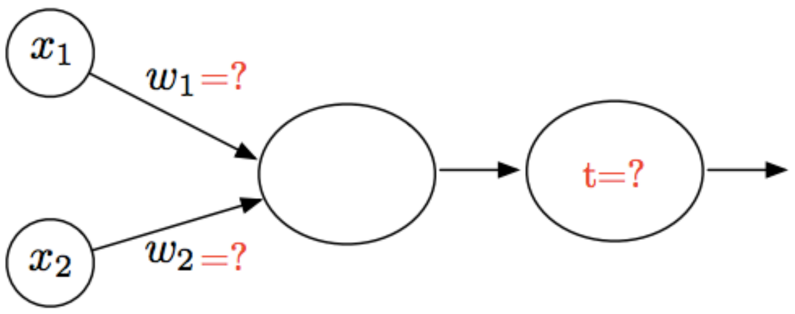
Logic NOT Gate

x_1	Out
0	1
1	0



Logic XOR Gate

x_1	x_2	Out
0	0	0
0	1	1
1	0	1
1	1	0



Rosenblatt Perceptron

- Binary classification task
- Positive class (1) vs. negative class (-1)
- Define activation function $\phi(z)$
- Takes as input a dot product of input and weights
- Net input: $z = w_1 x_1 + \cdots + w_n x_n$

$$\mathrm{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}, \mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Heaviside Step Function

- $\phi(z)$ known as activation
- if activation above some threshold, predict class 1
- predict class -1 otherwise

Heaviside Step Function

$$\phi(z) = \begin{cases} 1 \;\; if \; z \geq \theta \\ -1 \; otherwise \end{cases}$$

Step Function Simplified

- Bring the threshold θ to the left side of the equation and define a weight-zero as $w_0 = -\theta$ and $x_0 = 1$, so that we write z in a more compact form

$$z = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n = \mathbf{w}^T \mathbf{x}$$

and

$$\phi(z) = \begin{cases} 1 \ \ if \ \ z - \theta \geq 0 \\ -1 \ otherwise \end{cases}$$

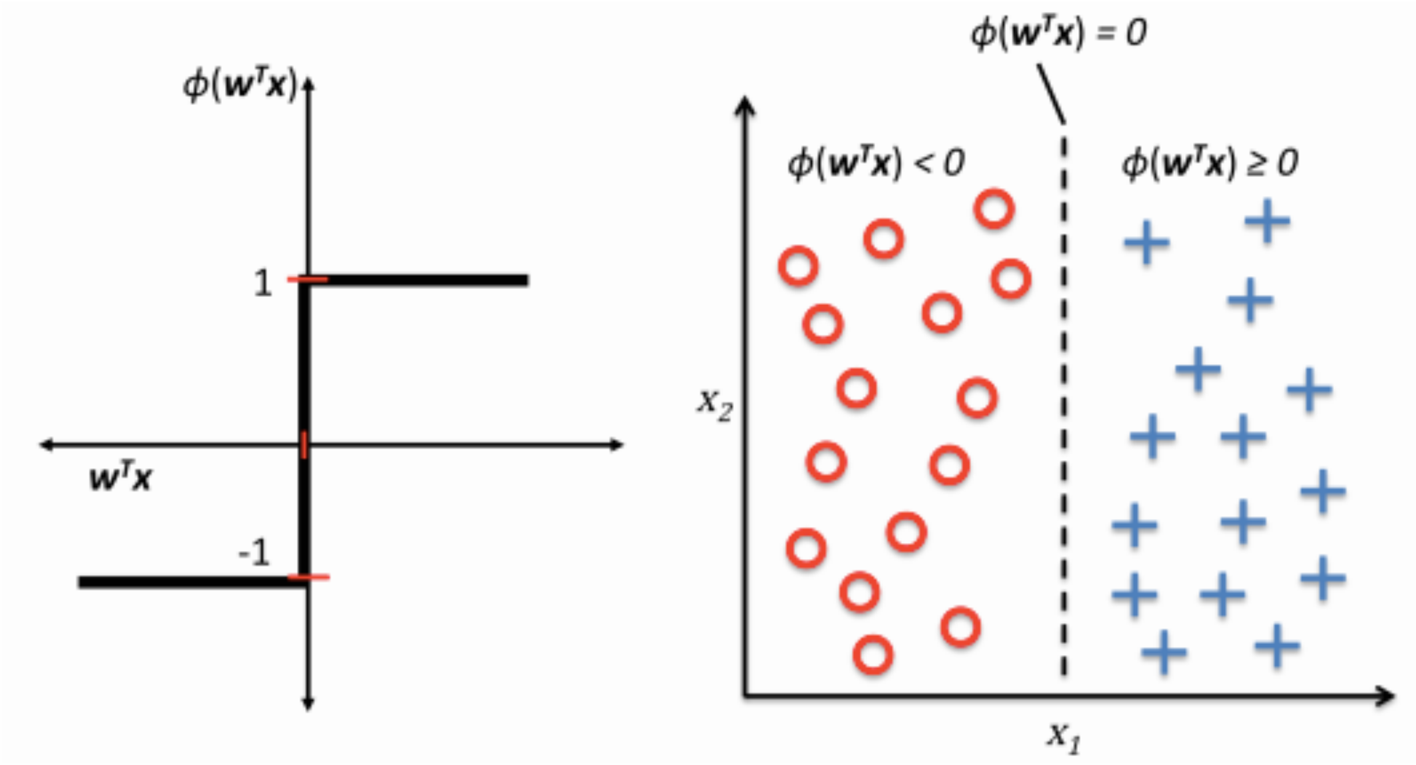
Basic Linear Algebra

Vector dot product

$$z = \mathbf{w}^T \mathbf{x} = \sum_{j=0}^{n} w_j x_j$$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = 1 \times 4 + 2 \times 5 + 3 \times 6 = 32.$$

Input Squashed Into a Binary Output



Interlude: Vectorization

In [1]:

```
x0, x1, x2 = 1., 2., 3.
bias, w1, w2 = 0.1, 0.3, 0.5

x = [x0, x1, x2]
w = [bias, w1, w2]
```

A simple for-loop:

In [2]:

```
z = 0.
for i in range(len(x)):
    z += x[i] * w[i]

print(z)
```

```
2.2
```

Interlude: Vectorization

A simple for-loop:

```
In [2]:
z = 0.
for i in range(len(x)):
    z += x[i] * w[i]

print(z)
```

```
2.2
```

A little bit better, list comprehensions:

```
In [3]:
z = sum(x_i*w_i for x_i, w_i in zip(x, w))
print(z)
```

```
2.2
```

Interlude: Vectorization

list comprehensions (still sequential):

In [3]:

```
z = sum(x_i*w_i for x_i, w_i in zip(x, w))
print(z)
```

```
2.2
```

A vectorized implementation:

In [4]:

```
import numpy as np

x_vec, w_vec = np.array(x), np.array(w)

z = (x_vec.transpose()).dot(w_vec)
print(z)

z = x_vec.dot(w_vec)
print(z)
```

```
2.2
2.2
```

Interlude: Vectorization

In [5]:
```
def forloop(x, w):
    z = 0.
    for i in range(len(x)):
        z += x[i] * w[i]
    return z


def listcomprehension(x, w):
    return sum(x_i*w_i for x_i, w_i in zip(x, w))


def vectorized(x, w):
    return x_vec.dot(w_vec)


x, w = np.random.rand(100000), np.random.rand(100000)
```

In [6]:
```
%timeit -r 100 -n 10 forloop(x, w)
```

```
38.9 ms ± 1.32 ms per loop (mean ± std. dev. of 100 r
uns, 10 loops each)
```

In [7]:
```
%timeit -r 100 -n 10 listcomprehension(x, w)
```

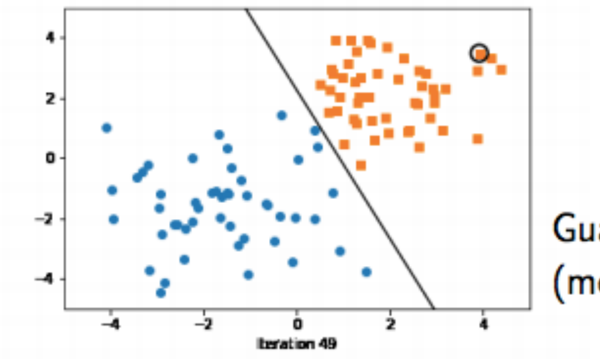
```
29.7 ms ± 842 µs per loop (mean ± std. dev. of 100 ru
ns, 10 loops each)
```

In [8]:
```
%timeit -r 100 -n 10 vectorized(x_vec, w_vec)
```

```
46.8 µs ± 8.07 µs per loop (mean ± std. dev. of 100 r
uns, 10 loops each)
```

The Perceptron Learning Algorithm

- If correct: Do nothing if the prediction if output is equal to the target
- If incorrect, scenario a) If output is 0 and target is 1, add input vector to weight vector
- If incorrect, scenario b) If output is 1 and target is 0, subtract input vector from weight vector



Guaranteed to converge if a solution exists
(more about that later...)

The Perceptron Learning Algorithm

Let

$$\mathcal{D} = \left(\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right) \in \left(\mathbb{R}^n \times \{0,1\}\right)^n$$

1. Initialize w:=0
2. For every training epoch:

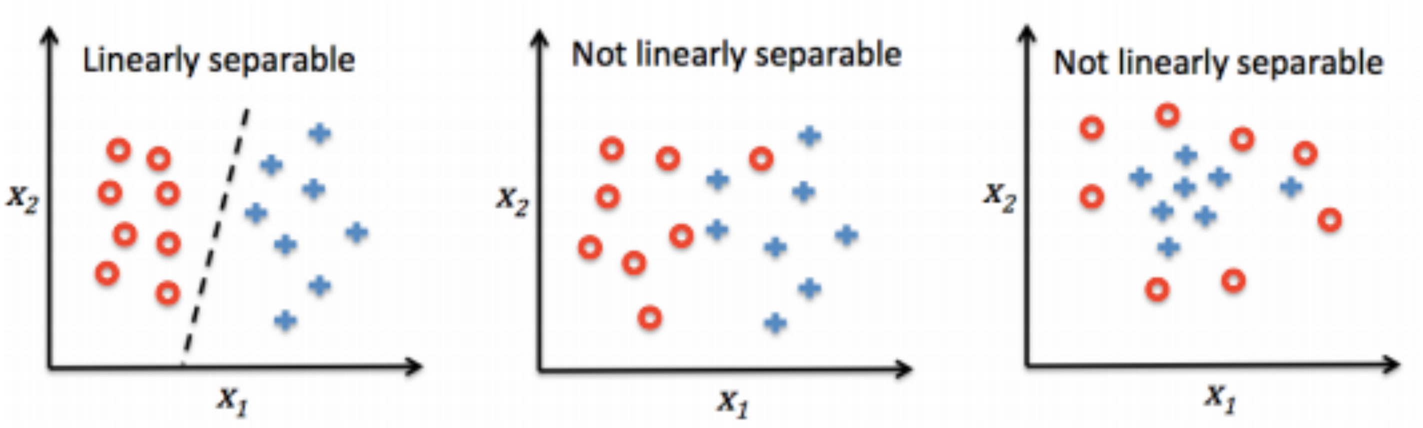
   A. For every $\left(x^{(i)}, y^{(i)}\right) \in \mathcal{D}$:

      (a) $\hat{y}^{(i)} := \sigma\left(x^{(i)T} w\right)$

      (b) $err := \left(y^{(i)} - \hat{y}^{(i)}\right)$

      (c) $w \coloneqq w + err \times x^{(i)}$

Linear Separability



Convergence

Convergence guaranteed if

- The two classes linearly separable
- Learning rate is sufficiently small

If classes cannot be separated:

- Set a maximum number of passes over the training dataset (epochs)
- Set a threshold for the number of tolerated misclassifications
- Otherwise, it will never stop updating weights (converge)

Perceptron Conclusion

The (classic) Perceptron has many problems

- Linear classifier, no non-linear boundaries possible
- Binary classifier, cannot solve XOR problems, for example
- Does not converge if classes are not linearly separable
- Many "optimal" solutions in terms of 0/1 loss on the training data, most will not be optimal in terms of generalization performance

Resources Used

- STAT 479: Deep Learning by Sebastian Raschka
- Python Machine Learning book by Sebastian Raschka